FITPED

Android application development

Ján Skalka
Kristián Fodor

www.fitped.eu

2021

Co-funded by the
Erasmus+ Programme
of the European Union

Work-Based Learning in Future
IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

Android Application Development

Published on

November 2021

Authors

Ján Skalka | Constantine the Philosopher University in Nitra, Slovakia

Kristián Fodor | Constantine the Philosopher University in Nitra, Slovakia

Reviewers

Cyril Klimeš | Mendel University in Brno, Czech Republic

Martin Drlík | Constantine the Philosopher University in Nitra, Slovakia

Eugenia Smyrnova-Trybulska | University of Silesia in Katowice, Poland

Piet Kommers | Helix5, Netherland

Graphics

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

David Sabol | Constantine the Philosopher University in Nitra, Slovakia

Erasmus+ FITPED

Work-Based Learning in Future IT Professionals Education

Project 2018-1-SK01-KA203-046382

Co-funded by the Erasmus+ Programme of the European Union

ISBN 978-80-558-1790-3

Table of Contents

Android

Chapter 1

1.1 Operating system Android

🕮 1.1.1

The Android operating system is:

- designed for mobile devices,
- kernel = Linux,
- since 5.0 is a 64-bit OS.

It was first presented in 2003 and was subsequently purchased by Google in 2007.

Why is OS Android successful?

- open community,
- a significant part of the system is available as open source,
- a number of devices (tablet phone, TV, wearable devices, Google glass, car, etc.),
- support for distributing applications via Google Play,
- alternatives: IOS, Windows, minor OS.

🕮 1.1.2

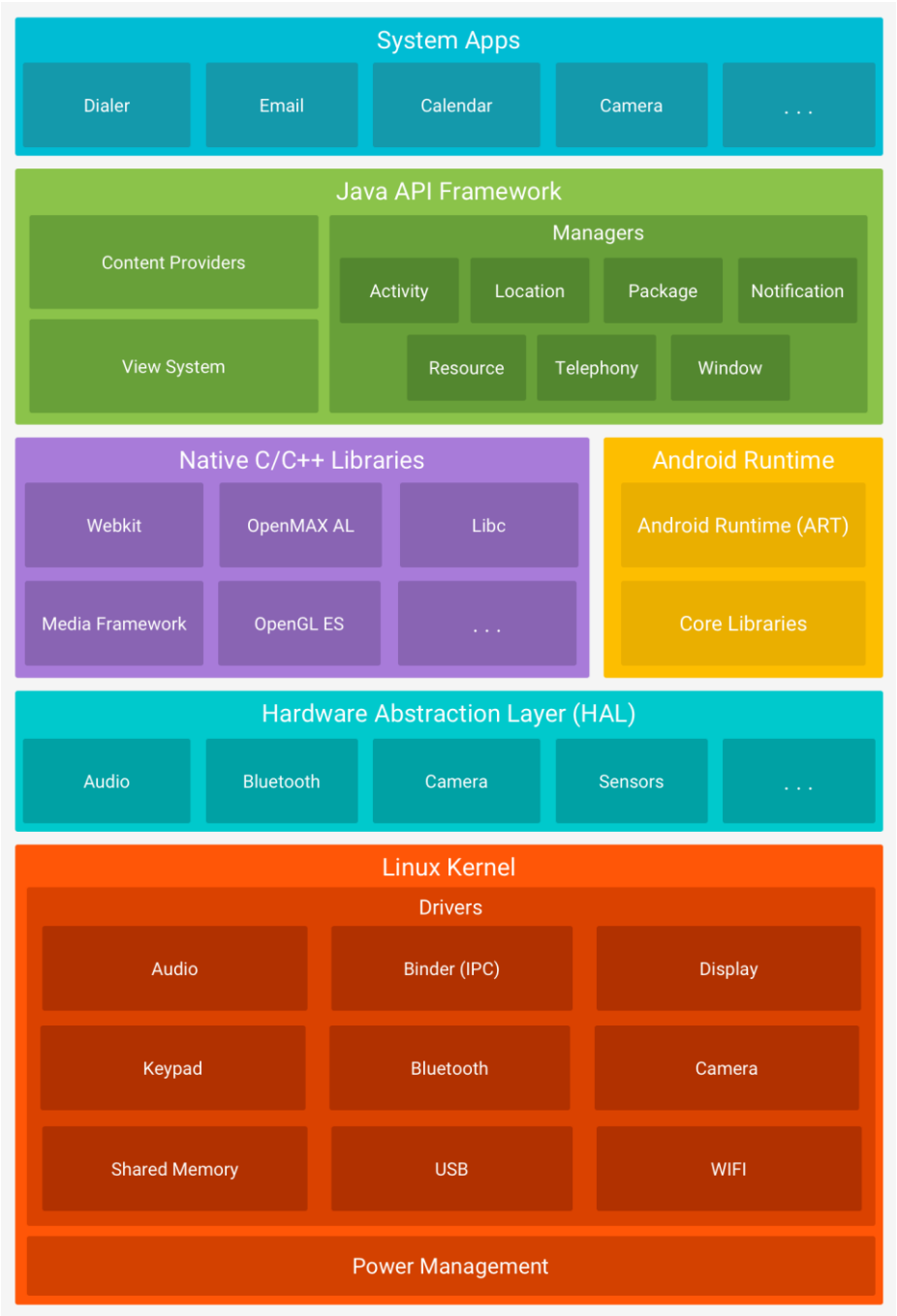
The OS is built on top of the following types of hardware components:

- processors
- CPU (multiple cores), GPU
- DSP (digital signal processor) - analog signal processing - sound, radio, etc.
- **storage**
- RAM
- flash memory (internal storage or SD card)
- transmitter / receiver (transceiver)
- wifi,
- bluetooth,
- near field communication (NFC)
- sensors - detect events from the environment
- movement: accelerometer, gyroscope
- environment: thermometer, touch sensor, pressure, humidity
- position: compass (magnetometer)

🕮 1.1.3

An architecture corresponding to the standard operating system model is built on top of the hardware:

- Linux Kernel
- HAL (Hardware Abstraction Layer)
- Native C / C ++ Libraries + Android Runtime
- Java API Framework
- System Apps



Android OS architecture by v. 4 (source: https://developer.android.com/guide/platform)

🕮 1.1.4

Linux kernel

Android Linux is a variant of GNU Linux. It is written in C and separated from other parts of Android. It is optimized for the needs of mobile devices, expecting different tracks and types of hardware. For other layers of architecture, it mediates through drivers hardware, memory sharing, power management, communication, etc.

Since Android is a Linux-type operating system, it holds that:

- each application is considered a separate user
- each application is assigned a unique Linux user ID, and the system sets the rights for all files within the application so that only that application can access them using its ID.
- application processes run in a separate virtual machine (VM), so application code is separate from other applications,
- each application runs in its own process, which starts when individual components need to be started and closes when it is no longer needed or the system needs to restore storage resources.

Android implements the principle of lowest rights:

- each application has access only to those components that it absolutely needs for its operation and nothing more,
- a well-secured environment is created in which applications do not have access to parts of the system to which they do not have the necessary rights.

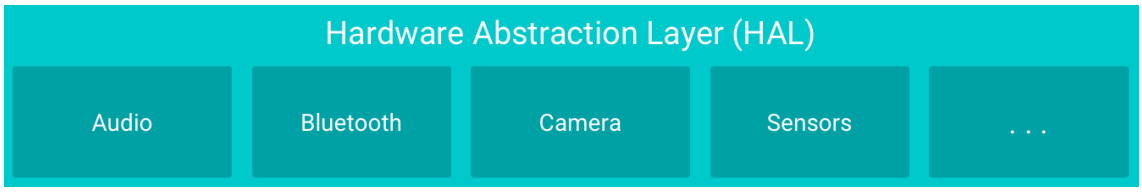
However, there are also options for how applications can share data or access system resources, e.g. the application can request rights to access data such as contacts, SMS, content on the SD card, camera, bluetooth, etc., which must be confirmed during installation.

🕮 1.1.5

Hardware abstraction layer

The hardware abstraction layer (HAL) is an interface that exposes the hardware of the Java API framework. It contains modules created as an interface that adapts to

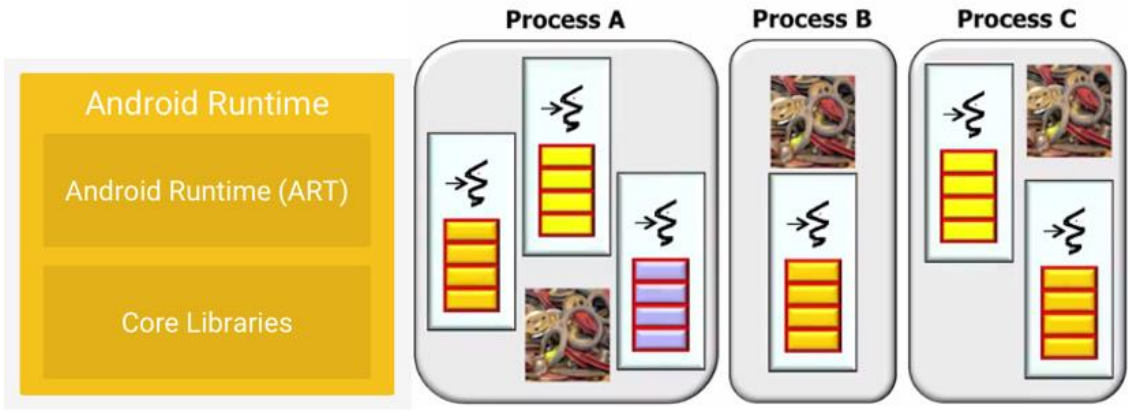
the specific hardware in the device. When the framework requires access to a device, the HAL provides the correct library for the component.



🕮 1.1.6

Android Runtime

The Android Runtime (ART) is a virtual machine that runs each application in a separate process = each application creates its own instances of ART (with the required number of processors threads). ART runs bytecode files with editing for Android.



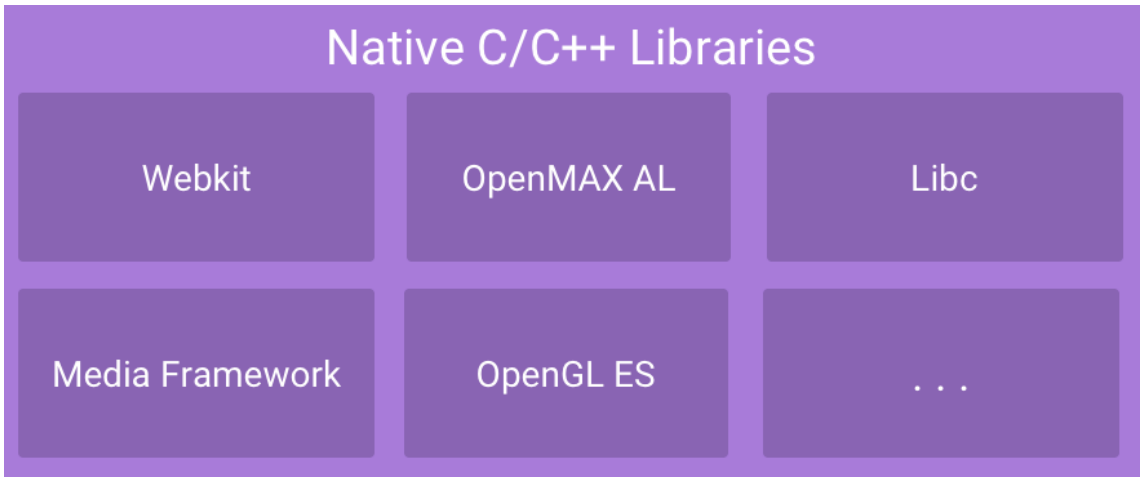
The structure of Android Runtime and the threads in ART.

🕮 1.1.7

Native C / C ++ libraries

The next layer is a group of libraries written in C and C++ used by HAL and ART.

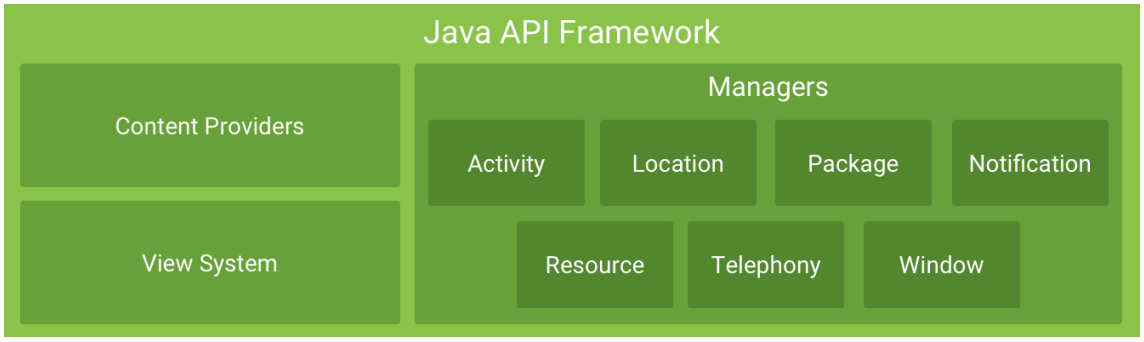
Some of these libraries are also provided by Java Framevork, e.g. OpenGL for 2D and 3D graphics, media playback, etc.



🕮 1.1.8

Java API Framework

The Java API Framework is a level close to the programmer and provides full access to all parts of the system via the Java API. It enables the use of hardware on the device, location services and information, running processes in the background, setting alarms, notifications, etc.



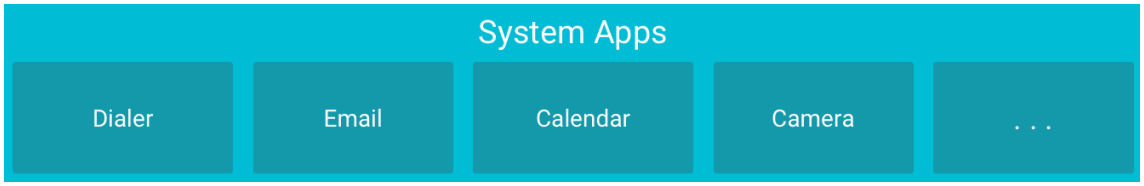
The basic components are represented by:

- View System allows you to create GUI applications using lists, text boxes, buttons, grids, etc.,
- Content Provider will provide access to data from other applications (eg Contacts, SMS…),
- Managers,
- Activity Manager ensures application lifecycle and switching between them,
- Resource Manager allows access to resources not included in the code (localized texts, graphics, sounds and xml files),
- Notification Manager provides access to an expandable panel with notifications and statuses.

🕮 1.1.9

System applications

System (core) applications are usually pre-installed applications or their parts, e.g. keyboard, mail client, SMS, calendar, web browser, etc.



1.2 Applications in the system

🕮 1.2.1

The Android application is a compiled source file packaged together with other resources into a zip file (with the apk extension), while each running application has its own virtual machine.

The basic feature of the OS is the ability to use parts of other applications in the application for the following reasons:

- restrict the creation of the same functionalities,
- simplify and speed up application development.

The application is designed as a package of components from which you can create instances separately. The basic types of components that are used in applications are:

- activity
- service
- broadcast receiver
- content provider

🕮 1.2.2

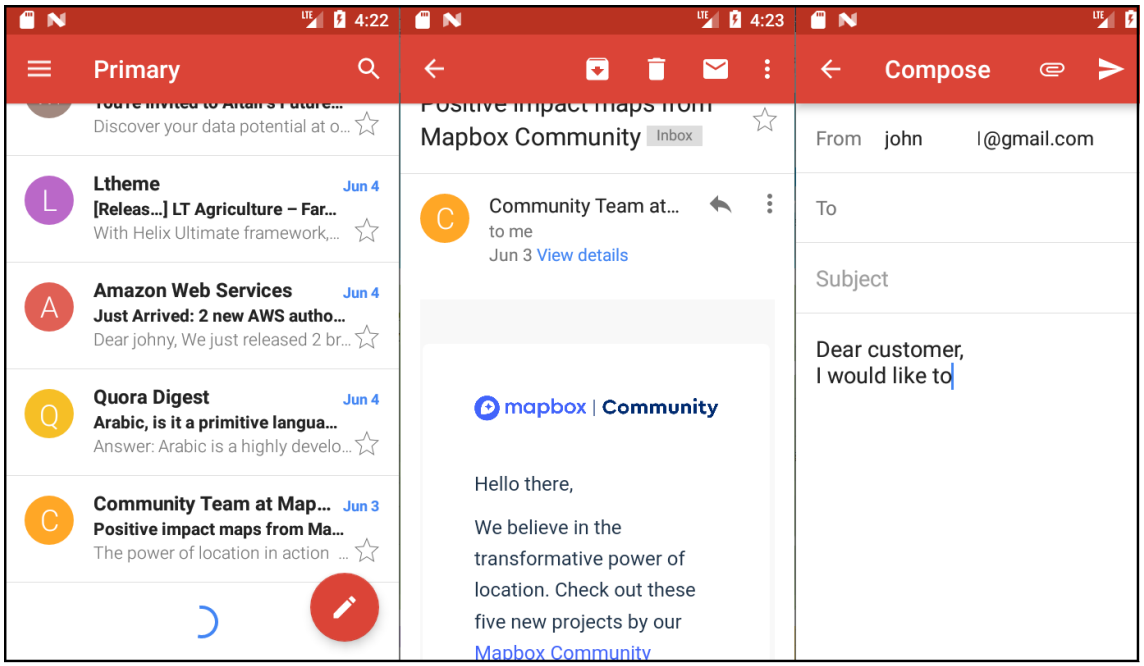
Activity

Activity is a basic visual component. By default, one activity represents one application screen (but it can have a different dimension).

The application consists of activities, e.g.: e-mail client:

- A1: list of received messages

- A2: message content
- A3: writing a message



The application could be programmed within one activity, but it is a bad and much more difficult way. In addition, keep in mind that we can use activities from other applications in our own application.

The content of the window (activity) is defined by rectangular visual elements (views). Some serve as containers (layouts) and may contain other elements. They thus create a certain hierarchical structure. The way elements are laid out determines the type of layout (e.g. horizontal, vertical, free).

🕮 1.2.3

Service

Service is an invisible (non-visual) service running in the background, providing for example:

- download of emails in the background
- sending location information
- checking whether the holder has not entered a restricted zone, etc.

We will discuss the services in more detail later.

🕮 1.2.4

Broadcast recevier

The broadcast receiver is a non-visual component used to process broadcast announcements. In response to an incoming message, it can e.g. run other components that may be system or user components.

Typical examples are:

- responding to a low battery condition starts saving data in an open application
- response to inserting or removing SD card
- receiver responds to an incoming SMS by opening it for reading

Broadcast receivers can also be system or user level (applications can have their own broadcast receivers).

🕮 1.2.5

Content provider

The content provider is used to make data available to other applications, for which it uses instances of the ContentResolver element. The transmitted data can be made available for reading or writing, while the permissions are taken from the permissions of the used applications.

Using a content resolver, it is possible to access e.g. the entries in the address book, photos, records of incoming and outgoing calls, SMS messages, etc.

🕮 1.2.6

Individual components of the application can run other components (from their own or other applications). They activate them through asynchronous messages - **intents**.

The storage of running components (tasks) is secured by a stack of instances of activities that have been started. The following rules apply:

- the newly started activity is stored on top of the tank,
- go back with the back button,
- There can be more tasks (usually for each application).

Android Studio

Chapter 2

2.1 Creating a project

🕮 2.1.1

Android studio is currently probably the most suitable tool for application development in the Android OS.

It is created above the core of the IntelliJ IDEA development environment - thanks to which it was not necessary to develop the whole structure again, just to select suitable parts and "dress" them in a new coat.

Android Studio offers the following benefits over other advanced Java application development environments:

- easier application localization,
- more convenient design creation - simultaneous display of xml and "graphics",
- built-in emulators with configurable parameters,
- Simultaneous preview of the application in several resolutions,
- responsive IDE,
- support for a number of devices (wear, google glass, car, etc.).

The development environment can be obtained from:
https://developer.android.com/sdk/index.html

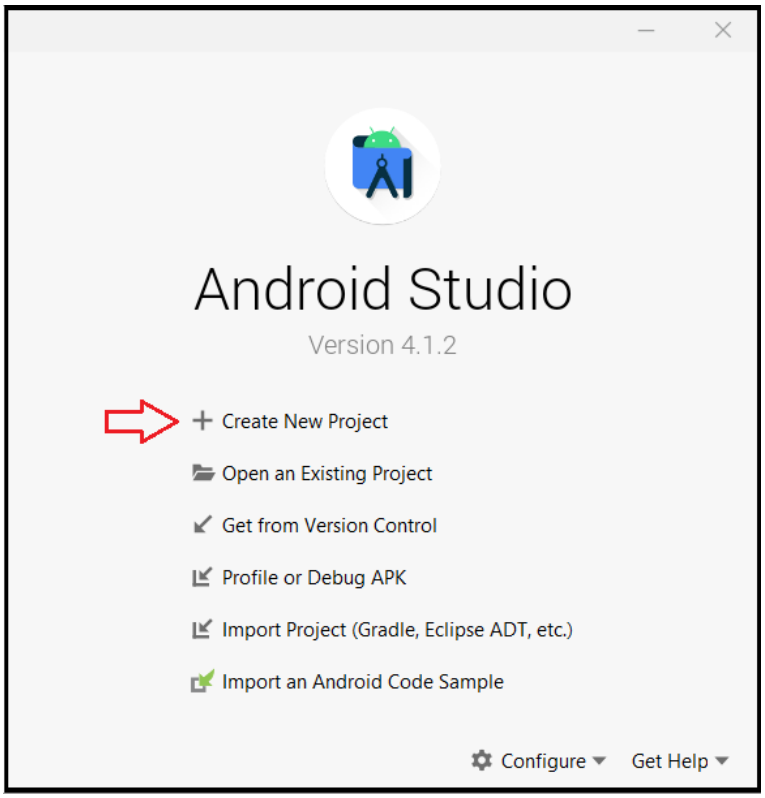
After the first installation, it will begin updating for a relatively long time, while further updates must be installed when creating emulators.

🕮 2.1.2

Create an application to display a greeting

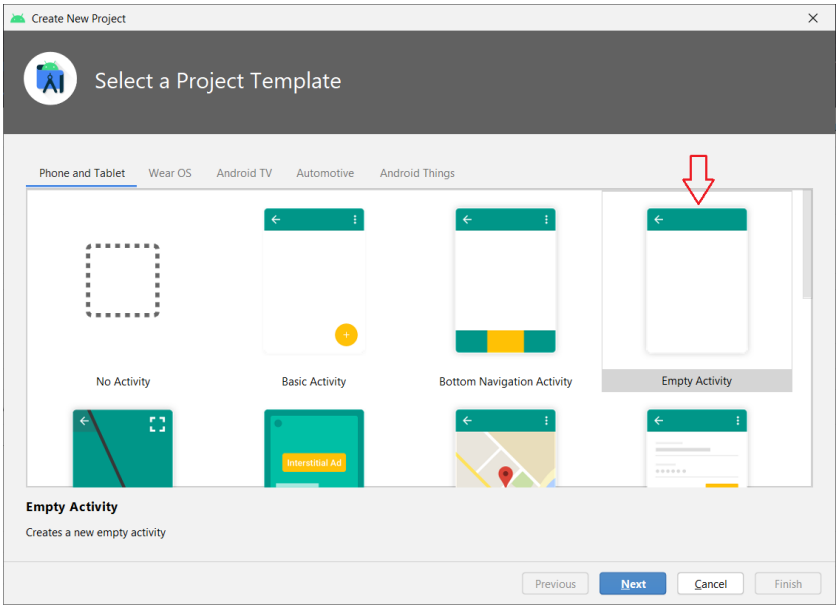
The first step in getting to know a new development environment is usually the task of displaying text in an application.

When Android Studio starts, it displays a basic menu in which we select a new project.



2.1.3

In the next step, we will select the default appearance of the application (activity). We have several templates available for different types of applications, **<empty activity>** will now be the most suitable for our needs.

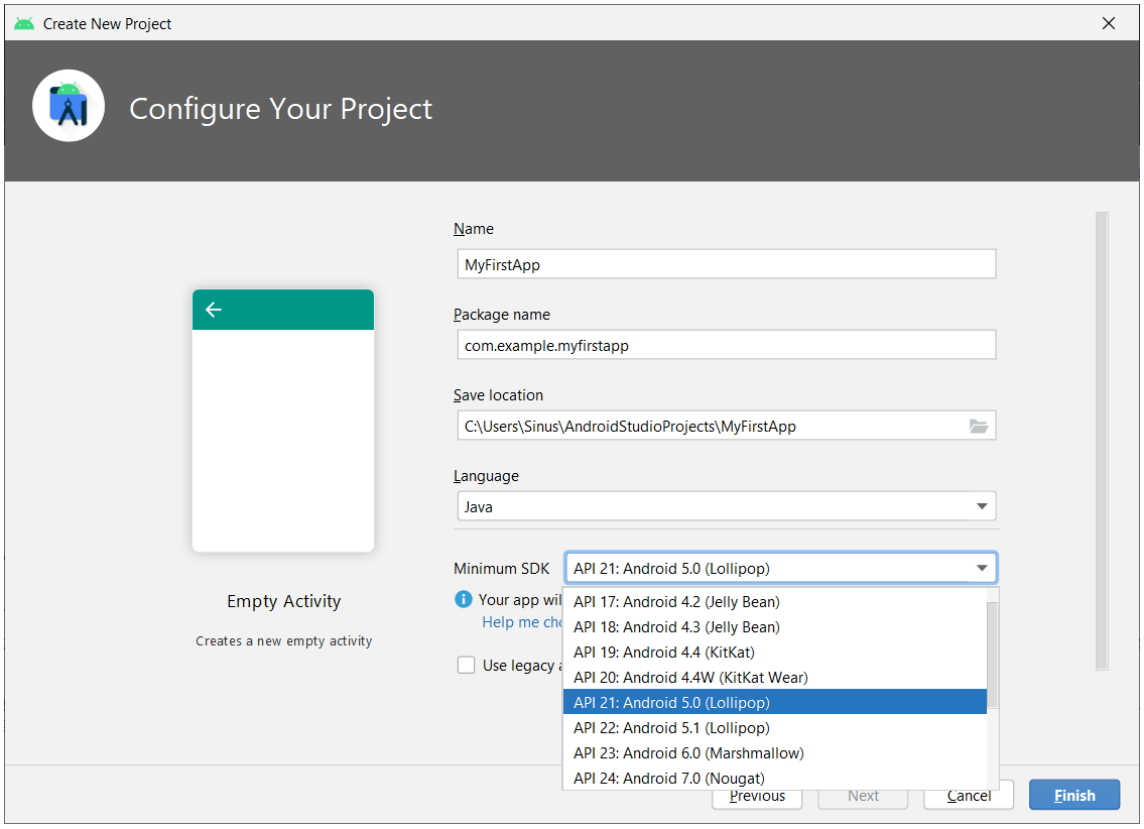


2.1.4

In the next step, we will enter the application name, the default package and the disk location.

An important step is the version of the API we want to use. The higher the version we select, the more modern elements of the operating system we can use, but on the other hand it will not be possible to run the application on devices with an older version.

The minimal version, which has at least some of the modern features, is Android 4.4. Applications created for this version will run on 98.1% of Android devices. If the application is to use a modern design (Material design), it is necessary to choose API 21 (Android 5.0) as a minimum, where we will still cover up to 94.1% of existing devices. Unless otherwise stated, API 21 will suffice for our needs.

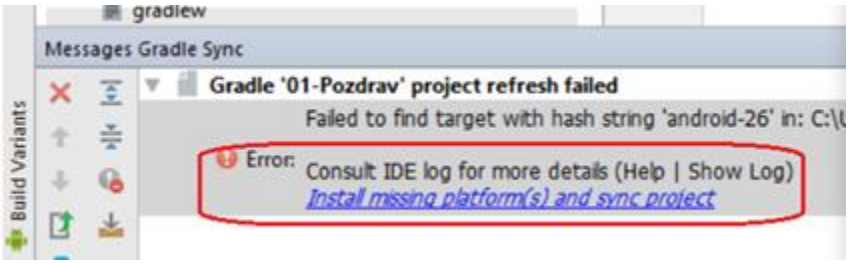


2.1.5

After building the project structure through **Gradle**, we have a development environment available.

Gradle is a build system that creates the structure of the application as well as a standalone application. It's hidden inside Android Studio, we are not interested in it yet.

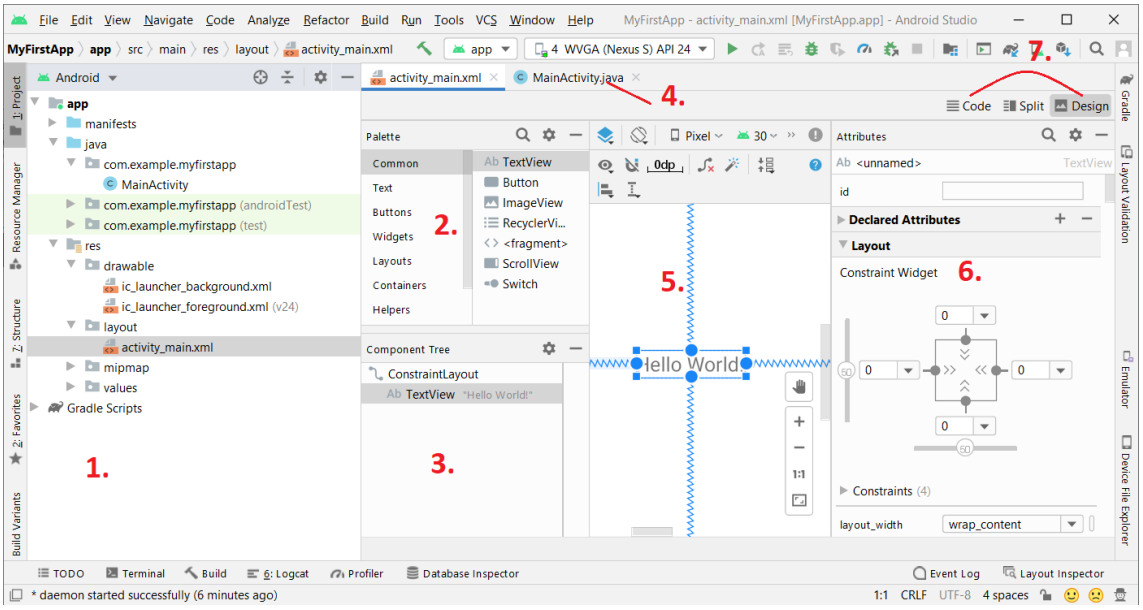
It is often necessary to install various packages the first time you run Android Studio, without which the compiler and builder report errors.



2.2 The structure of the development environment

📖 2.2.1

The basic parts of the development environment can be seen in the picture:



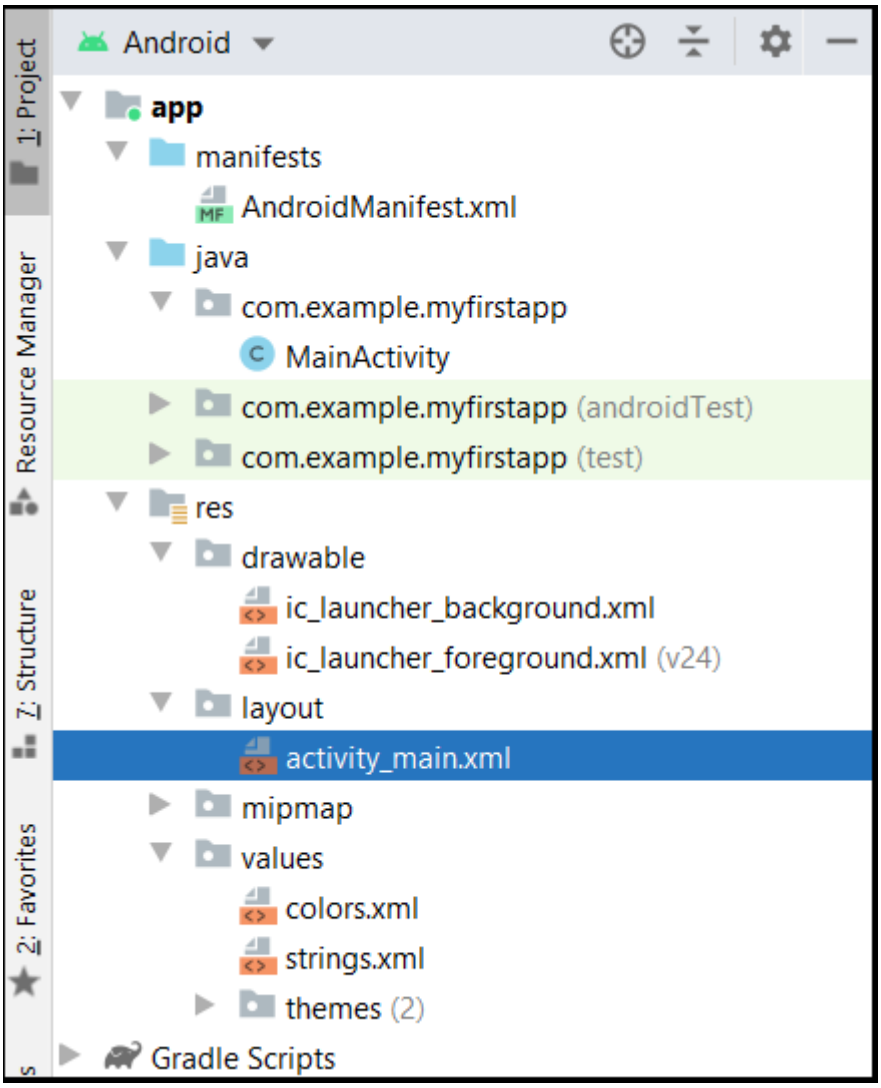
1. project structure
2. components
3. list of components used
4. application source code
5. the current appearance of the application
6. component parameters
7. switching between visual design and its written form using xml commands

An important feature of Android Studio is switching between displaying the design and the xml file with its definition. We will also have to get used to the fact that the appearance must be defined separately for each window (activity) and we write the service code in Java in a different file.

The other parts are similar to other development environments designed for RAD (Rapid Application Development).

🕮 2.2.2

The structure of the projects



The basic parts of the project are expanded in the picture and described in the following list:

- AndroidManifest.xml
- file with application properties and settings
- java - MainActivity
- source code for the relevant activity (window)
- res - folder for resources used by the application
- drawable - a list of application images, usually bitmaps
- layout - group for appearances of individual activities

- layout - activity_main.xml - contains the source code of the activity design
- values - defined constants used by the system e.g. for easier localization into other languages
- values - strings - values and definitions of resources (variables), the most common use in localization

🕮 2.2.3

Application

The application itself is represented by the Activity class, respectively **AppCompactActivity**, which takes into account the possibilities of use in various resolutions. Our activity is a descendant of this class.

```
package com.example.myfirstapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

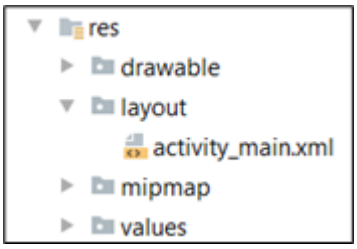
The **MainActivity** class contains mainly the **onCreate()** method, which is executed as one of the first when the application is started and provides:

- creattion of an instance (**super.onCreate()**),
- basic appearance setting (**setContentView**).

🕮 2.2.4

Appearance of the application

R.layout.activity_main is a link to the content of the so-called R-class that contains resource identifiers and is generated based on defined layouts just before the application is started. Using this class, we access individual resources or layouts - it is a "short" way of referring to the resources used.



In our program, we refer to the **activity_main.xml** file, which contains the definition of the main screen of the application.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

The content of the file is a list of defined elements, their settings and relative position to each other.

- **layout_width** and **layout_height** determine the width and height of the view:
- **match_parent** - expands the view to the maximum value allowed by its parent
- **wrap_content** - view takes the minimum value it needs to display its content
- **TextView** - a component stored in the activity
- **android: text** - defines the text content of the component
- /> - terminates the component - in this case **TextView**

🕮 2.2.5

Manifest

An essential part of the application is the definition of settings for its execution contained in the manifest.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myfirstapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyFirstApp">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

The first part defines the icon, name and appearance style of the application. Subsequently, a single activity (MainActivity) is defined, which is introduced first. The intent-filter defines to which message the activity is able to respond, and finally the combination of action and category sets that the activity is the entry point of the application and that it will be visible in the list of applications.

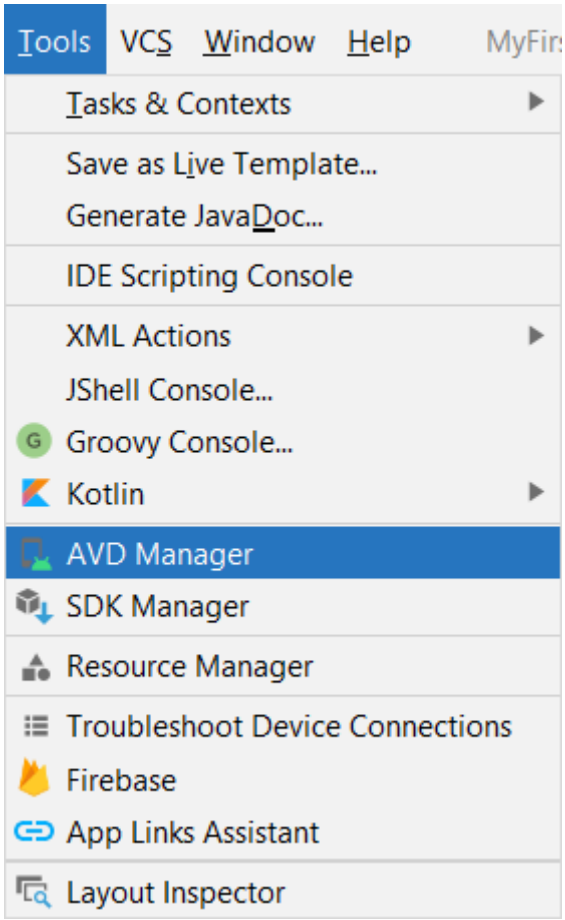
We don't have to change the manifest by default, Android Studio will set it up for us.

2.3 Emulator

🕮 2.3.1

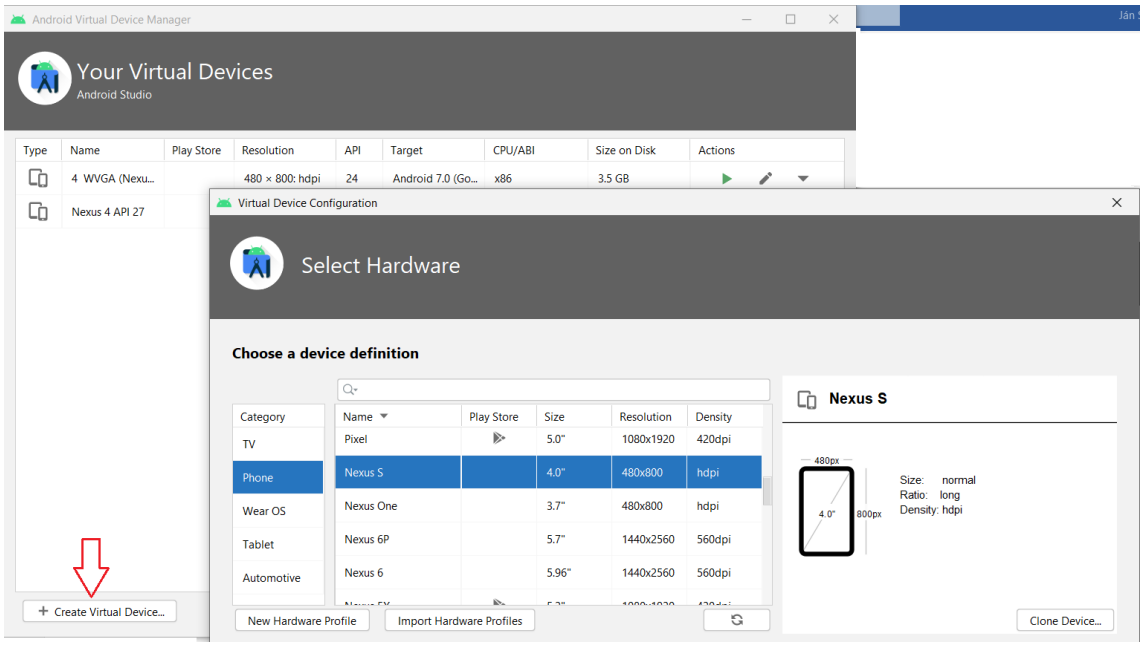
For the needs of debugging and testing the application on several devices, we can define a number of emulators and run the application on them. **Android Studio** from version 2 has an improved emulator - it can be used on weaker devices.

After the first start it is advisable to create one or more emulators and then after starting the application we can choose which one to use. We define the emulator in the **Android Studio** menu.



🕮 2.3.2

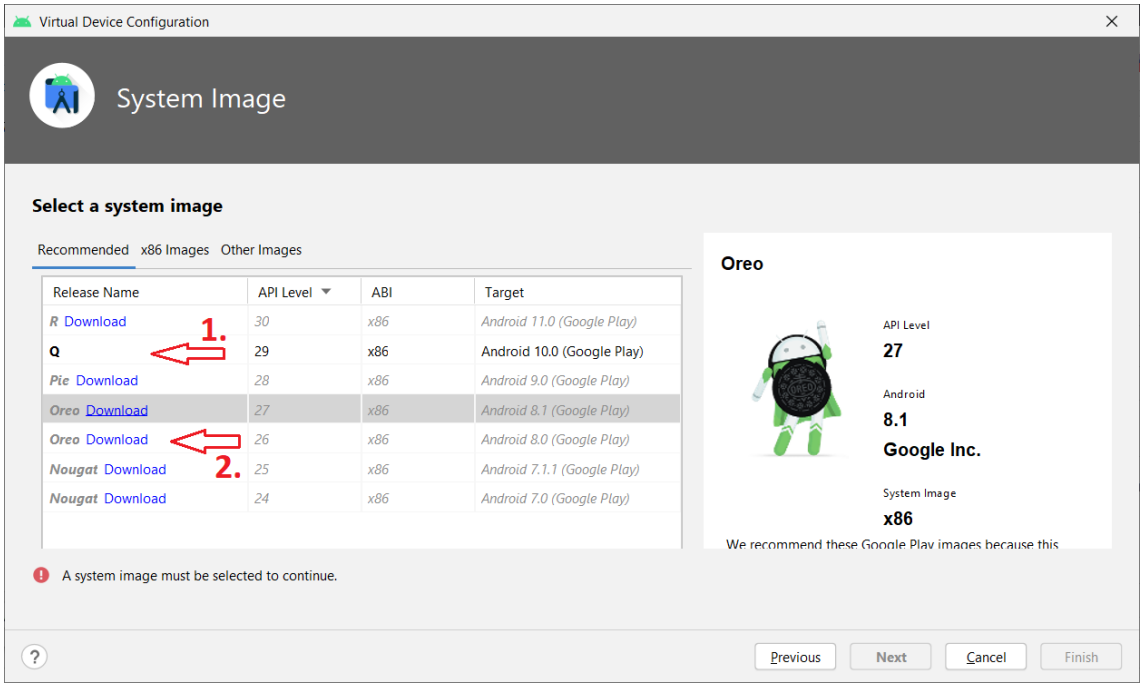
In the displayed window we can select from existing or create new devices.



For initial debugging and verification of the application's functionality, we recommend a lower resolution and poorer performance of the emulated device.

2.3.3

After adding a new device, we may not have an image of the appropriate version of the device's operating system (the appropriate API) and it needs to be downloaded.

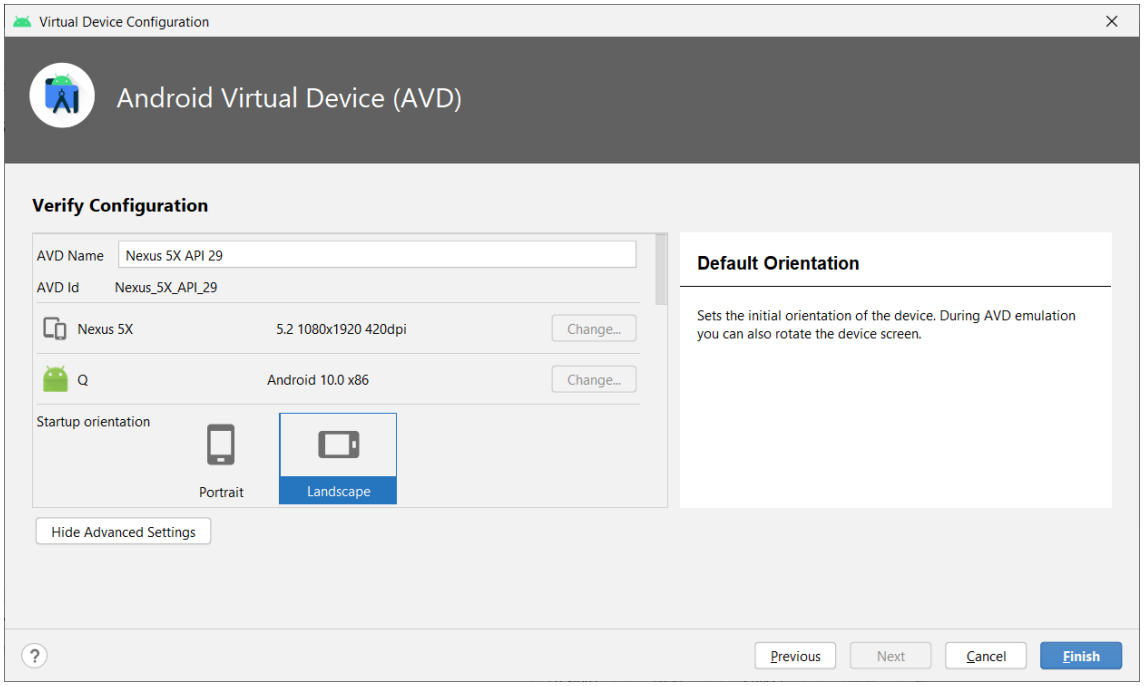


1. the image of the operating system version has been downloaded, we just need to select it,

2. the operating system version image is not available on our computer, we need to download it.

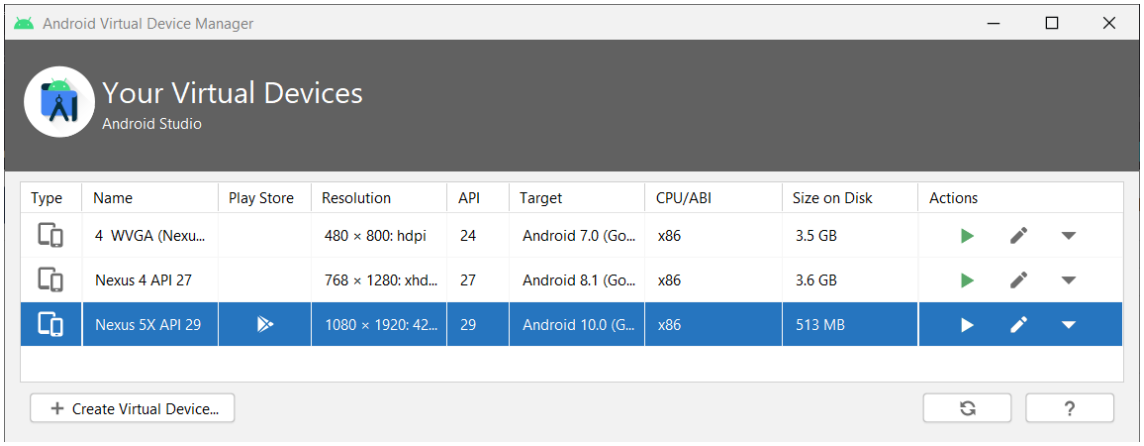
2.3.4

After setting or downloading the operating system image, we can set several device parameters ...



2.3.5

After adding the created device to the list of available devices, we can e.g. double click to start.

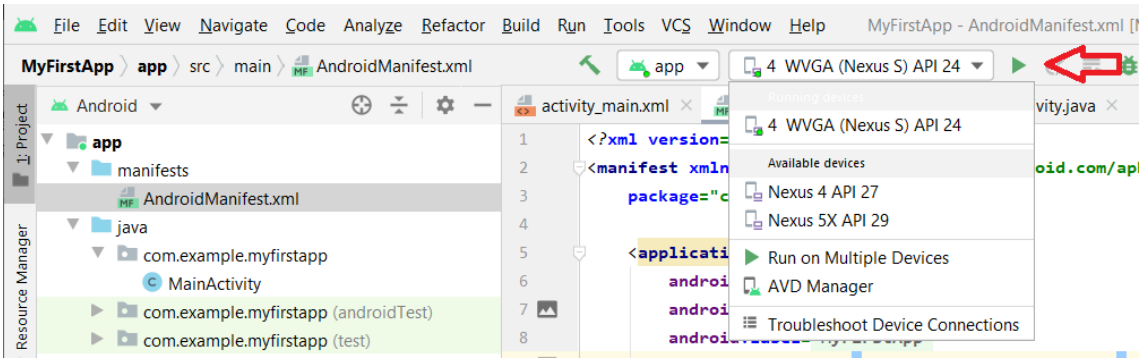


🕮 2.3.6

It usually takes a while for the emulator to start, so we don't recommend turning it off when debugging the application. Restarting the application always reloads it into the emulator and restarts it.

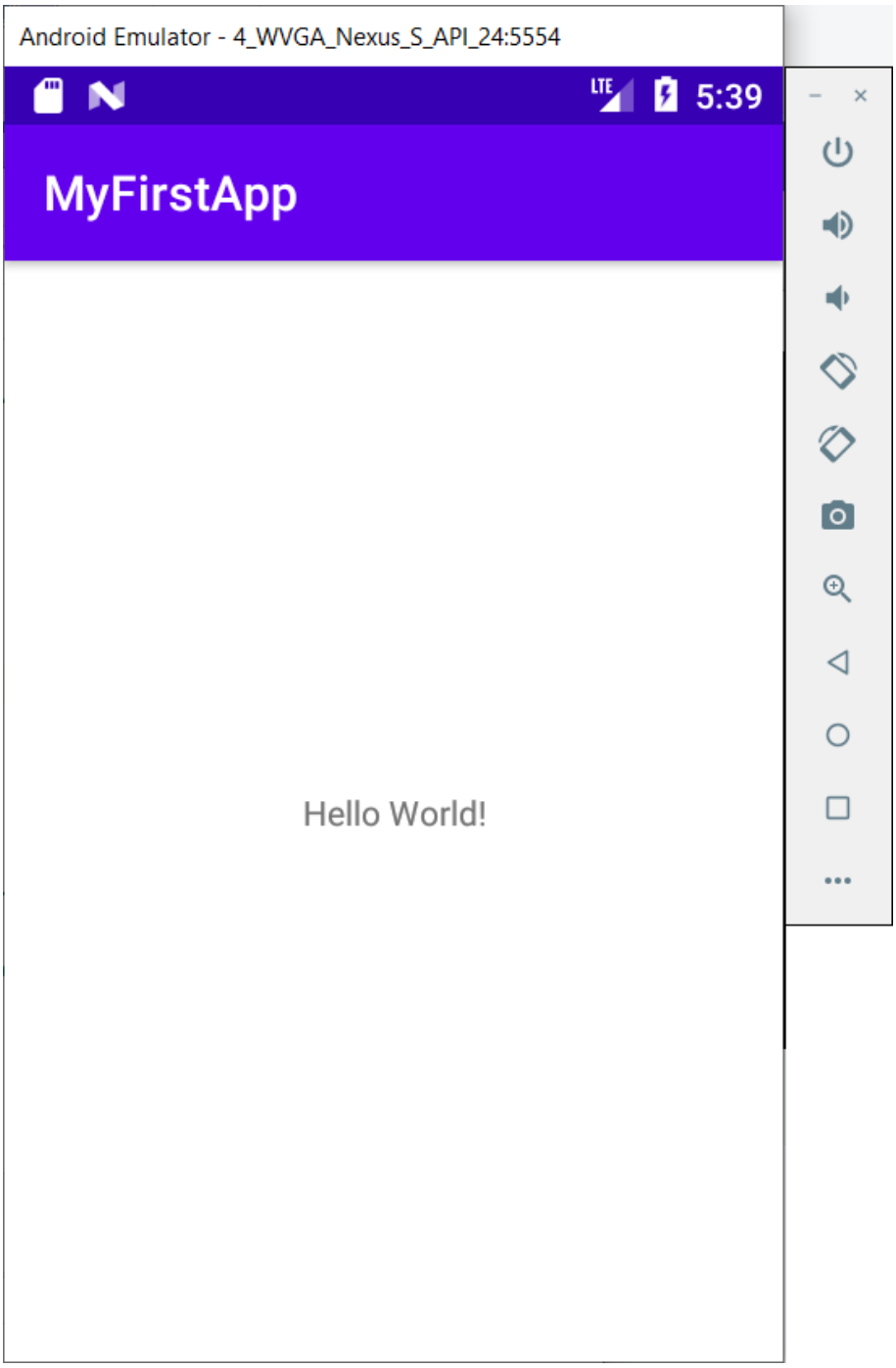
We run the application itself via the menu, by clicking on the appropriate icon, or we can select the emulator on which we want to run it.

If we have a physical device connected, this will also appear in the menu.



🕮 2.3.7

We didn't even write a line of code in the application, but we still completed the assignment. The authors of **Android Studio** have prepared an application with Hello World message as the default when creating an empty application.


Android Emulator - 4_WVGA_Nexus_S_API_24:5554
5:39
MyFirstApp
Hello World!

User Interface

Chapter 3

3.1 Input processing

🕮 3.1.1

Assignment:

Create an application in which:

- the user enters content in the text box,
- which is written backwards in the **TextView** component.

🕮 3.1.2

Design proposal

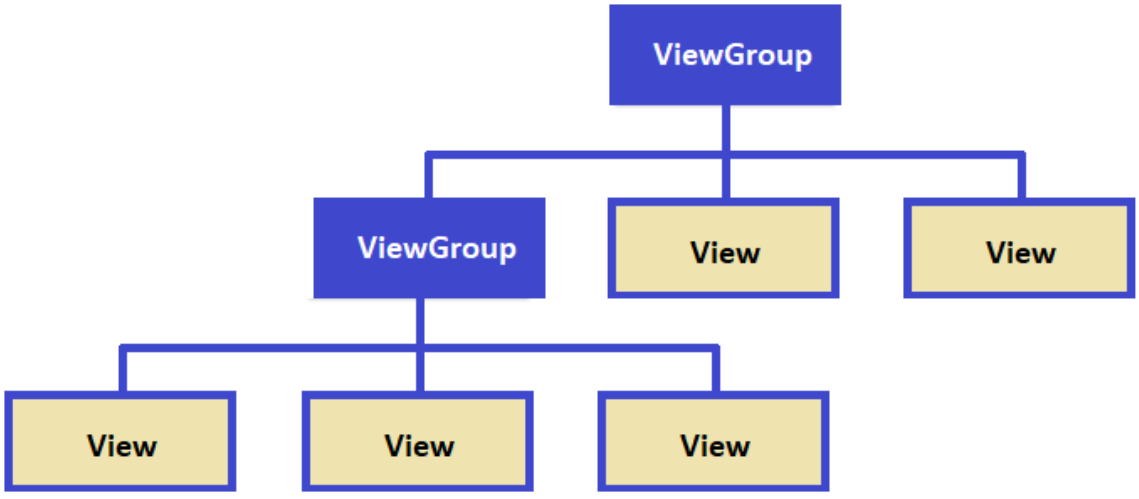
When creating a mobile application for Android, we usually start by designing it so that we can reference the relevant elements in the code.

The design of application activities can be created:

- by editing the xml,
- switching to **Graphical Layout** and inserting components with the mouse,
- via code.

We have two main groups of elements available for creating a design:

- **View** - a simple component,
- **ViewGroup** - a component that can contain other components



🕮 3.1.3

The components must be stored in containers in some way in the activity. Due to the different resolutions of the devices, several types of layouts are defined, which prescribe in which order the elements are stored as a priority if they have enough space (next to each other, below each other, etc.). By default, **Constraint Layout** is set as the default, which defines the position of elements relative to other elements.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

In the following text, we will refer to the components in the Android language - **view**.

🕮 3.1.4

Linear layout

After creating a new application, we will proceed to modify the design.

The simplest code suitable for our beginner's needs generates a **linear layout**, which ensures the placement of elements below or next to each other, regardless of the size of the screen. We have two alignments available: horizontal or vertical.

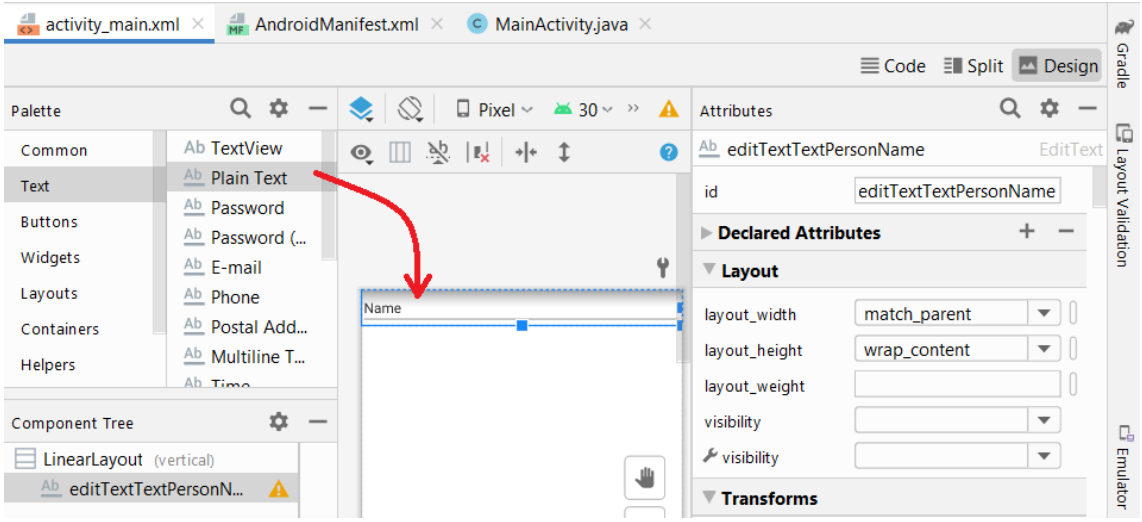
In **activity_main.xml** we change the layout to **Linear**, while the attributes **android:layout_width** and **android:layout_height** specify the size of the activity. We can exclude the automatically inserted **TextView** element from the design.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
</LinearLayout>
```

3.1.5

The text field represents the **PlainText view** element. Dragging and dropping into the activity layout places it in the top row.



3.1.6

As a result, the following will be added in the **xml** layout:

```
<EditText
    android:id="@+id/editTextTextPersonName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Name" />
```

- **id** - identifier of the **view**, through which we refer to the **view** in java code
- @ is always used when referring to the source listed in **xml**,
- + says that we are creating a new resource, it is no longer used in other places when referring to it
- **editTextTextPersonName** is the name of the **view** by which we refer to it - we can easily rewrite it for an easier name
- **layout_width** and **layout_height** define the dimensions of the **view**
- match_parent according to the size of the parent view in which it is located
- wrap_content - according to the text it contains

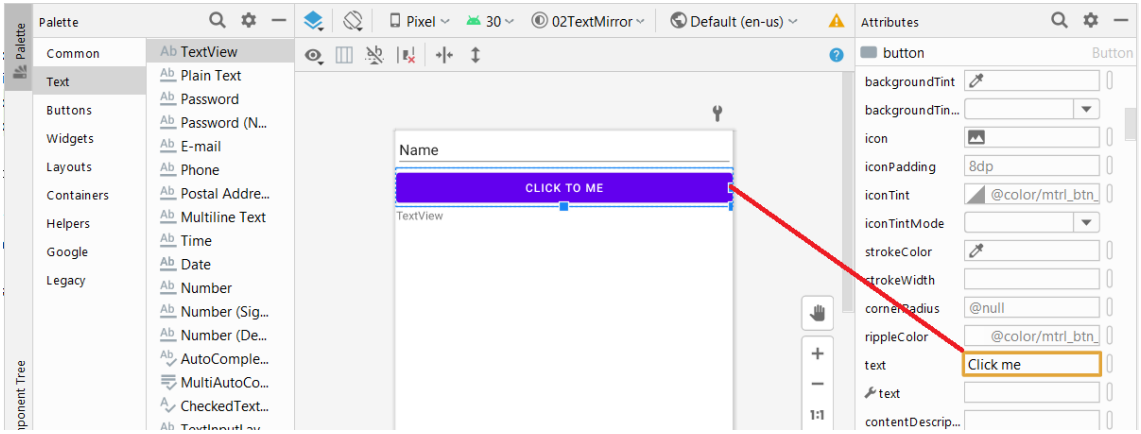
- **inputType** specifies the type of data that can be inserted into the field - in this case it is alphanumeric text (defined according to a more detailed **textPersonName** template)
- the **text** represents the content that appears in the field.

🕮 3.1.7

Then we add the **Button** element to the activity. The elements are stored below each other thanks to the fact that we have defined a vertical layout.

Finally, we will add a **TextView** into which we will insert the output.

We will overwrite the text content of the inserted items via the view properties (text item) or in the xml layout file.



🕮 3.1.8

In order to be able to refer to the **TextView** into which we want to insert the result, it is necessary to check whether it has a naming in **xml**. The layout can finally take the form of:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="" />

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click me" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="--- TextView for result ---" />

</LinearLayout>
```

3.2 Events

📖 3.2.1

The reaction of interacting (touching) a button can be defined in two ways:

- by adding an **onClickListener** to the button in the source code,
- by defining a method responding to clicks in the layout **xml** file.

📖 3.2.2

The first option is to embed the method in the code in the following form (you will find a description of the individual commands directly in the code):

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // we identify the button we want to activate
        Button b = findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {// for the onClick
event we define ...
                // we find an element named editText - it's
View,
                // so we will redefine it to EditText for
better work
                EditText ta = (EditText)
findViewById(R.id.editText);
                String a = ta.getText().toString(); // we read
what it contains
                // we find the element named textView and
redefine it to the type TextView
                TextView tv = (TextView)
findViewById(R.id.textView);
                String b = "";
                // we mirrored the input
                for(int i=0; i<a.length(); i++) b =
a.substring(i,i+1) + b;
                tv.setText(b); // we write the result
            }
        });

    }
}
```

⌨ 3.2.3 Source codes of the first solution

MainActivity.java

```
package com.example.a02_textmirror;

import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // we identify the button we want to activate
        Button b = findViewById(R.id.button);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {// for the onClick
event we define ...
                // we find an element named editText - it's a
View,
                // so we will redefine it to EditText for
better work
                EditText ta = (EditText)
findViewById(R.id.editText);
                String a = ta.getText().toString(); // we read
what it contains
                // we find the element named textView and
redefine it to the type TextView
                TextView tv = (TextView)
findViewById(R.id.textView);
                String b = "";
                // we mirrored the input
                for(int i=0; i<|a.length(); i++) b =
a.substring(i,i+1) + b;
                tv.setText(b); // we write the result
            }
        });

    }
}
```

ActivityMain.xml

```
<|?xml version="1.0" encoding="utf-8"?>
```

```
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click me" />

    <|TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="--- TextView for result ---" />

<|/LinearLayout>
```

AndroidManifest.xml

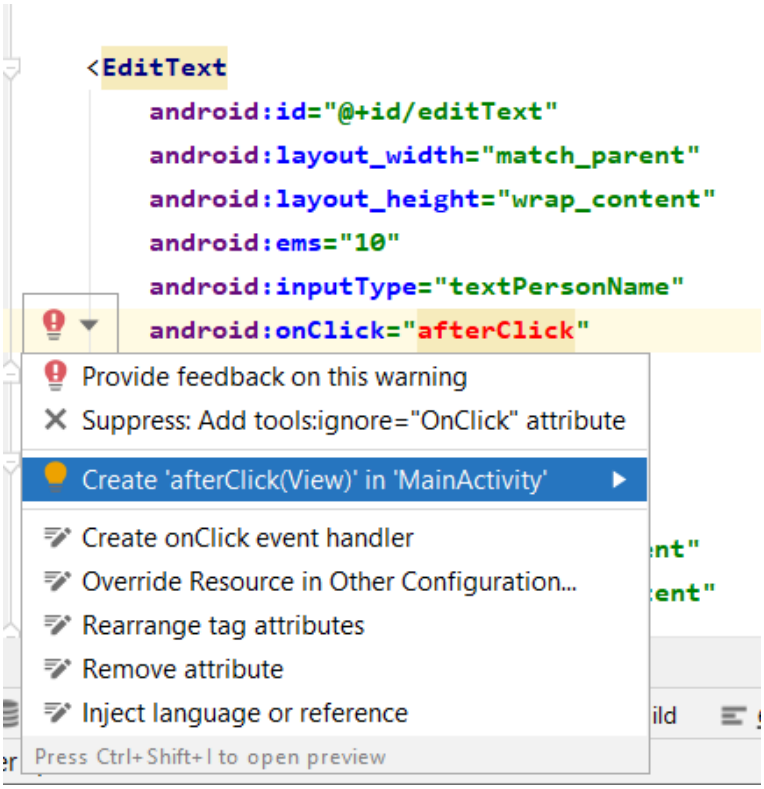
```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a02_textmirror">

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
```

```
        android:supportsRtl="true"
        android:theme="@style/Theme.02TextMirror">
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
    <|/application>

<|/manifest>
```

3.2.4

The second option is a bit clearer for a beginner.

The name of the method we want to call after clicking on the button can be defined in the **xml** file using the **android:onClick** attribute, and then this method is implemented in the activity that contains the button. Either we define its header manually or by clicking on the alert icon we leave the environment to create its empty body.



We will then add the necessary code in the application code located in the **MainActivity.java** file.

```
public void afterClick(View view) {
    // we find an element named editText - it's a View,
    // so we will redefine it to EditText for better work
    EditText ta = (EditText) findViewById(R.id.editText);
    String a = ta.getText().toString(); // we read what it
contains
    // we find the element named textView and redefine it to
the type TextView
    TextView tv = (TextView) findViewById(R.id.textView);
    String b = "";
    // we mirrored the input
    for(int i=0; i<a.length(); i++) b = a.substring(i,i+1) +
b;
    tv.setText(b); // we write the result
}
```

And our first communicating application is ready.

⌨ 3.2.5 Source code of the second solution

MainActivity.java

```
package com.example.a02_textmirror;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

```
    public void afterClick(View view) {
        // we find an element named editText - it's View,
        // so we will redefine it to EditText for better work
        EditText ta = (EditText) findViewById(R.id.editText);
        String a = ta.getText().toString(); // we read what it
contains
        // we find the element named textView and redefine it
to the type TextView
        TextView tv = (TextView) findViewById(R.id.textView);
        String b = "";
        // we mirrored the input
        for(int i=0; i<|a.length(); i++) b =
a.substring(i,i+1) + b;
        tv.setText(b); // we write the result
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:onClick="afterClick"
        android:text="" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click me" />

    <|TextView
```

```
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="--- TextView for result ---" />

<|/LinearLayout>
```

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a02_textmirror">

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.02TextMirror">
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
    <|/application>

<|/manifest>
```

3.3 Intents

🕮 3.3.1

Create an application that has two activities and it:

- displays the button in the first activity
- after clicking a button, it shows the second activity where the users can enter their username and confirm it
- then it returns to the first activity and greets the user.

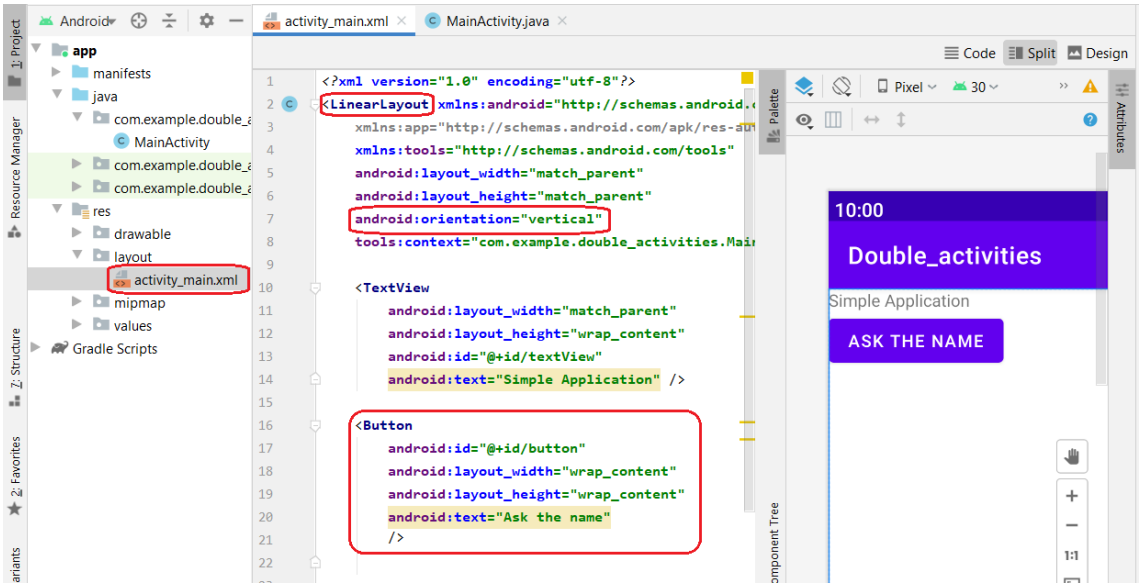
3.3.2

Activity layout

The appearance (design, layout) of the activity can be created in several ways:

- by editing the **xml** in the file associated with the activity and stored in the **res** folder
- by switching the display of the **xml** file to **Design**, where we see the appearance of the activity
- through code in a **java** file without affecting the content placed in the xml design.

Due to our agreement to create the simplest possible design, we will combine rewriting the **xml** (we rewrite the layout to **LinearLayout**) and inserting **views** in the design view (drag to insert the **Button** element).



The result can be as in the picture.

3.3.3

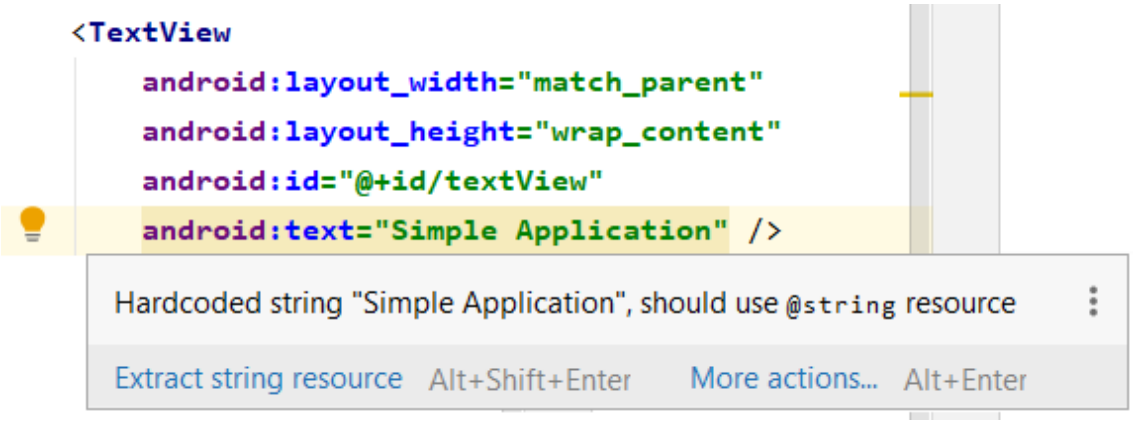
Multilingual applications

If you are creating applications that will be localized into other languages, it is advisable not to insert texts displayed in the system's views directly, but to use variables, resp. constants. The advantage of this approach is that when localizing, we only need to rewrite the data in one place or create a separate file of texts for each language. This is an approach commonly used in the development of other types of applications.

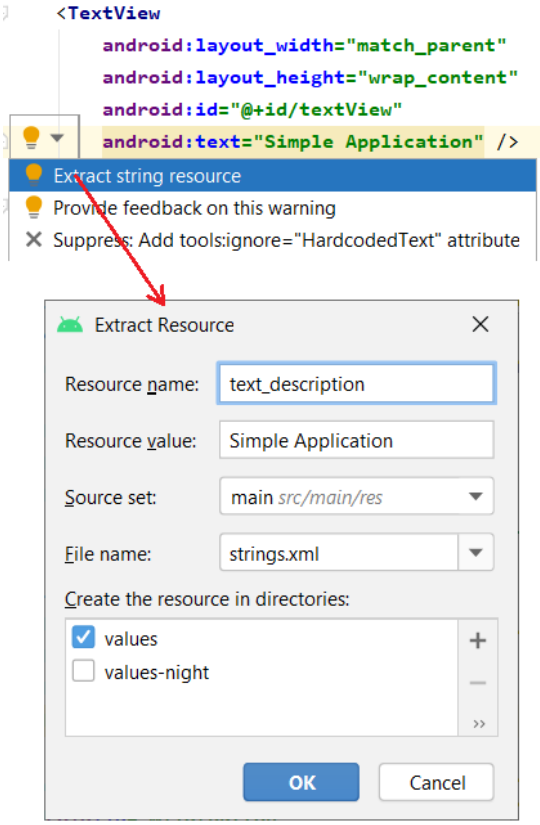
Although this is a common approach, we will not use it in the following text for the sake of code comprehensibility. However, we recommend that you follow it when creating applications to be used in multiple languages.

🕮 3.3.4

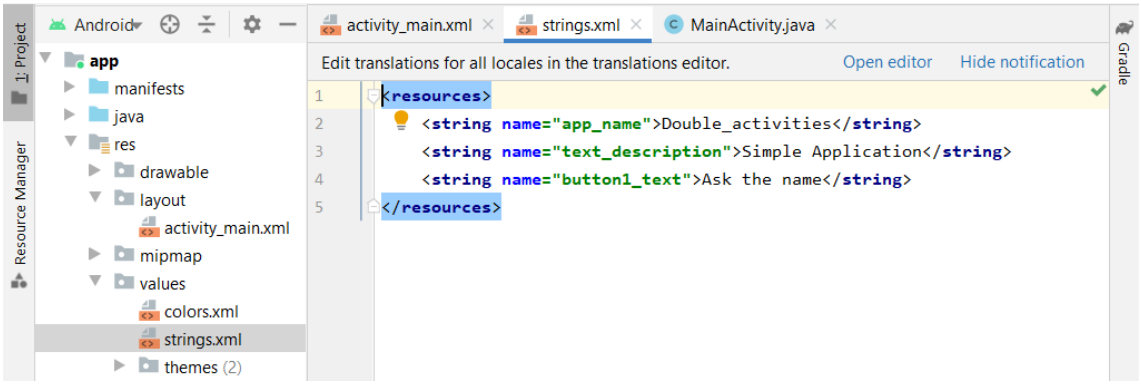
What would our application look like if we followed this rule?



By clicking on the **hardcoded string**, a light bulb will appear indicating the possibilities of solving a non-standard approach, from which we can select the first item (**Extract string resource**), which transforms the fixed **string** into a variable with a name that we enter so that in the future we will know which string it represents.



At the same time, we specify the name of the source in which the variable-value pair will be stored and with which we will refer to it. This source is usually set to values and can be found in **res-values-strings**.



Overwriting the values in the **resource** section also automatically changes the content of the linked components.

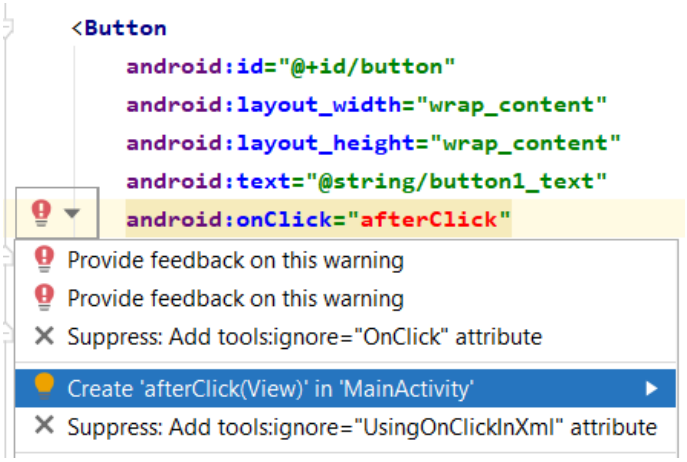
In addition, the code of the design is modified so that the strings in the **views** reference the variable defined in the **values** section.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:text="@string/text_description" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button1_text"
/>
```

📖 3.3.5

Processing of clicks

We expect the button we have defined to display the next activity, therefore we define the code that will be called after a click. We will use the system menu to create a method which name we have defined for the behavior of the **onClick** event.



Selecting the **Create** ... option creates a method in the activity code associated with the layout (**MainActivity**) and ensures that the event is linked to it.

```
public void afterClick(View view) {
    Intent intent = new Intent(MainActivity.this,
NameActivity.class);
    startActivityForResult(intent, 1);
}
```

To bring up a new window, we will create a new message - **intent**, which has two parameters:

- the first parameter is the message source (**MainActivity.this)**, i. e. the activity from which the message originates,
- the second parameter is the class to be created, in this case the activity to be run (**NameActivity.class**)

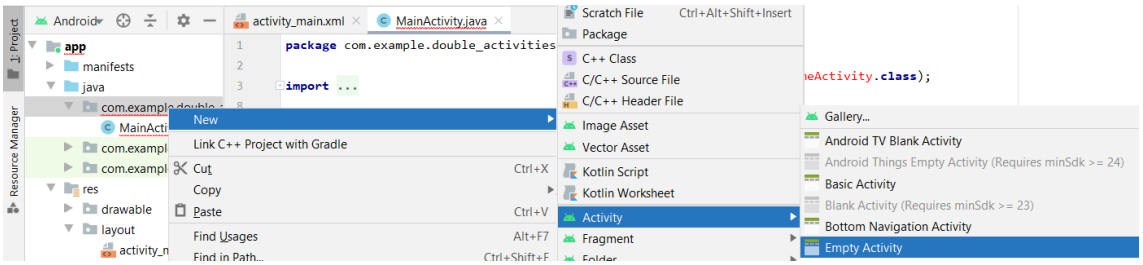
After creating the message, we send it with the expectation of the result (**start Activity For Result**). The function contains as parameters an intent defining what is to happen and a return identifier, on the basis of which we later identify that this is the answer to this request. If we called other activities from the activity, each should have a different identifier.

🕮 3.3.6

The second activity of the application

In the application, we do not need to distinguish whether the activity is the main or secondary, it is still just an activity that we create in a standard way. We need to create a new activity that we will use to enter the name.

We will create it via the context menu in the **java** package.



After rewriting the name in the wizard window and confirming, a new **java** file will be created in the java package and a new design file in the **res-layout** section.

3.3.7

In the created activity, we define the design and modify the content of the button in a known way. The activity will contain a **Plain Text** to enter the name and a **Button** to confirm the entry and close the activity.

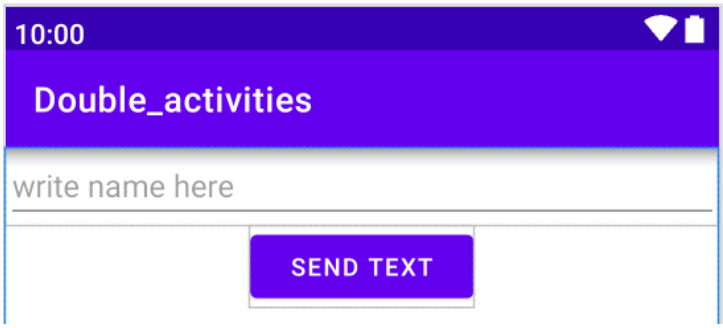
When using a linear layout, we recommend that you change the layout first and then insert the **views**.

We can also try some new features:

- define a **hint** in the text field that will be lost when you start typing a text,
- define the width of the button using **layout_width** or just simply resize it with the mouse,
- define the alignment of the button to the center of the screen by setting the **layout_gravity** to the **center**.

```
<EditText
    android:id="@+id/editText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="write name here" />
```

```
<Button
    android:layout_width="129dp"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:onClick="afterClick2"
    android:text="Send text" />
```



In order for the button to respond to a click, we must also add an **onClick** event to it, with the event that handles it defined in the second activity.

🕮 3.3.8

Return response

In order to read the data from the new activity and return it to the first activity, we need to take a few steps:

- first of all, we get a link to a text field in which the user enters a name so that we can read the entered text from it,
- create an intent through which we want to return the result (name),
- since it is not an essential resource or target, it can be created without parameters - we will send it as a return message,
- we add the data we need to transfer to the intent (putExtra),
- the data is entered in pairs as the variable name (**my_name**) and its value (from the text field).

```
public void afterClick2(View view) {
    EditText myName = (EditText)findViewById(R.id.editText);
    Intent intentBack = new Intent();
    intentBack.putExtra("my_name",
myName.getText().toString());
    setResult(RESULT_OK, intentBack);
    finish();
}
```

The **setResult** command sets or sends the result of the activity from which the request came - **MainActivity** and sends information that the communication was OK even with the whole intent to which we added the values.

The **finish()** method ends the activity.

🕮 3.3.9

Response processing

The **NameActivity** activity allowed us to read the value entered by the user and return it to the main activity (**MainActivity**). We must read and process the return value.

In the Android framework, a method is defined, which is called when returning from another activity and ensures the reading of returned values. This is the **onActivityResult()** method.

We will adjust it to the required form as follows:

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
    switch (requestCode) {
        case 1:
            if (resultCode == RESULT_OK) {
                Bundle returnedData = data.getExtras();
                String text =
returnedData.getString("my_name");
                TextView tv = (TextView)
findViewById(R.id.textView);
                tv.setText("Hello " + text);
            }
            break;
    }
}
```

Since this method is designed to process return values from all activities, it is necessary to distinguish from which activities the values are returned. We do this through **requestCode**, which is the code we entered when calling **NameActivity** as a value of 1 (by default, constants are used for better code reading).

To distinguish the values returned from different standard activities, we use the **switch** construct, in which we define a separate branch (**case**) for each case.

```
if (resultCode == RESULT_OK)...
```

In case of returning from each activity, it is advisable to check whether the activity was terminated by confirming the values (and pressing the **OK** button, which set the result of the termination as **RESULT_OK**). The activity could have been terminated in another way: by closing the window, by clicking on the cancel button or simply by some exception or disaster.

It only makes sense to read the returned data if the activity ended as expected, otherwise there would be no data in the incoming intent.

```
if (resultCode == RESULT_OK) {
    Bundle returnedData = data.getExtras();
    String text = returnedData.getString("my_name");
```

For the returned data, we will create an instance of the **Bundle** class, which is used to obtain the transferred data. In it we find the value corresponding to the name of the "variable" **my_name** and insert it into the text string.

```
TextView tv = (TextView) findViewById(R.id.textView);
tv.setText("Hello " + text);
```

Finally, we identify the field in which we want to insert the result (**TextView**) and insert the downloaded string into it with a greeting.

If you do not have a **TextView** with the given name, you can insert a new **view** into the activity.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:text="@string/text_description" />
```

The **view** name represents the **android:id** property, which is followed by the mandatory **@+id/** label, followed by the name.

3.3.10 Code of the application

MainActivity.java

```
package com.example.double_activities;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void afterClick(View view) {
        Intent intent = new Intent(MainActivity.this,
NameActivity.class);
        startActivityForResult(intent, 1);
    }

    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        switch (requestCode) {
            case 1:
                if (resultCode == RESULT_OK) {
                    Bundle returnedData = data.getExtras();
                    String text =
returnedData.getString("my_name");
                    TextView tv = (TextView)
findViewById(R.id.textView);
                    tv.setText("Hello " + text);
                }
                break;
        }
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
```

```
tools:context="com.example.double_activities.MainActivity">

    <|TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:text="@string/text_description" />

    <|Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button1_text"
        android:onClick="afterClick"
        />

<|/LinearLayout>
```

NameActivity.java

```
package com.example.double_activities;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class NameActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_name);
    }

    public void afterClick2(View view) {
        EditText myName =
(EditText)findViewById(R.id.editText);
        Intent intentBack = new Intent();
        intentBack.putExtra("my_name",
myName.getText().toString());
        setResult(RESULT_OK, intentBack);
        finish();
```

```
    }
}
```

activity_name.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".NameActivity">

    <|EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="write name here" />

    <|Button
        android:layout_width="129dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:onClick="afterClick2"
        android:text="Send text" />

<|/LinearLayout>
```

3.3.11 Exercise tasks

1. Create an application that allows you to add two numbers entered in the text fields and print the whole example (6 + 3 = 9) in a new window. Apply an adjustment to the field so that only numbers can be inserted (set the **inputType** property to **number** for the text field).

Lists

Chapter 4

4.1 Lists

🕮 4.1.1

Working with lists is the most common activity when working with mobile devices, and lists are also necessary means of solving tasks working with databases in any sense.

The most common requirement of common applications is to display a list from which the user can choose and ensure an adequate response when clicked. Because the screen size is usually limited, it is not possible to provide the user with all the information that is available, so details are usually not displayed until the list item is clicked.

Due to the fact that this is a very common request, **Android** has created its own mechanism to solve such a task with a minimum of code.

🕮 4.1.2

ListView

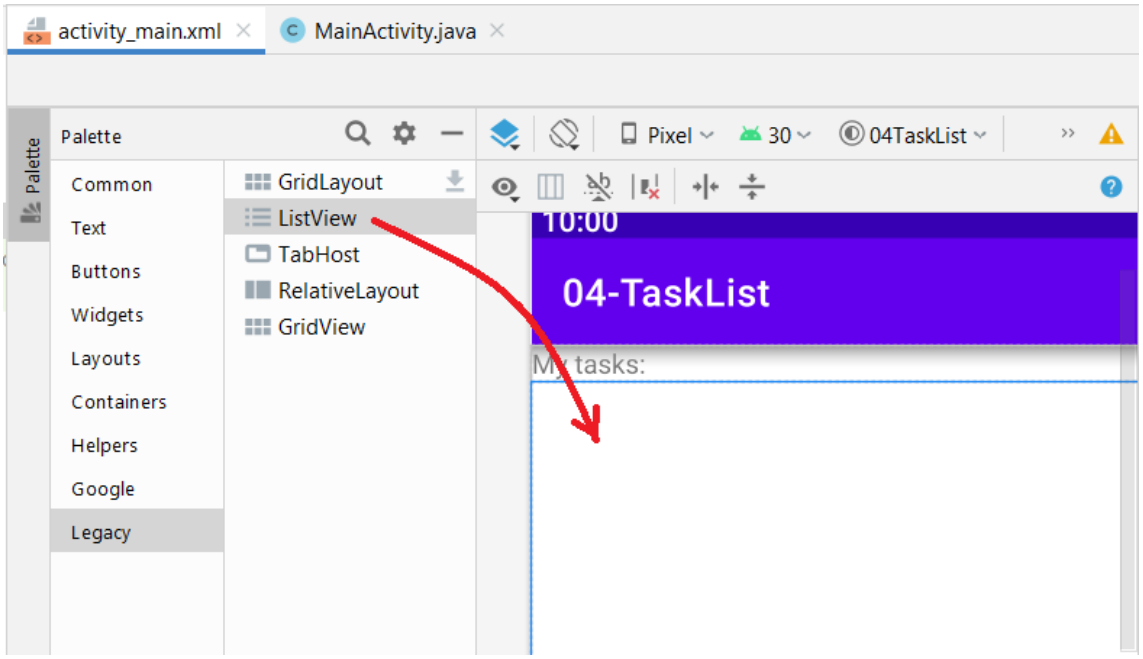
The most common **view** used to work with lists is **ListView**.

Its use is relatively simple, in addition to the elementary list consisting of one piece of data in one item, it also supports the creation of various structured items (combination of several record items, images, combination of texts with image, etc.).

📖 4.1.3

Create an application that displays a list of tasks to perform along with a description of how to do it. In the main activity, only display a list that, when clicked, displays the second activity along with a description of how to perform the activity.

First we will create an interface - we will insert a **ListView** element into an empty activity template (not the whole activity as in some examples) and above it the title: **My tasks**.



📖 4.1.4

Adapter

As with tables and lists in desktop **Java**, **Android** displays lists through a mechanism that separates data from displayed **views**. The reason for this approach, and at the same time a great advantage, is that different containers (eg **ListView** and **GridView**) can display the same data in different ways - as a vertical list or as a table.

The adapter that ensures the correct connection of views with data is an intermediary. Provides data from memory to be customized so that the visual component can read and display it.

Although **Android** has several adapters, the simplest one is usually sufficient - **ArrayAdapter**. Technically, it is an adapter modified into a form that transforms an array (list) of data into **TextView** elements.

```
ArrayAdapter<String> myAdapter = new
ArrayAdapter<String>(...);
```

🕮 4.1.5

ArrayAdapter is a generic class and can handle arrays of any type defined during creation. As a result, it calls the **toString()** method for each field item to display it.

In order to be able to work with data within the application, it is necessary to define them as class variables (activities). We will have the data stored in two **ArrayLists** - one for recording tasks, the other with a description. The task in the i-th position will correspond to the description in the i-th position.

The introductory code of the class will be in the form in which we create empty lists and declare **ArrayAdapter**:

```
public class MainActivity extends AppCompatActivity {
    ArrayList<String> taskArray = new ArrayList<String>();
    ArrayList<String> descriptionArray = new
ArrayList<String>();
    ArrayAdapter<String> myAdapter;
```

The connection of data to the adapter as well as the insertion of test data should be realized already when creating the activity, ideally by calling a separate method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    prepareData();
}
```

🕮 4.1.6

After filling the lists with basic tasks and their description, we will create an adapter for working with strings with the following parameters:

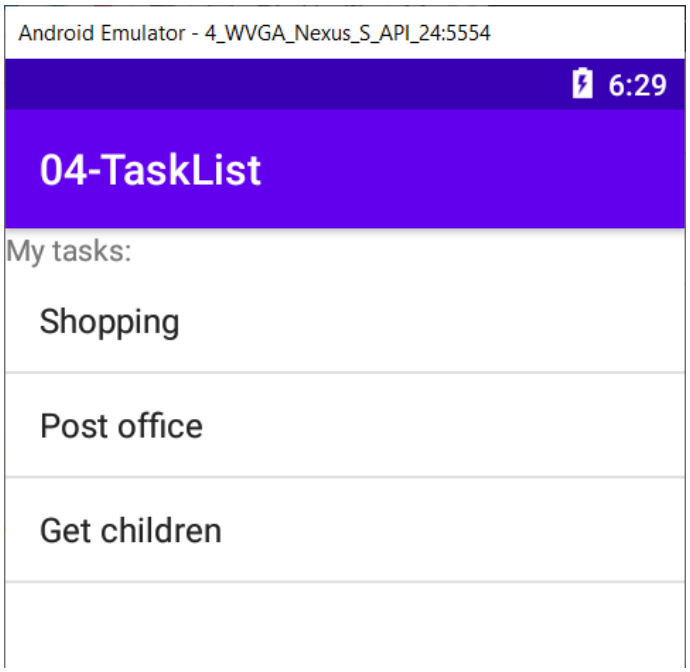
- the first parameter is the context, in our case it represents the activity,
- **simple_list_item1** is a template defined for simple display of lists, it is defined in the system, we do not have to extra work with it,
- a list represented by the **ArrayList** class and containing the data that will be displayed through the adapter.

```
private void prepareData() {
    // demo data
    taskArray.add("Shopping"); descriptionArray.add("food,
socks");
    taskArray.add("Post office"); descriptionArray.add("pay PO
BOX");
    taskArray.add("Get children"); descriptionArray.add("from
kindergarten - please, ours");
    // array adapter
    myAdapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            taskArray);

    // join to view
    ListView myList = (ListView) findViewById(R.id.listView);
    myList.setAdapter(myAdapter);
}
```

We will connect the created adapter to the view into which the data from the **ArrayList** is to be displayed - to the **listView**.

It is possible that the **ListView** that we inserted into the activity does not have an identifier assigned to it. We will check and, if necessary, set its name via the id parameter.

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



The result of our work is an application that displays data in a list and allows you to click on them, but the items do not respond to clicks.

4.2 Events in lists

🕮 4.2.1

Events in the list

The task of the created list is to display data and respond to clicks. In order for a view to be able to respond to a click on an item, it needs to have an added **listener** capable of handling such a click (the listener is an interface just like in **Java**). Since it is not possible to access the **ListView** items within the **xml** design proposal, we must create and connect the listener in the code.

We will create a separate method, which we will call when creating an activity for loading data and connecting an adapter.

```
private void addListener() {
    AdapterView.OnItemClickListener mMessageClickedHandler =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView parent,
                                        View v,
                                        int position,
                                        long id) {
                    // code to execute when the item is
clicked
```

```
                    // e.g. ((TextView)v).setText("selected");
                }
            };
    ListView myList = (ListView) findViewById(R.id.listView);
    myList.setOnItemClickListener(mMessageClickedHandler);
}
```

AdapterView is a view which items are populated through the adapter and it includes the **OnItemClickListener** class, which we create an instance of.

In the created instance, we define the **onItemClick()** method with the prescribed parameters:

- **View** is an object that the user clicked on - based on the template used, we know that it is a **TextView**,
- **position** is the position of the item in the list, it will serve us when we want to know which item in the list was clicked,
- **id** is the position of the item in the containers (for **ArrayAdapter** the **position** and **id** are the same)
- the code that we insert into this method is to ensure the opening of a new activity with the details of the selected item.

🕮 4.2.2

We need to send detailed information about the selected task to the new activity, but the user would certainly welcome the display of the task names. Therefore, we will send two text data, which we will then display in a user-friendly form.

We already know the principle - we create an intent, add variables with values to it and send it.

```
public void onItemClick(AdapterView parent,View v,int
position,long id) {
    // intent to open the detail activity
    Intent intent = new Intent(MainActivity.this,
DetailActivity.class);
    // read the data from the array according to the selected
item
    String task = taskArray.get(position);
    String description = descriptionArray.get(position);
    // put data into intent
    intent.putExtra("task",task);
    intent.putExtra("description",description);
    // show second activity
    startActivity(intent);
```

```
}
```

Finally, we connect the created **listener** to the **ListView**.

```
ListView myList = (ListView) findViewById(R.id.listView);
myList.setOnItemClickListener(mMessageClickedHandler);
```

We will not forget to call the created method in the **OnCreate()** method of the activity.

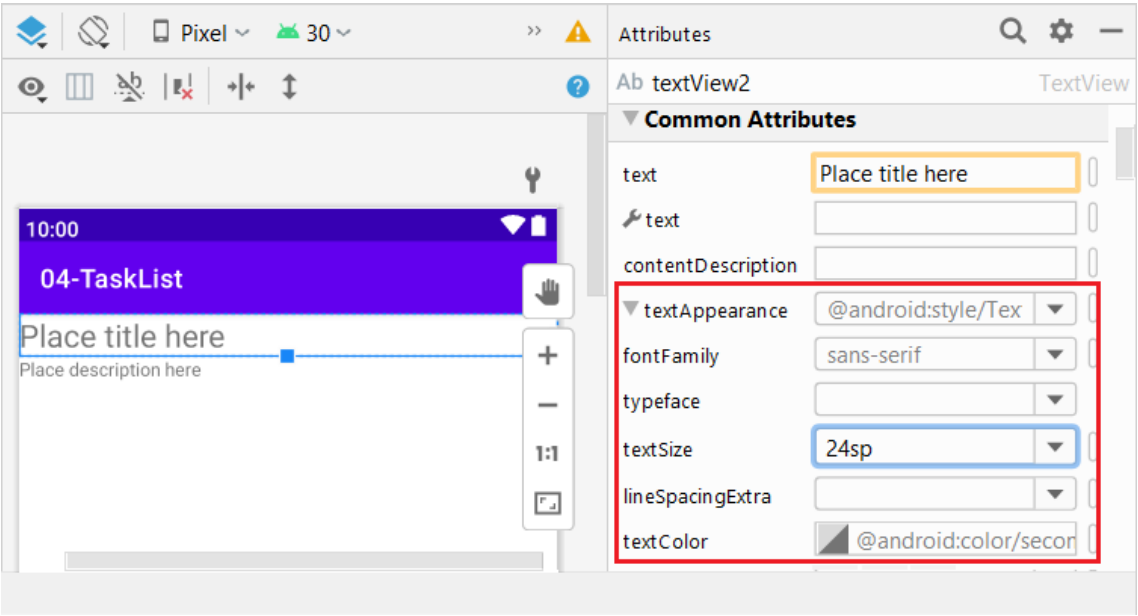
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    prepareData();
    addListener();
}
```

📖 4.2.3

Getting data when creating an activity

We will create an activity displaying the details of the task in the standard way as an empty activity with the name e.g. **DetailActivity**.

Within the displayed data, we can set different font sizes for displaying the task and its description, or add a button to close the activity, which, however, is not a standard element in applications for **Android OS** - activities are closed with the system button.



We will read and display the data we sent from the main activity. The ideal place is the method of creating an activity which, after setting up the layout, will also fill its views with the necessary content:

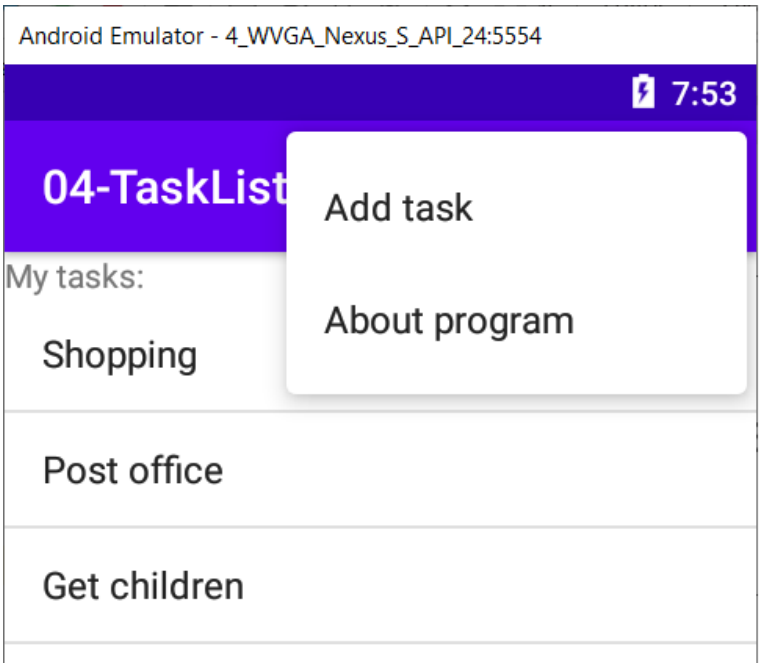
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detail);
    // views into which we display the content
    TextView tv_title = (TextView)
findViewById(R.id.textView2);
    TextView tv_descr = (TextView)
findViewById(R.id.textView3);
    // we read the intent by which the activity was evoked
    Intent intent = getIntent();
    // we read the contents of the submitted variables from
the intent
    String task = intent.getStringExtra("task");
    String descr = intent.getStringExtra("description");
    // output
    tv_title.setText(task);
    tv_descr.setText(descr);
}
```

Finally, to close the activity without having to return a value, use only the button with the **finish()** command.

4.3 Menu

🕮 4.3.1

Allow new tasks to be added through the new activity. Use the menu to call it up.

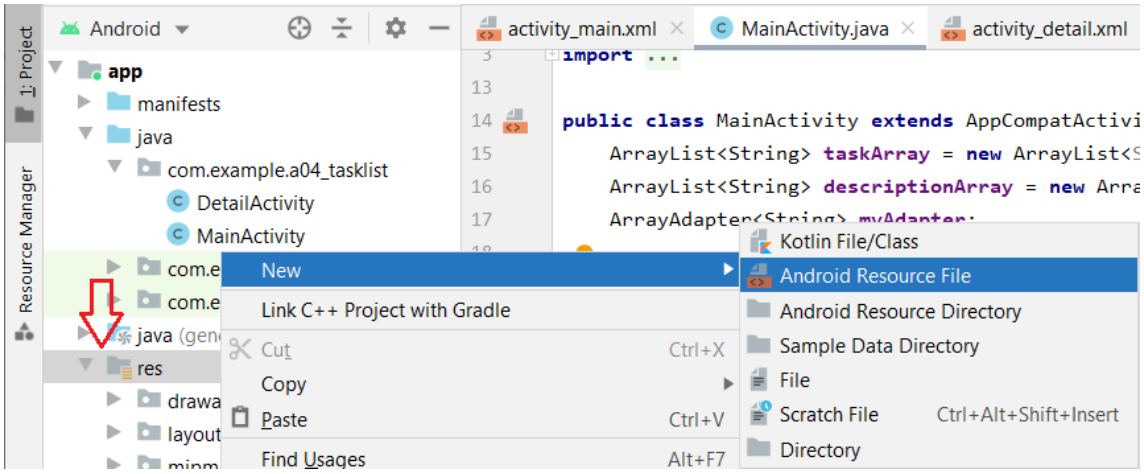


4.3.2

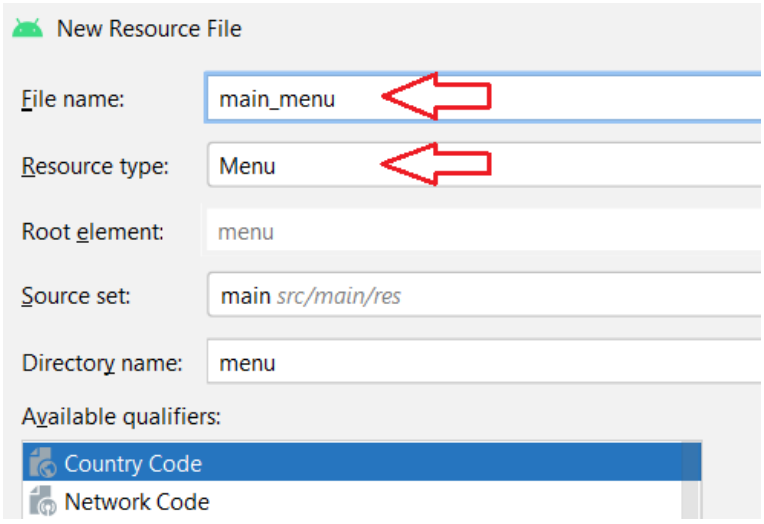
The menu offers an interface for accessing the functions and settings of the application. Android provides it to us in two basic forms:

- **Options Menu** - appears at the bottom of the screen when you press the "Menu" button on your device
- **Context Menu** - displayed in the center of the screen, activated by holding your finger on the component, e.g. on the list

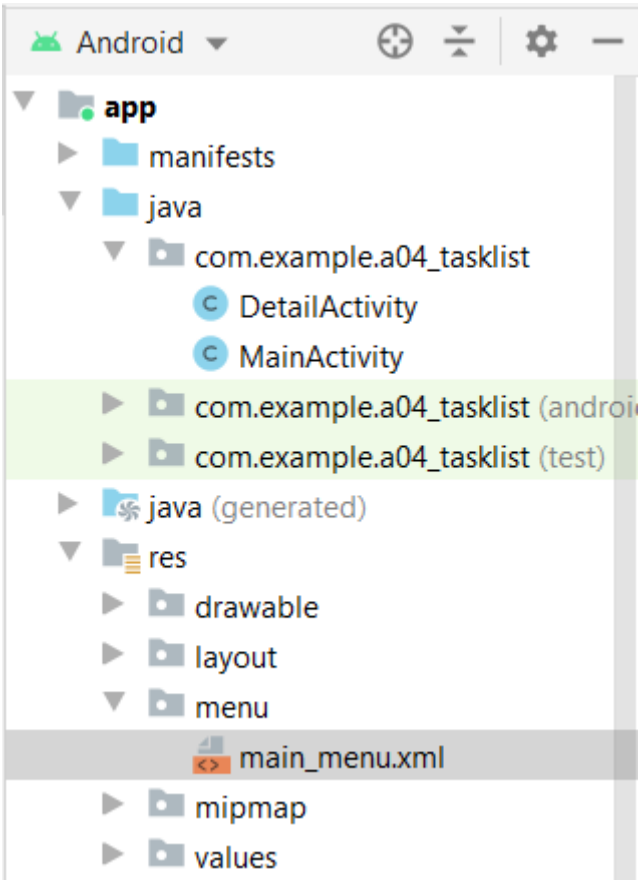
For our needs, it will be more appropriate to use the **Options menu**, for which a group in the resource was created in advance in older versions of the system, currently we have to create it via the context menu for res.



We specify the type of object (**menu**) and the name.



A menu subfolder item and a newly named object are created in the **res** folder.



📖 4.3.3

The content of the menu can be created in the same way as in the case of activity design either via the GUI or by editing the xml file. The menu with two items will look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/menu_add"
        android:title="Add task" />
    <item android:id="@+id/menu_about"
        android:title="About program" />
</menu>
```

In the **Android framework**, a menu is created in the **onCreateOptionsMenu()** method of the activity that the menu uses. We will add this to our main activity:

🕮 4.3.4

Connecting the menu and code or processing of a click on a menu item is provided by the **onOptionsItemSelected** method, into which the selected menu item enters as a parameter:

- it is enough for us to create the appropriate code for each item
- item names (**itemId**) are identical to menu.xml

```
public boolean onOptionsItemSelected(MenuItem item){
    switch (item.getItemId()) {
        case R.id.menu_add:
            startActivityForResult(new Intent(this,
AddActivity.class),1);
            return true;
        case R.id.menu_about:
            // just a fading bubble = toast
            Toast.makeText(getBaseContext(), "My info",
Toast.LENGTH_SHORT).show();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

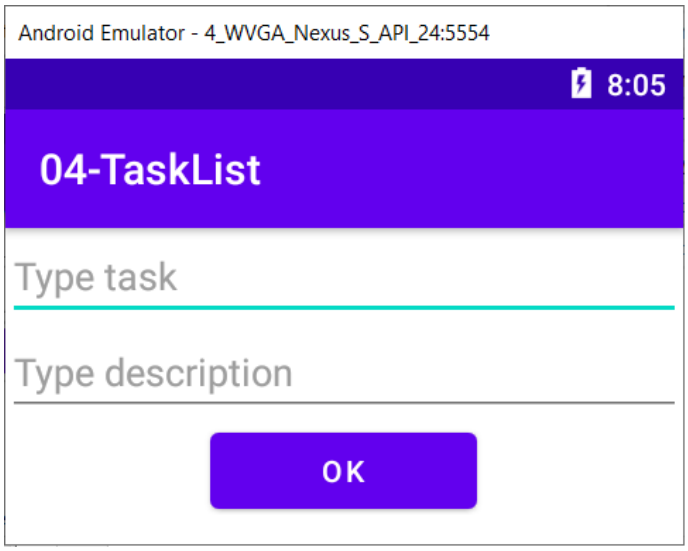
The first item invokes a new activity that allows you to insert a new task-description pair into the task list. The second item displays information about the program, which we display through a frequently used display element - **Toast**, which displays the entered text in the lower half of the screen in a separate rectangle, which is lost after a short time.

Toast is also a very useful tool for debugging an application, we can list the values of variables without affecting other elements in the activities.

🕮 4.3.5

Adding an item

By default, a separate activity is used to add data to lists. In our case, it will contain two edit fields and a button that sends the filled data to the main activity.



The code of the button creates an **intent**, appends the values from the textfields to it, returns the value to the main activity, and finally closes the current one.

```
public void button2Click(View view) {
    EditText task =
(EditText)findViewById(R.id.editTextTextPersonName);
    EditText desc =
(EditText)findViewById(R.id.editTextTextPersonName2);
    Intent intentBack = new Intent();
    intentBack.putExtra("task", task.getText().toString());
    intentBack.putExtra("description",
desc.getText().toString());
    setResult(RESULT_OK,intentBack);
    finish();
}
```

We've adapted the name of the method that handles the click on the button to the name of the button, and we'll continue to use this standardized approach in the future - so that when you look at the code, it's clear to which button the method belongs to.

Unless we have changed the field names, we will keep the original environment-defined name (editTextTextPersonName...)

🕮 4.3.6

Finally, in the main activity, we add the data obtained to the list in the method for processing the returned results:

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
    switch (requestCode) {
        case 1:
            if (resultCode == RESULT_OK) {
                String task = data.getStringExtra("task");
                String descr =
data.getStringExtra("description");
                // insert into task array
                taskArray.add(task);
                // insert into description array
                descriptionArray.add(descr);
                break;
            }
    }
}
```

⌨ 4.3.7 The code of the application

MainActivity.java

```
package com.example.a04_tasklist;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
```

```
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
    ArrayList<|String> taskArray = new ArrayList<|String>();
    ArrayList<|String> descriptionArray = new
ArrayList<|String>();
    ArrayAdapter<|String> myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        prepareData();
        addListener();
    }

    private void prepareData() {
        // demo data
        taskArray.add("Shopping"); descriptionArray.add("food,
socks");
        taskArray.add("Post office");
descriptionArray.add("pay PO BOX");
        taskArray.add("Get children");
descriptionArray.add("from kinder garden - please, our");
        // array adapter
        myAdapter = new ArrayAdapter<|String>(
                this,
                android.R.layout.simple_list_item_1,
                taskArray);

        // join to view
        ListView myList = (ListView)
findViewById(R.id.listView);
        myList.setAdapter(myAdapter);
    }

    private void addListener() {
        AdapterView.OnItemClickListener mMessageClickedHandler
=
                new AdapterView.OnItemClickListener() {
                    public void onItemClick(AdapterView
parent,View v,int position,long id) {
```

```
                        // intent to open the detail activity
                        Intent intent = new
Intent(MainActivity.this, DetailActivity.class);
                        // read the data from the array
according to the selected item
                        String uloha =
taskArray.get(position);
                        String popis =
descriptionArray.get(position);
                        // put data into intent
                        intent.putExtra("task",uloha);
                        intent.putExtra("description",popis);
                        // show second activity
                        startActivity(intent);
                    }
                };
        ListView myList = (ListView)
findViewById(R.id.listView);
        myList.setOnItemClickListener(mMessageClickedHandler);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main_menu, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item){
        switch (item.getItemId()) {
            case R.id.menu_add:
                startActivityForResult(new Intent(this,
AddActivity.class),1);
                return true;
            case R.id.menu_about:
                // just a fading bubble  = toast
                Toast.makeText(getBaseContext(), "My info",
Toast.LENGTH_SHORT).show();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

```
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        switch (requestCode) {
            case 1:
                if (resultCode == RESULT_OK) {
                    String task = data.getStringExtra("task");
                    String descr =
data.getStringExtra("description");
                    // insert into task array
                    taskArray.add(task);
                    // insert into description array
                    descriptionArray.add(descr);
                    break;
                }
        }
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="My tasks:" />

    <|ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />


<|/LinearLayout>
```

DetailActivity.java

```
package com.example.a04_tasklist;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        // view-y into which we display the content
        TextView tv_title = (TextView)
findViewById(R.id.textView2);
        TextView tv_descr = (TextView)
findViewById(R.id.textView3);
        // we read the intent by which the activity was evoked
        Intent intent = getIntent();
        // we read the contents of the submitted variables
from the intent
        String task = intent.getStringExtra("task");
        String descr = intent.getStringExtra("description");
        // output
        tv_title.setText(task);
        tv_descr.setText(descr);
    }
}
```

activity_detail.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".DetailActivity">

    <|TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Place title here"
        android:textSize="24sp" />

    <|TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Place description here" />

<|/LinearLayout>
```

AddActivity.java

```
package com.example.a04_tasklist;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class AddActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add);
    }

    public void button2Click(View view) {
        EditText task = (EditText)
findViewById(R.id.editTextTextPersonName);
        EditText desc =
(EditText)findViewById(R.id.editTextTextPersonName2);
        Intent intentBack = new Intent();
        intentBack.putExtra("task",
task.getText().toString());
        intentBack.putExtra("description",
desc.getText().toString());
        setResult(RESULT_OK,intentBack);
        finish();
    }

}
```

activity_add.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".AddActivity">

    <|EditText
        android:id="@+id/editTextTextPersonName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Type task"
        android:inputType="textPersonName" />

    <|EditText
        android:id="@+id/editTextTextPersonName2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Type description"
        android:inputType="textPersonName" />

    <|Button
        android:id="@+id/button"
        android:layout_width="126dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:onClick="button2Click"
        android:text="OK"/>
<|/LinearLayout>
```

main_menu.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|menu
xmlns:android="http://schemas.android.com/apk/res/android">

    <|item android:id="@+id/menu_add"
        android:title="Add task" />
    <|item android:id="@+id/menu_about"
```

```
        android:title="About program" />
<|/menu>
```

List Layouts

Chapter 5

5.1 Simple layout

🕮 5.1.1

The basic containers for displaying lists are ListView and GridView.

They represent the most common non-graphical tool in Android applications.

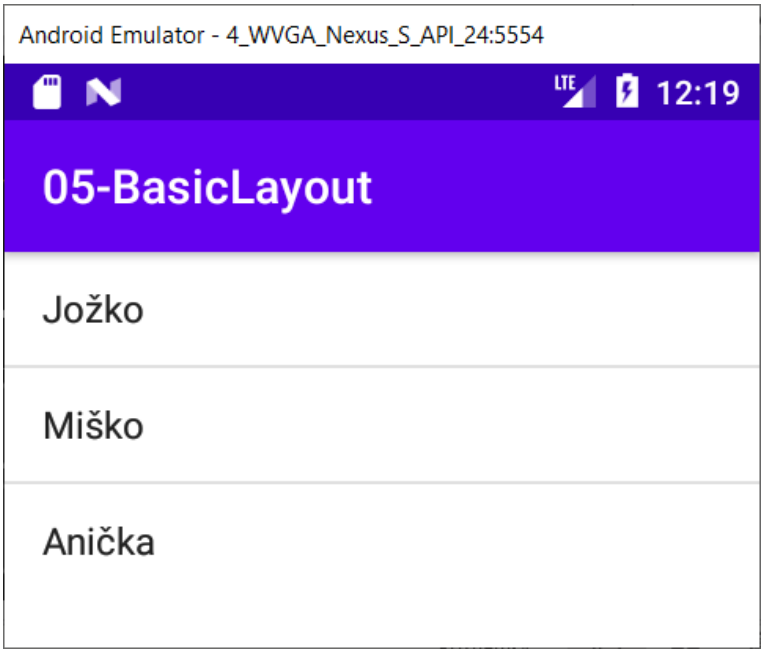
According to their use, they also provide many ways of displaying and manipulating data, while still separating data from display.

🕮 5.1.2

Although we have already displayed a list in ListView, we will introduce it again for completeness.

The most suitable data source for simple display is **ArrayList** or **ArrayList <String>**.

The layout only requires inserting a **ListView** into the activity



🕮 5.1.3

We usually define a list as generic.

We fill the data into it with standard commands - the data is only for demonstration, so that the content of the list is visible.

```
public class MainActivity extends AppCompatActivity {
    ArrayList<String> myList = new ArrayList<String>();
    ArrayAdapter myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fillMyData();
    }
```

Filling the data:

```
private void fillMyData() {
    myList.add("John");
    myList.add("Michael");
    myList.add("Annie");
...
```

🕮 5.1.4

The adapter that we use to link data to the display component requires information about:

- display pattern - in this case we will use the built-in one-line "template" - **android.R.layout.simple_list_item_1**
- data sources - we will present the created generic list

```
private void fillMyData() {
    myList.add("John");
    myList.add("Michael");
    myList.add("Annie");

    myAdapter = new ArrayAdapter(
            this,                                   // context,
usualy activity
            android.R.layout.simple_list_item_1,  // display
prescription
```

```
            myList                                          // data
source
    );
 ...
```

Finally, we will ensure the connection of the adapter to the component to which it will supply data.

```
    ...
    // a link to a view intended for data
    ListView lv = (ListView) findViewById(R.id.listView);
    lv.setAdapter(myAdapter); // connection via adapter to LV
}
```

🕮 5.1.5

Types of adapters

Android has three different adapters available:

- **ArrayAdapter** transforms an Array or List
- **SimpleAdapter** maps and transforms static data
- **CursorAdapter** expects a list of records (we will use it in the database section)

Multiple alternatives can often be used to solve the same problem, the choice depends on the specific situation.

⌨ 5.1.6 Code of the application

MainActivity.java

```
package com.example.a05_basiclayout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
```

```
    ArrayList<|String> myList = new ArrayList<|String>();
    ArrayAdapter myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fillMyData();
    }

    private void fillMyData() {
        myList.add("Joseph");
        myList.add("Michael");
        myList.add("Annie");

        myAdapter = new ArrayAdapter(
                this,                                   //
context, usualy activity
                android.R.layout.simple_list_item_1,  //
display prescription
                myList                                    //
data source
        );

        // a link to a view intended for data
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter); // connection via adapter to
LV
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <|ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:id="@+id/listView" />

<|/LinearLayout>
```

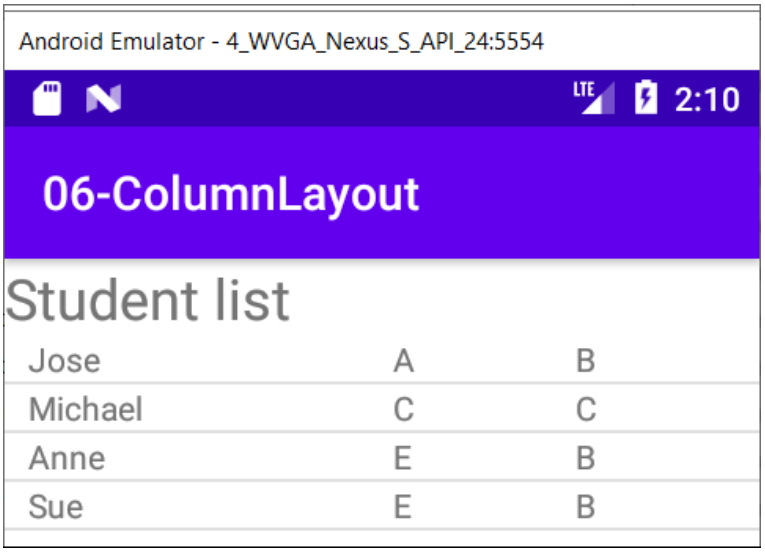
5.2 Multi-column layout

🕮 5.2.1

Assignment

Create a layout in which you display data about students and their results from the programming of the first year.

Let each line contain information about the student, his grade from the first and second semester.



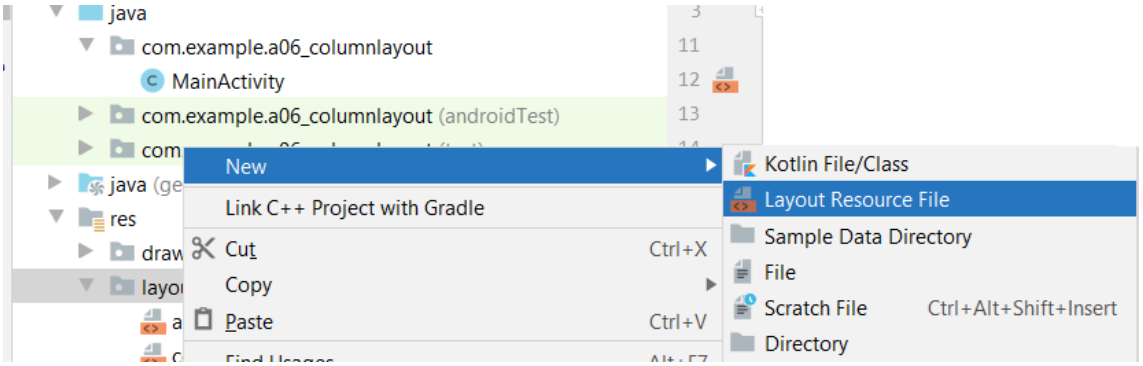
🕮 5.2.2

ListView as a container for a list will again be part of the activity, but for the content of its line we will have to define individual components, because it will no longer be just plain text.

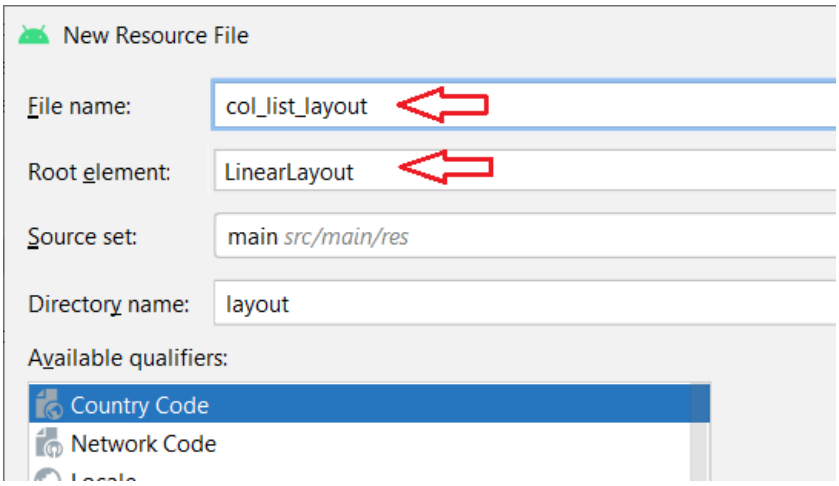
The **ListView** line will thus represent another container in which we will store and format individual (essentially arbitrary) components.

Since this is a separate rule (which we can use in other places), we need to create a new layout for it through a new layout file.

We will create a new file in the layout folder



and name it accordingly.



🕮 5.2.3

Layout definition

We define a template for displaying data by creating elements in which the data will be displayed within one **ListView** item.

The created template can represent one line or even more of them, we are not limited by any rules.

We define names and widths for individual objects into which data will be inserted.

The width of the object is calculated as the ratio of the **layout_weight** parameter to the sum of all **layout_weight**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="horizontal"
android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/item1"
        android:text="row_id"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:layout_weight="2"
        />
    <TextView android:id="@+id/item2"
        android:text="col_1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:layout_weight="1"
        />
    <TextView android:id="@+id/item3"
        android:text="col_2"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:layout_weight="1"
        />

</LinearLayout>
```

🕮 5.2.4

If we want to use the display of data in several components, we need:

- either a custom adapter
- **SimpleAdapter**, which requires the use of a child of the **Map** class in the constructor

It will be easier for us to choose the second option.

🕮 5.2.5

Map / **HashMap** is one of the map implementations that contains key and value value pairs (we know the principle, we have defined the variable name / variable content pairs e.g. for transfer between activities within the intent)

In the case of **HashMap**, the first value (key / variable name) in the list must not be repeated, which is understandable and acceptable to us.

The emphasized fact in the definition of **HashMap** is that the list of data may not be stored in memory in the same order as it was inserted, which is in our case an irrelevant information, because the location of elements will be defined through views

Each map will contain a list of values for one row in the **ListView**.

The map will contain pairs **String**, **String** (name, value), i.e. (name - Jose; PR1 - A; PR2 - C) or (name - Peter; PR1 - B; PR2 - B).

One map contains all data for one student.

ListView will need a list of such maps (list of students and their results), so we define an **ArrayList** of such maps for it.

And at the same time we define the already talked about **SimpleAdapter**.

```
public class ColumnLayoutActivity extends AppCompatActivity {
 ArrayList<HashMap<String, String>> myList = new
ArrayList<HashMap<String, String>>();
 SimpleAdapter myAdapter;
```

🕮 5.2.6

Populating the list with instances of the **HashMap** class consists of the following steps:

- create a map instance
- inserting all the variables + value pairs, which represent one row of the list, into the map instance
- inserting the created map instance into the list
- sequence repetition for all students

```
private void fillMyData() {
    HashMap<String, String> myRow = new HashMap<String,
String>();
    myRow.put("name", "Jose");
    myRow.put("PR1", "A");
    myRow.put("PR2", "B");
    myList.add(myRow);

    myRow = new HashMap<String, String>();
    myRow.put("name", "Michael");
    myRow.put("PR1", "C");
    myRow.put("PR2", "C");
    myList.add(myRow);
```

```
    myRow = new HashMap<String, String>();
    myRow.put("name", "Anne");
    myRow.put("PR1", "E");
    myRow.put("PR2", "B");
    myList.add(myRow);

    myRow = new HashMap<String, String>();
    myRow.put("name", "Sue");
    myRow.put("PR1", "E");
    myRow.put("PR2", "B");
    myList.add(myRow);
}
```

🕮 5.2.7

Connecting the adapter

For this type of adapter (**SimpleAdapter**) we need to define the connection between data and display elements.

- the **from** list will contain the **name of the variable in the map**
- the **to** list will contain the views in which the respective value is to be displayed (the name of the element represents its unique integer value)

```
private void connectMyAdapter() {
    String[] from = new String[]{"name", "PR1", "PR2"};
    int[] to = new int[]{R.id.item1, R.id.item2, R.id.item3};
```

🕮 5.2.8

The adapter then requires:

- data source - defined and filled with **ArrayList**, it must be a list of maps
- list definition and layout of elements in the **ListView row** (it is in the **layout** file, which we created manually)
- list of fields, from the order in which they are to be drawn from individual maps (list **from**)
- list of elements in the layout in the order in which the data defined in **from** are to be inserted into them (list **to**)

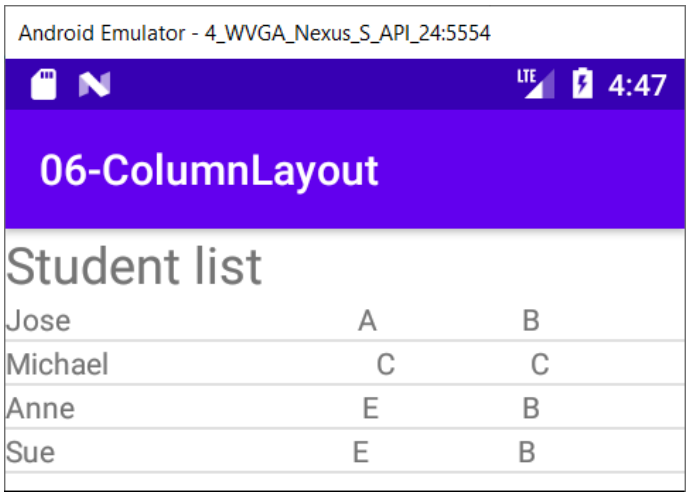
```
...
    myAdapter = new SimpleAdapter(
        this,                       // context, usualy activity
```

```
        myList,                         // datasource
        R.layout.col_list_layout,   // rules for visualisation
        from,                           // variable names in
hashmaps
        to                              // names of views in layout
file
    );
    // we get a link to the view
    ListView lv = (ListView) findViewById(R.id.listView);
    lv.setAdapter(myAdapter); // connect the data to the LV
via the adapter
}
```

🕮 5.2.9

The result

After launching, we get an application that displays the required data, but with a slightly unsatisfactory design, depending on the number of characters in the name at first glance.



🕮 5.2.10

Although the **layout_weight** property works, it is distorted by the **layout_width** property (in the case of a horizontal layout), which adjusts it to the size of the content.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...
android:orientation="horizontal"
```

```
 <TextView android:id="@+id/item1"
       android:layout_height="fill_parent"
       android:layout_width="wrap_content"
       android:layout_weight="2"
       />
   <TextView android:id="@+id/item2"
       android:layout_height="fill_parent"
       android:layout_width="wrap_content"
       android:layout_weight="1"
       />
   <TextView android:id="@+id/item3"
       android:layout_height="fill_parent"
       android:layout_width="wrap_content"
       android:layout_weight="1"
       />
</LinearLayout>
```

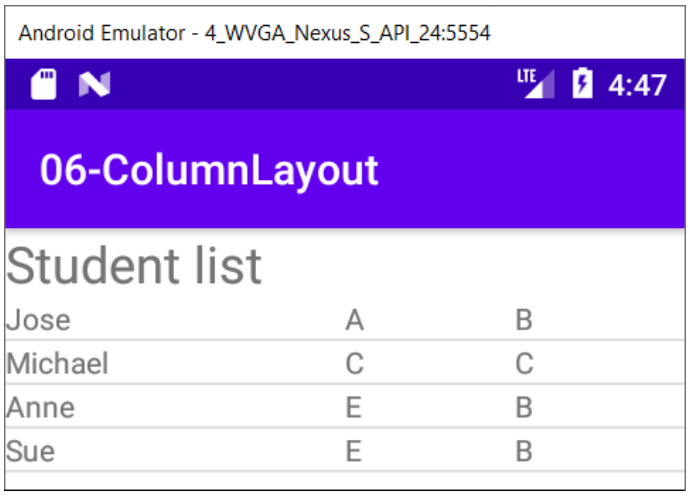
To make the width of the element depend only on the ratio specified in **weight**, we change the **layout_width** to a constant - **0 dip**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...
android:orientation="horizontal"

 <TextView android:id="@+id/item1"
       android:layout_height="fill_parent"
       android:layout_width="0dip"
       android:layout_weight="2"
       />
   <TextView android:id="@+id/item2"
       android:layout_height="fill_parent"
       android:layout_width="0dip"
       android:layout_weight="1"
...
```

📖 5.2.11

Result 2

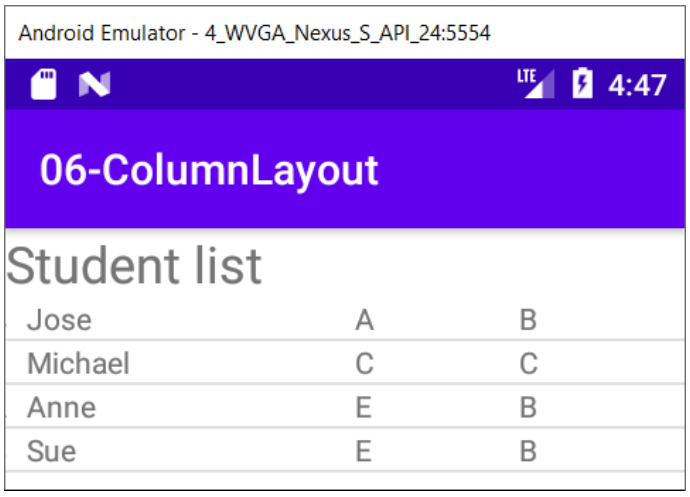
The layout is a bit better at first glance, the only thing that bothers is the text being too close to the left border.



We will modify this transgression by defining the edges of the first **TextView**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...
android:orientation="horizontal"
 <TextView android:id="@+id/item1"
  android:layout_height="fill_parent"
  android:layout_width="0dip"
  android:layout_marginLeft="10dip"
  android:layout_weight="2"
 />...
```

And finally:



🕮 5.2.12

DIP

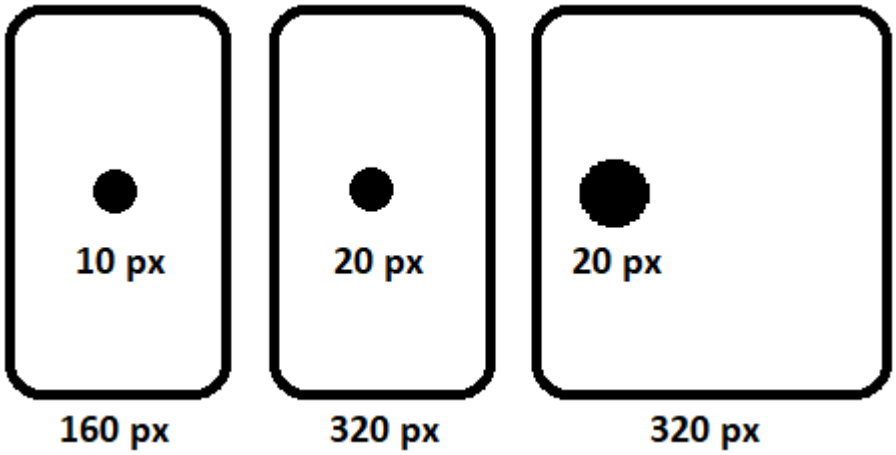
What is a dip or what are dips?

dip or **dp** (density-independent pixels) represents the unit of dimensions in Android.

This is actually the equivalent of one pixel with a screen width of 160 pixels.

For a screen with a smaller number of pixels, the number of dips will be converted to pixels so that the size is reduced, for a screen with a larger number of pixels, it will be converted to more pixels

The picture shows a circle with 10 dip on different devices. In principle, it is important to maintain the ratio of the size of the object to the size of the screen.



The conversion can be written relatively easily as

dp = (width in pixels * 160) / screen density

sp (Scaleable pixels) are working on the same principle and are used for fonts. However, they take their default settings from the font properties, so there may be variations between the same font size.

⌨ 5.2.13 Code of the application

MainActivity.java

```
package com.example.a06_columnlayout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;
```

```
import java.util.ArrayList;
import java.util.HashMap;

public class MainActivity extends AppCompatActivity {
    ArrayList<|HashMap<|String, String>> myList = new
ArrayList<|HashMap<|String, String>>();
    SimpleAdapter myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fillMyData();
        connectMyAdapter();
    }

    private void connectMyAdapter() {
        String[] from = new String[]{"name", "PR1", "PR2"};
        int[] to = new int[]{R.id.item1, R.id.item2,
R.id.item3};

        myAdapter = new SimpleAdapter(
                this,             // context, usualy activity
                myList,                   // datasource
                R.layout.col_list_layout, // rules for
visualisation
                from,
                to
        );

        // we get a link to the view
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter); // connect the data to the
LV via the adapter
    }

    private void fillMyData() {
        HashMap<|String, String> myRow = new HashMap<|String,
String>();
        myRow.put("name", "Jose");
        myRow.put("PR1", "A");
        myRow.put("PR2", "B");
        myList.add(myRow);
```

```
        myRow = new HashMap<|String, String>();
        myRow.put("name", "Michael");
        myRow.put("PR1", "C");
        myRow.put("PR2", "C");
        myList.add(myRow);

        myRow = new HashMap<|String, String>();
        myRow.put("name", "Anne");
        myRow.put("PR1", "E");
        myRow.put("PR2", "B");
        myList.add(myRow);

        myRow = new HashMap<|String, String>();
        myRow.put("name", "Sue");
        myRow.put("PR1", "E");
        myRow.put("PR2", "B");
        myList.add(myRow);
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

        <|TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Student list"
            android:textSize="24sp" />

        <|ListView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/listView"/>

    <|/LinearLayout>
```

col_list_layout.xml

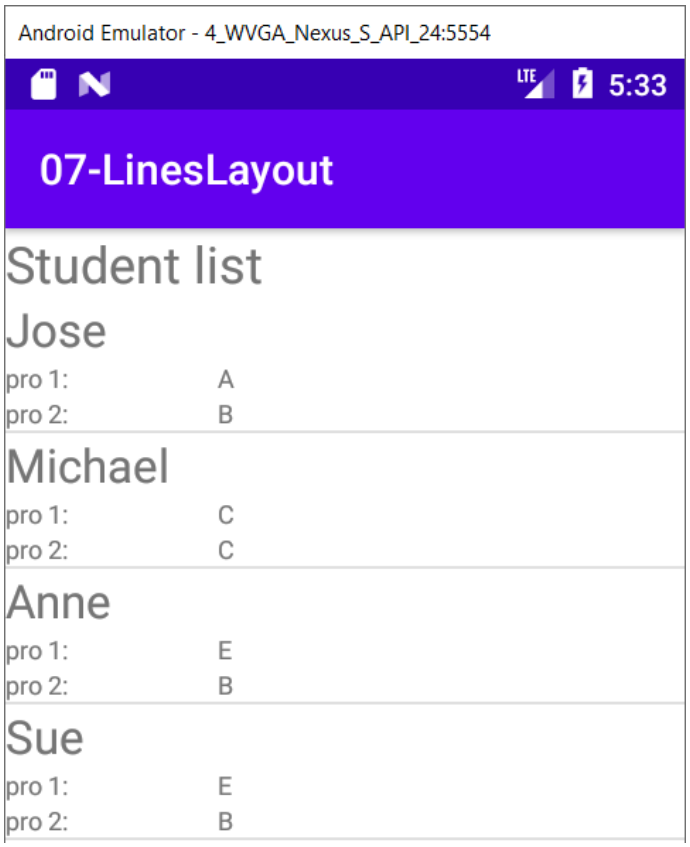
```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|TextView android:id="@+id/item1"
        android:text="row_id"
        android:layout_height="fill_parent"
        android:layout_marginLeft="10dip"
        android:layout_width="0dip"
        android:layout_weight="2"
        />
    <|TextView android:id="@+id/item2"
        android:text="col_1"
        android:layout_height="fill_parent"
        android:layout_width="0dip"
        android:layout_weight="1"
        />
    <|TextView android:id="@+id/item3"
        android:text="col_2"
        android:layout_height="fill_parent"
        android:layout_width="0dip"
        android:layout_weight="1"
        />

<|/LinearLayout>
```

5.3 Multi-row layout

🕮 5.3.1

Organize student data into **ListView** so that:

- in the first line the students' name will be displayed
- in the next lines there will be a list of subjects with the name of the subject and grade



📖 5.3.2

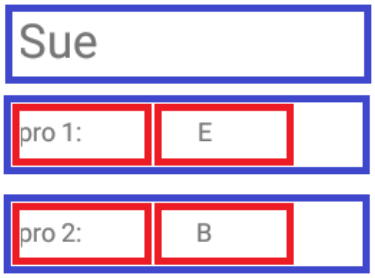
Layout definition

The philosophy of the code for filling with demo data is preserved, the application will differ from the previous one only in the layout definition.

Again, we define a separate xml file for the **ListView** item layout

For one line we define a vertical layout, which will consist of a text field and two layouts that place elements next to each other (horizontally)

It is quite common that in one layout we define another (independent) layout with its own behavior of its elements.



Elements of the vertical layout are marked in blue, elements of the horizontal within the already existing vertical are marked in red.

The code will have, for example: the following form

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/item1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:textSize="22sp"
        android:width="100dp"
        />

    <LinearLayout
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:layout_height="fill_parent">

        <TextView android:id="@+id/descr2"
            android:text="pro 1:"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

        <TextView android:id="@+id/item2"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:layout_height="fill_parent">

        <TextView android:id="@+id/descr3"
```

```
            android:text="pro 2:"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

        <TextView android:id="@+id/item3"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />
    </LinearLayout>

</LinearLayout>
```

📖 5.3.3

Connecting the data

The memory structure is the same as in the previous task.

When defining the adapter, we omit descriptive TextViews and fill in only those containing data.

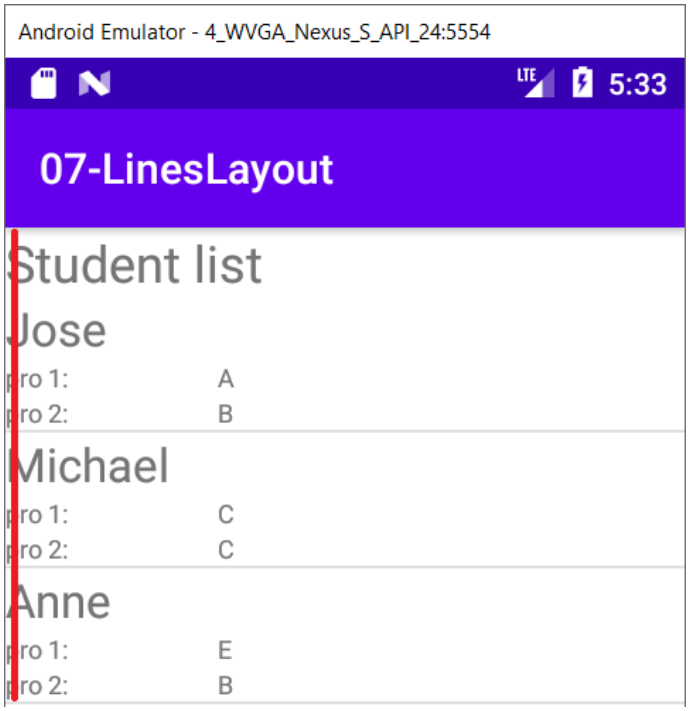
```
public class MainActivity extends AppCompatActivity {
    ArrayList<HashMap<String, String>> myList = new
ArrayList<HashMap<String, String>>();
    SimpleAdapter myAdapter;

...

    private void connectMyAdapter() {
        String[] from = new String[]{"name", "PR1", "PR2"};
        int[] to = new int[]{R.id.item1, R.id.item2,
R.id.item3};

        myAdapter = new SimpleAdapter(
                this,                       // context, usualy
activity
                myList,                     // datasource
                R.layout.lines_list_layout, // new layout file
                from,
                to
        );
```

```
        // we get a link to the view
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter); // connect the data to the
LV via the adapter
    }
```

📖 5.3.4

Visual adjustment

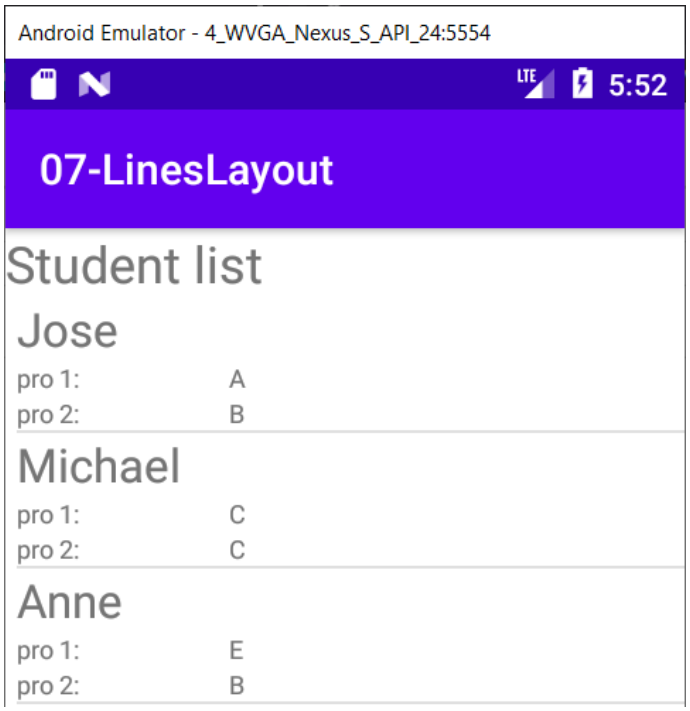
At first glance, a visual transgression strikes the eye - printing texts on the left edge of the screen.



However, we will not indent each item individually from the left, but we will indent the entire **ListView** in **activity_main.xml** by setting a value for **layout_marginLeft**.

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="5dp"
    android:id="@+id/listView"/>
```

The result:



5.3.5 The code of the application

MainActivity.java

```
package com.example.a07_lineslayout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import java.util.ArrayList;
import java.util.HashMap;

public class MainActivity extends AppCompatActivity {
    ArrayList<|HashMap<|String, String>> myList = new
ArrayList<|HashMap<|String, String>>();
    SimpleAdapter myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        fillMyData();
```

```
        connectMyAdapter();
    }

    private void connectMyAdapter() {
        String[] from = new String[]{"name", "PR1", "PR2"};
        int[] to = new int[]{R.id.item1, R.id.item2,
R.id.item3};

        myAdapter = new SimpleAdapter(
                this,             // context, usualy activity
                myList,                   // datasource
                R.layout.lines_list_layout, // rules for
visualisation
                from,
                to
        );

        // we get a link to the view
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter); // connect the data to the
LV via the adapter
    }

    private void fillMyData() {
        HashMap<|String, String> myRow = new HashMap<|String,
String>();
        myRow.put("name", "Jose");
        myRow.put("PR1", "A");
        myRow.put("PR2", "B");
        myList.add(myRow);

        myRow = new HashMap<|String, String>();
        myRow.put("name", "Michael");
        myRow.put("PR1", "C");
        myRow.put("PR2", "C");
        myList.add(myRow);

        myRow = new HashMap<|String, String>();
        myRow.put("name", "Anne");
        myRow.put("PR1", "E");
        myRow.put("PR2", "B");
        myList.add(myRow);

        myRow = new HashMap<|String, String>();
```

```
        myRow.put("name", "Sue");
        myRow.put("PR1", "E");
        myRow.put("PR2", "B");
        myList.add(myRow);
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <|TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Student list"
        android:textSize="24sp" />

    <|ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="5dp"
        android:id="@+id/listView"/>

<|/LinearLayout>
```

lines_list_layout.xml

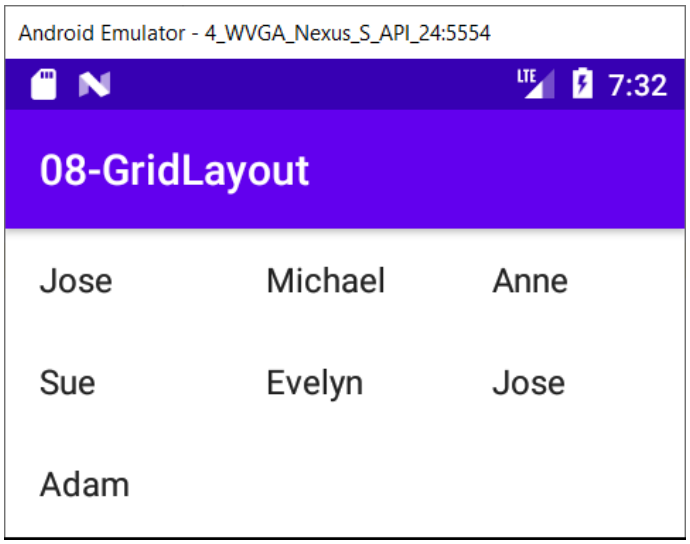
```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|TextView android:id="@+id/item1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:textSize="22sp"
```

```
        android:width="100dp"
        />

    <|LinearLayout
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:layout_height="fill_parent">

        <|TextView android:id="@+id/descr2"
            android:text="pro 1:"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

        <|TextView android:id="@+id/item2"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

    <|/LinearLayout>

    <|LinearLayout
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:layout_height="fill_parent">

        <|TextView android:id="@+id/descr3"
            android:text="pro 2:"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />

        <|TextView android:id="@+id/item3"
            android:layout_height="fill_parent"
            android:layout_width="wrap_content"
            android:textSize="12sp"
            android:width="100dip" />
    <|/LinearLayout>

<|/LinearLayout>
```

5.4 Grid layout

🕮 5.4.1

Arrange the list of students in a grid, so not only below each other, but also next to each other so that they fit on the screen as much as possible.



🕮 5.4.2

If we require the display of a single text data (name), it is sufficient for us to provide:

- a list of names stored in a list of type **ArrayList**
- a simple **ArrayAdapter** that always fills the data into the next item, regardless of whether the items are below or next to each other

The order in which the items of the display component are filled, which means whether it is first all the columns in a row and only then the next column, or vice versa, is given by the component.

```
public class MainActivity extends AppCompatActivity {
    ArrayList<String> myList = new ArrayList<String>();
    ArrayAdapter myAdapter;
```

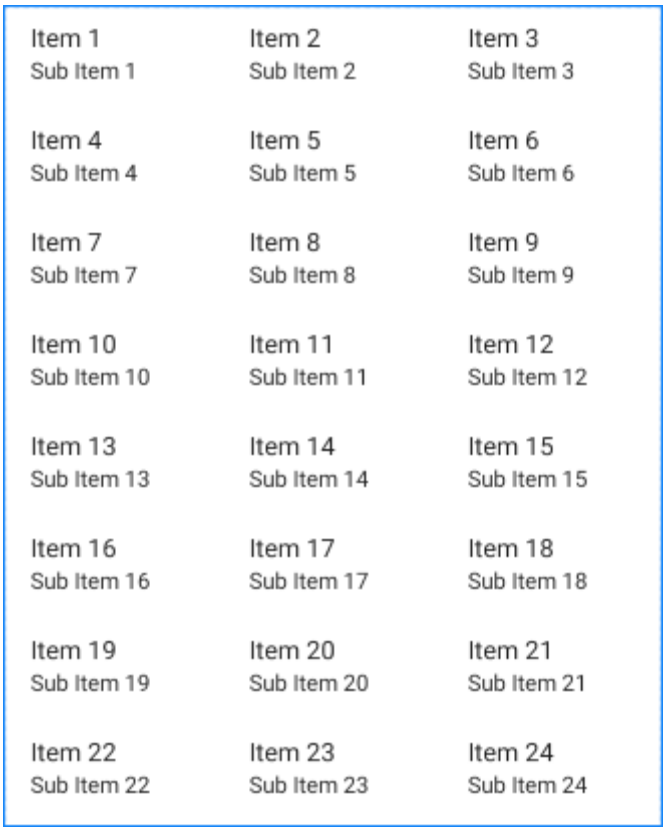
🕮 5.4.3

Defining the appearance of one item in the grid is not necessary, we just insert the **GridView** element into the activity instead of the **ListView**.

Of course, it is advisable to set the number of columns, without which one column would behave the same as the **ListView**.

```
<LinearLayout ...>
   <GridView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/gridView"
   android:numColumns="3" />
</LinearLayout>
```

With a preset appearance:



🕮 5.4.4

The connection between the data and the display component is provided by the standard simple_list_item_1

```
public class MainActivity extends AppCompatActivity {
    ArrayList<String> myList = new ArrayList<String>();
    ArrayAdapter myAdapter;
```

```
    private void fillMyData() {
        myList.add("Jose");
        myList.add("Michael");
        ...
    }

    private void connectMyAdapter() {
        myAdapter = new ArrayAdapter(
                this,             // context, usualy activity
                android.R.layout.simple_list_item_1,  //
visualisation template
                myList // data source
        );
        // we get a link to the view
        GridView gv = (GridView) findViewById(R.id.gridView);
        gv.setAdapter(myAdapter); // connect the data to the
LV via the adapter
    }
```

5.4.5 The code of the application

MainActivity.java

```
package com.example.a08_gridlayout;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.GridView;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import java.util.ArrayList;
import java.util.HashMap;

public class MainActivity extends AppCompatActivity {
    ArrayList<|String> myList = new ArrayList<|String>();
    ArrayAdapter myAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
        fillMyData();
        connectMyAdapter();
    }

    private void connectMyAdapter() {

        myAdapter = new ArrayAdapter(
                this,            // context, usualy activity
                android.R.layout.simple_list_item_1,
// datasource
                myList // rules for visualisation
        );

        // we get a link to the view
        GridView gv = (GridView) findViewById(R.id.gridView);
        gv.setAdapter(myAdapter); // connect the data to the
LV via the adapter
    }

    private void fillMyData() {
        myList.add("Jose");
        myList.add("Michael");
        myList.add("Anne");
        myList.add("Sue");
        myList.add("Evelyn");
        myList.add("Jose");
        myList.add("Adam");
    }
}
```

activity_main.xml

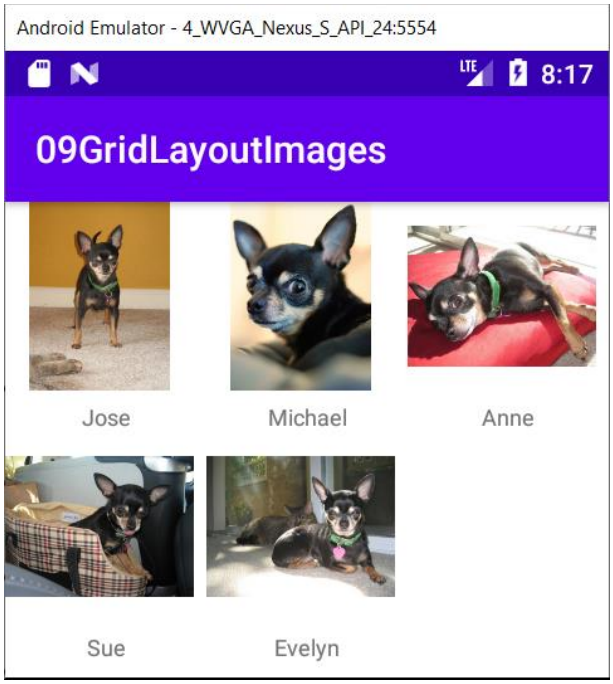
```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <|GridView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/gridView"
```

```
        android:numColumns="3"/>

<|/LinearLayout>
```

5.5 Grid layout (images)

🕮 5.5.1

Add photos to the students in the grid.



🕮 5.5.2

In addition to its simple form, **GridView** is also an element that can display any data in a two-dimensional and scrollable table.

In principle, our goal is to create a gallery which elements will include an image and a descriptive text.

🕮 5.5.3

We will create a standard layout of the activity with **GridView** and leave it with 3 columns set.

Each grid cell will contain an **ImageView** and a **TextView**, e.g. in the following form, which we define in a separate layout file - lets call it **grid_item_layout.xml**.

```
<LinearLayout ...
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:id="@+id/image"
        android:layout_width="100dp"
        android:layout_height="100dp" />

    <TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:gravity="center"
        android:textSize="12sp" />

</LinearLayout>
```

The size of the image, to which the content of the image will be adjusted without further settings, can be 100x100 pixels.

The **layout_marginTop** parameter for text sets the offset of the text from the image object so that they are not stuck on top of each other.

🕮 5.5.4

Structure for images

Since we want to display structured information, we will create a separate class with getters and setters to maintain it.

Bitmap is a class that allows us to store / save bitmap content that we obtain from any type of bitmap image (jpg, png ...).

```
public class ImageItem {
    private Bitmap image;
    private String title;

    public ImageItem(Bitmap image, String title) {
        super();
        this.image = image;
        this.title = title;
    }

    public Bitmap getImage() {
        return image;
    }

    public void setImage(Bitmap image) {
        this.image = image;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

🕮 5.5.5

List for items

ArrayList is enough for storing data, thanks to which we will be able to use **ArrayAdapter** as the default adapter. We declare it in another method, which will return the list to us as a result of its activities.

We will create a child from it by inheriting, for which we will describe how it should display the read data. The structure of the image and text is not a standard type for which an adapter would be ready without the need for modifications.

```
public class MainActivity extends AppCompatActivity {
   // ArrayList<ImageItem> imageItems = new ArrayList<>(); //
- will be created elsewhere
   private GridView gridView;
   private GridViewAdapter gridAdapter;
```

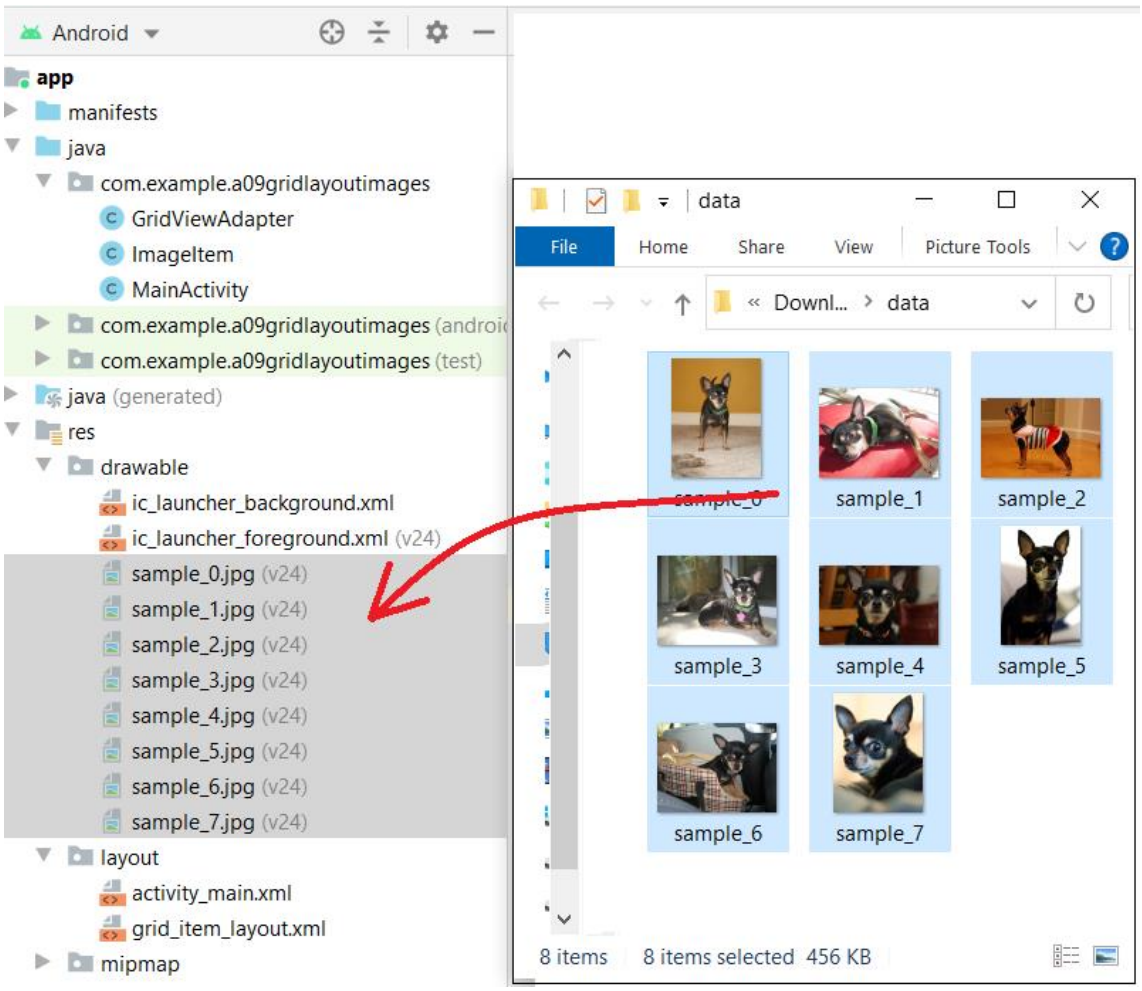
📖 5.5.6

Getting images

If we do not have a database, the images will probably be stored in files.

In order to make them available to the application, we need to insert them into the project structure.

Usually, physical copying to the **drawable** folder in the **res** section works.



📖 5.5.7

Adding elements to the list

The fact that we use images is nothing special

In the standard way, we get the source, decode the image from it, and place it in a list along with the name.

We will return the list created in this way as a result of our method, which we will call when creating the adapter. In this way, we want to show a different way of securing data than by using a class variable.

```
private ArrayList getData() {
     ArrayList<ImageItem> imageItems = new ArrayList<>();
     Bitmap b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_0);
     imageItems.add(new ImageItem(b, "Jose"));

     b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_7);
     imageItems.add(new ImageItem(b, "Michael"));

     b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_1);
     imageItems.add(new ImageItem(b, "Anne"));

     b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_6);
     imageItems.add(new ImageItem(b, "Sue"));

     b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_3);
     imageItems.add(new ImageItem(b, "Evelyn"));

    return imageItems;
}
```

🕮 5.5.8

GridView adapter

It will be a little more complicated to create an adapter to get the data into the grid.

We know that an adapter is a tool that transforms data from a data structure into a component and does it item by item (rows, cells) based on the prescription specified in the layout file.

In this case, we use two items - image and description.

So we will create our own class for "translation" of data / inflation derived from **ArrayAdapter**. For the needs of functionality we will remember:

- **context** - the activity used, because we refer to it in several places
- **layoutResourceID** - xml according to which we display the data in the cell
- **ArrayList** data - containing a list of data to be displayed

```
public class GridViewAdapter extends ArrayAdapter {
    private Context context;
    private int layoutResourceId;
    private ArrayList data = new ArrayList();
```

🕮 5.5.9

Constructor

In almost every derived class, we need to define a constructor, we select one of the simpler ones, and in it we call the parent and set the already mentioned values.

```
public GridViewAdapter(Context context, int layoutResourceId,
ArrayList data)  {
    super(context, layoutResourceId, data);
    this.layoutResourceId = layoutResourceId;
    this.context = context;
    this.data = data;
}
```

🕮 5.5.10

getView

Overriding the **getView()** method is necessary because in its body, it provides the creation of new items in the grid.

It is called when each item is displayed, and the elements defined in the template are inserted into the grid cell each time it is called.

Part of the method is to check that the object to be displayed is not null. If it is, it must be inserted - this situation occurs with elements that emerge from above or below when scrolling.

For filling with content, the so-called holder is used, which contains a list of views that are in the grid item.

By default, it is defined as a static structure that is part of the adapter.

```
static class ViewHolder {
    TextView imageTitle;
    ImageView image;
}
```

🕮 5.5.11

getView - code

The whole code of the method has the following form, the explanation is in the code.

```
public View getView(int position, View convertView, ViewGroup
parent) {
    View cell = convertView;
    ViewHolder holder = null;

    if (cell == null) { // if a cell is empty
        // we get inflater to prepare element
        LayoutInflater inflater = ((Activity)
context).getLayoutInflater();
        // we get a prescription according to which the cell
should be filled
        cell = inflater.inflate(layoutResourceId, parent,
false);
        // we will create a data space of a new cell in the
memory
        holder = new ViewHolder();
        // we point the holder at the components in the given
cell / row
        holder.imageTitle = (TextView)
cell.findViewById(R.id.text);
        holder.image = (ImageView)
cell.findViewById(R.id.image);
        // for the list / grid item we set the connection to
the appropriate values
        cell.setTag(holder);
    } else {
        // if the cell has already been created, we will only
return the connection to views
        holder = (ViewHolder) cell.getTag();
    }
    // in the holder variable are stored components that we
are going to fill
```

```
    // we read values from the source
    ImageItem item = (ImageItem)data.get(position);
    holder.imageTitle.setText(item.getTitle()); //
    holder.image.setImageBitmap(item.getImage());
    // returns the prepared row
    return cell;
}
```

🕮 5.5.12

getView - Inflater

In the adapter we get a **LayoutInflater**, which allows us to adjust the output display.

LayoutInflater class is used to create an instance of the layout of an XML file in the appropriate View object, a prescription enters from an XML file and creates Views from it

The **getView()** process basically consists of only two steps:

- get inflater
- use it to fill the resource

```
public View getView(int position, View convertView, ViewGroup
parent) {
       View cell = convertView;
       ViewHolder holder = null;
       if (cell == null) { // if the row is empty
           // we get the inflater
           LayoutInflater inflater = ((Activity)
context).getLayoutInflater();
           // and the prescription which defines how the cells
should look like
           cell = inflater.inflate(layoutResourceId, parent,
false);
```

🕮 5.5.13

Connecting the adapter

The adapter is ready at this point, we just need to use it to connect the data to the **GridView** in **MainActivity**.

```
private void connectMyAdapter() {
    gridView = (GridView) findViewById(R.id.gridView);
    gridAdapter = new GridViewAdapter(
            this,
            R.layout.grid_item_layout,
            getData()
    );
    gridView.setAdapter(gridAdapter);
}
```

⌨ 5.5.14 The code of the application

MainActivity.java

```
package com.example.a09gridlayoutimages;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.widget.GridView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
    private GridView gridView;
    private GridViewAdapter gridAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        connectMyAdapter();
    }

    private void connectMyAdapter() {
        gridView = (GridView) findViewById(R.id.gridView);
        gridAdapter = new GridViewAdapter(
                this,
                R.layout.grid_item_layout,
                getData()
        );
        gridView.setAdapter(gridAdapter);
```

```
    }

    private ArrayList getData() {
         ArrayList<|ImageItem> imageItems = new
ArrayList<|>();
         Bitmap b =
BitmapFactory.decodeResource(getResources(),
R.drawable.sample_0);
         imageItems.add(new ImageItem(b, "Jose"));

         b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_7);
         imageItems.add(new ImageItem(b, "Michael"));

         b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_1);
         imageItems.add(new ImageItem(b, "Anne"));

         b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_6);
         imageItems.add(new ImageItem(b, "Sue"));

         b = BitmapFactory.decodeResource(getResources(),
R.drawable.sample_3);
         imageItems.add(new ImageItem(b, "Evelyn"));

        return imageItems;
    }

}
```

ImageItem.java

```
package com.example.a09gridlayoutimages;

import android.graphics.Bitmap;

public class ImageItem {
    private Bitmap image;
    private String title;

    public ImageItem(Bitmap image, String title) {
        super();
        this.image = image;
        this.title = title;
    }
```

```
    public Bitmap getImage() {
        return image;
    }

    public void setImage(Bitmap image) {
        this.image = image;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}
```

GridViewAdapter.java

```
package com.example.a09gridlayoutimages;

import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;

public class GridViewAdapter extends ArrayAdapter {
    private Context context;
    private int layoutResourceId;
    private ArrayList data = new ArrayList();

    public GridViewAdapter(Context context, int
layoutResourceId, ArrayList data)  {
        super(context, layoutResourceId, data);
        this.layoutResourceId = layoutResourceId;
        this.context = context;
        this.data = data;
    }
```

```
    static class ViewHolder {
        TextView imageTitle;
        ImageView image;
    }

    public View getView(int position, View convertView,
ViewGroup parent) {
        View cell = convertView;
        ViewHolder holder = null;

        if (cell == null) { // if a cell is empty
            // we get inflater to prepare element
            LayoutInflater inflater = ((Activity)
context).getLayoutInflater();
            // we get a prescription according to which the
cell should be filled
            cell = inflater.inflate(layoutResourceId, parent,
false);
            // we will create a data space of a new cell in
the memory
            holder = new ViewHolder();
            // we point the holder at the components in the
given cell / row
            holder.imageTitle = (TextView)
cell.findViewById(R.id.text);
            holder.image = (ImageView)
cell.findViewById(R.id.image);
            // for the list / grid item we set the connection
to the appropriate values
            cell.setTag(holder);
        } else {
            // if the cell has already been created, we will
only return the connection to views
            holder = (ViewHolder) cell.getTag();
        }
        // in the holder variable are stored components that
we are going to fill
        // we read values from the source
        ImageItem item = (ImageItem)data.get(position);
        holder.imageTitle.setText(item.getTitle()); //
        holder.image.setImageBitmap(item.getImage());
        // returns the prepared row
        return cell;
    }
```

```
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<|GridView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/gridView"
    android:numColumns="3"/>

<|/LinearLayout>
```

grid_item_layout.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|ImageView
        android:id="@+id/image"
        android:layout_width="100dp"
        android:layout_height="100dp" />

    <|TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:gravity="center"
        android:textSize="12sp" />

<|/LinearLayout>
```

Databases

Chapter 6

6.1 Database

🕮 6.1.1

In order not to have to fill in the data in the lists after each launch of the application, it is best to use the database tools offered to us directly by the Android operating system.

The system has an integrated open source database system SQLite, which supports relational database standards:

- SQL
- transactions

Its great advantage is that after startup it needs only about 250 kB for operation.

The database system is part of every Android OS without the need for setup or administration.

The data stored in the database, which is assigned to your application by default, is secured and is not accessible to other applications unless we explicitly allow it.

🕮 6.1.2

Support of datatypes

As this is a simple system, we pay for its size with minor restrictions.

Provides data types:

- text / string
- integer / long
- real / double
- blob - saves data in the form in which they arrive

All other types must be converted to these basic ones (e.g. **datetime**)

When providing data, we can read data of one type as another type, e.g. data from an **int** field can be read as **String**.

🕮 6.1.3

Database in the system

Access to the database is provided at the file system level - data or files with them are created in the structure at **DATA/data/APP_NAME/databases/FILENAME**

- DATA - path obtainable from **Environment.getDataDirectory()**
- APP_NAME - The name of the application
- FILENAME - The name of the database

🕮 6.1.4

Assignment: database of students

Create an application capable of working with the student database, including an e-mail contact and age.

Provide the viewing of the list, adding, editing and deleting records.

🕮 6.1.5

Good practice 1

For Android database applications, it is common for table names and field names to be defined as static constants.

- older approach: in the class where a table is defined, there were constants for the given table
- new approach: contract class - common container for all application components - allows the use of constants in all application classes

The definition of a contract-class can take the form of:

```
public final class MyContract {
   public static class Student {
      public static final String TABLE_NAME = "students";
      public static final String COLUMN_ID = "id";
      public static final String COLUMN_NAME = "name";
      public static final String COLUMN_EMAIL = "email";
      public static final String COLUMN_AGE = "age";
   }
 // next tables
}
```

📖 6.1.6

Good practice 2

For each data entity we define a separate class with constructor, getters and setters:

```
public class Student {
 private long ID;
 private String name;
 private String email;
 private int age;

 public Student(long ID, String name, String email, int age) {
   this.ID = ID;
   this.name = name;
   this.email = email;
   this.age = age;
 }

 public long getID() { return ID; }
 public void setID(long ID) { this.ID = ID; }

 public String getName() { return name;}
 public void setName(String name) { this.name = name; }

 public String getEmail() { return email; }
 public void setEmail(String email) { this.email = email; }

 public int getAge() { return age; }
 public void setAge(int age) { this.age = age; }
}
```

📖 6.1.7

SQLiteOpenHelper

By default, a child element of the SQLiteOpenHelper class is used to work with the database, which:

- ensures the creation of a database
- updates the structure in case of version changes
- usually also works with data in a database

For use in a particular application, it requires the redefining of a pair of methods:

- **onCreate()** - executes if the database does not exist, ensures its creation
- **onUpgrade()** - one of the parameters of the database is also its version. If the structure changes or tables are added, this method makes the adjustments prescribed by the programmer to ensure the upgrade.

🕮 6.1.8

DBHelper

We will create a **DBHelper** class that will provide the **SQLiteOpenHelper** functions.

The use and the purpose of each part is described in the code:

```
public class DBHelper extends SQLiteOpenHelper {
   // version - for upgrades
 private static final int DATABASE_VERSION = 4;
   // database name – we use this when creating a new database
   // or open it if it already exists
 private static final String DATABASE_NAME = "students";

 public DBHelper(Context context) { // the constructor of the
database
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
 }

 @Override
 public void onCreate(SQLiteDatabase db) {
   //  we will return here - now we leave it empty
 }

 @Override
 public void onUpgrade(SQLiteDatabase db, int oldVers, int
newVers){
   // the input is the database and information about the
current database version
   // and the version number to which we want to upgrade to
 }
}
```

📖 6.1.9

onCreate()

The method ensures that the database is created, provided it does not exist.

If the database already exists in some form, **it will not start**.

It uses constants defined in contract-class when creating via SQL statements

```
@Override
public void onCreate(SQLiteDatabase db) {
    // every needed table is created here
    // - constants are used - from the MyContract class
  String SQL_CREATE_TABLE_STUDENT = "CREATE TABLE " +
MyContract.Student.TABLE_NAME + "("
       + MyContract.Student.COLUMN_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT,"
       + MyContract.Student.COLUMN_NAME + " TEXT, "
       + MyContract.Student.COLUMN_EMAIL + " TEXT, "
       + MyContract.Student.COLUMN_AGE + " INTEGER )";
  db.execSQL(SQL_CREATE_TABLE_STUDENT);
}
```

Contact-class had the form:

```
public final class MyContract {
  public static class Student {
     public static final String TABLE_NAME = "students";
     public static final String COLUMN_ID = "id";
     public static final String COLUMN_NAME = "name";
     public static final String COLUMN_EMAIL = "email";
     public static final String COLUMN_AGE = "age";
  }
}
```

📖 6.1.10

onUpgrade()

The method ensures that the table structure changes between database versions.

The main reason for using it is that the application does not deprive the user of data. If the user uses the application and it is updated for some reason (and its

version is upgraded), then the user should not lose the data obtained and saved through the older versions.

The approach of a programmer who would only create new tables and delete old ones with data would quickly make him unemployed.

Since the programmer does not know whether the user had a version of one or 10 lower (older), he needs to ensure a gradual transition from the user's version (**oldVers**) to the current one (**newVers**).

The content of the method is therefore usually a description of:

- if it's version 1 and you're going to 2 do this
- if it is version 2 and you go to 4 do this etc.

Sometimes fields are added, other times they are removed, or new tables are added and it is necessary to exchange data between them, etc.

Usually the update is carried out in a chain, e.g. between versions 1 to 4, the database is first upgraded to version 2, then from 2 to version 3, and finally from 3 to 4. And this approach should be considered when writing code.

🕮 6.1.11

onUpgrade() - example

In simple tasks, or in those where we do not want user data (we use a database to store data needed for the program, not for the user), we simply delete the table and call the **onCreate()** method to recreate it.

What is also the case with beginner tasks:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVers, int
newVers){
   // we remove the table with this name, if it exists
 db.execSQL("DROP TABLE IF EXISTS " +
MyContract.Student.TABLE_NAME);
   // we create a new table with the defined structure in
onCreate
 onCreate(db);
}
```

6.2 Access to data

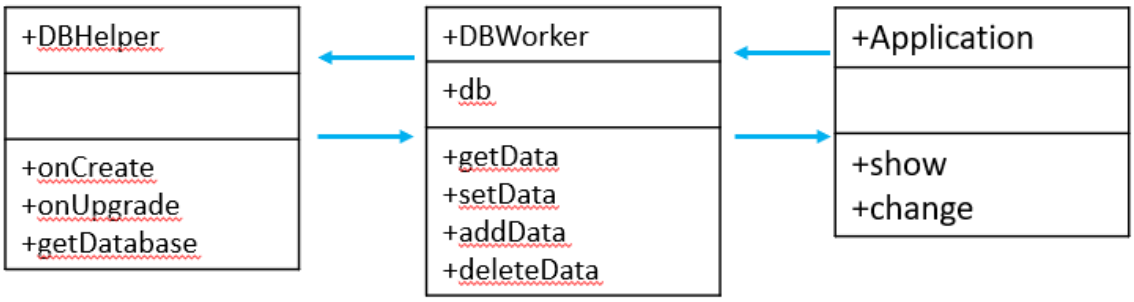
🕮 6.2.1

DBHelper

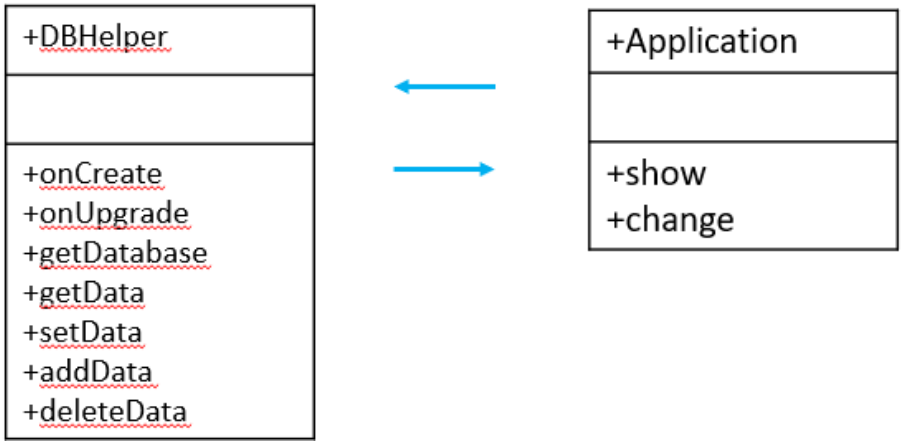
DBHelper is a descendant of **SQLiteOpenHelper** and represents a class that provides communication with the database.

We can use it in two ways:

- the **DBHelper** class will only represent the layer making the database accessible for data operations, i.e. it will not provide operations for adding, deleting, updating data - on the outside it will only provide access to the database. In this case, we will need another class, which will already contain specific operations for working with data.



- the DBHelper class can be extended compared to the initial version by methods working with data => all operations will be integrated into one class.



We will use the first option if we have several different classes for working with data (e.g. due to the complexity of the application), the second option in the case of simple applications working with one or a few tables.

In the current situation, we will use the second alternative.

🕮 6.2.2

So we need tools to:

- add student
- return the student data (e.g. based on ID)
- delete student (e.g. based on ID)
- change of student data (e.g. based on ID)

🕮 6.2.3

Adding a student

Within **DBHelper**, we will use the built-in method of the database called **insert**, which uses the named pairs of fields + value to insert, which represent the column into which the data is to be inserted and the data itself.

The input to the method is an instance of the **Student** class, from which we read the individual values and prepare them for insertion into the database.

The ID entry is ignored, the database inserts it automatically

```
public void addStudent(Student s) {
      // I create a map of values – column name + value
    ContentValues values = new ContentValues();
    values.put(MyContract.Student.COLUMN_NAME, s.getName());
    values.put(MyContract.Student.COLUMN_EMAIL, s.getEmail());
    values.put(MyContract.Student.COLUMN_AGE, s.getAge());

      // I get access to the database
    SQLiteDatabase db = getWritableDatabase();
      // I add a new line, where the return value is the ID
(primary key), which got assigned to the new record
    long newRowId = db.insert(
            MyContract.Student.TABLE_NAME, // názov tabuľky
            null,    // if we use
FeedEntry.COLUMN_NAME_NULLABLE,
                     // values of empty fields are null,
                     // if I use null, nothing will be in
there
```

```
            values); // the defined map of values
    db.close();
}
```

🕮 6.2.4

Obtaining student data

We will acquire the student data on the basis of their ID, other procedures (e.g. name, e-mail, etc.) are very similar, but they do not always guarantee the unambiguous acquisition of the right person.

```
public Student getStudent(long ID) {
      // I get database access
    SQLiteDatabase db = getWritableDatabase();

      // list of arrays to acquire
    String[] projection=
{MyContract.Student.COLUMN_NAME,MyContract.Student.COLUMN_EMAI
L, MyContract.Student.COLUMN_AGE};
    String selection = MyContract.Student.COLUMN_ID;       //
array in the condition (WHERE)
    String[] selectionArgs = {""+ID};        // list of values
for the condition

    Cursor c = db.query(
            MyContract.Student.TABLE_NAME,  // the table for
the query
            projection,     // lust if arrays for return
            selection,      // WHERE is going to be above this
column
            selectionArgs,  // values for the WHERE condition
            null,           // fields for GROUP BY
            null,           // HAVING
            null );        // sort order
...
```

🕮 6.2.5

Cursor

An instance of the **Cursor** class is the result of a query and represents a pointer that points to exactly one line of the query result (list of records).

In this way, Android can buffer data relatively efficiently - it does not need to load the entire result at once, but allows the user to move through the list through the cursor and keep in memory only what it needs

methods:

- **c.getCount()** - number of list elements
- **c.moveToFirst()** - move to the first record
- **c.moveToNext()** - move to the next record
- **c.isAfterLast()** - check if the last record was read
- **c.get*()** methods: getLong (columnIndex), getString (columnIndex) - return value from the column with the given sequence number

The use of **Cursor** must be closed after finishing the work - **c.close()**.

🕮 6.2.6

Reading students' data

We assume that the cursor in our case contains only one result - a student with the required ID.

```
Student s = new Student(ID,
  c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)
),
  c.getString(2),
  c.getInt(c.getColumnIndex(MyContract.Student.COLUMN_AGE));

c.close(); // closing the cursor
db.close(); // closing the database
return s;  // return the filled instance of the student
}
```

The column index number is used as an argument to retrieve data from a column using the getX() methods.

However, relying on the fact that the sequence number of the column will always be the same even after changes to the table is quite risky, so it is more appropriate to use the method to find the sequence number by the name of the column.

Via **getColumnIndex(column_name)** we find the index number of the column in the query and use it as an argument for the **getX** function.

📖 6.2.7

Deleting a student

Again, we will use the built-in method to delete the record according to the condition:

```
public void deleteStudent(int ID) {

  // standard approach
  db.delete(
       MyContract.Student.TABLE_NAME,           // name of the
table
       MyContract.Student.COLUMN_ID + "=" +ID,  // condition
       null);                                   // list of
parameters

 // approach with parameters (more secure)
 db.delete(
       MyContract.Student.TABLE_NAME, // table name
       MyContract.Student.COLUMN_ID + "= ?", // condition with
parameter
       new String[] { ""+ID }); // value of parameter
}
```

📖 6.2.8

Updating students data

Approaches to changing the data can be different.

Suppose we have a student's ID stored in the student instance, which is identical to the corresponding ID in the database, and in addition the instance contains new values for name, email, and age.

What we need to ensure in this case is updating the content of the individual fields for the given ID.

Again, we will use the built-in method (update) similar to the previous one:

```
public void updateStudent(Student s) {
      // we create a map of values - name of column + value
    ContentValues values = new ContentValues();
    values.put(MyContract.Student.COLUMN_NAME, s.getName());
    values.put(MyContract.Student.COLUMN_EMAIL, s.getEmail());
```

```
    values.put(MyContract.Student.COLUMN_AGE, s.getAge());

      // get access to the database
    SQLiteDatabase db = getWritableDatabase();
    db.update(
            MyContract.Student.TABLE_NAME,        // table
name
            values,  // defined pair values
            MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameter
            new String[] { ""+s.getID() }); // value of
parameters
    db.close(); // closing the database
}
```

📖 6.2.9

Why don't we use SQL?

We can use SQL through the **rawQuery()** and **execSQL()** methods, but we risk:

- error rate: if a semicolon occurs in the SQL text, some of the statements after it are "silently ignored"
- security: easier to get into raw SQL than into parameters which contents are secured
- current trends: database manipulation is increasingly common at the object level than at the relational level

📖 6.2.10

Displaying of data

Although we have an application capable of working with data in a database, we do not see it. In order to test what we have created, we need to ensure:

- filling the data
- displaying data in a list (**ListView**)

📖 6.2.11

Filling the data

We can create a method that runs after creating the application, preparing the database via **DBHelper**, and populating the table:

```
public class MainActivity extends AppCompatActivity {
     // getting the instance, which has access to the database
     // and I will use it in the activity
   DBHelper dbh = new DBHelper(this);

   private void PrepareStudents() {
       // id in these three are irrelevant
       // we dont use them when adding Students
     dbh.addStudent(new Student(1, "Mike", "mm", 15));
     dbh.addStudent(new Student(2, "Sue", "mm", 17));
     dbh.addStudent(new Student(3, "Peter", "mm", 16));
       // here the data will change for the student with the
entered ID with the entered values
     dbh.updateStudent(new Student(3, "Peterson", "mm", 14));
 }
```

🕮 6.2.12

List for visualization

In the case of a simple list of students, it is enough to create an ordinary list from the data from the database, which we get as a return value for **ListView** and **ArrayAdapter**.

```
public class DBHelper extends SQLiteOpenHelper {
...
 public ArrayList<String> getStudentNames() {
     // the creation of a list of strings, which are filled
with names
   ArrayList<String> zoznam = new ArrayList<String>();
   SQLiteDatabase db = getWritableDatabase(); // access to
database
   Cursor c = db.rawQuery( // reading the records from the
table
           "select * from " + MyContract.Student.TABLE_NAME,
null);
   if (c.moveToFirst()) { // if we can move to the first
record - it's not empty
     do { // we are adding to the list
```

```
       zoznam.add(c.getString(c.getColumnIndex(MyContract.Stud
ent.COLUMN_NAME)));
     } while (c.moveToNext()); // while another record exists
   }
   c.close();
   db.close(); // we close the cursor and the database
   return zoznam; // we return the list
 }
}
```

🕮 6.2.13

Connection to the adapter

We will provide it in the MainActivity activity:

```
public class MainActivity extends AppCompatActivity {
    DBHelper dbh = new DBHelper(this);
    ArrayAdapter myAdapter;

    private void connectAdapter() {
          // we use the predefined template of one item list
        myAdapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_1,
                dbh.getStudentNames() ); // list<String> of
students
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter);
    }
```

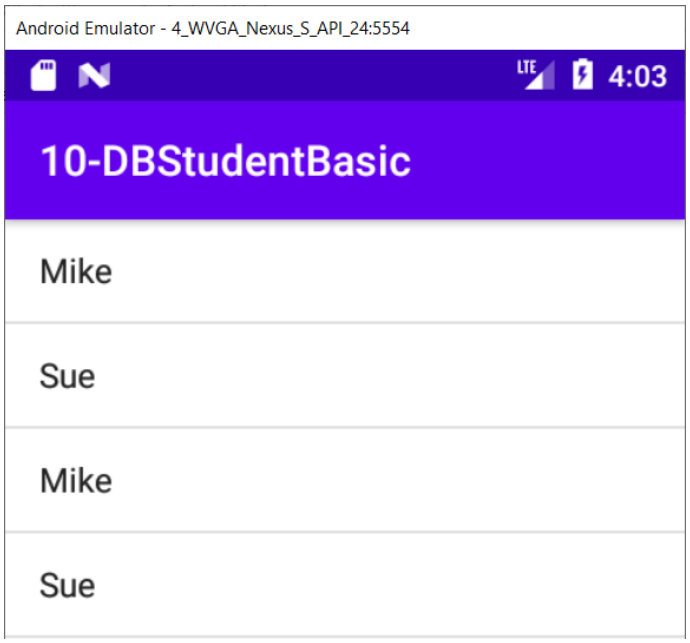
🕮 6.2.14

The result

The result in the form of displaying entries in the list is now a real thing.

Although the entries inserted at the start of the application are added every time it is started, they are not lost when it is closed and started as it was in the previous chapters.

The constant increase is due to calling the method to add students each time the application is started.



6.2.15 The code of the application

MainActivity.java

```
package com.example.a10_dbstudentbasic;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {
    ArrayAdapter myAdapter;
    // I get the instance, which has access to the database
    DBHelper dbh = new DBHelper(this);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        PrepareStudents();
        connectAdapter();
    }

    private void PrepareStudents() {
        dbh.addStudent(new Student(1,"Mike","mm",15));
```

```
        dbh.addStudent(new Student(1,"Sue","mm@ukf.dk",21));
    }

    private void connectAdapter() {
        //I use the predefined scheme for one-item layout
        myAdapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_1,
                dbh.getStudentNames() ); // list<|String> of
students
        ListView lv = (ListView) findViewById(R.id.listView);
        lv.setAdapter(myAdapter);
    }
}
```

MyContract.java

```
package com.example.a10_dbstudentbasic;

public final class MyContract {

    public static class Student {
        public static final String TABLE_NAME = "students";
        public static final String COLUMN_ID = "id";
        public static final String COLUMN_NAME = "name";
        public static final String COLUMN_EMAIL = "email";
        public static final String COLUMN_AGE = "age";
    }
}
```

Student.java

```
package com.example.a10_dbstudentbasic;

public class Student {
    private long ID;
    private String name;
    private String email;
    private int age;

    public Student(long ID, String name, String email, int
age) {
        this.ID = ID;
        this.name = name;
        this.email = email;
        this.age = age;
    }
```

```
    public long getID() {
        return ID;
    }

    public void setID(long ID) {
        this.ID = ID;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "ID=" + ID +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", age=" + age +
                '}';
    }
}
```

DBHelper.java

```
package com.example.a10_dbstudentbasic;
```

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import java.util.ArrayList;

public class DBHelper extends SQLiteOpenHelper {
    // version - due to upgrades
    private static final int DATABASE_VERSION = 1;
    //database  name - the database is created using this, if
it doesn't exist
    // or open if exists
    private static final String DATABASE_NAME = "students";

    public DBHelper(Context context) { // the constrctor of
the database
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        // every table is created here
        // - constants are used from the classes of the
defined tables
        String SQL_CREATE_TABLE_STUDENT = "CREATE TABLE " +
MyContract.Student.TABLE_NAME  + "("
                + MyContract.Student.COLUMN_ID  + " INTEGER
PRIMARY KEY AUTOINCREMENT ,"
                + MyContract.Student.COLUMN_NAME + " TEXT, "
                + MyContract.Student.COLUMN_EMAIL + " TEXT, "
                + MyContract.Student.COLUMN_AGE + " INTEGER
)";

        Log.d("",SQL_CREATE_TABLE_STUDENT);
        sqLiteDatabase.execSQL(SQL_CREATE_TABLE_STUDENT);
        Log.d("","ok");
    }

    @Override
```

```
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int
i, int i1){
        // we delete the table with this name, if it exists
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " +
MyContract.Student.TABLE_NAME);
        // we recreate the table based on the structure
defined in onCreate
        onCreate(sqLiteDatabase);
    }

    public void addStudent(Student s) {
        // we create a map of values - column name + value
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getName());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getAge());

        // getting access to the database
        SQLiteDatabase db = getWritableDatabase();
        // I enter a new row, while the return value is the ID
      // (Primary key), which was assigned to the new record
        long newRowId = db.insert(
                MyContract.Student.TABLE_NAME, // name of the
table
                null,    // if we use
FeedEntry.COLUMN_NAME_NULLABLE,
                // values of empty textfields will be null
                // if we use null, nothing will be in there
                values); // defined pairs - values
        db.close();
    }

    public Student getStudent(long ID) {
        // access to the database
        SQLiteDatabase db = getWritableDatabase();

        // list of fields to select
        String[] projection=
{MyContract.Student.COLUMN_NAME,MyContract.Student.COLUMN_EMAI
L, MyContract.Student.COLUMN_AGE};
        String selection = MyContract.Student.COLUMN_ID;
// the condition (WHERE)
```

```
        String[] selectionArgs = {""+ID};         // list of
values for the condition

        Cursor c = db.query(
                MyContract.Student.TABLE_NAME,   // table for
the query
                projection,      // the fields we want to be
returned
                selection,       // WHERE will be here
                selectionArgs,   // values for the WHERE
condition
                null,            // fields for GROUP BY
                null,            // HAVING
                null  );         // sort order

        Student s = new Student(ID,

c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)),
                c.getString(2),

c.getInt(c.getColumnIndex(MyContract.Student.COLUMN_AGE)));
        c.close();  // closing the cursor
        db.close(); // closing the database
        return s;   // return the instance of the Student
object
    }

    public void deleteStudent(int ID) {
        // access to the database
        SQLiteDatabase db = getWritableDatabase();

        // delete with parameters (more secure)
        db.delete(
                MyContract.Student.TABLE_NAME,       // table
name
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameter
                new String[] { ""+ID }); // values of
parameters
        db.close(); // closing the database
    }

    public void updateStudent(Student s) {
        // we create a map of values - column name + value
```

```
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getName());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getAge());

        // access to the database
        SQLiteDatabase db = getWritableDatabase();
        db.update(
                MyContract.Student.TABLE_NAME,        // name
of the table
                values,  // defined field pairs - value
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameter
                new String[] { ""+s.getID() }); // value of
parameters
        db.close(); // closing the database
    }
    public ArrayList<|String> getStudentNames() {
        // list of strings with names
        ArrayList<|String> myList = new ArrayList<|String>();

        // access to the database
        SQLiteDatabase db = getWritableDatabase();

        // reading the data from the table
        Cursor c = db.rawQuery("select * from " +
MyContract.Student.TABLE_NAME, null);

        if (c.moveToFirst()) { // if we can move to the start
            do { // we will be adding to the list

myList.add(c.getString(c.getColumnIndex(MyContract.Student.COL
UMN_NAME)));
            } while (c.moveToNext()); // while we can move to
the next item
        }

        c.close(); // closing the cursor
        db.close(); // closing the database
        return myList; // returning the list
    }
}
```

activity_main.xml

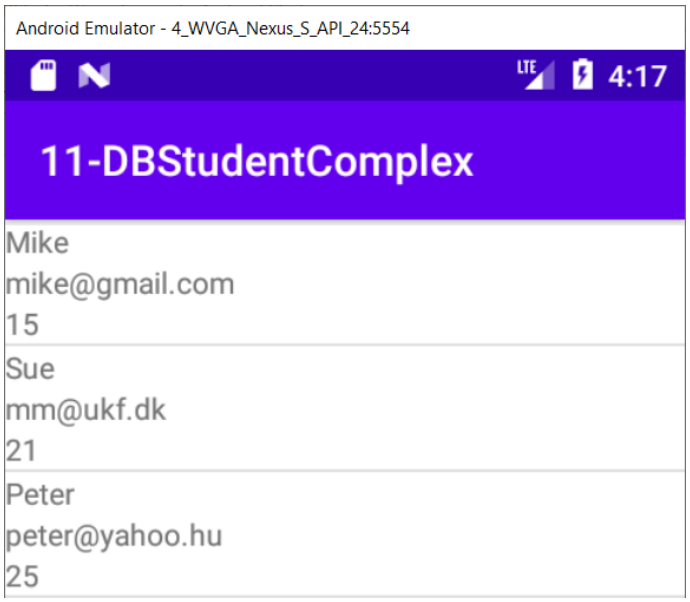
```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <|ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listView" />

<|/LinearLayout>
```

6.3 The visualisation of the table contents

🕮 6.3.1

Multi-item layout

Create a design that displays all the information about the students, not just the name.



🕮 6.3.2

We can use the **SimpleAdapter** known from previous lessons to provide data, which allows us to fill **ListView** items with multiple lines or views.

We therefore define the appearance of one **ListView** item via a layout file, e.g. list_layout.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/id1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:visibility="gone"/>

    <TextView android:id="@+id/name1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>

    <TextView android:id="@+id/email1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>

    <TextView android:id="@+id/age1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>
</LinearLayout>
```

We will need the ID item to identify the record, but it is not advisable to see it - we will set it to invisible via the **visibility** parameter set to **gone**.

🕮 6.3.3

In order to use the **SimpleAdapter**, we first need to prepare the data in the form of a map (**HashMap**).

We will create a method that returns a list of maps as part of **DBHelper**:

```
public List<HashMap<String, String>> getStudents() {
```

```
    SQLiteDatabase db = getWritableDatabase(); // access to
database
      // declaration of hashmaps that the SimpleAdapter needs
    List<HashMap<String, String>> mylist = new
ArrayList<HashMap<String, String>>();
      // we get the cursor to the result of the query entered
directly
    Cursor c = db.rawQuery("select * from " +
MyContract.Student.TABLE_NAME, null);
    if (c.moveToFirst()) { // if we can move to the start = a
result exists
        do {  // we create a new map
            HashMap<String, String> map = new HashMap<String,
String>();
            // we insert values from the query fields into the
fields with the specified names
            map.put("id",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_ID)));
            map.put("name",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)))
;

map.put("email",c.getString(c.getColumnIndex(MyContract.Studen
t.COLUMN_EMAIL)));
            map.put("age",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_AGE)));
            mylist.add(map); // we add the map to the list
        } while (c.moveToNext());  // if there are more, we
continue, if not, the loop ends here
    }
    c.close(); db.close(); // we close the cursor and database
    return zoznam;   // we return the list of maps for the
SimpleAdapter
}
```

📖 6.3.4

The display of items in the **ListView** itself is in charge of a suitably configured **SimpleAdapter**:

We create lists **from** (from where to get values) and **to** (where to insert them) directly in the command to create an adapter by naming the values of the list.

```
public class MainActivity extends AppCompatActivity {
```

```
    SimpleAdapter myAdapter;
    DBHelper dbh = new DBHelper(this); // access to database

    private void connectAdapter() {
        myAdapter = new SimpleAdapter(
                this,
                  // we get the list of maps = information
about the students
                dbh.getStudents(),
                  // the information is added to this layout
                R.layout.list_layout,
                  // name of the keys/variables from the
created map
                new String[] { "id","name","age","email"},
                  // the content of the variables are added to
these elements
                new int[] {R.id.id1, R.id.name1, R.id.age1,
R.id.email1 }
        );
        ListView lv= (ListView)findViewById(R.id.listView);
        lv.setAdapter(myAdapter);
    }
```

📖 6.3.5

Discussion

This solution is correct, but it does not correspond to the characteristics of application development, which we are used to when working with Android. Until now, it was true that frequently performed operations can be secured with a minimum amount of code ...

... and working with a database to display values from the database to lists seems to be a common operation.

⌨ 6.3.6 The code of the application

MainActivity.java

```
package com.example.a11_dbstudentcomplex;

import androidx.appcompat.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;

public class MainActivity extends AppCompatActivity {
    SimpleAdapter myAdapter;
    // instance that has access to the database
    DBHelper dbh = new DBHelper(this);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        PrepareStudents();
        connectAdapter();
    }

    private void PrepareStudents() {
        dbh.addStudent(new
Student(1,"Mike","mike@gmail.com",15));
        dbh.addStudent(new Student(2,"Sue","mm@ukf.dk",21));
        dbh.addStudent(new
Student(3,"Peter","peter@yahoo.hu",25));
    }

    private void connectAdapter() {
        myAdapter = new SimpleAdapter(
                this,
                dbh.getStudents(),// we get the list of maps =
needed information about the students
                R.layout.list_layout, // the information is
added to this layout
                new String[] { "id","name","age","email"}, //
names of key/value pairs
                // from the created map
                new int[] {R.id.id1, R.id.name1, R.id.age1,
R.id.email1 } // the content of
                // the variables
                // to be displayed in these containers
        );
        ListView lv= (ListView)findViewById(R.id.listView);
        lv.setAdapter(myAdapter);
    }
```

```
}
```

MyContract.java

```
package com.example.a11_dbstudentcomplex;

public final class MyContract {

    public static class Student {
        public static final String TABLE_NAME = "students";
        public static final String COLUMN_ID = "id";
        public static final String COLUMN_NAME = "name";
        public static final String COLUMN_EMAIL = "email";
        public static final String COLUMN_AGE = "age";
    }
}
```

Student.java

```
package com.example.a11_dbstudentcomplex;

public class Student {
    private long ID;
    private String name;
    private String email;
    private int age;

    public Student(long ID, String name, String email, int
age) {
        this.ID = ID;
        this.name = name;
        this.email = email;
        this.age = age;
    }

    public long getID() {
        return ID;
    }

    public void setID(long ID) {
        this.ID = ID;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "ID=" + ID +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", age=" + age +
                '}';
    }
}
```

DBHelper.java

```
package com.example.a11_dbstudentcomplex;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
public class DBHelper extends SQLiteOpenHelper {
    // version - due to upgrades
    private static final int DATABASE_VERSION = 1;
    // name of the database - used to create the database if
it doesn't exists
    // or open, if it does
    private static final String DATABASE_NAME = "students";

    public DBHelper(Context context) { // the constructor of
the database
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        // every table is created here
        // - constants are used from the specified structure
of the table
        String SQL_CREATE_TABLE_STUDENT = "CREATE TABLE " +
MyContract.Student.TABLE_NAME  + "("
                + MyContract.Student.COLUMN_ID  + " INTEGER
PRIMARY KEY AUTOINCREMENT ,"
                + MyContract.Student.COLUMN_NAME + " TEXT, "
                + MyContract.Student.COLUMN_EMAIL + " TEXT, "
                + MyContract.Student.COLUMN_AGE + " INTEGER
)";

        Log.d("",SQL_CREATE_TABLE_STUDENT);
        sqLiteDatabase.execSQL(SQL_CREATE_TABLE_STUDENT);
        Log.d("","ok");
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int
i, int i1){
        // delete the table with this name if it exists
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " +
MyContract.Student.TABLE_NAME);
        // we recreate the table based on the structure
defined in onCreate
        onCreate(sqLiteDatabase);
    }
```

```
    public void addStudent(Student s) {
        // we create the map of values - column name + value
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getName());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getAge());

        // we get access to the database
        SQLiteDatabase db = getWritableDatabase();
        // we create a new row, where the return value is the
ID (primary key), which was assigned                    to
the new record
        long newRowId = db.insert(
                MyContract.Student.TABLE_NAME, // name of the
table
                null,    // if we use
FeedEntry.COLUMN_NAME_NULLABLE,
                // values for empty textfields are null,
                // if we use null, nothing will be in there
                values); // defined pairs - values
        db.close();
    }

    public Student getStudent(long ID) {
        // access to the database
        SQLiteDatabase db = getWritableDatabase();

        // list of fields to select
        String[] projection=
{MyContract.Student.COLUMN_NAME,MyContract.Student.COLUMN_EMAI
L, MyContract.Student.COLUMN_AGE};
        String selection = MyContract.Student.COLUMN_ID;
// fields in the condition (WHERE)
        String[] selectionArgs = {""+ID};         // list of
values for the condition

        Cursor c = db.query(
                MyContract.Student.TABLE_NAME,   //  table for
the query
                projection,      // list of fields to return
                selection,       // WHERE will be here
```

```
                selectionArgs,   // values for the WHERE
condition
                null,            // fields for GROUP BY
                null,            // HAVING
                null  );         // sort order

        Student s = new Student(ID,

c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)),
                c.getString(2),

c.getInt(c.getColumnIndex(MyContract.Student.COLUMN_AGE)));
        c.close();  // closing the cursor
        db.close(); // closing the database
        return s;   // return the instance of the Student
object
    }

    public void deleteStudent(int ID) {
        // accessing the database
        SQLiteDatabase db = getWritableDatabase();

        // delete with parameters (more secure)
        db.delete(
                MyContract.Student.TABLE_NAME,        // name
of the table
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameters
                new String[] { ""+ID }); // values of
parameters
        db.close(); // closing the database
    }

    public void updateStudent(Student s) {
        // creating a map of values - column name + value
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getName());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getAge());

        // accessing the database
        SQLiteDatabase db = getWritableDatabase();
```

```
        db.update(
                MyContract.Student.TABLE_NAME,       // name
of the table
                values,  // defined pairs - values
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameter
                new String[] { ""+s.getID() }); // values of
parameters
        db.close(); // closing the database
    }
    public ArrayList<|String> getStudentNames() {
        // we create a list of strings that will be filled
with names
        ArrayList<|String> myList = new ArrayList<|String>();

        // accessing the database
        SQLiteDatabase db = getWritableDatabase();

        // reading the data from the table
        Cursor c = db.rawQuery("select * from " +
MyContract.Student.TABLE_NAME, null);

        if (c.moveToFirst()) { // if we can move to the start
            do { // I will be adding to the list

myList.add(c.getString(c.getColumnIndex(MyContract.Student.COL
UMN_NAME)));
            } while (c.moveToNext()); // while we can move to
the next element
        }

        c.close(); // we close the cursor
        db.close(); // closing the database
        return myList; // returning the list
    }

    public List<|HashMap<|String, String>> getStudents() {
        SQLiteDatabase db = getWritableDatabase(); //
accessing the database
          // declaring the list of maps needed by the
SimpleAdapter
        List<|HashMap<|String, String>> myList = new
ArrayList<|HashMap<|String, String>>();
          // get the cursor to the result of the raw query
```

```
        Cursor c = db.rawQuery("select * from
"+MyContract.Student.TABLE_NAME, null);
        if (c.moveToFirst()) { // if we can move to the start
= a result exists
            do {  // we create a new map
                HashMap<|String, String> map = new
HashMap<|String, String>();
                // to the field we add the values from the
query
                map.put("id",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_ID)));
                map.put("name",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)))
;

map.put("email",c.getString(c.getColumnIndex(MyContract.Studen
t.COLUMN_EMAIL)));
                map.put("age",
c.getString(c.getColumnIndex(MyContract.Student.COLUMN_AGE)));
                myList.add(map); // we add the map to the list
            } while (c.moveToNext());  // if we can move to
the next we do, if not, the loop ends
        }
        c.close(); db.close(); // we close the cursor and the
database
        return myList;   // we return the list of maps for the
SimpleAdapter
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <|ListView
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listView" />

<|/LinearLayout>
```

list_layout.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|TextView android:id="@+id/id1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:visibility="gone"/>

    <|TextView android:id="@+id/name1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>

    <|TextView android:id="@+id/email1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>

    <|TextView android:id="@+id/age1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"/>
<|/LinearLayout>
```

Database Application

Chapter 7

7.1 Cursor adapter

🕮 7.1.1

Cursor Adapter

CursorAdapter is a type of adapter that directly processes data obtained from a cursor. The input for it is the cursor and the description of the data link: data about which fields go to which views.

In this sense, it is again a variant of **SimpleAdapter**.

As a first solution, we will try to use the built-in **SimpleCursorAdapter** class.

🕮 7.1.2

Appearance of the output

The layout of the **ListView** items could basically be left the same as in the previous case. The items are placed below each other.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView android:id="@+id/id1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:visibility="gone"
        android:width="100dip" />

    <TextView android:id="@+id/name1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />

    <TextView android:id="@+id/email1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />

    <TextView android:id="@+id/age1"
```

```
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />
</LinearLayout>
```

📖 7.1.3

Creating an adapter

to see the similarity with SimpleAdapter, we only slightly modify the connection to the Cursor adapter.

```
public class MainActivity extends AppCompatActivity {
    SimpleCursorAdapter myAdapter;
    DBHelper dbh = new DBHelper(this);// we get the instance,
which has access to the database

    private void connectAdapter() {
       myAdapter = new SimpleCursorAdapter(
            this,
            R.layout.list_layout, // defined layout
            dbh.getMyCursor(),    // gets the cursor = pointer
to the student data
            // = for the query
            new String[] {
MyContract.Student.COLUMN_ID,MyContract.Student.COLUMN_NAME,
                          MyContract.Student.COLUMN_EMAIL,
                          MyContract.Student.COLUMN_AGE }, //
list from - column names from the query
            new int[] {R.id.id1, R.id.name1, R.id.email1,
R.id.age1}, // display destinations
            0); // flag defines adapter behavior - 0 is OK

       ListView lv = (ListView)findViewById(R.id.listView);
       lv.setAdapter(myAdapter);
    }
```

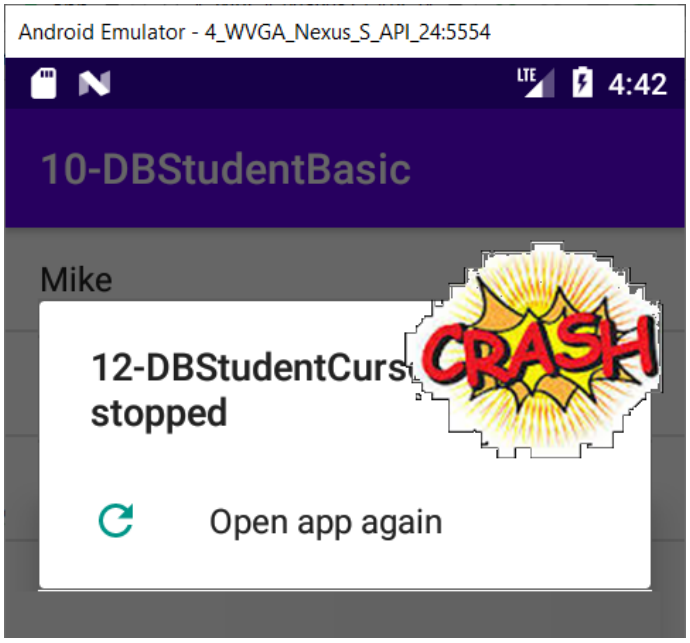
📖 7.1.4

Retrieving data from a database

getMyCursor() is a method of the **DBHelper** class that returns a cursor to the retrieved list of students.

```
public Cursor getMyCursor() {
      // access to the database
    SQLiteDatabase db = getWritableDatabase();
    String sstr = "select * from " +
MyContract.Student.TABLE_NAME;
      // getting the cursor using a query
    Cursor c = db.rawQuery(sstr, null);
      // usually we have to point the cursor to the first /
before the first record
    c.moveToFirst();
    db.close();
    return c;
 }
```

This method is really very simple, it just gets the result of the query and returns it as a parameter. With this result = list of data, the adapter can playfully deal with it... or does it?



... the application or activity crashes after starting. Finding a reason is a non-trivial, primitive reason:

_id, not id, must be used as the primary key field name.

🕮 7.1.5

New rule

We must, of course, respect this requirement and define the primary key field with an underscore: **_id**

So let's modify the definition of the **MyContract** class.

```
public final class MyContract {
   public static class Student {
       public static final String TABLE_NAME = "students";
       public static final String COLUMN_ID = "_id";
       public static final String COLUMN_NAME = "name";
       public static final String COLUMN_EMAIL = "email";
       public static final String COLUMN_AGE = "age";
   }
}
```

🕮 7.1.6

Database version change

Remember to change / increase the constant defining the database version before starting.

```
public class DBHelper extends SQLiteOpenHelper {
      // version - due to upgrades increase version
    private static final int DATABASE_VERSION = 2;
    private static final String DATABASE_NAME = "students.db";
```

Without this change, the application does not know after launch that it has to re-create the table with students and works with what is currently on the device - the key field is named **id** and not **_id**.

Changing the version will ensure calling the **onUpgrade()** method, in which we took care of deleting the original tables and creating them again according to the currently set constants in the **MyContract** class.

```
@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i,
int i1){
      // we delete the table with this name if it exists
    sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " +
MyContract.Student.TABLE_NAME);
      // we create a new table with the structure defined in
onCreate
    onCreate(sqLiteDatabase);
}
```

onCreate() has the form known from previous tasks.

```
public void onCreate(SQLiteDatabase sqLiteDatabase) {
    String SQL_CREATE_TABLE_STUDENT = "CREATE TABLE " +
MyContract.Student.TABLE_NAME  + "("
            + MyContract.Student.COLUMN_ID  + " INTEGER
PRIMARY KEY AUTOINCREMENT ,"
            + MyContract.Student.COLUMN_NAME + " TEXT, "
            + MyContract.Student.COLUMN_EMAIL + " TEXT, "
            + MyContract.Student.COLUMN_AGE + " INTEGER )";

    sqLiteDatabase.execSQL(SQL_CREATE_TABLE_STUDENT);
}
```

🕮 7.1.7

By default, the **SimpleCursorAdapter** is sufficient for displaying data from the database, only in special cases, we need to use our own **CursorAdapter (e.g.** for different data formatting). The redefining is based on the same principle as the grid image adapter.

7.2 CRUD operations

🕮 7.2.1

Adding an item

Adding is done by loading data from a separate (new) activity. After obtaining the individual attributes of the student, we will use the **addStudent()** method from **DBHelper** for the addition itself.

After adding a new item, it is necessary to refresh the list = to get the cursor again, which will already contain the newly added element in its list.

The cursor cannot identify that changes have occurred in the database and the programmer must take care of re-creating the cursor and notify the user.

🕮 7.2.2

Editing an item

Editing consists of:

- identification of the record we want to edit,
- display of student data,
- overwriting data by the users
- overwriting of data in the database in case the editing is confirmed

To find out which row it is, we capture the click event on the **ListView** row.

For this row, we find the position of the **cursor** and read the **ID** according to it.

We will send the obtained **ID** to another (again separate) activity and process it.

We add an item click listener to the adapter connection:

```
ListView lv= (ListView)findViewById(R.id.listView);
...
lv.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
        // I find my listview
        ListView lv= (ListView)findViewById(R.id.listView);
        // from the listview we get its cursor
        Cursor cursor = ((SimpleCursorAdapter)
lv.getAdapter()).getCursor();
        // position is the parameter from the events
on&long&click
        cursor.moveToPosition(position);
        // we get the ID
        long ID =
cursor.getLong(cursor.getColumnIndex(MyContract.Student.COLUMN
_ID));
        Toast.makeText(MainActivity.this, ""+ID,
Toast.LENGTH_SHORT).show();
    }
});
```

🕮 7.2.3

Deleting an item

Since a standard click will cause edits, we must use another action to try to delete, which can also identify a specific record.

Most often, a long click on the item is used, which in our case opens a dialog to confirm the deletion.

The listener for capturing a long click is **OnItemLongClickListener**, we will add it to our **ListView**.

🕮 7.2.4

Dialog

We should ask for user confirmation before deleting the record.

The answer to a simple question is easiest to get through dialogs

for dialogs we can use:

- **AlertDialog** - simple designs with text and button (buttons)
- **Dialog** - for customized dialogs

🕮 7.2.5

AlertDialog

AlertDialog uses a builder to create a window for its dialog, which, after its creation, sets the parameters of the dialog.

We can create it

either by gradually setting the parameters

```
     // we create the builder, which builds a dialog
  AlertDialog.Builder builder = new
AlertDialog.Builder(parent.getContext());
    // the dialog is built with set parameters
  builder.setMessage("Do you really want to delete?");
  builder.setTitle("Delete");
```

or by concatenation into a single command

```
       // we create the builder, which builds a dialog
  AlertDialog.Builder builder = new
AlertDialog.Builder(parent.getContext());
     // the dialog is built with set parameters
  builder.setMessage("Do you really want to delete?") //
without semicolon, the definition continues
         .setTitle("Delete")...
```

📖 7.2.6

AlertDialog - Buttons

Like other parameters, we add buttons.

As soon as we add buttons for them, we also define the code for the click event (onClick):

```
builder.setMessage("Do you really want to delete?")
     .setTitle("Delete")
     .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int
which) {
           // I can call the record deletion here
           dialog.dismiss(); // close
         }
      })
     .setNegativeButton("No", new
DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int
which) {
           dialog.dismiss(); // close
         }
     });
```

📖 7.2.7

AlertDialog - initialize

After setting all the parameters of the dialog, we can display it

```
builder.setMessage("Do you really want to delete?")
      ...
builder.show();
....
```

Be careful, the dialogue is not modal

If there is code after **builder.show()**, it is executed immediately after the dialog box is displayed without waiting for the button to be pressed.

If the programmers does not like this approach, they can create his own activity with his own buttons and open it as a modal window.

🕮 7.2.8

Deletion - code

In order to be able to identify the ID of the item we want to delete, we need to repeat the same procedure as for editing:

```
lv.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
  @Override
  public boolean onItemLongClick(AdapterView<?> parent, View
view, final int position, long id) {
      // we create he builder, which builds the dialog
    AlertDialog.Builder builder = new
AlertDialog.Builder(parent.getContext());
      // the dialog is built with specific parameters
    builder.setMessage("Do you really want to delete?")
         .setTitle("Delete")
         .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
            @Override
             public void onClick(DialogInterface dialog, int
which) {
               // I find my ListView
             ListView lv=
(ListView)findViewById(R.id.listView);
               // I get the cursor connected to the ListView
             Cursor cursor = ((SimpleCursorAdapter)
lv.getAdapter()).getCursor();
```

```
                // position is parameter from events
on&long&cick
                cursor.moveToPosition(position);
                  // I get the ID and call delete
                long ID =
cursor.getLong(cursor.getColumnIndex(MyContract.Student.COLUMN
_ID));
                dialog.dismiss(); // close
                Toast.makeText(MainActivity.this, "deleting",
Toast.LENGTH_SHORT).show();
            }
          })
         .setNegativeButton("No", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int
which) {
              dialog.dismiss(); // close
              Toast.makeText(MainActivity.this, "nothing is
happening", Toast.LENGTH_SHORT).show();
            }
          });
      // Show the dialog
    builder.show();
    return false; // return from onItemLongClick
  }});
}
```

⌨ 7.2.9 The code of the application

MainActivity.java

```
package com.example.a12_dbstudentcursor;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
```

```
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    SimpleCursorAdapter myAdapter;
    DBHelper dbh = new DBHelper(this);// get the instance
which has access to the database

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        PrepareStudents();
        connectAdapter();
    }

    private void PrepareStudents() {
        dbh.addStudent(new
Student(1,"Mike","mike@gmail.com",15));
        dbh.addStudent(new Student(2,"Sue","mm@ukf.dk",21));
        dbh.addStudent(new
Student(3,"Peter","peter@yahoo.hu",25));
    }

    private void connectAdapter() {
        myAdapter = new SimpleCursorAdapter(
                this,
                R.layout.list_layout, // the defined layout
                dbh.getMyCursor(),     // cursor - pointing at
student data
                // = for query
                new String[] {
MyContract.Student.COLUMN_ID,MyContract.Student.COLUMN_NAME,My
Contract.Student.COLUMN_EMAIL,MyContract.Student.COLUMN_AGE},
// list from - names
                // fields from the query
                new int[] {R.id.id1, R.id.name1, R.id.email1,
R.id.age1}, // target containers
                // displaying
                0); // flag defines the behavior of the
adapter, 0 is OK

        ListView lv= (ListView)findViewById(R.id.listView);
        lv.setAdapter(myAdapter);
```

```
        lv.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<|?> parent,
View view, int position, long id) {
                // we get the reference to our ListView
                ListView lv=
(ListView)findViewById(R.id.listView);
                // we get the cursor of the ListView
                Cursor cursor = ((SimpleCursorAdapter)
lv.getAdapter()).getCursor();
                // position is the parameter from the events
on&long&cick
                cursor.moveToPosition(position);
                // we get the ID
                long ID =
cursor.getLong(cursor.getColumnIndex(MyContract.Student.COLUMN
_ID));
                Toast.makeText(MainActivity.this, ""+ID,
Toast.LENGTH_SHORT).show();
            }
        });

        lv.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<|?>
parent, View view, final int position, long id) {
                // we create a builder, which builds a dialog
                AlertDialog.Builder builder = new
AlertDialog.Builder(parent.getContext());

                // create the dialog with defined parameters
                builder.setMessage("Do you really want to
delete?");
                builder.setTitle("Delete")
                        .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                            @Override
                            public void
onClick(DialogInterface dialog, int which) {
                                // reference to our ListView
                                ListView lv=
(ListView)findViewById(R.id.listView);
```

```
                                    // cursor from the ListView
                                    Cursor cursor =
((SimpleCursorAdapter) lv.getAdapter()).getCursor();
                                    // position is the parameter
from the event on&long&cick

cursor.moveToPosition(position);
                                    // get the ID
                                    long ID =
cursor.getLong(cursor.getColumnIndex(MyContract.Student.COLUMN
_ID));
                                    // and then we call the
deletion here
                                    dialog.dismiss(); // close


Toast.makeText(MainActivity.this, "Deleting",
Toast.LENGTH_SHORT).show();
                                }
                            })
                            .setNegativeButton("No", new
DialogInterface.OnClickListener() {
                                @Override
                                public void
onClick(DialogInterface dialog, int which) {
                                    dialog.dismiss(); // close

Toast.makeText(MainActivity.this, "Nothing is happening",
Toast.LENGTH_SHORT).show();
                                }
                            });
                    // show the created dialog
                    builder.show();
                    return false;
                }});
    }

}
```

DBHelper.java

```
package com.example.a12_dbstudentcursor;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DBHelper extends SQLiteOpenHelper {
    // version - due to upgrades
    private static final int DATABASE_VERSION = 2;
    // database name - the database is created with this name
    // or opens with it, if it exists
    private static final String DATABASE_NAME = "students.db";

    public DBHelper(Context context) { // the constrctor of
the database
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        // every table is created here
        // - constants are called from the class defininig
the table
        String SQL_CREATE_TABLE_STUDENT = "CREATE TABLE " +
MyContract.Student.TABLE_NAME  + "("
                + MyContract.Student.COLUMN_ID  + " INTEGER
PRIMARY KEY AUTOINCREMENT ,"
                + MyContract.Student.COLUMN_NAME + " TEXT, "
                + MyContract.Student.COLUMN_EMAIL + " TEXT, "
                + MyContract.Student.COLUMN_AGE + " INTEGER
)";

        sqLiteDatabase.execSQL(SQL_CREATE_TABLE_STUDENT);
        Log.d("create","created");
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int
i, int i1){
        // we delete the table with this name, if it exists
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " +
MyContract.Student.TABLE_NAME);

        // we recreate the table using the defined structure
in onCreate
        onCreate(sqLiteDatabase);
```

```
    }

    public void addStudent(Student s) {
        // we create a map of values - column name + value
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getMeno());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getVek());

        // accessing the database
        SQLiteDatabase db = getWritableDatabase();
        // adding a new row, while the return value is an ID
(primary key), which was assigned to the new record
        long newRowId = db.insert(
                MyContract.Student.TABLE_NAME, // name of the
table
                null,    // if we use
FeedEntry.COLUMN_NAME_NULLABLE,
                // values of empty fields will be null
                // null means nothing will be in them
                values); // defined pairs field - value
        db.close();
    }

    public Student getStudent(long ID) {
        // accessing the database
        SQLiteDatabase db = getWritableDatabase();

        // list of fields to be returned
        String[] projection=
{MyContract.Student.COLUMN_NAME,MyContract.Student.COLUMN_EMAI
L, MyContract.Student.COLUMN_AGE};
        String selection = MyContract.Student.COLUMN_ID;
// field in the condition (WHERE)
        String[] selectionArgs = {""+ID};         // list of
values for the condition

        Cursor c = db.query(
                MyContract.Student.TABLE_NAME,   // table for
the query
                projection,      // list of fields to be
returned
```

```
                selection,       // WHERE will be here
                selectionArgs,   // values for the WHERE
condition
                null,            // fields for GROUP BY
                null,            // HAVING
                null  );         // sort order

        Student s = new Student(ID,

c.getString(c.getColumnIndex(MyContract.Student.COLUMN_NAME)),
                c.getString(2),

c.getInt(c.getColumnIndex(MyContract.Student.COLUMN_AGE)));
        c.close();  // closing the cursor
        db.close(); // closing the database
        return s;   // returned the ready instance of the
Student
    }

    public void deleteStudent(int ID) {
        // accessing the database
        SQLiteDatabase db = getWritableDatabase();

        // delete with parameters (more secure)
        db.delete(
                MyContract.Student.TABLE_NAME,       // table
name
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with the parameter
                new String[] { ""+ID }); // value of the
parameters
        db.close(); // closing the database
    }

    public void updateStudent(Student s) {
        // creating a map of values - column name + value
        ContentValues values = new ContentValues();
        values.put(MyContract.Student.COLUMN_NAME,
s.getMeno());
        values.put(MyContract.Student.COLUMN_EMAIL,
s.getEmail());
        values.put(MyContract.Student.COLUMN_AGE, s.getVek());

        // accessing the database
```

```
        SQLiteDatabase db = getWritableDatabase();
        db.update(
                MyContract.Student.TABLE_NAME,        // table
name
                values,  // defined pairs field - value
                MyContract.Student.COLUMN_ID + "= ?",  //
condition with parameter
                new String[] { ""+s.getID() }); // value of
parameters
        db.close(); // closing the database
    }

    public Cursor getMyCursor() {
        // accessing the database
        SQLiteDatabase db = getWritableDatabase();
        String sstr = "select * from " +
MyContract.Student.TABLE_NAME;

        Cursor c = db.rawQuery(sstr, null);
        // to be sure, we can move to the first row
        c.moveToFirst();
        db.close();
        return c;
    }
}
```

MyContract.java

```
package com.example.a12_dbstudentcursor;

public final class MyContract {

    public static class Student {
        public static final String TABLE_NAME = "students";
        public static final String COLUMN_ID = "_id";
        public static final String COLUMN_NAME = "name";
        public static final String COLUMN_EMAIL = "email";
        public static final String COLUMN_AGE = "age";
    }
}
```

Student.java

```
package com.example.a12_dbstudentcursor;

public class Student {
    private long ID;
```

```
    private String meno;
    private String email;
    private int vek;

    public Student(long ID, String meno, String email, int
vek) {
        this.ID = ID;
        this.meno = meno;
        this.email = email;
        this.vek = vek;
    }

    public long getID() {
        return ID;
    }

    public void setID(long ID) {
        this.ID = ID;
    }

    public String getMeno() {
        return meno;
    }

    public void setMeno(String meno) {
        this.meno = meno;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getVek() {
        return vek;
    }

    public void setVek(int vek) {
        this.vek = vek;
    }
```

```
    @Override
    public String toString() {
        return "Student{" +
                "ID=" + ID +
                ", meno='" + meno + '\'' +
                ", email='" + email + '\'' +
                ", vek=" + vek +
                '}';
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <|ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listView"/>

<|/LinearLayout>
```

list_layout.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|TextView android:id="@+id/id1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:visibility="gone"
        android:width="100dip" />

    <|TextView android:id="@+id/name1"
```

```
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />

    <|TextView android:id="@+id/email1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />

    <|TextView android:id="@+id/age1"
        android:layout_height="fill_parent"
        android:layout_width="wrap_content"
        android:width="100dip" />
<|/LinearLayout>
```

Broadcast Receiver

Chapter 8

8.1 Broadcast receiver

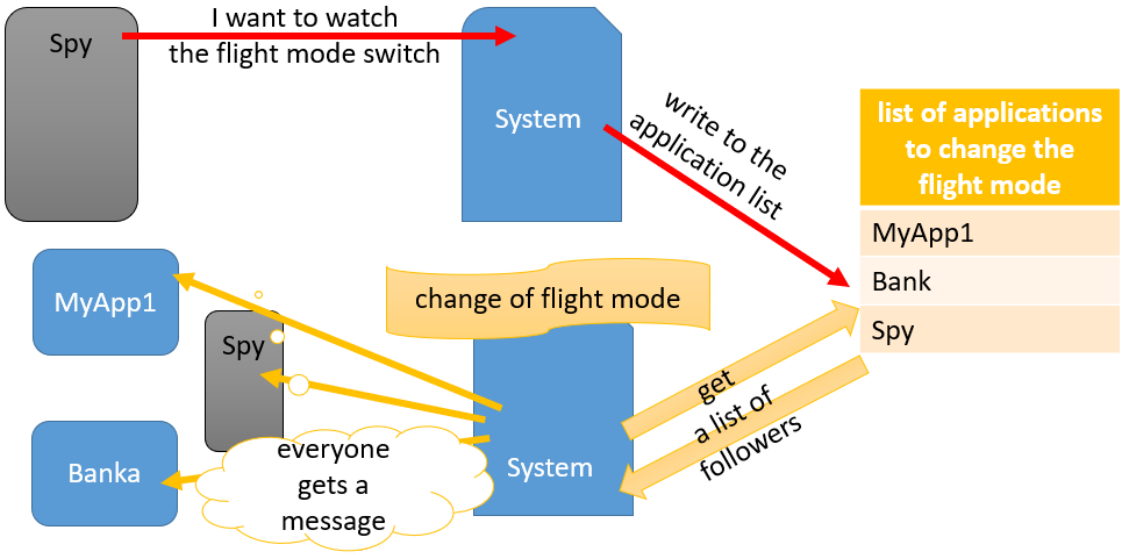
🕮 8.1.1

Broadcast receiver

The broadcast receiver is one of the basic parts of the Android system. It is basically a "sleeping" element that is activated by a message (intent) sent within the system.

In order for the application to be able to respond to an action / message in the system, it must be registered to process a specific type of message. If an event / action then occurs, the system sends a message to all registered applications.

The principle of operation is illustrated by the following figure:



An application usually registers to process one or a few types of messages

Registration options:

1. statically (via manifest) - via the <received> tag - outdated principle and not all system activities currently support it
2. dynamically - via the **registerReceiver()** method in the right place of the application, while **unregisterReceiver()** is also expected

After registration, the application or the respective broadcast receiver is activated on the basis of the received information and its **onReceive()** method is executed.

On execution, **onRecieve()** is assigned a **short time interval**, after which its operation is terminated => for a more complex operation, it is necessary to send an asynchronous message, start a service or application.

🕮 8.1.2

Assignment:

Ensure that the user (or application) is informed when airplane mode is activated.

This transition blocks the entire communication of the device with the environment, it is often necessary for the application to deal with it.

Just display the information in a **Toast** message.

🕮 8.1.3

We have an application ready with activity. To achieve functionality, we will create our own **BroadcastReceiver** (BR) as a separate class (and a separate file).

Its job is only to display the message when it is activated.

We determine when this happens when it is registered, so the receiver code is very simple.

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "change detected in airplane
mode",
                       Toast.LENGTH_SHORT).show();
    }
 }
```

🕮 8.1.4

The registration of the **broadcast receiver** determines which message it will respond to. We place registration and deregistration in "paired" methods, we know the principle:

- if we want to respond only if the application is active, then **onResume()** and **onPause()**

- if we want the application to respond even if it is in the background (covered by another activity or another application) then we use onCreate() and unregistering in onDestroy()
- if we want to respond even if the application is not running, it is necessary to create a **service**

To illustrate, we will only use the *broadcast* **receiver** if the application is in the foreground:

```
public class MainActivity extends AppCompatActivity {

 MyReceiver myRec = new MyReceiver(); // we declare and create

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(myRec,
            new
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED));
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(myRec);
    }
}
```

The key command is

```
 registerReceiver(mojRec, new
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED));
```

It is true that activities, services and receivers are activated by intents (we know that an intent is a message with additional parameters). The moment a message is generated and sent in an application, the system processes it, scans the list of components in the system that have requested its processing, and sends it to them.

🕮 8.1.5

In order for each component not to process everything and not respond to all messages, it only has specific message types defined via the filter

For our BR, we define the ability to select (process) messages that are of type **ACTION_AIRPLANE_MODE_CHANGED**. The BR reaction is defined in its **onReceive** method.

We add a selection of the most used types of messages that are processed by applications:

- Intent.ACTION_BATTERY_LOW
- Intent.ACTION_BATTERY_OKAY
- Intent.ACTION_BOOT_COMPLETED
- Intent.ACTION_DEVICE_STORAGE_LOW
- Intent.ACTION_DEVICE_STORAGE_OK
- Intent.ACTION_HEADSET_PLUG
- Intent.ACTION_LOCALE_CHANGED
- Intent.ACTION_MY_PACKAGE_REPLACED
- Intent.ACTION_PACKAGE_ADDED
- Intent.ACTION_POWER_CONNECTED
- Intent.ACTION_POWER_DISCONNECTED
- KeyChain.ACTION_STORAGE_CHANGED
- BluetoothDevice.ACTION_ACL_CONNECTED
- AudioManager.ACTION_AUDIO_BECOMING_NOISY

⌨ 8.1.6 The code of the application

MainActivity.java

```
package com.example.a13_airplanereceiver;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    MyReceiver myRec = new MyReceiver(); // declare and create

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

```
    @Override
    protected void onResume() {
        super.onResume();
        registerReceiver(myRec,
                new
IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED));
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(myRec);
    }
}
```

MyReceiver.java

```
package com.example.a13_airplanereceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "change in airplane mode
detected",
                Toast.LENGTH_SHORT).show();
    }
}
```

main_activity.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <|TextView
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<|/androidx.constraintlayout.widget.ConstraintLayout>
```

Bluetooth

Chapter 9

9.1 Bluetooth adapter

🕮 9.1.1

Bluetooth is a communication protocol designed for short-range, bandwidth peer-to-peer communication. It was created in 1994 by Ericson as a wireless replacement for a serial wired interface.

It is defined as a layer made up of several basic protocols

- LMP, L2CAP, SDP, HCI, RFCOMM
- radio frequency communication (RFCOMM) is a replacement for the cable protocol, which is used to create a virtual serial data stream

Many applications use Bluetooth RFCOMM for its extensive support and publicly available APIs on most operating systems. As of Android 2.1, only encrypted communication is supported, which means we can only create connections between paired devices.

🕮 9.1.2

The basic functionalities of the device consist at the bluetooth level of:

- turning bluetooth on/off
- visibility of the search device
- search for nearby devices
- list of paired devices
- pairing
- connection establishment and communication

Android provides us with the following classes to provide these features:

- **BluetoothAdapter** - represents the local Bluetooth device = Android device on which the application is running,
- **BluetoothDevice** - represents a remote (other) device with which we want to communicate,
- **BluetoothSocket** and **BluetoothServerSocket** - provide access to the communication channel to which the devices are connected to.

🕮 9.1.3

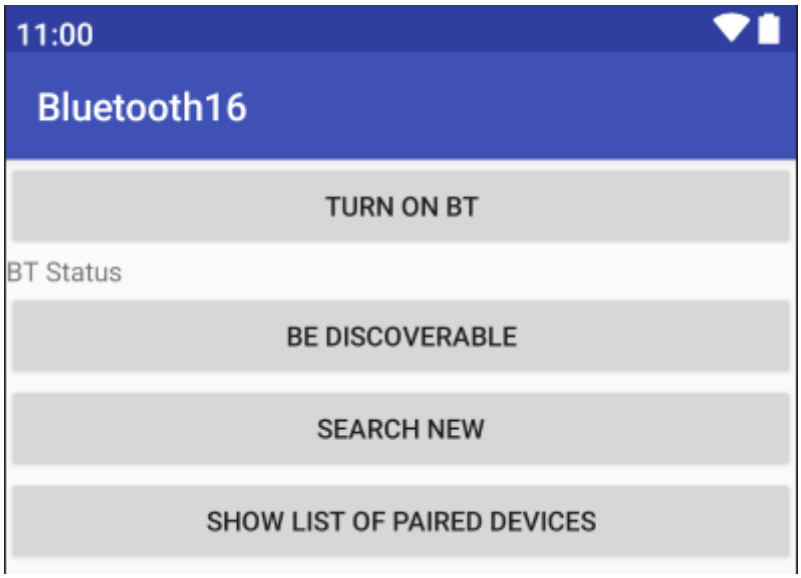
Assignment:

Create an application that can turn the Bluetooth adapter in the device on/off.

🕮 9.1.4

In order to cover all required functionalities, we need to:

- find out if bluetooth (BT) exists in the device,
- turn on BT,
- turn on the ability to be seen,
- look for visible devices in the area,
- view the list of devices that are currently paired with our device.



🕮 9.1.5

To use a BT adapter, we define an instance of the **BluetoothAdapter** class, and to verify the existence of the device, we use the loading of the base adapter via **BluetoothAdapter.getDefaultAdapter()**.

```
public class MainActivity extends AppCompatActivity {
    private BluetoothAdapter BA;

@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    zobrazBTStatus();
}

private void zobrazBTStatus() {
    BA = BluetoothAdapter.getDefaultAdapter();

    TextView tv = (TextView) findViewById(R.id.textView4);
    if (BA == null) {
        tv.setText("Adapter exists");
    } else {
        if (BA.isEnabled())
            tv.setText("turned on");
        else
            tv.setText("turned off");
    }
}
```

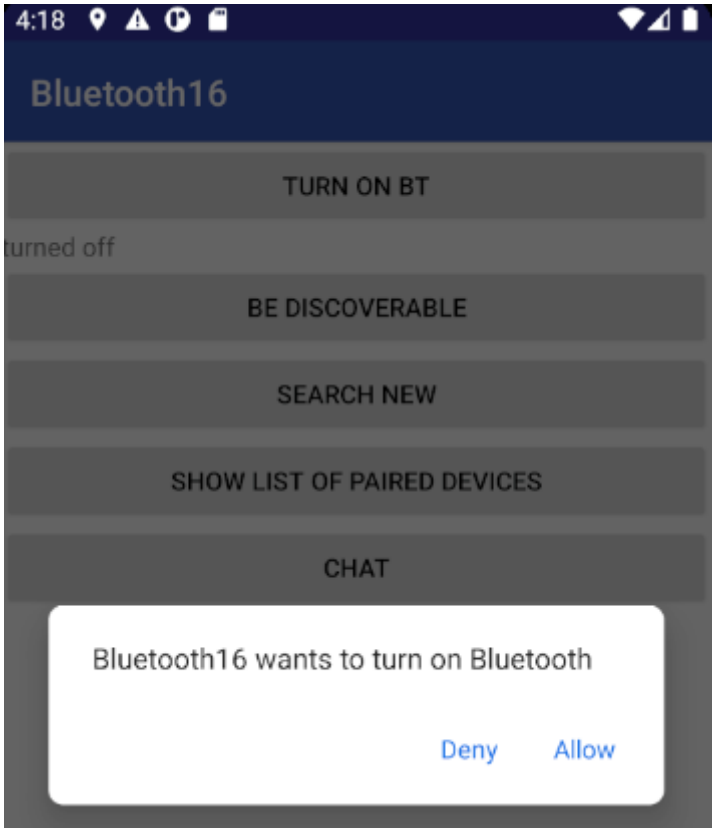
🕮 9.1.6

Turning on the adapter is a non-trivial process, first we need to call the system window for permission and when returning from it, switch the state (if turning on was enabled) or view it.

```
public void turnOnClick(View view) {
    if (!BA.isEnabled()) {
        Intent turnOn = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(turnOn, 5);
        Toast.makeText(getApplicationContext(), "Trying to
turn on",
                       Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getApplicationContext(), "Already
turned on",
                        Toast.LENGTH_LONG).show();
    }
}


@Override
```

```
 protected void onActivityResult(int requestCode, int
resultCode,
                                 Intent data) {
    switch (requestCode) {
        case 5:
            showBTStatus();
            break;
    }
 }
```

The displayed system dialog will prompt us to turn it on, and if we enable it, it will switch the status. We cannot provide this operation directly in the application.



🕮 9.1.7

In order for the application to gain the permissions to access the BT adapter and for it to be able to change its parameters, it is necessary to add permissions to the manifest.

```
<?xml version="1.0" encoding="utf-8"?>
 <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sinus.bluetooth16">
```

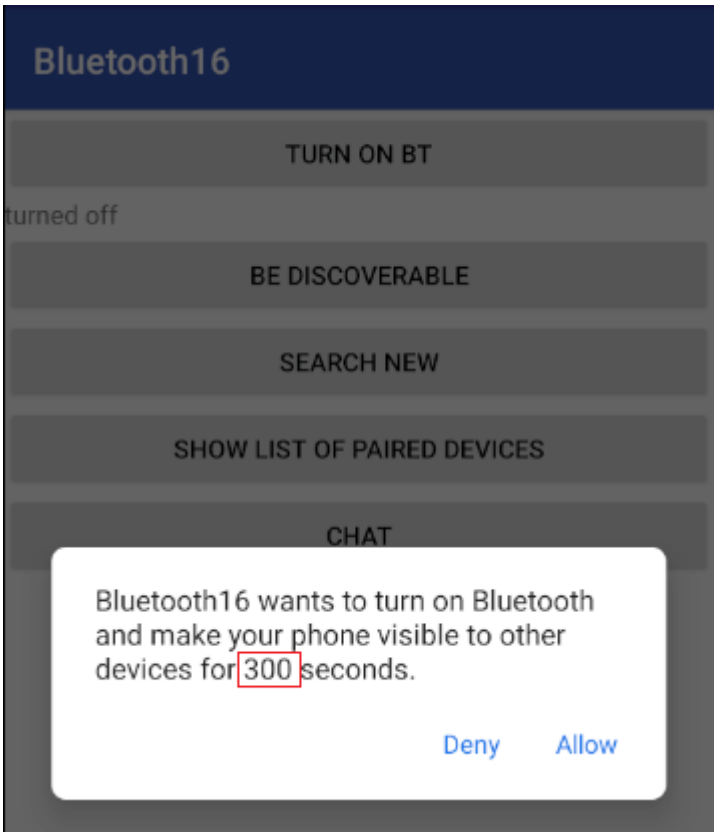
```
    <uses-permission
android:name="android.permission.BLUETOOTH" />
    <uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />
...
```

🕮 9.1.8

Device visibility

Security requires that only paired devices communicate, and at the same time that devices that do not need the same are not visible even though they have BT turned on. The device can thus be instructed to be accessible to searching devices for a certain period of time, but again it is necessary to use the system function.

We can define the time during which the device should be visible through the intent



If the device is currently looking for available devices, we will interrupt this activity - it slows down everything else and we want to be found, not search.

We will add the availability time to the intent (by default it is 120s and over 300s is not allowed).

```
public void discoverClick(View view) {
  BA.cancelDiscovery();
  Intent disIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
  disIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURAT
ION, 300);
  startActivityForResult(disIntent, 7);
 }
```

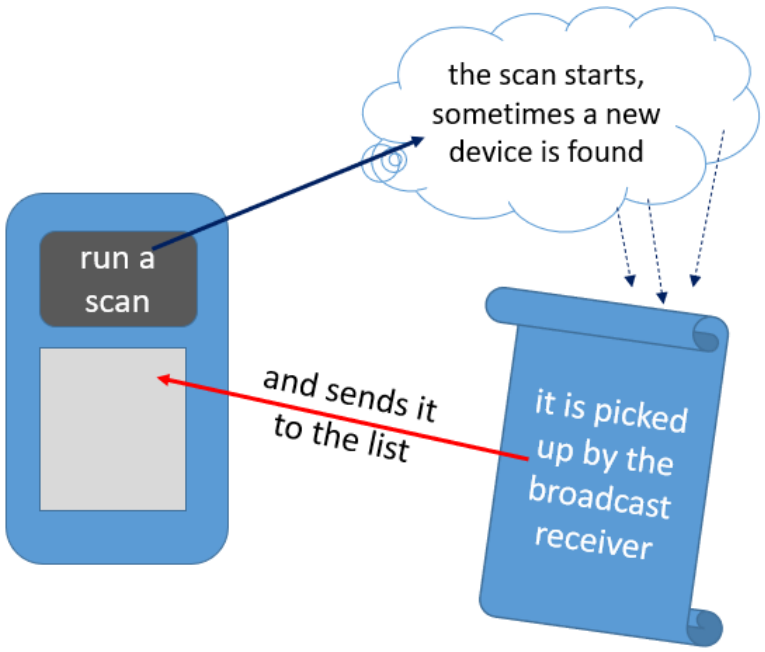
🕮 9.1.9

Searching for a device

On the other hand, there must be a device that will be able to find the object as "accessible". Due to the fact that there may be more available objects, we will need a list to display them. We can then select the device from the created list and pair with it.

Search engine:

- if a device is found, a message is sent
- captured by a broadcast receiver, set to capture found devices,
- adds the found device to the list.



🕮 9.1.10

To separate this functionality from the others, we will create a separate activity with a scan button and a list to which devices will be gradually added.

In the created activity, we define a list (**ArrayList**), in which the names of the devices will be displayed to the user. In parallel, we will use a list of devices to identify the pairing device.

We will create an **ArrayAdapter** for displaying data and connect a listener to the list items.

```
public class ScanActivity extends AppCompatActivity {
    BluetoothAdapter BA;
    ArrayList<String> myList = new ArrayList();
    ArrayList<BluetoothDevice> listDev = new ArrayList();
    ArrayAdapter adapter;

@Override
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_scan);

    connectAdapter();
    connectOnItemClickListener();
 }

private void pripojAdapter() {
    adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1, myList)
;
    ListView lv = (ListView) findViewById(R.id.listView3);
    lv.setAdapter(adapter);
 }

 private void pripojOnItemClickListener() {
    AdapterView.OnItemClickListener mMessageClickedHandler =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView parent,
                                        View v,
                                        int position,
                                        long id) {
                    BA.cancelDiscovery();
                    pairDevice(listDev.get(position));
                }
```

```
            };
    ListView lv = (ListView) findViewById(R.id.listView3);
    lv.setOnItemClickListener(mMessageClickedHandler);
 }
```

📖 9.1.11

In the list of devices, we will provide the processing of clicks and pairing with the selected device (will be described below):

We still lack the search itself, for which it is necessary to define the events that will be processed and the receiver that processes them. Let's register the receiver only when the use is invoked and unregister it when leaving the activity.

```
public void scanAroundClick(View view) {
    IntentFilter filter = new IntentFilter();

    filter.addAction(BluetoothDevice.ACTION_FOUND);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED
);
    filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHE
D);

    registerReceiver(mReceiver, filter);
    BA = BluetoothAdapter.getDefaultAdapter();
    BA.startDiscovery();
 }

 public void onDestroy() {
    unregisterReceiver(mReceiver);
    super.onDestroy();
 }
```

📖 9.1.12

To simplify the connection of the receiver with activity views, we define it as part of the activity:

```
private final BroadcastReceiver mReceiver = new
BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
       String action = intent.getAction();
```

```
        if
(BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
            Toast.makeText(context, "Starting search ...
", Toast.LENGTH_SHORT).show();
        } else
         if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
             Toast.makeText(context, "... End of search",
Toast.LENGTH_SHORT).show();
         } else if
(BluetoothDevice.ACTION_FOUND.equals(action)) {
             //bluetooth device found
             BluetoothDevice device = (BluetoothDevice)
             intent.getParcelableExtra(BluetoothDevice.EXTRA_DE
VICE);

             Toast.makeText(context, "Found: " +
device.getName(), Toast.LENGTH_SHORT).show();
             myList.add(device.getName());
             devList.add(device);
             adapter.notifyDataSetChanged();
         }
     }
 };
```

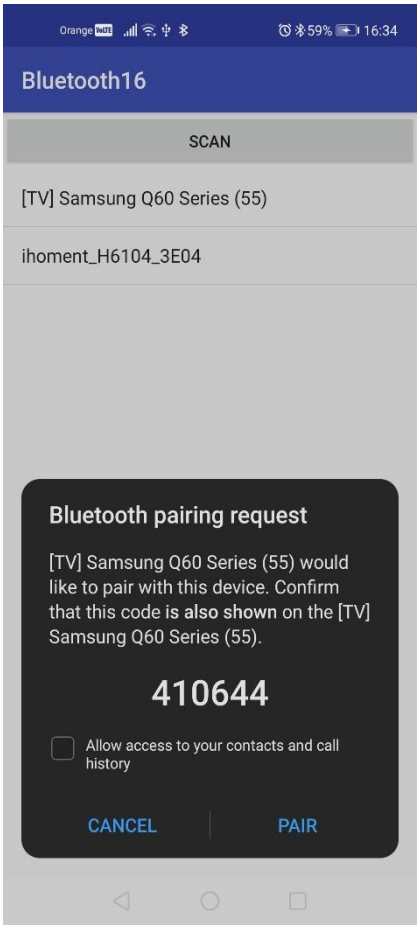
The receiver responds to the start and end of the search, with the most important activity being finding the device. In this case, it adds the device name from the **myList** variable (displayed in the list), adds the device to the **devList** as an object with all the data (it performs a pairing based on it), and finally sends information to the adapter that the content needs to be updated.

🕮 9.1.13

Device pairing is an important and serious operation because we need to confirm that we are pairing with the device we want and not with some random device that has managed to intercept our communication. Pairing takes place by known hand shaking (as in the case of ssl communication)

```
private void pairDevice(BluetoothDevice device) {
  try {
     Method method = device.getClass().getMethod("createBond",
(Class[]) null);
     method.invoke(device, (Object[]) null);
  } catch (Exception e) {
```

```
        Toast.makeText(getApplicationContext(), "Error!" ,
Toast.LENGTH_SHORT).show();
    }
 }
```



🕮 9.1.14

The list of paired devices represents devices with which we can establish communication without confirmation and are stored in the database of the device (smartphone).

To display them, we will create a separate activity with the activity returning the selected device, e.g. to establish communication. The list contains devices with which we can establish communication without confirmation. Devices are stored in the device database as a set and are obtained via **getBondedDevices()**.

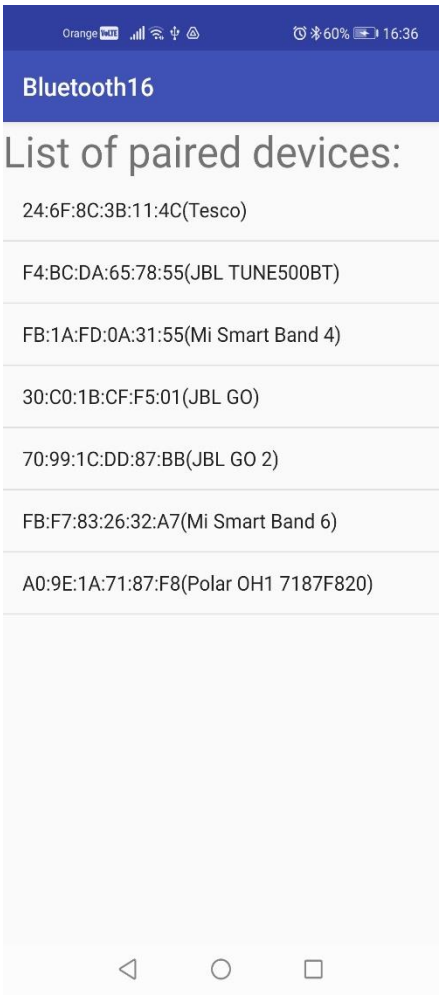
To display these devices, we will create a separate activity with the activity returning the selected device, e.g. to establish communication.

```
public class DeviceListActivity extends AppCompatActivity {
    ArrayAdapter adapter;
    ArrayList<String> myList;
    private Set<BluetoothDevice> pairedDevices;
    BluetoothAdapter BA;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        BA = BluetoothAdapter.getDefaultAdapter();
        pairedDevices = BA.getBondedDevices();
        showList();             // bonded list
        addListener();         // add listener
    }
```

📖 9.1.15

We create the list by iterating through all the elements of the set, which are of the BluetoothDevice type. We can obtain the MAC address and device name from the stored data. We will therefore save them in a list that will be displayed in the **ListView** container.

```
private void UkazZoznam() {
    myList = new ArrayList();
    for(BluetoothDevice bt : pairedDevices) {
        myList.add(bt.getAddress() + "(" + bt.getName() +
")");
    }
    adapter =
new ArrayAdapter(this,android.R.layout.simple_list_item_1,
myList);
    ListView lv = (ListView) findViewById(R.id.listView2);
    lv.setAdapter(adapter);
 }
```



📖 9.1.16

After selecting a specific device, we return the MAC address, which is unique - we get it as the first 17 characters from the text in the row during the processing of the click event in the list.

```
private void addListener() {
    AdapterView.OnItemClickListener mMessageClickedHandler =
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView parent,
                                        View v,
                                        int position,
                                        long id) {

                    Intent intentBack = new Intent();
                    intentBack.putExtra("device",
myList.get(position).substring(0, 17));
                    setResult(RESULT_OK, intentBack);
```

```
                    finish();
                }
            };
    ListView lst = (ListView) findViewById(R.id.listView2);
    lst.setOnItemClickListener(mMessageClickedHandler);
 }
```

After this step, we have the device we want to communicate with.

9.1.17 The code of the application

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sinus.bluetooth16">

    <|uses-permission
android:name="android.permission.BLUETOOTH" />
    <|uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <|activity android:name=".MainActivity">
          <|intent-filter>
            <|action android:name="android.intent.action.MAIN"
/>

            <|category
android:name="android.intent.category.LAUNCHER" />
          <|/intent-filter>
        <|/activity>
        <|activity android:name=".DeviceListActivity" />
        <|activity android:name=".ScanActivity" />
        <|activity
android:name=".CommunicationActivity"><|/activity>
```

```
    <|/application>

<|/manifest>
```

MainActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {
    private BluetoothAdapter BA;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        showBTStatus();
    }

    private void showBTStatus() {
        BA = BluetoothAdapter.getDefaultAdapter();

        TextView tv = (TextView) findViewById(R.id.textView4);
        if (BA == null) {
            tv.setText("the device doesn't have Bluetooth");
        } else {
            if (BA.isEnabled())
                tv.setText("turned on");
            else
                tv.setText("turned off");
        }
    }

    public void turnOnClick(View view) {
        if (!BA.isEnabled()) {
```

```
            Intent turnOn = new

Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(turnOn, 5);
            Toast.makeText(getApplicationContext(), "Trying to
turn on",
                           Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(getApplicationContext(), "Already
turned on",
                           Toast.LENGTH_LONG).show();
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int
resultCode,
                                    Intent data) {
        switch (requestCode) {
            case 5:
                showBTStatus();
                break;
        }
    }

    public void listClick(View view) {
       Intent z = new
Intent(MainActivity.this,DeviceListActivity.class);
       startActivityForResult(z, 6);
    }

    public void makeVisibleClick(View view) {
        BA.cancelDiscovery();
        Intent disIntent = new

Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

disIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATIO
N,
                           300);
        startActivityForResult(disIntent, 7);
    }

    public void scanClick(View view) {
```

```
        Intent z = new Intent(MainActivity.this,
ScanActivity.class);
        startActivityForResult(z, 8);
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|Button
        android:id="@+id/button4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="turnOnClick"
        android:text="Turn on BT" />

    <|TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BT Status" />


    <|Button
        android:id="@+id/button6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="makeVisibleClick"
        android:text="Turn discoverability on" />

    <|Button
        android:id="@+id/button7"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="scanClick"
```

```
        android:text="Search for new devices" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="listClick"
        android:text="Show paired devices" />

<|/LinearLayout>
```

ScanActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.lang.reflect.Method;
import java.util.ArrayList;

public class ScanActivity extends AppCompatActivity {
    BluetoothAdapter BA;
    ArrayList<|String> myList = new ArrayList();
    ArrayList<|BluetoothDevice> devList = new ArrayList();
    ArrayAdapter adapter;

    private final BroadcastReceiver mReceiver = new
BroadcastReceiver() {
      public void onReceive(Context context, Intent intent) {
          String action = intent.getAction();

          if
(BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
```

```
                Toast.makeText(context, "Starting search ...
",
                                Toast.LENGTH_SHORT).show();
          } else if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.
                    equals(action)) {
                Toast.makeText(context, "... End of search",
                                Toast.LENGTH_SHORT).show();
          } else if
(BluetoothDevice.ACTION_FOUND.equals(action)) {
              //bluetooth device found
              BluetoothDevice device = (BluetoothDevice)

intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                Toast.makeText(context, "Found " +
device.getName(),
                                Toast.LENGTH_SHORT).show();
                myList.add(device.getName());
                devList.add(device);
                adapter.notifyDataSetChanged();
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scan);

        connectAdapter();
        connectOnItemClickListener();
    }

    private void connectOnItemClickListener() {
        AdapterView.OnItemClickListener mMessageClickedHandler
=
                new AdapterView.OnItemClickListener() {
                    public void onItemClick(AdapterView
parent,
                                            View v,
                                            int position,
                                            long id) {
                        BA.cancelDiscovery();
```

```
                        pairDevice(zoznamDev.get(position));
                    }
                };
        ListView lv = (ListView) findViewById(R.id.listView3);
        lv.setOnItemClickListener(mMessageClickedHandler);
    }

    private void pairDevice(BluetoothDevice device) {
        try {
            Method method =
device.getClass().getMethod("createBond",
                            (Class[]) null);
            method.invoke(device, (Object[]) null);
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), "Error!" ,
                           Toast.LENGTH_SHORT).show();
        }
    }

    private void connectAdapter() {
        adapter = new ArrayAdapter(this,
                       android.R.layout.simple_list_item_1,
zoznam);
        ListView lv = (ListView) findViewById(R.id.listView3);
        lv.setAdapter(adapter);
    }

    public void scanAroundClick(View view) {
        IntentFilter filter = new IntentFilter();

        filter.addAction(BluetoothDevice.ACTION_FOUND);

filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);

filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);

        registerReceiver(mReceiver, filter);
        BA = BluetoothAdapter.getDefaultAdapter();
        BA.startDiscovery();
    }

    public void onDestroy() {
        unregisterReceiver(mReceiver);
        super.onDestroy();
```

```
    }
}
```

activity_scan.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ScanActivity">

    <|Button
        android:id="@+id/button5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="scanAroundClick"
        android:text="Scan" />

    <|ListView
        android:id="@+id/listView3"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
<|/LinearLayout>
```

DeviceListActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;

public class DeviceListActivity extends AppCompatActivity {
```

```
    ArrayAdapter adapter;
    ArrayList<|String> myList;
    private Set<|BluetoothDevice> pairedDevices;
    BluetoothAdapter BA;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        BA = BluetoothAdapter.getDefaultAdapter();
        pairedDevices = BA.getBondedDevices();
        showList();              // bonded list
        addListener();          // add listener
    }

    private void showList() {
        myList = new ArrayList();
        for(BluetoothDevice bt : pairedDevices) {
            myList.add(bt.getAddress() + "(" + bt.getName() +
")");
        }
        adapter = new ArrayAdapter(this,
                        android.R.layout.simple_list_item_1,
myList);
        ListView lv = (ListView) findViewById(R.id.listView2);
        lv.setAdapter(adapter);
    }

    private void addListener() {
        AdapterView.OnItemClickListener mMessageClickedHandler
=
                new AdapterView.OnItemClickListener() {
                    public void onItemClick(AdapterView
parent,
                                            View v,
                                            int position,
                                            long id) {

                        Intent intentReturn = new Intent();
                        intentReturn.putExtra("device",

myList.get(position).substring(0, 17));
                        setResult(RESULT_OK, intentReturn);
```

```
                        finish();
                    }
                };
        ListView ml = (ListView) findViewById(R.id.listView2);
        ml.setOnItemClickListener(mMessageClickedHandler);
    }
}
```

activity_device_list.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".DeviceListActivity">


    <|TextView
        android:id="@+id/textView5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="List of paired:"

android:textAppearance="@style/TextAppearance.AppCompat.Displa
y1" />

    <|ListView
        android:id="@+id/listView2"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

<|/LinearLayout>
```

9.2 Chat

🕮 9.2.1

Assignment:

Create a chat application which uses the Bluetooth communication channel. Send messages with a button and view them in a list.

🕮 9.2.2

Pairing devices is only the first step in communication. In the next step, we need to create a connection that is based on the client-server model:

- server - listens for incoming connections,
- client - trying to connect to server (should know which device is listening)

The connection will create a channel that is accessible to both sides and through which they will exchange messages. The client and server part should be implemented in one application, while in a specific situation, live users will determine who will be the client and who the server - this fact is important only for establishing a connection, later the role of client and server is lost.

🕮 9.2.3

Server

The device that will represent the server must open a **BluetoothServerSocket** to listen for incoming requests.

The result of a successful connection will be a **BluetoothSocket**, through which data will then be exchanged.

In order not to connect two different applications, it is necessary to specify the application Universally Unique Identifier (UUID), which is basically a port: it is a 128-bit format in the form of a string and we can convert it from a **String**. We will add it to the main activity of the application.

```
public class MainActivity extends AppCompatActivity {
    private BluetoothAdapter BA;
    public static final UUID MY_UUID =
                UUID.fromString("00112233-afac-1234-abcd-
abcdef012345");
```

🕮 9.2.4

The server will listen thanks to a server-socket created by a bluetooth adapter (BA) and a listening method in which the first parameter is a text identifier, e.g. the second application name is uuid.

```
BluetoothServerSocket bluetoothServerSocket =
BA.listenUsingRfcommWithServiceRecord("MyChatApp", uuid);
```

By calling the **accept()** method, we gain access to the communication socket (and we can close the serversocket).

```
BluetoothSocket bluetoothSocket =
bluetoothServerSocket.accept();
bluetoothServerSocket.close();
```

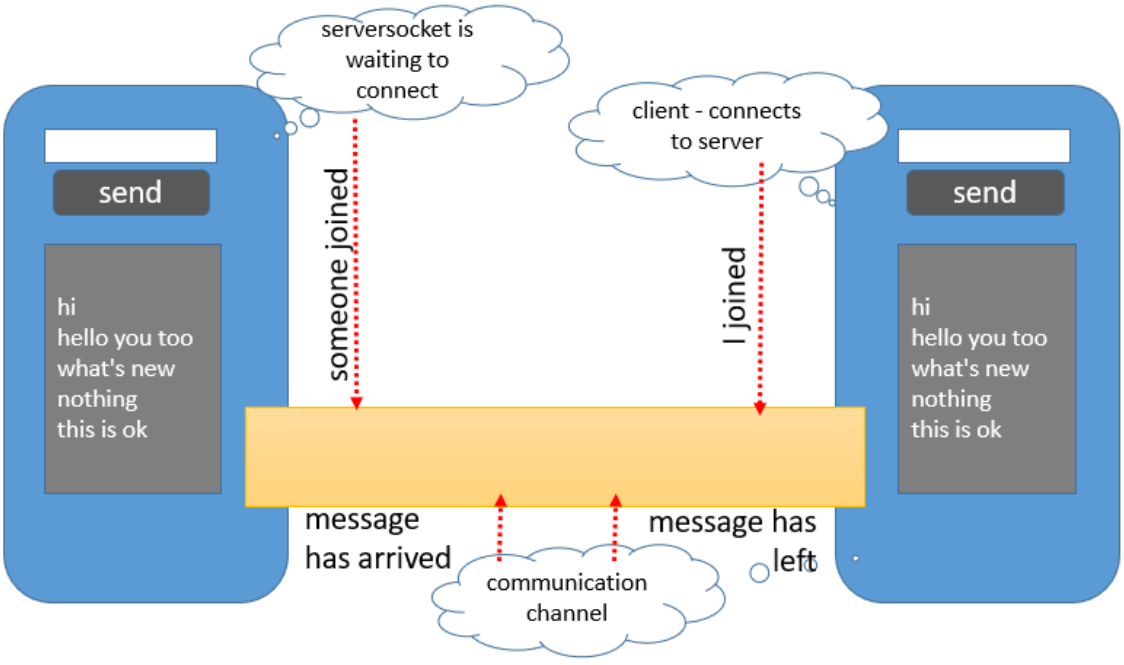
🕮 9.2.5

```
BluetoothSocket bluetoothSocket =
bluetoothServerSocket.accept();
bluetoothServerSocket.close();
```

However, the problem with this connection is that **accept()** blocks any communication unless the connection is established, which is likely to cause problems when using the application. The solution is to send a wait message to accept the connection to a separate thread and ensure that the changes from the thread are reflected in the activity in use.

This approach is not foreign to us, the rendering of the animation works similarly, so we will create a handler in the activity, which will process the information from the threads.

The structure of bluetooth client-server communication is shown in the picture:



🕮 9.2.6

We will also use a thread for client work and another thread that will take over the communication after the connection is established.

So together we will need the following threads:

- server waiting for **ListeningThread** connection
- when the device is connected, it exits and passes control to **ConnectedThread**
- a client trying to connect to the **ConnectingThread** server
- when the device is connected, it exits and passes control to **ConnectedThread**
- communicating clients guarding and controlling I/O operations within the **ConnectedThread** thread

All information from the threads will be displayed in the activity thanks to the **Handler** receiving their messages.

🕮 9.2.7

The activity can take the following form and have implemented functions:

- the ability to turn on server behavior
- the ability to turn on client behavior by selecting the server to which it wants to connect to
- space for writing text
- space for displaying communication

🕮 9.2.8

It will include a handler that will capture messages from threads, while all information will be displayed in the message window (but this may not be the case, it is possible to use e.g. **TextView** to display the status).

```
private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        TextView tv = (TextView) findViewById(R.id.textView);
        switch (msg.what) {
            case CONNECTED:
                adapter.add("Connected:  ");
                break;
            case ACCEPTED:
                adapter.add("Accepted:  ");
                break;
            case MESSAGE_WRITE:
                byte[] writeBuf = (byte[]) msg.obj;
                // construct a string from the buffer
                String writeMessage = new String(writeBuf);
                adapter.add("ME:  " + writeMessage);
                break;
            case MESSAGE_READ:
                byte[] readBuf = (byte[]) msg.obj;
                // construct a string from the valid bytes in
the buffer
                String readMessage = new String(readBuf, 0,
msg.arg1);
                adapter.add(readMessage);
                break;
        }
    }
};
```

In the code, the buffer is an array of characters that we convert to **String**, and **msg** is the message we sent from the thread.

🕮 9.2.9

In the communication activity, we define constants for messages, threads with which we will communicate and create a main thread (**ConnectedThread**), to which we pass control after the connection is established.

```
public class CommunicationActivity extends AppCompatActivity {
    public static final int CONNECTED = 0;
    public static final int ACCEPTED = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;

    BluetoothDevice device2 = null;
    BluetoothAdapter BA;
    ListeningThread lt = null;
    ConnectingThread ct = null;
    ConnectedThread conThread = null;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_communication);
        BA = BluetoothAdapter.getDefaultAdapter();

        adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1);
        ListView lv = (ListView) findViewById(R.id.listView5);
        lv.setAdapter(adapter);
        if (conThread != null) conThread.cancel();
        conThread = new ConnectedThread(mHandler); // control
thread
    }
```

🕮 9.2.10

We start the server with a button, and if the server thread (**ListeningThread**) has already started, it must be stopped. **ListeningThread** is the listening thread to which we send

- our BT adapter
- UUID
- a thread that takes control after a connection has been estabilsihed
- the handler to which the messages will go

and then run the newly created thread.

```
public void serverClick(View view) {
    if (lt != null) lt.cancel();
    lt = new
ListeningThread(BluetoothAdapter.getDefaultAdapter(),
                              MainActivity.MY_UUID, conThread,
mHandler);
    lt.start();
}
```

📖 9.2.11

Server thread

ListeningThread provides listening to connecting clients. In the constructor, we set the parameters (handler and thread that takes control) and get the serversocket.

```
public class ListeningThread extends Thread {
    private final BluetoothServerSocket bluetoothServerSocket;
    ConnectedThread comThread;
    Handler mHandler;

    public ListeningThread(BluetoothAdapter BA, UUID uuid,
ConnectedThread ci, Handler ih) {
       BluetoothServerSocket temp = null;
       comThread = ci;
       mHandler = ih;
       try {
         temp =
BA.listenUsingRfcommWithServiceRecord("MyChatApp", uuid);
       } catch (IOException e) {e.printStackTrace();}
       bluetoothServerSocket = temp;
    }
```

📖 9.2.12

Subsequently, the thread runs and waits until a connection is established. If the connection is established, we set the obtained socket into the communication thread, send a message to the handler that someone has connected to and start the communication thread. Since we no longer need the server socket, we close it when canceling a thread in the **cancel()** method.

```
...
```

```
public void run() {
    BluetoothSocket bluetoothSocket;
    while (true) { // its running or waiting for a connected
client
        try {
            bluetoothSocket = bluetoothServerSocket.accept();
        } catch (IOException e) { break; }
          // if connection is successfull
        if (bluetoothSocket != null) {
              // we add the socked to the communicating thread
            comThread.setSocket(bluetoothSocket);
              // we send a message to the Handler - someone
has connected
            mHandler.obtainMessage(CommunicationActivity.CONNE
CTED, -1, -1, null).sendToTarget();
            comThread.start(); // we start the communication
thread
            try {
                  // serversocket is no longer needed
                bluetoothServerSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            break; // end of the never ending loop
        }
    }
}

// cancels the socket and stops the listening thread
public void cancel() {
    try {
        bluetoothServerSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

📖 9.2.13

Client

The client role starts an activity with a list of paired devices, in which the user selects the server to which he wants to connect.

```
public class CommunicationActivity extends AppCompatActivity {
    public static final int CONNECTED = 0;
    public static final int ACCEPTED = 1;
...
public void connectMeClick(View view) {
    Intent z = new Intent(this, DeviceListActivity.class);
    startActivityForResult(z, 9);
}

@Override
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
    switch (requestCode) {
        case 9:
            Bundle returnedData = data.getExtras();
            String MAC = returnedData.getString("device");
            device2 = BA.getRemoteDevice(MAC);
            // Initiate a connection request in a separate
thread
            if (ct != null) ct.cancel();
            ct = new
ConnectingThread(BluetoothAdapter.getDefaultAdapter(),
                        device2, MainActivity.MY_UUID,
conThread, mHandler);
            ct.start();
            break;
    }
}
```

From the induced activity, we obtain the MAC address of the device and then the identification of the entire device, which we then send to the connection thread. We create a new connecting thread (**ConnectingThread**) with parameters: our BT adapter, the device to which we connect, UUID, the thread that will take control after the connection, the handler to which messages will go.

🕮 9.2.14

Client thread

The client thread connects to the server's listening thread and then leaves control to the communication thread. The constructor sets the parameters and creates a record for the connection.

```
class ConnectingThread extends Thread {
```

```
    private final BluetoothSocket bluetoothSocket;
    private final BluetoothDevice device2;
    BluetoothAdapter BA;
    ConnectedThread comThread;
    Handler mHandler;

    public ConnectingThread(BluetoothAdapter iBA,
BluetoothDevice dev2, UUID uuid, ConnectedThread ci, Handler
ih) {
        BluetoothSocket temp = null;
        device2 = dev2;
        BA = iBA;
        comThread = ci;
        mHandler = ih; // setting the parameters
        // socket creation for the device to which we want to
connect to
        try {
            temp =
dev2.createRfcommSocketToServiceRecord(uuid);
        } catch (IOException e) {
            e.printStackTrace();
        }
        bluetoothSocket = temp; // if we succeeded, remember
it
    }
```

📖 9.2.15

In the body of the thread:

- we will cancel the search, if any is in progress,
- we will try to connect, which blocks further thread activity,
- if everything went well, we have a communication socket and send it to the control thread,
- we will send the info to the handler,
- we start the communication thread,
- we will cancel the unnecessary connecting thread.

```
public void run() {
    // we cancel the search
    BA.cancelDiscovery();
    try {
```

```
          // we try to connect, which blocks the other
activites of the thread
        bluetoothSocket.connect();
    } catch (IOException connectException) {
        connectException.printStackTrace();
        try {
            bluetoothSocket.close();
        } catch (IOException closeException) {
            closeException.printStackTrace();
        }
    }

      // if everything went well, we have a communication
socket
      // we send it to the control thread
    comThread.setSocket(bluetoothSocket);
    try {
          // we send the information to the handler
        mHandler.obtainMessage(CommunicationActivity.ACCEPTED,
-1,-1,null)
                .sendToTarget();
    } catch (Exception e) {
        Log.e("", "error", e);
    }
    comThread.start(); // start the communication thread
}

// we cancel the unnecessary connecting thread
public void cancel() {
    try {
        bluetoothSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

🕮 9.2.16

Communication thread

A separate thread for subsequent communication is in charge of communication after the connection is established. Although it is created as the first thread, it will not start running until the other threads have finished (when their **run()** method runs out).

It provides a separate method for writing and guards the input buffer in an endless loop, and if something is found in it, it reads it and sends it to the activity via a handler.

In its constructor, we set only the handler and after setting the socket from the connecting threads, we gain access to the input/output channels.

```
public class ConnectedThread extends Thread {
    private BluetoothSocket socket;
    private InputStream inStream;
    private OutputStream outStream;
    private final Handler mHandler;

    public ConnectedThread(Handler ih) {
        mHandler = ih;
    }

    public void setSocket(BluetoothSocket s) {
        socket = s;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
          // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e("", "temp sockets not created", e);
        }
        inStream = tmpIn;
        outStream = tmpOut;
    }
```

📖 9.2.17

The communication itself is relatively simple - the thread listens for input in an endless loop and if a character arrives, it sends it to the handler for display.

To write, the **write()** method is called, which sends the data to the output channel and to the handler for display.

```
public void run() {
    byte[] buffer = new byte[1024]; // I need a buffer for the
text
```

```
    int bytes; // only one character
      // always listen for input
    while (true) {
        try {
              // reading all the recieved characters
            bytes = inStream.read(buffer);
              // and send it to the Handler to display it in
the activity
            mHandler.obtainMessage(CommunicationActivity.MESSA
GE_READ, bytes, -1, buffer)
                    .sendToTarget();
        } catch (IOException e) {
            Log.e("", "disconnected", e);
            break; // if the connection is cancelled, we
cancel the process
        }
    }
}

  // write the recieved text to the buffer and send it to the
Handler
public void write(byte[] buffer) {
    try {
        outStream.write(buffer);
        mHandler.obtainMessage(CommunicationActivity.MESSAGE_W
RITE, -1, -1, buffer).sendToTarget();
    } catch (IOException e) {
        Log.e("", "Exception during write", e);
    }
}

public void cancel() {
    try {
        socket.close(); // close the socket and end the
communication
    } catch (IOException e) {
        Log.e("", "close() of connect socket failed", e);
    }
}
```

⌨ 9.2.18 The code of the application

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sinus.bluetooth16">

    <|uses-permission
android:name="android.permission.BLUETOOTH" />
    <|uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
        <|activity android:name=".DeviceListActivity" />
        <|activity android:name=".ScanActivity" />
        <|activity
android:name=".CommunicationActivity"><|/activity>
    <|/application>

<|/manifest>
```

MainActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
```

```
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {
    private BluetoothAdapter BA;
    public static final UUID MY_UUID =
UUID.fromString("00112233-afac-1234-abcd-abcdef012345");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        showBTStatus();
    }


    private void showBTStatus() {
        BA = BluetoothAdapter.getDefaultAdapter();

        TextView tv = (TextView) findViewById(R.id.textView4);
        if (BA == null) {
            tv.setText("the device doesn't have BlueTooth");
        } else {
            if (BA.isEnabled())
                tv.setText("turned on");
            else
                tv.setText("turned off");
        }
    }

    public void turnOnClick(View view) {
        if (!BA.isEnabled()) {
            Intent turnOn = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(turnOn, 5);
            Toast.makeText(getApplicationContext(), "Trying to
turn on", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(getApplicationContext(), "Already
turned on", Toast.LENGTH_LONG).show();
        }
    }
```

```
    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        switch (requestCode) {
            case 5:
                showBTStatus();
                break;
            case 6:

                break;
        }
    }

    public void myListClick(View view) {
        Intent z = new Intent(MainActivity.this,
DeviceListActivity.class);
        startActivityForResult(z, 6);
    }

    public void showClick(View view) {
        BA.cancelDiscovery();
        Intent disIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

disIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATIO
N, 300);
        startActivityForResult(disIntent, 7);
    }

    public void scanClick(View view) {
        Intent z = new Intent(MainActivity.this,
ScanActivity.class);
        startActivityForResult(z, 8);
    }

    public void chatClick(View view) {
        Intent in = new Intent(MainActivity.this,
CommunicationActivity.class);
        startActivity(in);
    }
}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|Button
        android:id="@+id/button4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="turnOnClick"
        android:text="Turn on BT" />

    <|TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BT status" />


    <|Button
        android:id="@+id/button6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="showClick"
        android:text="Be discoverable" />

    <|Button
        android:id="@+id/button7"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="scanClick"
        android:text="Search new" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:onClick="myListClick"
        android:text="Show the list of paired devices" />

    <|Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="chatClick"
        android:text="Chat" />

<|/LinearLayout>
```

ScanActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.lang.reflect.Method;
import java.util.ArrayList;

public class ScanActivity extends AppCompatActivity {
    BluetoothAdapter BA;
    ArrayList<|String> myList = new ArrayList();
    ArrayList<|BluetoothDevice> devList = new ArrayList();
    ArrayAdapter adapter;

    private final BroadcastReceiver mReceiver = new
BroadcastReceiver() {
        public void onReceive(Context context, Intent intent)
{
            String action = intent.getAction();
```

```
            if
(BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
                Toast.makeText(context, "Starting search ...
", Toast.LENGTH_SHORT).show();
            } else if
(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                Toast.makeText(context, "... end of search",
Toast.LENGTH_SHORT).show();
            } else if
(BluetoothDevice.ACTION_FOUND.equals(action)) {
                //bluetooth device found
                BluetoothDevice device = (BluetoothDevice)
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                Toast.makeText(context, "Found " +
device.getName(), Toast.LENGTH_SHORT).show();
                myList.add(device.getName());
                devList.add(device);
                adapter.notifyDataSetChanged();
            }
        }
    };


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scan);

        connectAdapter();
        addOnItemClickListener();
    }

    private void addOnItemClickListener() {
        AdapterView.OnItemClickListener mMessageClickedHandler
=
                new AdapterView.OnItemClickListener() {
                    public void onItemClick(AdapterView
parent,
                                            View v,
                                            int position,
                                            long id) {
                        BA.cancelDiscovery();
                        pairDevice(devList.get(position));
```

```
                    }
                };
        ListView lv = (ListView) findViewById(R.id.listView3);
        lv.setOnItemClickListener(mMessageClickedHandler);
    }

    private void pairDevice(BluetoothDevice device) {
        try {
            Method method =
device.getClass().getMethod("createBond", (Class[]) null);
            method.invoke(device, (Object[]) null);
        } catch (Exception e) {
            Toast.makeText(getApplicationContext(), "Error!" ,
Toast.LENGTH_SHORT).show();
        }
    }

    private void connectAdapter() {
        adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1,
myList);
        ListView lv = (ListView) findViewById(R.id.listView3);
        lv.setAdapter(adapter);
    }

    public void scanAroundClick(View view) {
        IntentFilter filter = new IntentFilter();

        filter.addAction(BluetoothDevice.ACTION_FOUND);

filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);

filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);

        registerReceiver(mReceiver, filter);
        BA = BluetoothAdapter.getDefaultAdapter();
        BA.startDiscovery();
    }

    public void onDestroy() {
        unregisterReceiver(mReceiver);
        super.onDestroy();
    }
}
```

activity_scan.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ScanActivity">

    <|Button
        android:id="@+id/button5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="scanAroundClick"
        android:text="Scan" />

    <|ListView
        android:id="@+id/listView3"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
<|/LinearLayout>
```

DeviceListActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;

public class DeviceListActivity extends AppCompatActivity {
    ArrayAdapter adapter;
    ArrayList<|String> myList;
```

```
    private Set<|BluetoothDevice> pairedDevices;
    BluetoothAdapter BA;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        BA = BluetoothAdapter.getDefaultAdapter();
        pairedDevices = BA.getBondedDevices();
        showList();             // bonded list
        addListener();          // add listener
    }

    private void showList() {
        myList = new ArrayList();
        for(BluetoothDevice bt : pairedDevices) {
            myList.add(bt.getAddress() + "(" + bt.getName() +
")");
        }
        adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1,
myList);
        ListView lv = (ListView) findViewById(R.id.listView2);
        lv.setAdapter(adapter);
    }

    private void addListener() {
        AdapterView.OnItemClickListener mMessageClickedHandler
=
                new AdapterView.OnItemClickListener() {
                    public void onItemClick(AdapterView
parent,
                                            View v,
                                            int position,
                                            long id) {

                        Intent intentNavrat = new Intent();
                        intentNavrat.putExtra("device",
myList.get(position).substring(0, 17));
                        setResult(RESULT_OK, intentNavrat);
                        finish();
                    }
                };
```

```
        ListView myList = (ListView)
findViewById(R.id.listView2);
        myList.setOnItemClickListener(mMessageClickedHandler);
    }
}
```

activity_device_list.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".DeviceListActivity">


    <|TextView
        android:id="@+id/textView5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="List of paired:"

android:textAppearance="@style/TextAppearance.AppCompat.Displa
y1" />

    <|ListView
        android:id="@+id/listView2"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

<|/LinearLayout>
```

CommunicationActivity.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class CommunicationActivity extends AppCompatActivity {
    public static final int CONNECTED = 0;
    public static final int ACCEPTED = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;

    BluetoothDevice device2 = null;
    BluetoothAdapter BA;

    ListeningThread lt = null;
    ConnectingThread ct = null;
    ConnectedThread conThread = null;

    private ArrayAdapter<|String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_communication);
        BA = BluetoothAdapter.getDefaultAdapter();

        adapter = new ArrayAdapter<|String>(this,
android.R.layout.simple_list_item_1);
        ListView lv = (ListView) findViewById(R.id.listView5);
        lv.setAdapter(adapter);
        if (conThread != null) conThread.cancel();
        conThread = new ConnectedThread(mHandler);
    }

    public void serverClick(View view) {
        if (lt != null) lt.cancel();
        lt = new
ListeningThread(BluetoothAdapter.getDefaultAdapter(),MainActiv
ity.MY_UUID, conThread, mHandler);
        lt.start();
    }

    public void connectMeClick(View view) {
```

```
        Intent z = new Intent(this, DeviceListActivity.class);
        startActivityForResult(z, 9);
    }


    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        switch (requestCode) {
            case 9:
                Bundle myData = data.getExtras();
                String MAC = myData.getString("device");
                device2 = BA.getRemoteDevice(MAC);
                // Initiate a connection request in a separate
thread
                if (ct != null) ct.cancel();
                ct = new
ConnectingThread(BluetoothAdapter.getDefaultAdapter(),device2,
MainActivity.MY_UUID, conThread, mHandler);
                ct.start();
                break;
        }
    }

    // The Handler that gets information back from the
BluetoothChatService
    // The Handler that gets information back from the
BluetoothChatService
    private final Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            TextView tv = (TextView)
findViewById(R.id.textView);
            switch (msg.what) {
                case CONNECTED:
                    adapter.add("Connected:  ");
                    break;
                case ACCEPTED:
                    adapter.add("Accepted:  ");
                    break;
                case MESSAGE_WRITE:
                    byte[] writeBuf = (byte[]) msg.obj;
                    // construct a string from the buffer
```

```
                    String writeMessage = new
String(writeBuf);
                    adapter.add("ME:  " + writeMessage);
                    break;
                case MESSAGE_READ:
                    byte[] readBuf = (byte[]) msg.obj;
                    // construct a string from the valid bytes
in the buffer
                    String readMessage = new String(readBuf,
0, msg.arg1);
                    adapter.add(readMessage);
                    break;
            }
        }
    };

    public void sendMsgClick(View view) {
        EditText ed = (EditText) findViewById(R.id.editText);
        String s = ed.getText().toString();
        ed.setText("");
        byte[] writeBuf = s.getBytes();
        if (conThread == null)
            Toast.makeText(this, "Thread missing",
Toast.LENGTH_SHORT).show();
        else
            conThread.write(writeBuf);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (lt != null) lt.cancel();
        if (ct != null) ct.cancel();
        if (conThread != null) conThread.cancel();
    }
}
```

activity_communication.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:orientation="vertical"
    tools:context=".CommunicationActivity">


    <|Button
        android:id="@+id/button10"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="serverClick"
        android:text="I am going to be the server" />

    <|Button
        android:id="@+id/button11"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="connectMeClick"
        android:text="Connecting me to..." />

    <|EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="text" />

    <|Button
        android:id="@+id/button12"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="sendMsgClick"
        android:text="Sent" />

    <|ListView
        android:id="@+id/listView5"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
<|/LinearLayout>
```

ListeningThread.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
```

```
import android.os.Handler;
import android.util.Log;

import java.io.IOException;
import java.util.UUID;

public class ListeningThread extends Thread {
    private final BluetoothServerSocket bluetoothServerSocket;
    ConnectedThread comThread;
    Handler mHandler;

    public ListeningThread(BluetoothAdapter BA, UUID uuid,
ConnectedThread ci, Handler ih) {
        BluetoothServerSocket temp = null;
        comThread = ci;
        mHandler = ih;

        try {
            temp =
BA.listenUsingRfcommWithServiceRecord("MyChatApp", uuid);
        } catch (IOException e) {e.printStackTrace();}
        bluetoothServerSocket = temp;
    }

    public void run() {
        BluetoothSocket bluetoothSocket;
        // This will block while listening until a
BluetoothSocket is returned
        // or an exception occurs
        while (true) {
            try {
                bluetoothSocket =
bluetoothServerSocket.accept();
            } catch (IOException e) { break; }

            // If a connection is accepted
            if (bluetoothSocket != null) {
                // Manage the connection in a separate thread
                comThread.setSocket(bluetoothSocket);
                    // Send the obtained bytes to the UI
Activity

mHandler.obtainMessage(CommunicationActivity.CONNECTED, -1, -
1, null)
```

```
                                .sendToTarget();
                    comThread.start();

                    try {
                        bluetoothServerSocket.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    break;
                }
            }
        }

        // Cancel the listening socket and terminate the thread
        public void cancel() {
            try {
                bluetoothServerSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```

ConnectingThread.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.os.Handler;
import android.util.Log;

import java.io.IOException;
import java.util.UUID;

class ConnectingThread extends Thread {
    private final BluetoothSocket bluetoothSocket;
    private final BluetoothDevice device2;
    BluetoothAdapter BA;
    ConnectedThread comThread;
    Handler mHandler;

    public ConnectingThread(BluetoothAdapter iBA,
BluetoothDevice dev2, UUID uuid, ConnectedThread ci, Handler
ih) {
```

```
        BluetoothSocket temp = null;
        device2 = dev2;
        BA = iBA;
        comThread = ci;
        mHandler = ih;

        // Get a BluetoothSocket to connect with the given
BluetoothDevice
        try {
            temp =
dev2.createRfcommSocketToServiceRecord(uuid);
        } catch (IOException e) {
            e.printStackTrace();
        }
        bluetoothSocket = temp;
    }

    public void run() {
        // Cancel any discovery as it will slow down the
connection
        BA.cancelDiscovery();

        try {
            // This will block until it succeeds in connecting
to the device
            // through the bluetoothSocket or throws an
exception
            bluetoothSocket.connect();
        } catch (IOException connectException) {
            connectException.printStackTrace();
            try {
                bluetoothSocket.close();
            } catch (IOException closeException) {
                closeException.printStackTrace();
            }
        }

        // Code to manage the connection in a separate thread
        comThread.setSocket(bluetoothSocket);
        try {
            // Send the obtained bytes to the UI Activity

mHandler.obtainMessage(CommunicationActivity.ACCEPTED, -1, -1,
null)
```

```
                    .sendToTarget();
        } catch (Exception e) {
            Log.e("", "error", e);
        }
        comThread.start();
    }

    // Cancel an open connection and terminate the thread
    public void cancel() {
        try {
            bluetoothSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ConnectedThread.java

```
package com.example.sinus.bluetooth16;

import android.bluetooth.BluetoothSocket;
import android.os.Handler;
import android.util.Log;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;


public class ConnectedThread extends Thread {
    private BluetoothSocket socket;
    private InputStream inStream;
    private OutputStream outStream;
    private final Handler mHandler;

    public ConnectedThread(Handler ih) {
        mHandler = ih;
    }

    public void setSocket(BluetoothSocket s) {
        socket = s;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
```

```
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e("", "temp sockets not created", e);
        }
        inStream = tmpIn;
        outStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];
        int bytes;
        // Keep listening to the InputStream while connected
        while (true) {
            try {
                // Read from the InputStream
                bytes = inStream.read(buffer);
                // Send the obtained bytes to the UI Activity

mHandler.obtainMessage(CommunicationActivity.MESSAGE_READ,
bytes, -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                Log.e("", "disconnected", e);
                break;
            }
        }
    }
    /**
     * Write to the connected OutStream.
     * @param buffer  The bytes to write
     */
    public void write(byte[] buffer) {
        try {
            outStream.write(buffer);
            // Share the sent message back to the UI Activity

mHandler.obtainMessage(CommunicationActivity.MESSAGE_WRITE, -
1, -1, buffer)
                    .sendToTarget();
        } catch (IOException e) {
            Log.e("", "Exception during write", e);
        }
    }
    public void cancel() {
```

```
        try {
            socket.close();
        } catch (IOException e) {
            Log.e("", "close() of connect socket failed", e);
        }
    }
}
```

Permissions

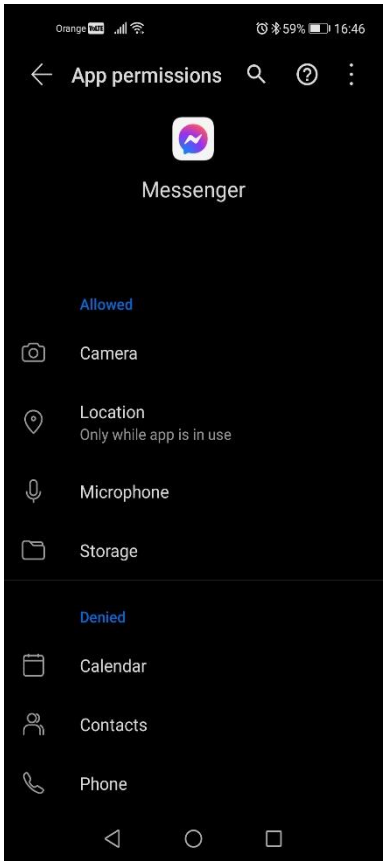
Chapter 10

10.1 Definition of permissions

🕮 10.1.1

Permissions before

Android is version by version a more and more secure system. From the first versions of the application, they informed which elements they require access to, but after approving the confirmation, we could not do anything with these permissions.



🕮 10.1.2

Permissions (new approach)

Each Android application runs in a separate instance of the virtual machine (sandbox). If it wants to collect information from outside the environment, it must ask for permissions.

In the case of standard permissions, it is sufficient to include them in the manifest.

From API version 24, the situation on the retention of permission information has changed compared to previous versions. While standard permissions are confirmed when installing an application, as was the case with earlier versions, "dangerous" permissions must be defined in the manifest and then requested by the application and confirmed by the user.

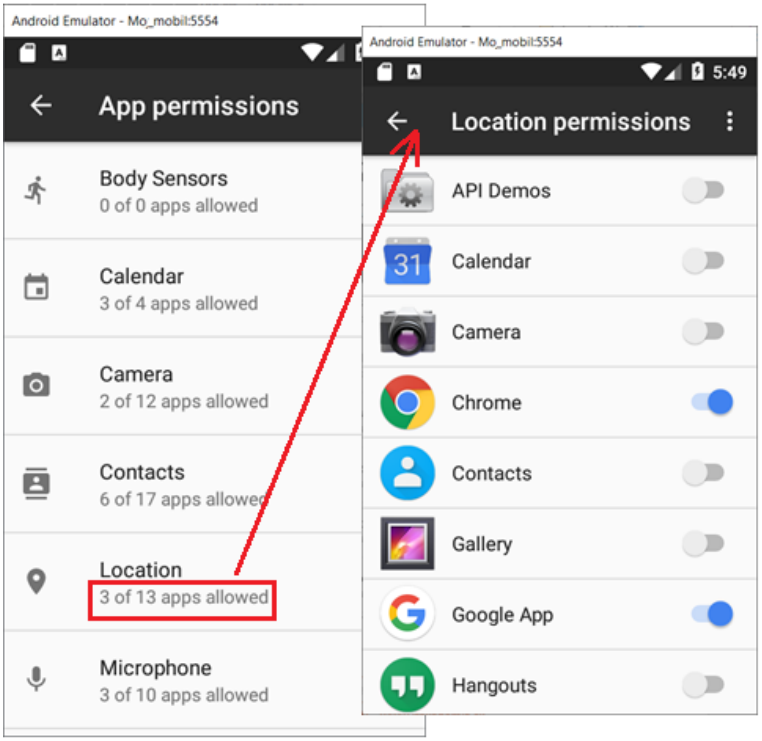
These authorizations include, for example:

- READ_CALENDAR
- WRITE_CALENDAR
- CAMERA
- READ_CONTACTS
- WRITE_CONTACTS
- GET_ACCOUNTS
- ACCESS_FINE_LOCATION
- ACCESS_COARSE_LOCATION
- RECORD_AUDIO
- READ_PHONE_STATE
- CALL_PHONE
- READ_CALL_LOG
- WRITE_CALL_LOG
- ADD_VOICEMAIL
- USE_SIP
- PROCESS_OUTGOING_CALLS
- BODY_SENSORS
- SEND_SMS
- RECEIVE_SMS
- READ_SMS
- RECEIVE_WAP_PUSH
- RECEIVE_MMS
- RECEIVE_MMS
- READ_EXTERNAL_STORAGE
- WRITE_EXTERNAL_STORAGE

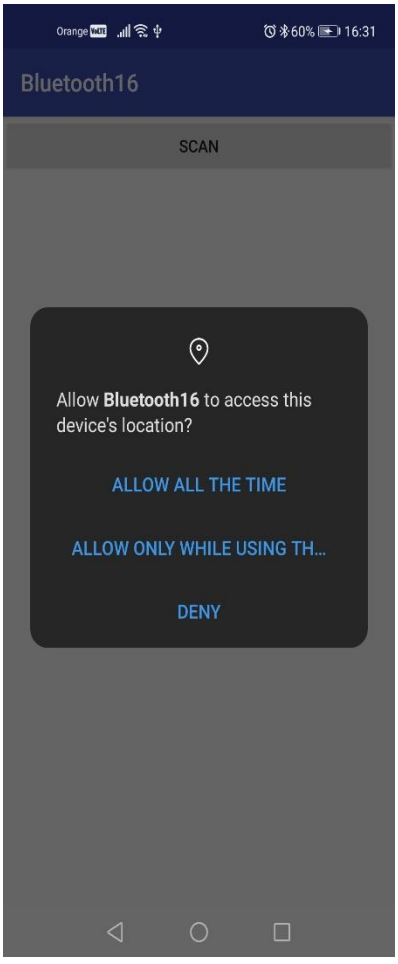
🕮 10.1.3

User options

After switching to the item in the settings, we have a list of applications that have access to sensitive hardware or sensitive data, and we can revoke the authorization at any time.



So even if the app used a camera yesterday or was allowed to use GPS, it can't assume that it has such a permission today.



🕮 10.1.4

Requesting permissions

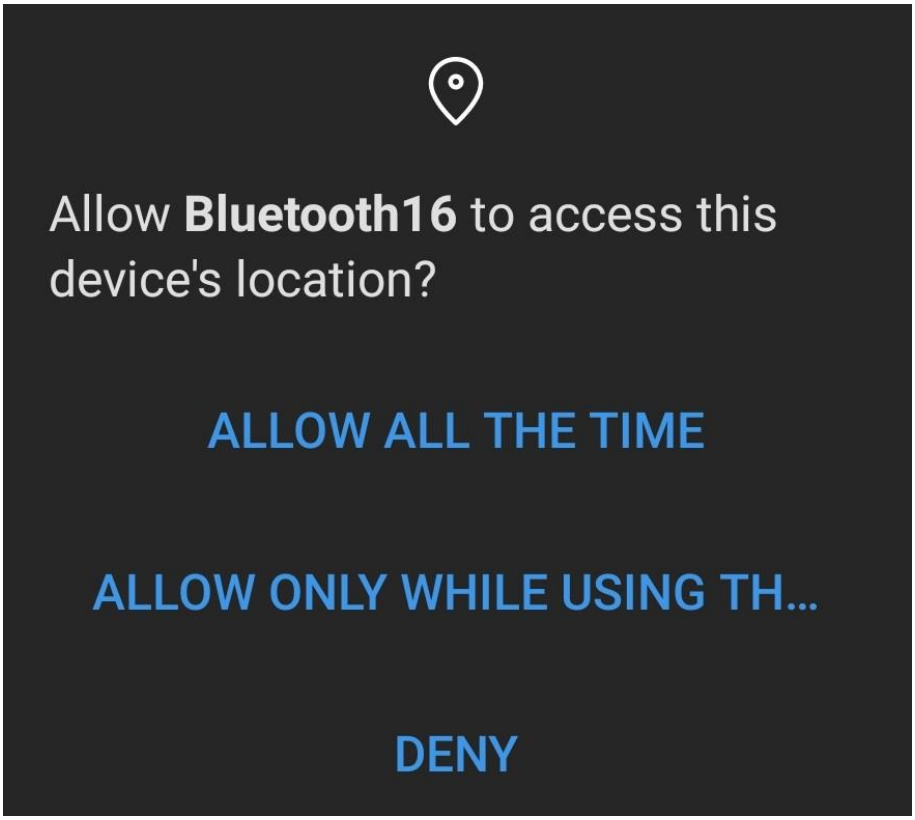
If we do not have permissions, we need to request them, for which the **ActivityCompat.requestPermissions()** method is used, in which we define one or a group of permissions that the application requires.

The **requestPermission()** method is a request to obtain permission:

- the first parameter is the activity
- the second parameter is an array of strings representing one or more permissions
- the last parameter is the code identifying the request in the **onRequestPermissionsResult** method

```
if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.CAMERA)
    != PackageManager.PERMISSION_GRANTED {
      ActivityCompat.requestPermissions(this,
                            new
String[]{Manifest.permission.CAMERA}, 5);
}
```

Upon request, a dialog for assigning permissions is displayed:



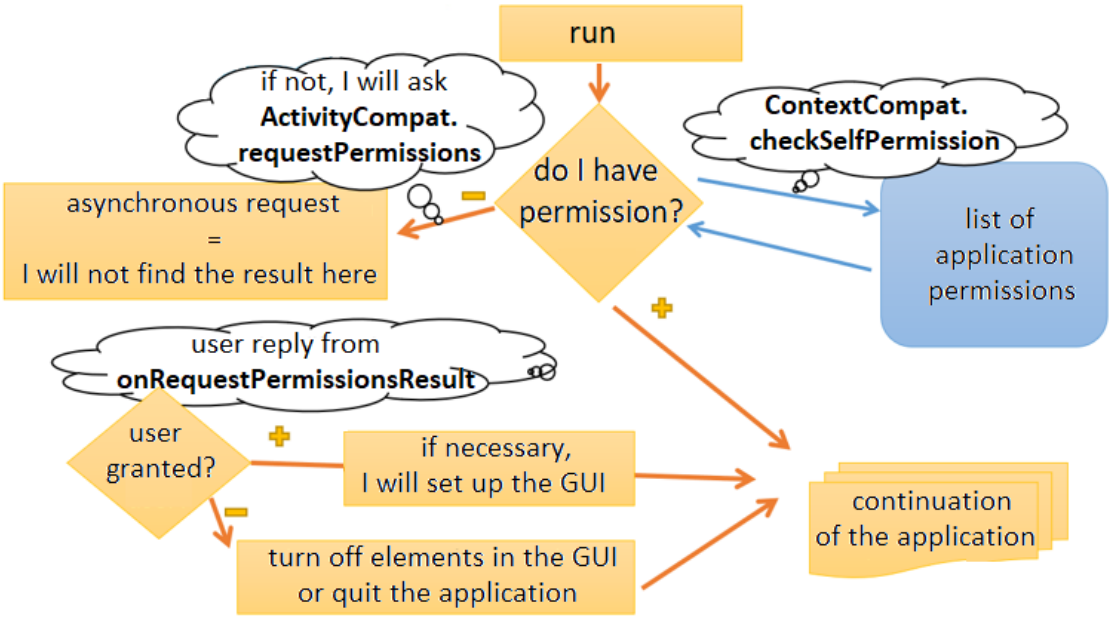
🕮 10.1.5

Permissions verification scheme

For the specified **requestCode**, we will go through the list of required permissions, for the **grantResult** specified by the user, we will list whether the permission has been assigned or not, and at the same time we will list from the **permissions** which permission it is.

If we lack some permissions, the device cannot be used and it is necessary to either close the application or adjust its functionality accordingly (e.g. it is not possible to take pictures - we deactivate the button).

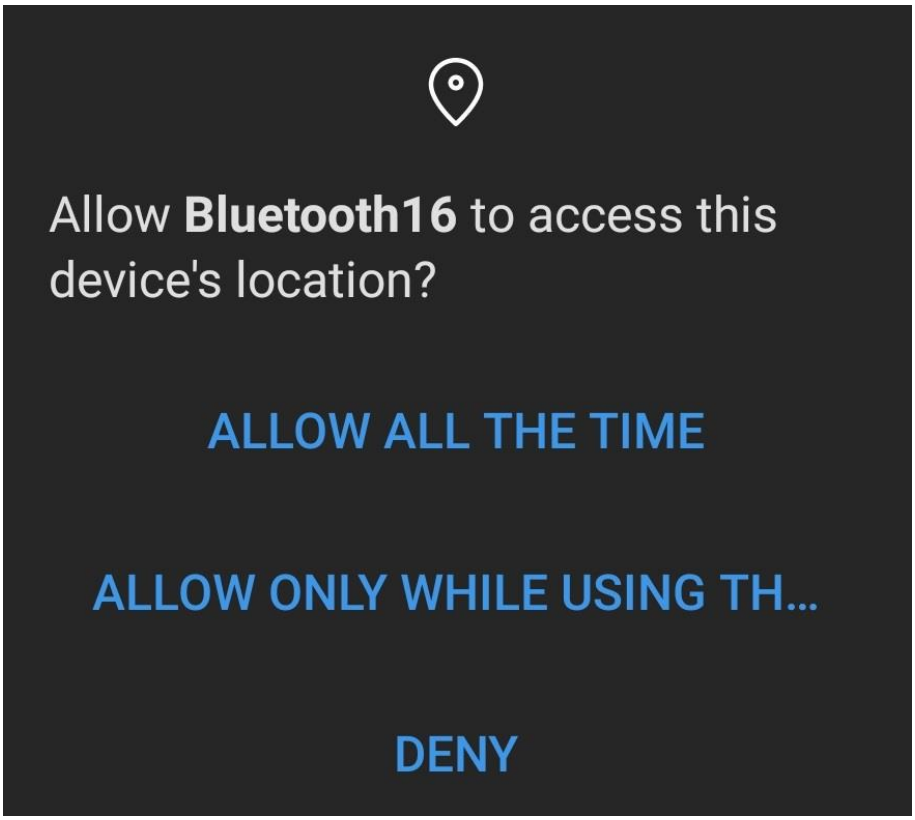
The verification and acquisition of permissions scheme is as follows:



🕮 10.1.6

Verification of the acquisition of permissions

The call displays a dialog for individual permissions and expects a user response => the request is asynchronous, if we want to respond to acceptance or rejection, we must do so in the **onRequestPermissionResult** method.



After the user's response, we identify the parameters in the **onRequestPermissonsResult** method,

- requestCode identifies the request, which was defined by the programmer by the number when calling the request,
- permissions is a list of required permissions - now only one
- grantResults is information about assigning permissions to the given permission(s)

```
public void onRequestPermissionsResult(int requestCode,
                                       String[] permissions, int[]
grantResults) {
    switch (requestCode) {
        case 5: if (grantResults.length > 0 && // identifying
0. permission
                grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                   Toast.makeText(this, "Camera - OK",
Toast.LENGTH_SHORT).show();
        } else {
            buttonImage.setEnabled(false); // deactivating the
camera button
```

```
            Toast.makeText(this, "Camera won't be used",
Toast.LENGTH_SHORT).show();
        }
    }
}
```

The return from the permissions allocation request is the **grantResults** array, which contains a list of permissions from the communication call.

If we specify only one item in the request, we only need to check the value for permissions in the 0th position (there is no longer a list of permissions in the array, but only the values (allowed, denied) in the sequence as the request was called). So if some permissions were returned (grantResults.length> 0), then if the first permission was confirmed (in the meantime it was written to the database of confirmed permissions), we can continue.

If the application has not obtained permissions, then e.g. I turn off the elements that are associated with the permission or quit the application

🕮 10.1.7

Permissions in code

Usually, a permission is requested in the **onCreate()** method of the application, or when accessing the device for the first time. The application then remembers the permission.

The permission check takes place within **ContextCompat.checkSelfPermission()** and returns PERMISSION_GRANTED or PERMISSION_DENIED. The code needs to be adapted to ensure that the application requests permission to use the dangerous permissions and deals with the situation if it does not receive this permission.

The following code checks the permission to access the camera if the permission is not stored in the repository, the application can save it after request and confirmation - if it is then stored, it does not ask for it again.

```
if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.CAMERA)
    != PackageManager.PERMISSION_GRANTED {
...
}
```

If the permission is not granted, it is necessary to ask for it or notify the user that the functions in the application are missing or disabled, etc.

The easiest way to change permissions when testing an application is to uninstall the application -> new application = new permissions.

Server Communication

Chapter 11

11.1 HTTP request

📖 11.1.1

A web browser is a standard part of the application equipment of smartphones. Allows us to receive and send data most often via http and https protocols.

In addition, many applications communicate with "their server" within the functionalities, e.g. for profile storage, content download, communication with teammates, social networks, etc.

📖 11.1.2

Assignment

Create an application, that can read data from a website.

📖 11.1.3

Classes for communication

The basic class for communication is **HttpURLConnection**, which input is a URL address. Opening a connection creates a connection to the given URL and gains access to the input channel from which the content at the given URL is read.

```
URL url = new URL("http://cnn.com/");
 HttpURLConnection urlConnection = (HttpURLConnection)
url.openConnection();
 try {
   InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
   readStream(in);
  } finally {
    urlConnection.disconnect();
  }
```

Many examples of communication via http can be found on the web, but they are no longer up to date.

- as of API version 22 (Android 5.1) **HttpPost**, **HttpGet** are deprecated
- as of API level 23, the **Apache HTTP client** is not used either

🕮 11.1.4

Permissions

To access communication channels (e-mail, web), it is necessary to set permissions in the manifest.

We place the definition above the <**application**> setting:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a17_webreader">
    <uses-permission
android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

🕮 11.1.5

The process of download

Downloading content from the Internet is one of the more time consuming operations. It is not used as part of GUI code (threads for the GUI), because long operations would give the impression of an application freezing and the application would not respond to user interactions or communicate during the download.

The **AsyncTask** class is generally used for more time-consuming operations.

🕮 11.1.6

AsyncTask

An instance of the **AsyncTask** class allows you to perform operations:

- out of thread UI = application responds
- in the background, in a separate thread
- with output to the GUI elements of the activity
- without the need to create a mechanism for working with threads
- ideally designed for shorter operations (several seconds)

It's a good idea for an instance to be part of the activity in which it works, because it allows it to directly access the components to which it displays messages or retrieved content.

🕮 11.1.7

AsyncTask structure

It is necessary to define them by a pair of **doInBackground** and **onPostExecute** methods:

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
   ...
   public class GetMyData extends AsyncTask<String, Void,
String> {
     @Override
     protected String doInBackground(String... params){
         // gets the result
       return vysledok;
     }
     protected void onPostExecute(String result){
       // usually operations in GUI
     }
  } // kend of GetMyData
 ...
} // end of activity
```

The output obtained within the **doInBackground** method is the input to the **onPostExecute** method => the **doInBackground** method type is the same as the parameter type in **onPostExecute**.

In the first method, the result is obtained/downloaded, in the second, it is usually written to the GUI activity.

🕮 11.1.8

If we look at the **AsyncTask** classes in more detail, we notice three generic parameters in the child header:

```
public class GetMyData extends AsyncTask<String, Integer,
Long> {
  @Override
  protected Long doInBackground(String... params){
    return my_result;
  }

  protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]); // set e.g. to
progress bar
    }

  protected void onPostExecute(Long result){ }
}
```

The three parameters in the header are used as:

- the first parameter defines the type of parameters entering the **doInBackground** method
- the second parameter defines the types of inputs (parameters) to the **onProgressUpdate** method
- the third parameter defines the type of the result (type of the return method in doInBackgroud and input to the **onPostExecute** method)

The type of parameters sent to **doInBackground** (parameter **String** ...) means that their number is not defined and any number of them can be sent (but all must have the type **String**)

Progress during the process is provided by the optional **onProgressUpdate()** method and is used for display. It is updated from **doInBackground** by calling **publishProgress()**.

Unused parameters can be specified as **Void** in the child header.

🕮 11.1.9

Initialization of content acquisition

Initialization consists of e.g. in processing a click on a button to create an instance of **AsyncTask** and call the **execute()** method with parameters sent to **doInBackground()**.

Our child **AsyncTask** will take care of the download and display after its completion.

```
public void onClick(View view) {
    // URL of the website
  String myUrl = "https://cnn.com";
    // instance creation for download
  GetMyData getRequest = new GetMyData();
    // calling the method to start the AsyncTask
    // here we could use any number of string parameters
  getRequest.execute(myUrl);
}
```

🕮 11.1.10

Calling **getRequest.execute(myUrl)** ensures that the internal process runs the doInBackground method, which:

- reads the first parameter entering the **doInBackground()** method
- creates a URL and gets a connection to it

```
public class GetMyData extends AsyncTask<String, Void, String>
{
 @Override
 protected String doInBackground(String... params){
  String stringUrl = params[0]; // reading the first parameter
  String result = "", row;
 try {
     // creating the URL
   URL myUrl = new URL(stringUrl);
    // estabilishing connection
   HttpURLConnection con =(HttpURLConnection)
myUrl.openConnection();
...
```

🕮 11.1.11

Then the connection parameters are set and the client connects.

When reading web content, **BufferedReader** is opened and in the standard way (as from a regular file/stream) it ensures reading of content (= responses) from the destination address.

```
    // setting the method and the timeout
   con.setRequestMethod("POST");
   con.setReadTimeout(1500000);
con.setConnectTimeout(1500000);
     // connecting to the URL
   con.connect();
     // Buffered reader for reading the input = response
   InputStreamReader sR = new
InputStreamReader(con.getInputStream());
   BufferedReader reader = new BufferedReader(sR);
     // reading while there is something to read
   while((row = reader.readLine()) != null){
     result = result + row;
 }

 reader.close(); // the end
 streamReader.close();
} catch(Exception e){
   Log.d("error: ",e.toString());
   result = null; }
return result;
}
```

🕮 11.1.12

View acquired content

The result obtained and processed by the **doInBackground()** method is taken over by the **onPostExecute()** method and displayed in the GUI views.

```
public class GetMyData extends AsyncTask<String, Void, String>
{
  @Override
  protected String doInBackground(String... params){
    ...
    return result;
```

```
  }

  protected void onPostExecute(String result){
    super.onPostExecute(result);
    TextView tv = (TextView) findViewById(R.id.textView);
    tv.setText(result);
    Toast.makeText(MainActivity.this, "OK",
Toast.LENGTH_SHORT).show();
 }
}
```

Please note that the result will not be in the form of a web page - we only obtained the HTML code.

11.1.13 The code of the application

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a17_webreader">
    <|uses-permission
android:name="android.permission.INTERNET" />

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.17WebReader">
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
    <|/application>

<|/manifest>
```

MainActivity.java

```
package com.example.a17_webreader;

import androidx.appcompat.app.AppCompatActivity;

import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.InputStreamReader;
```

```
import java.net.HttpURLConnection;
import java.net.URL;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        // the URL adress of the website
        String myUrl = "http://cnn.com";
        // instance creation for download
        GetMyData getRequest = new GetMyData();
        // calling the method to start the AsyncTask
        // here we could use any number of String parameters
        getRequest.execute(myUrl);
    }

    public class GetMyData extends AsyncTask<|String, Void,
String> {

        @Override
        protected String doInBackground(String... params) {
            String stringUrl = params[0]; // reading the first
parameter
            String result = "", row;

            try {
                // create URL
                URL myUrl = new URL(stringUrl);

                // create connection
                HttpURLConnection con = (HttpURLConnection)
myUrl.openConnection();

                // setting the method and timeout parameters
                con.setRequestMethod("POST");
                con.setReadTimeout(1500000);
                con.setConnectTimeout(1500000);

                // connect to the URL
```

```
                con.connect();

                // Buffered reader for reading the input =
result
                InputStreamReader sR = new
InputStreamReader(con.getInputStream());
                BufferedReader reader = new
BufferedReader(sR);

                // we read while there is something to read
                while ((row = reader.readLine()) != null) {
                    result = result + row;
                }

                reader.close(); // the end
                reader.close();
            } catch (Exception e) {
                Log.d("x", e.toString());
                result = null;
            }
            return result;
        }

        protected void onPostExecute(String result) {
            super.onPostExecute(result);
            TextView tv = (TextView)
findViewById(R.id.textView);
            tv.setText("my text: " + result);
            Toast.makeText(MainActivity.this, "OK",
Toast.LENGTH_SHORT).show();
        }
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Start download" />

    <|TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="TextView" />
<|/LinearLayout>
```

11.2 Working with server-side data

🕮 11.2.1

Data sharing

By default, data between users of the same application is shared through databases located on publicly available servers.

Although direct access to the database is possible, it carries many security risks and is therefore strongly discouraged.

The ideal state for creating an infrastructure for working with data is a server with Java (servlets) and the exchange of entire instances with data.

Common status: PHP (+ My SQL) and communication via text/json data.

🕮 11.2.2

Assignment

Create a client-server application that can store data about contacts and the length of communication with them in a central database

- client on a smartphone with the ability to send a record and read all data from the database

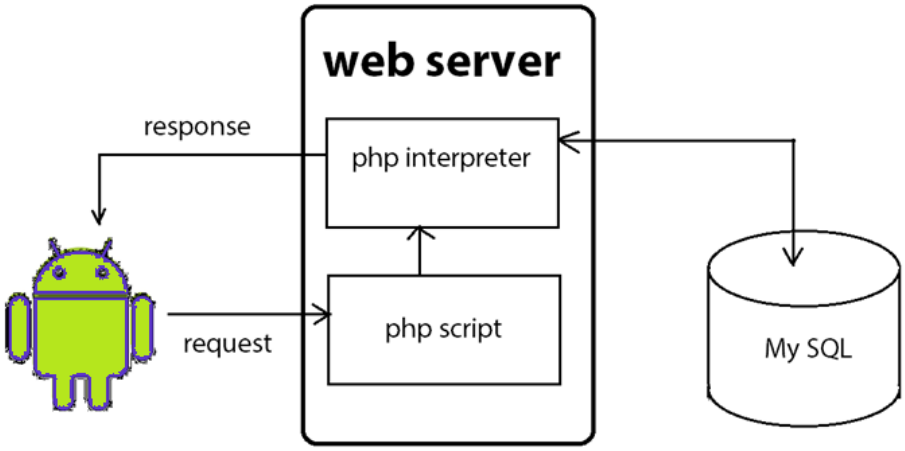
- scripts on the server (can be separate) for downloading data, storing data and sending (= displaying) data from the database to the smartphone

🕮 11.2.3

Architecture

The architecture of the application is obvious:

- the smartphone sends a request to run the php script
- the php script sends a web page to the client as a result of processing (= text)



🕮 11.2.4

Server side

The server provides:

- running a database server with a database containing shared data
- web server with support for running PHP scripts
- PHP script for storing data obtained from a smartphone
- PHP script for reading data from a database and generating a document with them
- as an additional alternative - HTML form to test the functionality of the scripts.

🕮 11.2.5

Database

Customers table in My SQL database with fields:

- ID - Autoincrement
- name - varchar
- surname - varchar
- spent_time - float.

🕮 11.2.6

Script for storing data

It consists of reading configuration data to connect to the database, reading data obtained through POST and then writing data to the database.

The text printed via the **echo** command generates a page = response for the client.

```
<?php
  include 'config.php';
    // create the connection
  $conn = mysqli_connect($servername, $username, $password,
$dbname);
  if ($conn->connect_error) {
     die("Connection failed: " . $conn->connect_error);
  }

     // read the data sent by POST
  $_name = $_POST["name"];
  $_surname = $_POST["surname"];
  $_spent_time = $_POST["spent_time"];
    // query for the database
  $sql= "INSERT INTO customers (name, surname, time) VALUES
        '" . $_name . "','" . $_surname . "','" . $spent_time
. "')";
    // feedback, if the query succeeded or not
  if (mysqli_query($conn,$sql)) { echo "OK ".$sql;
    } else { echo "0".$sql; }
    // closing the connection
  mysqli_close($conn);
?>
```

🕮 11.2.7

Connection configuration

To ensure communication with the database, PHP uses an elegant solution that provides database configuration to all scripts by linking a configuration file.

This is initially included in all scripts that need to connect to the database.

Contains name, password, server and database, e.g. in the form of:

```
<?php
  $servername = "localhost";
  $username = "myLogin";
  $password = " I don't have it, I won't give it, I will keep
it ";
  $dbname = "myDatabase";
?>
```

The server is usually localhost because the database system runs on the same server as PHP.

🕮 11.2.8

Printing the data

We will not format the content for better/easier data processing on the client side.

```
<?php
  include 'config.php';
    // trying to connect...
  $conn = mysqli_connect($servername, $username,
$password,  $dbname);
  if (!$conn) { die("Connection failed: " .
mysqli_connect_error()); }
    // selecting the data
  $sql = "SELECT name, surname, spent_time FROM zoznam";
  $result = mysqli_query($conn, $sql);
  if (mysqli_num_rows($result) == 0) {
       // if the table is empty
     echo "0";
   } else {
       // output record by record separated by the character
„|“
```

```
     while($row = mysqli_fetch_assoc($result)) {
        echo $row["name"]. ";" . $row["surname"]. ";" .
$row["spent_time"]."|";
    }
  }
  mysqli_close($conn);
?>
```

11.2.9

Test form

To test the functionality of our code, it is advisable to have an HTML page that sends data to the database:

```
<form action="write.php" method="post">
   <input type="text" name ="name">
   <input type="text" name ="surname">
   <input type="text" name ="spent_time">
   <input type="submit" value="Send data">
</form>
```

The parameter names are identical to the data read in PHP:

```
 // reading the data sent by POST
 $_name = $_POST["name"];
 $_surname = $_POST["surname"];
 $_spent_time = $_POST["spent_time"]
```

11.2.10 The code of the application

write.php

```
<|?php
  include 'config.php';
    // trying to connect
  $conn = mysqli_connect($servername, $username, $password,
$dbname);
  if ($conn->connect_error) {
     die("Connection failed: " . $conn->connect_error);
  }

     // reading the POST data
  $_name = $_POST["name"];
```

```
  $_surname = $_POST["surname"];
  $_spent_time = $_POST["spent_time"];
    // query for the database
  $sql= "INSERT INTO customers (name, surname, time) VALUES
        '" . $_name . "','" . $_surname . "','" . $spent_time
. "')";
    // feedback = whether the query was successfull or not
  if (mysqli_query($conn,$sql)) { echo "OK ".$sql;
    } else { echo "0".$sql; }
    // closing the connection
  mysqli_close($conn);
?>
```

output.php

```
<|?php
  include 'config.php';
    // trying to connect
  $conn = mysqli_connect($servername, $username, $password,
$dbname);
  if (!$conn) { die("Connection failed: " .
mysqli_connect_error()); }
    // selecting the data
  $sql = "SELECT name, surname, spent_time FROM zoznam";
  $result = mysqli_query($conn, $sql);
  if (mysqli_num_rows($result) == 0) {
       // if the table is empty
     echo "0";
   } else {
       // the output record by record separated by „|“
     while($row = mysqli_fetch_assoc($result)) {
        echo $row["name"]. ";" . $row["surname"]. ";" .
$row["spent_time"]."|";
   }
  }
  // closing the connection
  mysqli_close($conn);
?>
```

my_form.html

```
<|form action="write.php" method="post">
   <|input type="text" name ="name">
   <|input type="text" name ="surname">
   <|input type="text" name ="spent_time">
   <|input type="submit" value="Send data">
<|/form>
```

11.3 Communication with the server

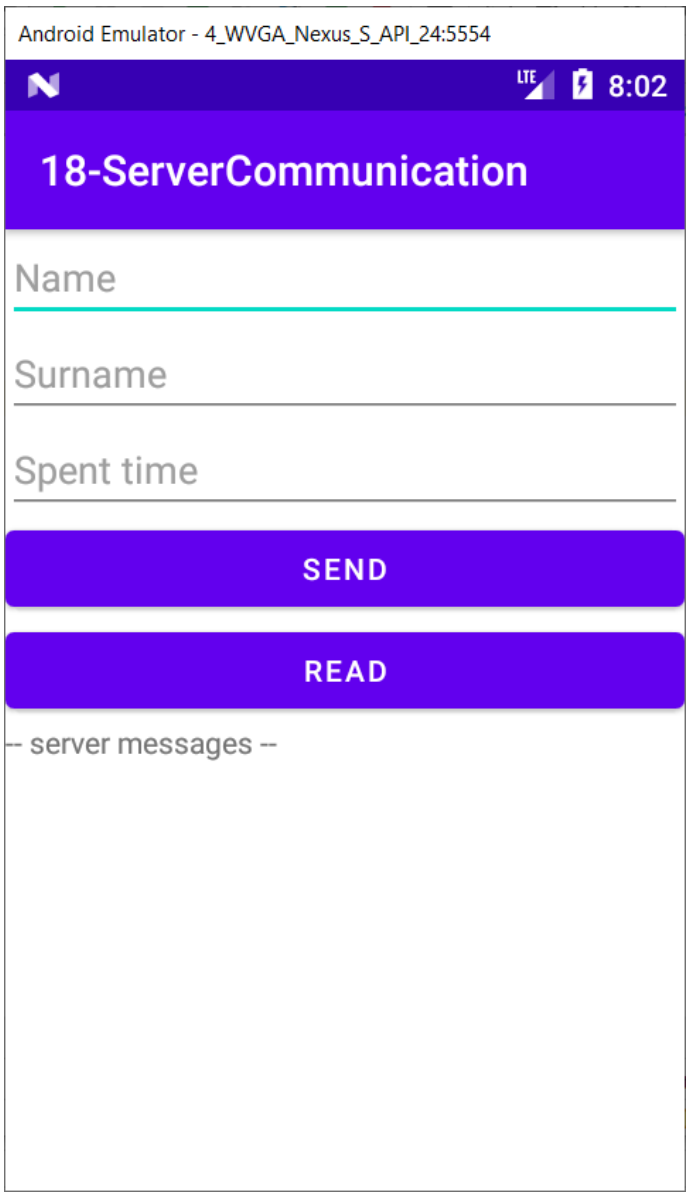
🕮 11.3.1

Client - android application

It remains to create "only" the client part of the application.

We will design an interface that will have:

- at the top - fields for entering data sent via post to the server
- button to read data from the database
- in the lower part - **TextView**, in which the data will be displayed or all responses from the server



📖 11.3.2

Data acquisition

A simpler operation is to load data, which in its simplest form is actually just reading the content of a website

To process the request, we will use the same child element of **AsyncTask** as in the case of the previous task:

```
public void onClick(View view) {
    // the URL of the website
    String myUrl = "https://.../android/output.php";
    // creating the instance for download
    GetMyData getRequest = new GetMyData();
    // calling the method to start AsyncTask
    // here we can use any number of String parameters
    getRequest.execute(myUrl, "1");
 }
```

So that we don't have to create a special **AsyncTask** child to write, we add a parameter to the call to specify what to do.

📖 11.3.3

In the **doInBackground()** method, set the connection parameters and send a request to the given address:

```
protected String doInBackground(String... params) {
    String stringUrl = params[0]; // reading the URL
    int task_type = Integer.parseInt(params[1]); // 2.
parameter - what must be done
    String result = "", row;

    try {
          // creating the URL
        URL url = new URL(stringUrl);
          // connecting to the specified URL
        URLConnection conn = url.openConnection();
        conn.setDoOutput(true); // setting the method to POST
          // connection parameters
        conn.setReadTimeout(15000);
conn.setConnectTimeout(15000);
```

```
        switch (task_type) {
            case 1: // create the connection - everything is
ready, we only send the query
                conn.connect();
                break;
        }
```

We read the data and return the result:

```
          // creating a response stream
        InputStreamReader streamReader = new
InputStreamReader(conn.getInputStream());
          // creating the reader
        BufferedReader reader = new
BufferedReader(streamReader);
          // reading the data from the stream
        while ((row = reader.readLine()) != null) {
            result += row;
        }
          // close
        reader.close();
        streamReader.close();
    } catch (Exception e) {
        Log.d("x", e.toString());
        result = null;
    }
    return result;
}
```

📖 11.3.4

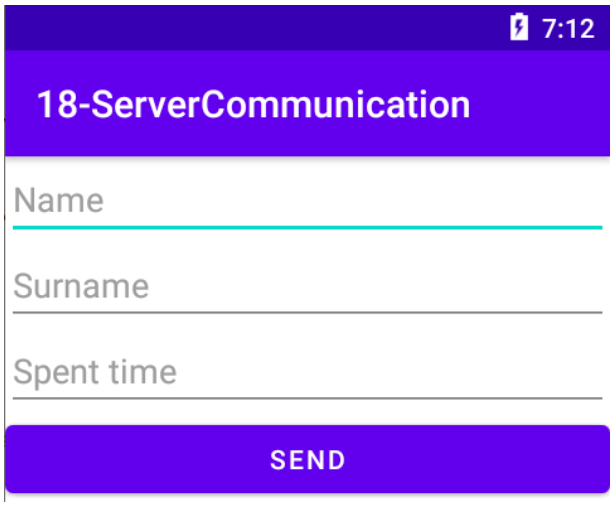
The actual display of the data takes place in the method in the **onPostExecute** method. We will use the same method both when displaying data and when displaying the response to inserting data into the database. The **doInBackground()** method generates a response, this method (**onPostExecute**) only displays it.

```
protected void onPostExecute(String result) {
  super.onPostExecute(result);
  TextView tv = (TextView) findViewById(R.id.textView);
  tv.setText(result);
  Toast.makeText(MainActivity.this, "Done",
Toast.LENGTH_SHORT).show();
}
```

🕮 11.3.5

Data storing

The data is stored in response to a button click.



An instance of **AsyncTask** is re-created in the appropriate method and all data is sent to it.

Thanks to the ability to use a different number of parameters, we can name the data without having to modify the header of the **doInBackground()** method.

```
public void onClickSend(View view) {
    String myUrl = "http://.../android/write.php";
    String _name = ((EditText)
(findViewById(R.id.editName))).getText().toString();
    String _surname = ((EditText)
(findViewById(R.id.editSurname))).getText().toString();
    String _ttime = ((EditText)
(findViewById(R.id.editTime))).getText().toString();
    _ttime.replace(',','.'); // to have a decimal point, not a
comma
      // creating the instance
    GetMyData getRequest = new GetMyData();
      // sending the parameters
    getRequest.execute(myUrl, "2", _name, _surname, _ttime);
}
```

🕮 11.3.6

In the **doInBackground()** method, we just add the part for task "2" in the switch structure so that it reads all sent parameters.

Call:

```
getRequest.execute(myUrl, "2", _name, _surname, _ttime);
```

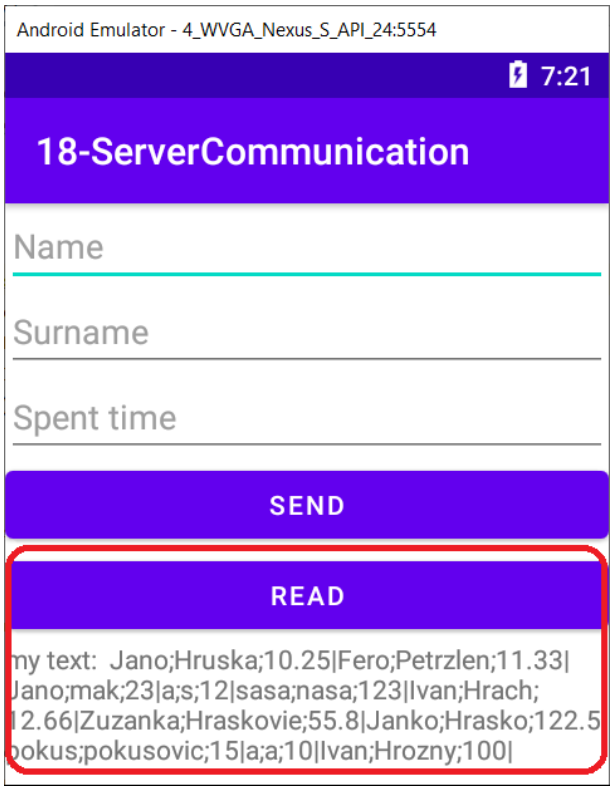
will be processed as:

```
switch (task) {
 case 1: // task to get the server data
   conn.connect();
   break;
 case 2: // reading the parameters from the method
   String first_name = params[2];
   String last_name = params[3];
   String time = params[4];
      // accessing the output stream of the request, creating
parameters
   OutputStream output = conn.getOutputStream();
   String data = URLEncoder.encode("first_name", "UTF-8") +
"=" +
                  URLEncoder.encode(first_name, "UTF-8");
      data += "&" + URLEncoder.encode("last_name", "UTF-8") +
"=" +
                     URLEncoder.encode(last_name, "UTF-8");
      data += "&" + URLEncoder.encode("time", "UTF-8") + "=" +
                     URLEncoder.encode(time, "UTF-8");
     // write to the stream
   output.write(data.getBytes("UTF-8"));
     // send
   output.flush();
   output.close();
   break;
}
```

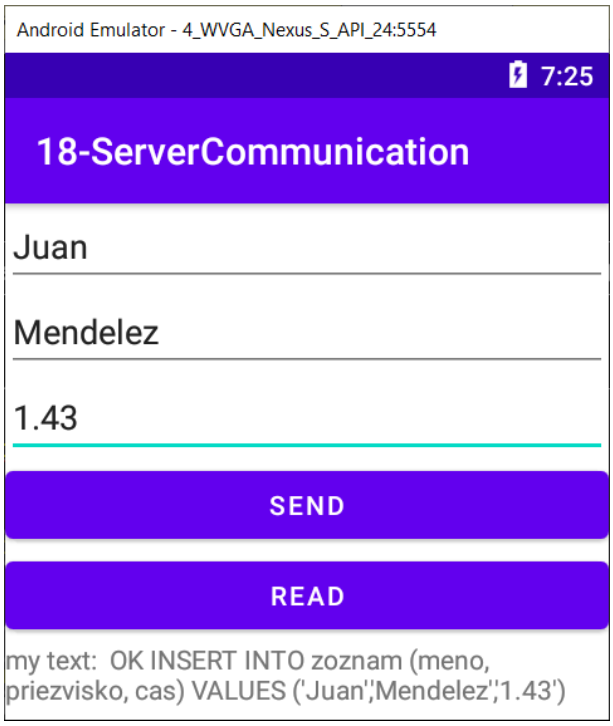
📖 11.3.7

The resulting application

either reads and displays the data:



or returns OK when writing, or a query with which the data was entered:



11.3.8 The code of the application

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a18_servercommunication">
    <|uses-permission
android:name="android.permission.INTERNET" />

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.18ServerCommunication">
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
    <|/application>

<|/manifest>
```

MainActivity.java

```
package com.example.a18_servercommunication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        // the URL of the website
        String myUrl = "https://.../android/output.php";
        // Creating the instance for download
        GetMyData getRequest = new GetMyData();
        // calling the method to start the AsyncTask
        // here we could use any number of String parameters
        getRequest.execute(myUrl, "1");
    }

    public void onClickSend(View view) {
        String myUrl = "https://.../android/write.php";
        String _name = ((EditText)
(findViewById(R.id.editName))).getText().toString();
        String _surname = ((EditText)
(findViewById(R.id.editSurname))).getText().toString();
        String _ttime = ((EditText)
(findViewById(R.id.editTime))).getText().toString();
        _ttime.replace(',','.'); // to have a floating point,
instead of a comma
          // creating the instance
        GetMyData getRequest = new GetMyData();
          // sending the parameters
        getRequest.execute(myUrl, "2", _name, _surname,
_ttime);
    }

    public class GetMyData extends AsyncTask<|String, Void,
String> {
```

```
        @Override
        protected String doInBackground(String... params) {
            String stringUrl = params[0]; // reading the URL
            int task_type = Integer.parseInt(params[1]); // 2.
parameter - what must be done
            String result = "", row;

            try {
                // creating the URL
                URL url = new URL(stringUrl);
                // connecting to the URL
                URLConnection conn = url.openConnection();
                conn.setDoOutput(true); // setting the method
to POST
                // parameters of the connection
                conn.setReadTimeout(15000);
conn.setConnectTimeout(15000);

                switch (task_type) {
                    case 1: // create the connection -
everything is ready, only the query is sent
                        conn.connect();
                        break;
                    case 2: // reading the parameters from the
method call
                        String first_name = params[2];
                        String last_name = params[3];
                        String time = params[4];
                        // accessing the output stream of the
request, creating parameters
                        OutputStream output =
conn.getOutputStream();
                        String data =
URLEncoder.encode("first_name", "UTF-8") + "=" +
                                URLEncoder.encode(first_name,
"UTF-8");
                        data += "&" +
URLEncoder.encode("last_name", "UTF-8") + "=" +
                                URLEncoder.encode(last_name,
"UTF-8");
                        data += "&" +
URLEncoder.encode("time", "UTF-8") + "=" +
```

```
                                URLEncoder.encode(time, "UTF-
8");
                        // write to the stream
                        output.write(data.getBytes("UTF-8"));
                        output.flush(); // send
                        output.close();
                        break;
                }

                  // creating the response stream
                InputStreamReader streamReader = new
InputStreamReader(conn.getInputStream());
                  // creating the reader
                BufferedReader reader = new
BufferedReader(streamReader);
                  // reading data from the stream
                while ((row = reader.readLine()) != null) {
                    result += row;
                }
                  // close
                reader.close();
                streamReader.close();
            } catch (Exception e) {
                Log.d("x", e.toString());
                result = null;
            }
            return result;
        }

        protected void onPostExecute(String result) {
            super.onPostExecute(result);
            TextView tv = (TextView)
findViewById(R.id.textView);
            tv.setText("my text: " + result);
            Toast.makeText(MainActivity.this, "OK",
Toast.LENGTH_SHORT).show();
        }
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|EditText
        android:id="@+id/editName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Name" />

    <|EditText
        android:id="@+id/editSurname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Surname" />

    <|EditText
        android:id="@+id/editTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Spent time" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickSend"
        android:text="Send" />

    <|Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Read" />

    <|TextView
```

```
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="-- server messages --" />

<|/LinearLayout>
```

11.4 JSON

🕮 11.4.1

JSON

The abbreviation JSON represents the type of object originating from the javascript environment - JavaScript Object Notation.

It is currently one of the most widely used formats for data exchange.

As an example we could mention:

```
[{
   "name": "John",
   "age": "99"
 }, {
   "name": "Peter",
   "age": "18"
}]
```

Code written in JSON format contains the name of the entity and its value.

It is basically transmitted between the client and the server as plain text, it is nothing special.

The advantage of this format is that it is currently supported by practically all programming languages, thanks to which we can use ready-made encoders and decoders => we save time.

🕮 11.4.2

Examples

In the following examples, we express qualitatively the same information by different notations. In practice, it is always up to the programmers and their decision to decide how to encode the data.

List of colors in an array:

```
{
 "colorsArray":[{
    "red":"#f00",
    "green":"#0f0",
    "blue":"#00f",
    "cyan":"#0ff",
    "magenta":"#f0f",
    "yellow":"#ff0",
    "black":"#000"
  }]
}
```

Listed colors:

```
{
 "red":"#f00",
 "green":"#0f0",
 "blue":"#00f",
 "cyan":"#0ff",
 "magenta":"#f0f",
 "yellow":"#ff0",
 "black":"#000"
}
```

List of properties (name, hexadecimal code) in an array:

```
{
 "colorsArray":[{
    "colorName":"red",
    "hexValue":"#f00"
  },{
    "colorName":"green",
    "hexValue":"#0f0"
  },{
     "colorName":"blue",
   "hexValue":"#00f"
```

```
  },{
    "colorName":"cyan",
    "hexValue":"#0ff"
  },{
    "colorName":"magenta",
    "hexValue":"#f0f"
  },{
    "colorName":"yellow",
    "hexValue":"#ff0"
  },{
    "colorName":"black",
    "hexValue":"#000"
  }
 ]
}
```

🕮 11.4.3

Assignment

Modify the application so that communication takes place via data in json format.

🕮 11.4.4

Server side

We will modify the php script to send the contents of the database so that it encodes the data from the database into json format.

The easiest approach is to store data from the database in a field and use built-in functions that encode the field in json format:

```
<?php
  include 'config.php';
  $conn = mysqli_connect($servername, $username, $password,
$dbname);
  if (!$conn) { die("Connection failed: " .
mysqli_connect_error());}

  $sql = "SELECT id, name, surname, spent_time FROM
customers";
  $result = mysqli_query($conn, $sql);
```

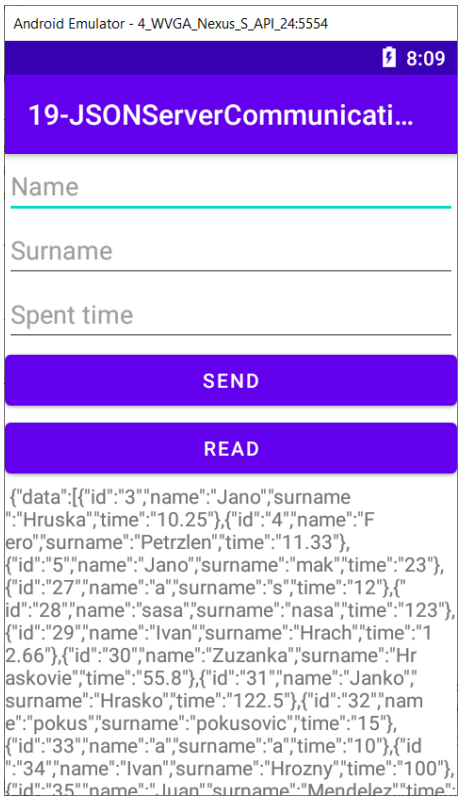
```
  $return_arr = array(); // returned array
  while ($row = mysqli_fetch_array($result)) {
    $row_array['id'] = $row['id'];
    $row_array['name'] = $row['name'];
    $row_array['surname'] = $row['surname'];
    $row_array['spent_time'] = $row['spent_time'];
    array_push($return_arr,$row_array); // adding an array to
an array
  }
    // adding the data label, coding the array and printing it
= sending
  echo json_encode(array("data"=>$return_arr));
  mysqli_close($conn);
?>
```

11.4.5

Client side

The result we get is in the form of text containing valid JSON code.

At first glance, decoding it is a bit more complicated than in the case of the original text, but if we use suitable libraries, it will be faster.



11.4.6

In addition to appropriate decoding, the resulting application should also be able to display data in some form of list.

The appropriate library for json is the basis. If it is not part of the environment, it must be obtained e.g. from json.org or through dependencies in the development environment.

11.4.7

The resulting application

The read data is displayed in a list.

It would be appropriate to adjust the communication in the opposite direction too - add items using JSON.

The advantage over "raw" data (cvs) is that when changing the data structure (e.g. new field, omitting a field), the data is not written to the fields in the wrong order - each data is identified by its key (the field to which it belongs).

🕮 11.4.8

Displaying the data

To display the data, we create a separate activity with **ListView** and load the data into it as soon as it is opened in the **onCreate()** method.

```
public class ListActivity extends AppCompatActivity {
    String myJSON; // string for the data from the server
    JSONArray people = null; // array to decode the data
      // a list where the "translated" data will be stored
    ArrayList<HashMap<String, String>> personList;
    ListView list; // list for viewing
    SimpleAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list);
        list = (ListView) findViewById(R.id.listView);
```

```
        personList = new ArrayList<HashMap<String, String>>();
        getData();
    }
```

📖 11.4.9

Connection via adapter

After the loop is executed, all stored data is in the **HashMap** list.

To connect to the display view, create a layout for the list and connect it to the data via **SimpleAdapter**.

```
protected void showList(String jsonString) {
    try {
      ...
        adapter = new SimpleAdapter(
                this,
                personList,
                R.layout.list_layout,
                new String[]{"id", "name", "surname", "time"},
                new int[]{R.id._id, R.id._name, R.id._surname,
R.id._time}
        );

        list.setAdapter(adapter);

    } catch (JSONException e) {
        Log.d("decode error", e.toString());
    }
}
```

📖 11.4.10

Discussion

We often encounter communication treatment (server response) so that in the JSON structure there is also information on whether the communication ended successfully.

e.g.:

- {"query_result": "ERROR"}
- {"query_result": "SUCCESS"}

- {"query_result": "FAILURE"}

in the form:

```
{
   "query_result": "success",
   "data":   [{
     "id": "3",
     "name": "John",
     "surname": "Doe",
     "time": "10.25"
    }, {
     "id": "4",
     "name": "John",
     "surname": "Cena",
     "time": "11.33"
    }, {
     "id": "5",
     "name": "Jack",
     "surname": "Black",
     "time": "23"
  }]
}
```

🕮 11.4.11

Data acquisition

We read the data via a standard instance of the AsyncTask class, in which case we make the child of the class part of the getData() method - such an approach is also possible.

```
public void getData() {
    class GetDataJSON extends AsyncTask<String, Void, String>
{
        @Override
        protected String doInBackground(String... params) {
            String result = "", row;

            try {
                URL url = new
URL("https://.../android/vypisjson.php");
                  // create the connection
                URLConnection conn = url.openConnection();
```

```
                conn.setDoOutput(true); // the method is post
                  // connection properties
                conn.setReadTimeout(15000);
                conn.setConnectTimeout(15000);
                conn.connect(); // connection
                InputStreamReader streamReader = new
InputStreamReader(conn.getInputStream());

                   // read the data
                BufferedReader reader = new
BufferedReader(streamReader);
                while ((row= reader.readLine()) != null) {
                    result += row;
                }
                reader.close();
                streamReader.close();
            } catch (Exception e) {
                Log.d("x", e.toString());
                result = null;
            }
            return result;
        }

        @Override
        protected void onPostExecute(String result) {
            showList(result); // decode the data and add them
to the list
        }
    }
       // code of the getData method, which creates the
instance of the AsyncTask and starts it
    GetDataJSON g = new GetDataJSON();
    g.execute();
}
```

📖 11.4.12

Data decoding

Data decoding consists of reading data from a string into a **JSONObject**. This insertion takes into account the structure of the text string defined by the **JSON** tags and allows us to read the data in the **data** item from the decoded string (see below):

```
{"data": [{
  "id": "3",
  "name": "John",
  "surname": "Doe",
  "time": "10.25"
 }, {
  "id": "4",
  "name": "John",
  "surname": "Cena",
  "time": "11.33"
 }, {
  "id": "5",
  "name": "Jack",
  "surname": "Black",
  "time": "23"
 }]
}
```

Then individual items from the **data** part - we insert them into the **people** field of the **JSONArray** type, read them via **getString()** and insert them into the **HashMap**.

```
protected void showList(String jsonString) {
    try {
          // string is added to the object
        JSONObject jsonObj = new JSONObject(jsonString);
          // object or it's parts are added to an array
        people = jsonObj.getJSONArray("data");

        for (int i = 0; i < people.length(); i++) { // from
the field we read the data as Strings
            JSONObject c = peoples.getJSONObject(i);
            String id = c.getString("id");
            String _name = c.getString("name");
            String _surname = c.getString("surname");
            String _time = c.getString("time");

              // from the Strings we create hash elements for
the adapter
            HashMap<String, String> person = new
HashMap<String, String>();
            person.put("id", id);
            person.put("name", _name);
            person.put("surname", _surname);
            person.put("time", _time);
```

```
            personList.add(person);
        }
...
```

11.4.13 Code of the application - server

output.php

```
<|?php
  include 'config.php';
  $conn = mysqli_connect($servername, $username, $password,
$dbname);
  if (!$conn) { die("Connection failed: " .
mysqli_connect_error());}

  $sql = "SELECT id, name, surname, spent_time FROM
customers";
  $result = mysqli_query($conn, $sql);

  $return_arr = array(); // resulting array
  while ($row = mysqli_fetch_array($result)) {
    $row_array['id'] = $row['id'];
    $row_array['name'] = $row['name'];
    $row_array['surname'] = $row['surname'];
    $row_array['spent_time'] = $row['spent_time'];
    array_push($return_arr,$row_array); // array in array
  }
    // adding "data" label, coding of the array, and
"printing" = sending
  echo json_encode(array("data"=>$return_arr));
  mysqli_close($conn);
?>
```

11.4.14 Code of the application - client

AndroidManifest.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.a19_jsonservercommunication">
```

```
    <|uses-permission
android:name="android.permission.INTERNET" />

    <|application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"

android:theme="@style/Theme.19JSONServerCommunication">
        <|activity android:name=".ListActivity"><|/activity>
        <|activity android:name=".MainActivity">
            <|intent-filter>
                <|action
android:name="android.intent.action.MAIN" />

                <|category
android:name="android.intent.category.LAUNCHER" />
            <|/intent-filter>
        <|/activity>
    <|/application>

<|/manifest>
```

MainActivity.java

```
package com.example.a19_jsonservercommunication;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import org.json.JSONArray;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.URL;
```

```
import java.net.URLConnection;
import java.net.URLEncoder;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View view) {
        // the URL of the website
        String myUrl = "https://.../android/outputjson.php";
        // created instance for download
        GetMyData getRequest = new GetMyData();
        // calling the method to start the AsyncTask
        // here we can use any number of String parameters
        getRequest.execute(myUrl, "1");
    }

    public void onClickSend(View view) {
        String myUrl = "https://.../android/write.php";
        String _name = ((EditText)
(findViewById(R.id.editName))).getText().toString();
        String _surname = ((EditText)
(findViewById(R.id.editSurname))).getText().toString();
        String _ttime = ((EditText)
(findViewById(R.id.editTime))).getText().toString();
        _ttime.replace(',','.'); // to have floating points
instead of commas
        // creating the instance
        GetMyData getRequest = new GetMyData();
        // sending the parameters
        getRequest.execute(myUrl, "2", _name, _surname,
_ttime);
    }

    public void onShowList(View view) {
        Intent i = new Intent(this, ListActivity.class);
        startActivity(i);
    }

    public class GetMyData extends AsyncTask<|String, Void,
String> {
```

```
        @Override
        protected String doInBackground(String... params) {
            String stringUrl = params[0]; // reading the URL
            int task_type = Integer.parseInt(params[1]); // 2.
parameter - what to do
            String result = "", row;

            try {
                // creating the URL
                URL url = new URL(stringUrl);
                // connecting to the URL
                URLConnection conn = url.openConnection();
                conn.setDoOutput(true); // setting the method
to post
                // connection parameters
                conn.setReadTimeout(15000);
conn.setConnectTimeout(15000);

                switch (task_type) {
                    case 1: // create the connection -
everything is ready, send the query
                        conn.connect();
                        break;
                    case 2: // we read the parameters from the
method
                        String first_name = params[2];
                        String last_name = params[3];
                        String time = params[4];
                         // access to the output stream of the
request, creating the parameters
                        OutputStream output =
conn.getOutputStream();
                        String data =
URLEncoder.encode("first_name", "UTF-8") + "=" +
                                URLEncoder.encode(first_name,
"UTF-8");
                        data += "&" +
URLEncoder.encode("last_name", "UTF-8") + "=" +
                                URLEncoder.encode(last_name,
"UTF-8");
                        data += "&" +
URLEncoder.encode("time", "UTF-8") + "=" +
```

```
                                URLEncoder.encode(time, "UTF-
8");
                        // writing to stream
                        output.write(data.getBytes("UTF-8"));
                        output.flush(); // sending
                        output.close();
                        break;
                }

                // creating the response stream
                InputStreamReader streamReader = new
InputStreamReader(conn.getInputStream());
                // creating the reader
                BufferedReader reader = new
BufferedReader(streamReader);
                // reading data from the stream
                while ((row = reader.readLine()) != null) {
                    result += row;
                }
                // close
                reader.close();
                streamReader.close();
            } catch (Exception e) {
                Log.d("x", e.toString());
                result = null;
            }
            return result;
        }

        protected void onPostExecute(String result) {
            super.onPostExecute(result);
            TextView tv = (TextView)
findViewById(R.id.textView);
            tv.setText(result);
            Toast.makeText(MainActivity.this, "OK",
Toast.LENGTH_SHORT).show();
        }
    }

}
```

activity_main.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">


    <|EditText
        android:id="@+id/editName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Name" />

    <|EditText
        android:id="@+id/editSurname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Surname" />

    <|EditText
        android:id="@+id/editTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Spent time" />

    <|Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClickSend"
        android:text="Send" />

    <|Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Read" />

    <|Button
```

```
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onShowList"
        android:text="ShowList" />

    <|TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="-- server messages --" />

<|/LinearLayout>
```

ListActivity.java

```
package com.example.a19_jsonservercommunication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.HashMap;

public class ListActivity extends AppCompatActivity {
    String myJSON; // string for the data from the server
    JSONArray people = null; // array for decoding the data
    // list where the "translated" data will be stored
    ArrayList<|HashMap<|String, String>> personList;
    ListView list; // list for viewing
    SimpleAdapter adapter;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list);
        list = (ListView) findViewById(R.id.listView);
        personList = new ArrayList<|HashMap<|String,
String>>();
        getData();
    }

    public void getData() {
        class GetDataJSON extends AsyncTask<|String, Void,
String> {
            @Override
            protected String doInBackground(String... params)
{
                String result = "", row;

                try {
                    URL url = new
URL("https://.../android/outputjson.php");
                    // creating a connection
                    URLConnection conn = url.openConnection();
                    conn.setDoOutput(true); // the method is
post
                    // connection properties
                    conn.setReadTimeout(15000);
                    conn.setConnectTimeout(15000);
                    conn.connect(); // connecting...
                    InputStreamReader streamReader = new
InputStreamReader(conn.getInputStream());

                    // reading data
                    BufferedReader reader = new
BufferedReader(streamReader);
                    while ((row = reader.readLine()) != null)
{
                        result += row;
                    }
                    reader.close();
                    streamReader.close();
                } catch (Exception e) {
                    Log.d("x", e.toString());
                    result = null;
```

```
                }
                return result;
            }

            @Override
            protected void onPostExecute(String result) {
                showList(result); // we translate the data to
a list
            }

        }
             // code of the getData method, which creates an
instance of the AsyncTask and starts it
        GetDataJSON g = new GetDataJSON();
        g.execute();
    }

    protected void showList(String jsonString) {
        try {
            // string vlozime do objektu
            JSONObject jsonObj = new JSONObject(jsonString);
            // the object, or part of it is sent to an array
            people = jsonObj.getJSONArray("data");

            for (int i = 0; i <| people.length(); i++) { //
from the array, the data is read as Strings
                JSONObject c = people.getJSONObject(i);
                String id = c.getString("id");
                String _name = c.getString("name");
                String _surname = c.getString("surname");
                String _time = c.getString("time");

                // from the Strings, we create Hash elements
for the adapter
                HashMap<|String, String> person = new
HashMap<|String, String>();
                person.put("id", id);
                person.put("name", _name);
                person.put("surname", _surname);
                person.put("time", _time);

                personList.add(person);
            }
            adapter = new SimpleAdapter(
```

```
                    this,
                    personList,
                    R.layout.list_layout,
                    new String[]{"id", "name", "surname",
"time"},
                    new int[]{R.id._id, R.id._name,
R.id._surname, R.id._time}
            );

            list.setAdapter(adapter);

        } catch (JSONException e) {
            Log.d("decode error", e.toString());
        }
    }
}
```

activity_list.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".ListActivity">


    <|ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

<|/LinearLayout>
```

list_layout.xml

```
<|?xml version="1.0" encoding="utf-8"?>
<|LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
android:layout_width="match_parent"
    android:layout_height="match_parent">

    <|TextView
```

```
        android:id="@+id/_id"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />

    <|TextView
        android:id="@+id/_name"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />

    <|TextView
        android:id="@+id/_surname"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />

    <|TextView
        android:id="@+id/_time"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />
<|/LinearLayout>
```

PRISCILLA

priscilla.fitped.eu