Guideline for your first Android App

Hammer Clemens MSc
hammer@technikum-wien.at

University of Applied Sciences Technikum Wien
Department of Embedded Systems
April 1, 2017

Contents

1 First Android Application

The following chapters show you how to create your first Android App. The Application, which will be built is called *HealthQuiz*.

With the App HealthQuiz, the knowledge of the human body will be proven. This will be done by simply *True* and *False* questions.
As you can see in figure 1.1 our first application will look like the following.

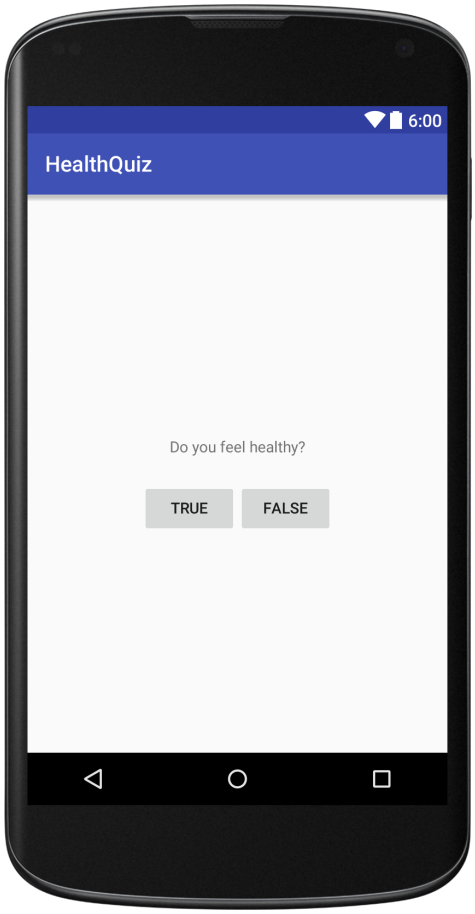

Figure 1.1: First application prototype

1.1 Create an Android Project

To create an Android Application the first step is to open *Android Studio*. From the dialog choose $File \rightarrow NewProject$.
Inside the open dialog you can define the application name and the company domain. Notice that the company domain will be used as preamble of your packages and classes. Therefore we define the company domain as:

<LAST_NAME>.bsa2.fhtw

For the application name we choose:

HealthQuiz

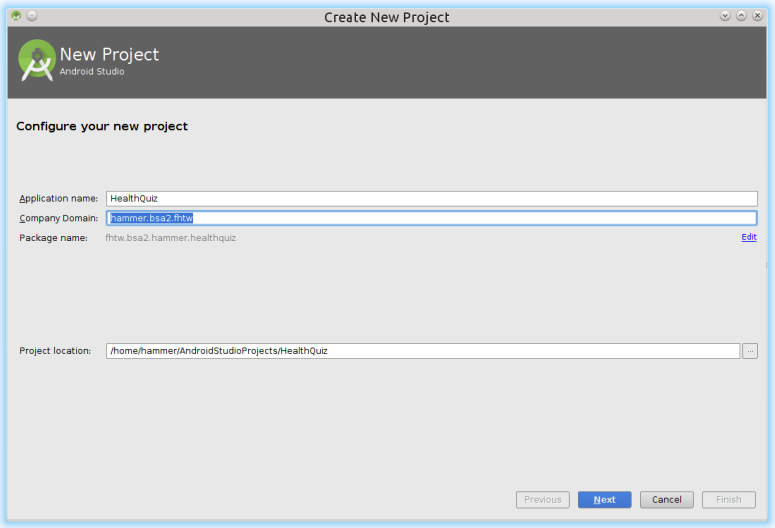

Figure 1.2: Create Project 1

Afterwards click *Next* to switch to the next widget where you can specify, which devices you will support and further, which minimal OS version your application requires.

In our case we want to develop an app for tablets and mobile phones and further work with an SDK version of *API 21: Android 5.0*.

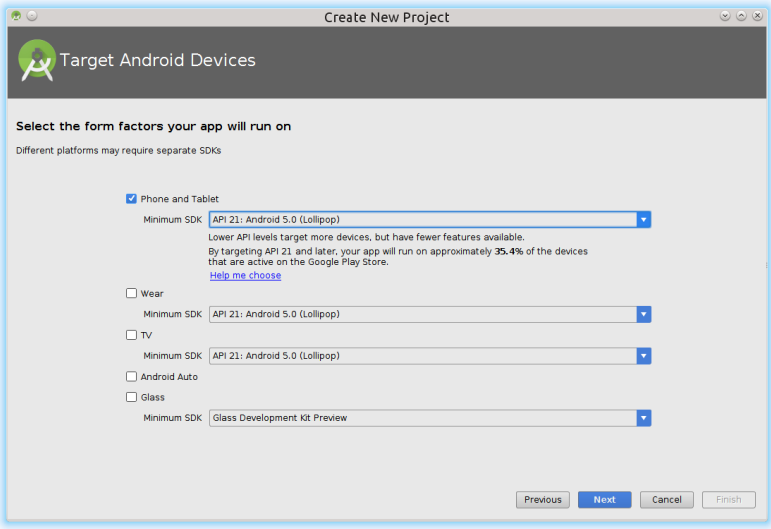

Figure 1.3: Specify device categorie and SDK version

If all configurations are done, click *Next*. On the next widget we are choosing a template for the application. For our first app we simply choose an *Empty Activity*. Afterwards click *Next*. As *Activity Name* we choose:

QuizActivity

The *Layout Name* will be automatically updated.

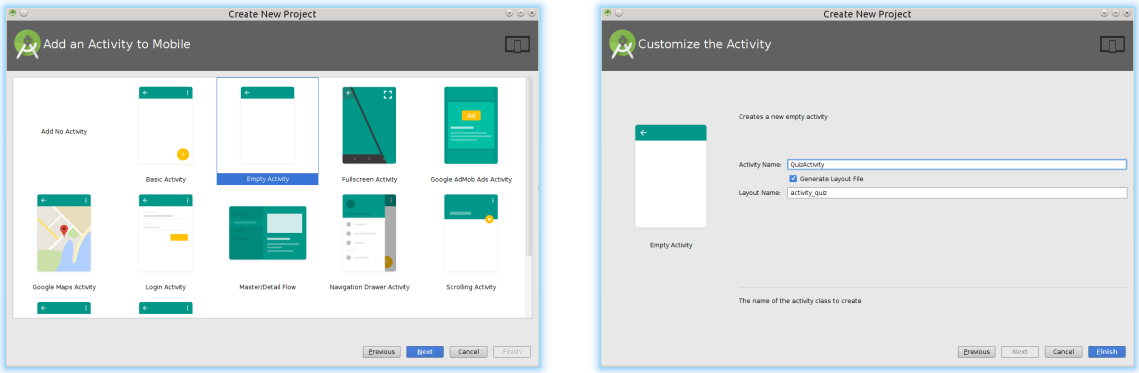

Figure 1.4: App template

After these steps we have done all configurations, which are basically necessary to implement an Android App.

1.2 Layout

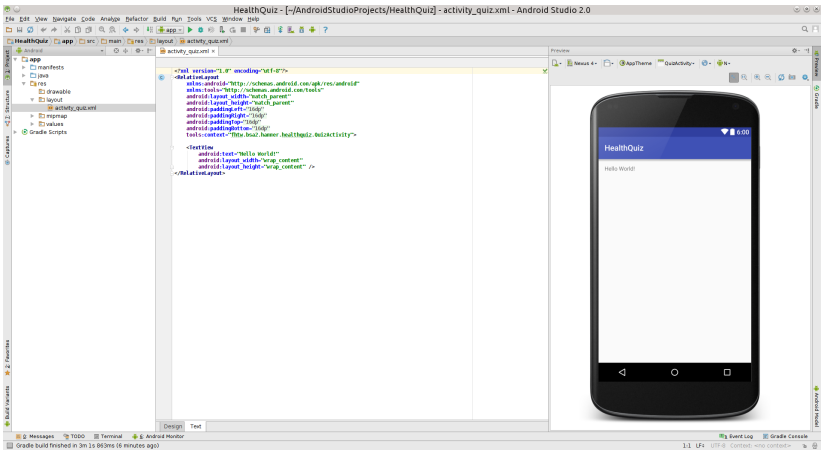

Figure 1.5: App prototype

As you can see in figure 1.5 a *Hello World* application is automatically built by the Android Studio. The current layout is defined in the file *activity_quiz.xml*.
To create an App, which looks like our predefined App from the beginning (see figure 1.1), we have to change the current layout.
This application includes five different types of widgets:

- vertical LinearLayout
- Textview
- horizontal LinearLayout
- Button

To change the current layout to our layout, we modify the file *activity_quiz.xml* to the following:

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

    <TextView
        android:text="@string/question_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp" />
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
```

```
        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />
        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />
    </LinearLayout>
</LinearLayout>
```

The Android Studio will show you possible errors at the XML tags *android:text*. This occurs, because we are using the attribute *@string*. This attributes are stored in an additional file, named *strings.xml*.
This file can be found inside the directory *app/res/values*. Inside this file we will store all strings, which are needed for our application. This additional file has the advantage that you can easily change the displayed text for the whole application.
Therefore open this file and add the following XML tags:

```
<resources>
    <string name="app_name">HealthQuiz</string>
    <string name="question_text">Do you feel healthy?</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

Now you should see an application preview like the predefined layout of our application. If so

CONGRATULATION

1.3 Implement functionality

During the creation of this project the IDE implements not only the folders, which contains the information about the layout further the IDE implements also the corresponding class file (QuizActivity) inside the *app/java* directory.

Resource
A resource is the equivalent of the layout, it contains all these parts or items, which are not a **code**, like string files, XML files and such like these.
To access these recourses from the code, we use the corresponding resource IDs. As you can see in section Layout we defined for each button a recourse ID. Over these IDs we can later connect our code with the widgets.

Code implementation
Now we change to the *QuizActivity.java* file to implement our own functionality. You should see the minimal code, which is basically needed to build an app.
We extend this code to the following:

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }
}
```

After this step we have created our objects for the two buttons and link them to the view over their recourse ID.
Now we want to implement our functionality to interact with the button. Therefore we have to implement the so called Listener for both buttons. Inside the inner class of the Listener we have to implement the abstract method *public void onClick(View v)*.
This method is called everytime when the user presses the button on the screen. Additionally, it is common that every item has it's own Listener.

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

    mTrueButton = (Button) findViewById(R.id.true_button);
    mFalseButton = (Button) findViewById(R.id.false_button);

    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });

    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });
}
```

Pop-up message

To create pop-up messages or so called *toast* we have to extend our *strings.xml* file with the following content.

```
<string name="correct_toast">Correct!</string>
<string name="incorrect_toast">Incorrect!</string>
```

Now we have to implement the functionality to call these toast messages. Therefore we have to extend the *OnClick* methods, so that these messages are shown if a button is pressed. The function must be extended as the following (respectively with incorrect):

```
public void onClick(View v) {
        Toast.makeText(QuizActivity.this, R.string.correct_toast,
                Toast.LENGTH_SHORT).show();
}
```

2 Run on the Emulator

To test your application you can use the so called emulator. Via this tool it is possible to run the app on different devices or software versions. The app itself can be emulated on a simulated or a real device.
The following steps explain you how to deploy your app on a simulated device. Therefore click on the **green play** symbol to run your application.
If a device is previously configured, you can choose the device where you want to deploy the app. If no device is configured execute the steps from the paragraph **Create a new device**.

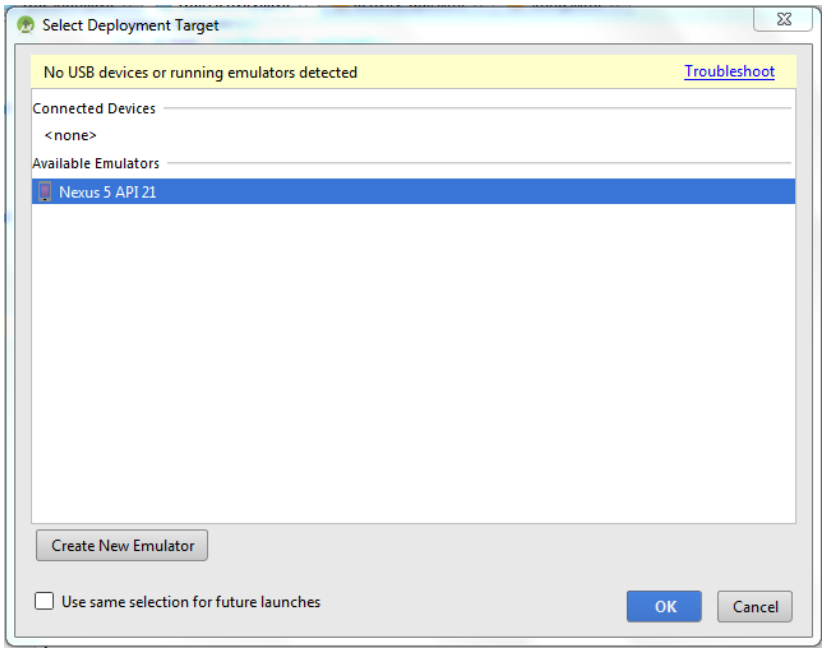

Figure 2.1: Choose device

Create a new device
If a new device should be configured, click first on the button *Create New Emulator*. Afterwards you should see the following screen: **Create a new device**.

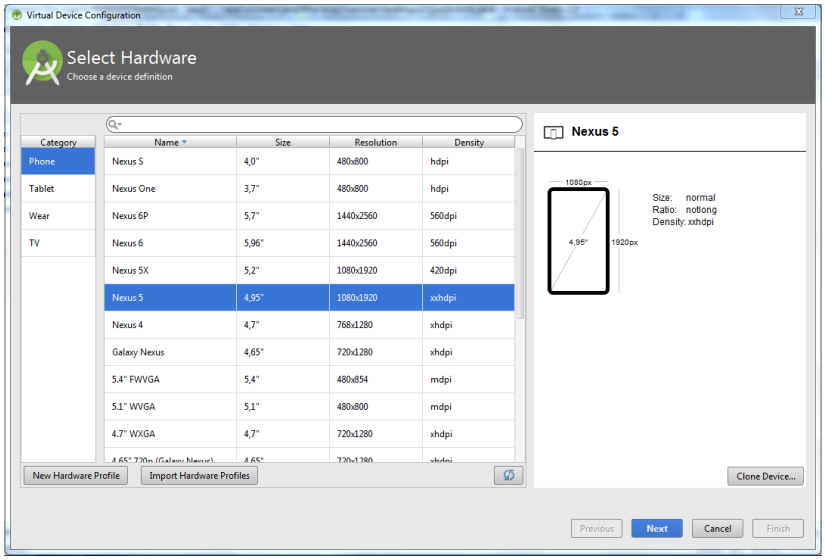

Figure 2.2: Choose Hardware

In the new opened widget you can choose the hardware model, where your application should run. Afterwards click *Next*. The last step is to choose the OS version, which should be installed on your device. If no OS was previously downloaded, just click download on the OS version you want. Afterwards click *Next*, check the summary and click *Finish*..

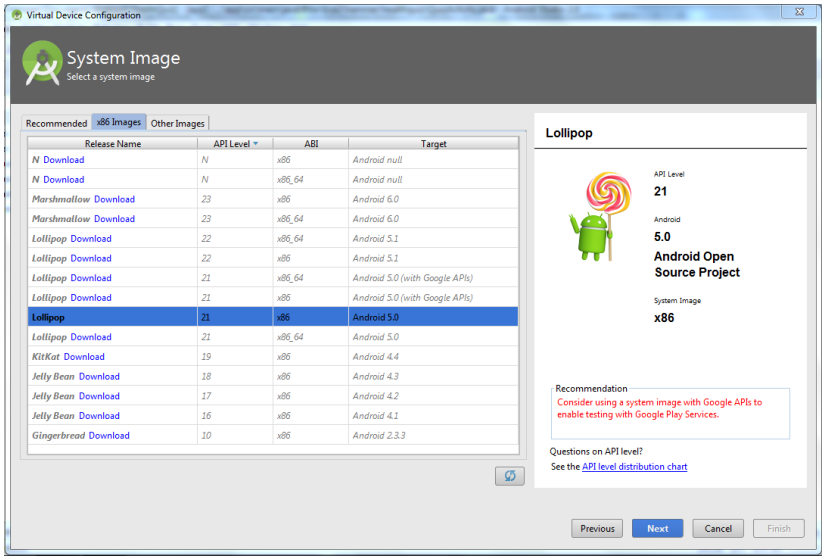

Figure 2.3: Choose OS version

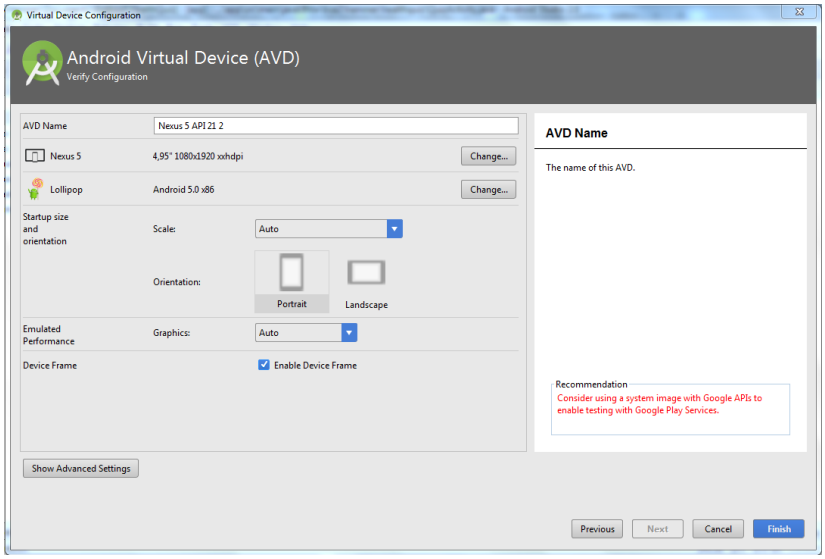
Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name
Nexus 5 API 21 2
Nexus 5
4,95" 1080x1920 xxhdpi
Change...
Lollipop
Android 5.0 x86
Change...
Startup size and orientation
Scale:
Auto
Orientation:
Portrait
Landscape
Emulated Performance
Graphics:
Auto
Device Frame
Enable Device Frame
Show Advanced Settings
AVD Name
The name of this AVD.
Recommendation
Consider using a system image with Google APIs to enable testing with Google Play Services.
Previous
Next
Cancel
Finish

Figure 2.4

3 First app extension

This chapter shows you how to extend the first app. In detail this means, that the application should implement the functionality to show more than only one question.
Therefore we have to redesign our first layout with an additional button (see figure 3.1). This implementation of the new button can be done via the *activity_quiz.xml* or respectively this can be done by the drag and drop functionally of the IDE.
Additionally we want to reuse the previous defined Textview. Therefore we have to assign an ID to the Textview. This could be done via the *activity_quiz.xml* file or via the graphical user interface designer. The ID itself should be assigt to:

question_text_view

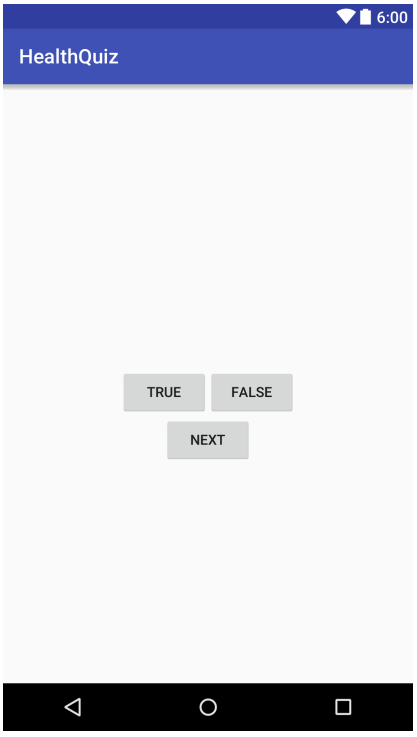

Figure 3.1: Layout 2.0

To process more than one function, we have further to create a new class called **Question**. This class contains all parameters, which are necessary to answer a question. This means in detail the class contains a private int variable with the question ID and a boolean with the answer (see listing Question.java).

Listing 3.1: Question.java

```
package fhtw.bsa2.hammer.healthquiz;

public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;

    public Question(int textResId, boolean answerTrue) {
        mTextResId = textResId;
        mAnswerTrue = answerTrue;
    }

    public int getTextResId() {
        return mTextResId;
    }

    public void setTextResId(int textResId) {
        this.mTextResId = textResId;
    }

    public boolean isAnswerTrue() {
        return mAnswerTrue;
    }

    public void setAnswerTrue(boolean answerTrue) {
        this.mAnswerTrue = answerTrue;
    }
}
```

The Questions themselves are stored inside the *string.xml* file, so that they can easily be accessed and respectively modified. To add these questions, the *string.xml* file has to be extended with the following content:

```
<string name="question1">Question1</string>
                        .
                        .
                        .
<string name="question10">Question10</string>
```

To link our new button to the source code and to implement the new functionality, the file *QuizActivity.java* has to be modified.
Therefore we have to declare the new button, the textview, the questions and some auxiliary variables.

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;

    private Question[] mQuestionBank = new Question[]{
            new Question(R.string.question1, true),
            new Question(R.string.question2, false),
            new Question(R.string.question3, true),
            new Question(R.string.question4, false),
            new Question(R.string.question5, true),
            new Question(R.string.question6, false),
            new Question(R.string.question7, true),
            new Question(R.string.question8, false),
            new Question(R.string.question9, true),
            new Question(R.string.question10, false)
    };

    private int mCurrentIndex = 0;
                                        .
                                        .
                                        .
```

At the end of the file we add some additional functions to process the questions. The first method will display the current question onto the screen, depending on the current index. The second method checks if the defined answer (True or False) equals the Button pressed (Button True or False).

```
private void updateQuestion() {
  mQuestionTextView.setText(mQuestionBank[mCurrentIndex].getTextResId());
}

private void checkAnswer(boolean userPressedTrue) {
  boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
  int messageId = 0;

  if(userPressedTrue == answerIsTrue) {
    messageId = R.string.correct_toast;
  } else {
    messageId = R.string.incorrect_toast;
  }
  Toast.makeText(QuizActivity.this,messageId,Toast.LENGTH_SHORT).show();
}
```

The last step is to connect the Textview and the new created button to our source code. Therefore we have to initialize the corresponding class members and further modify the Listeners.

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
        mNextButton = (Button) findViewById(R.id.next_button);
        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);

        updateQuestion();

        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                checkAnswer(true);
            }
        });

        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                checkAnswer(false);
            }
        });

        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
               updateQuestion();
            }
        });
```

4 Logging

Logging is one of the easiest function mechanism to debug during runtime. Via logging you can print information about your applications at nearly any point you want. To print the information you can use different levels of logging functions.

Log Level	Method	Notes
ERROR	Log.e(...)	Errors
WARNING	Log.w(...)	Warnings
INFO	Log.i(...)	Information messages
DEBUG	Log.d(...)	Debug output; may be filtered out
VERBOSE	Log.v(...)	For development only!

To use these methods it is recommended to define a *TAG* in each class, where you want to use the logging functionality.
This *TAG* can be used later on to filter the information according to a specific class.

Example usage

```
private static final String TAG ="CLASS_NAME";
        .
        .
        .
Log.d(TAG, "METHODE_NAME");
```

5 Landscape

By default the design is only defined for the *Portrait* perspective. This design is also mapped to the *Landscape* perceptive. This means if the device is rotated by $90°$, the app will use the same layout design.
Hereby it could happen that the design is displaced. To prevent this a separate layout can be defined for the *Landscape* mode.
Therefore click on *File* $\rightarrow$ *New* $\rightarrow$ *Android resource directory*.
Afterwards choose *Orientation* and click on the right arrows in the middle.

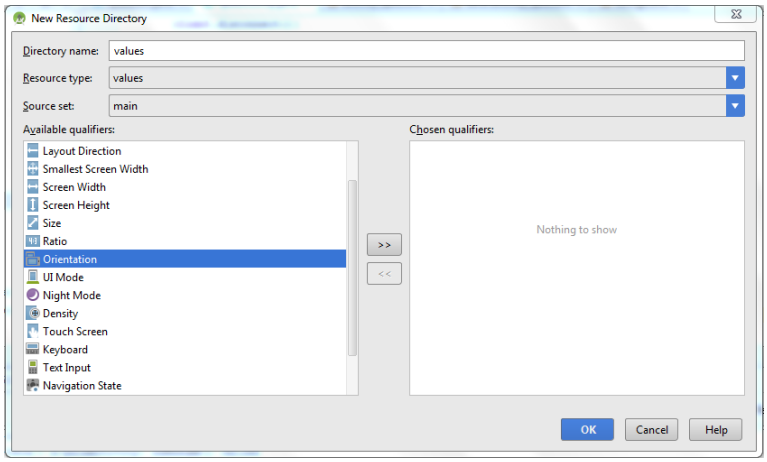

Figure 5.1: Create landscape 1

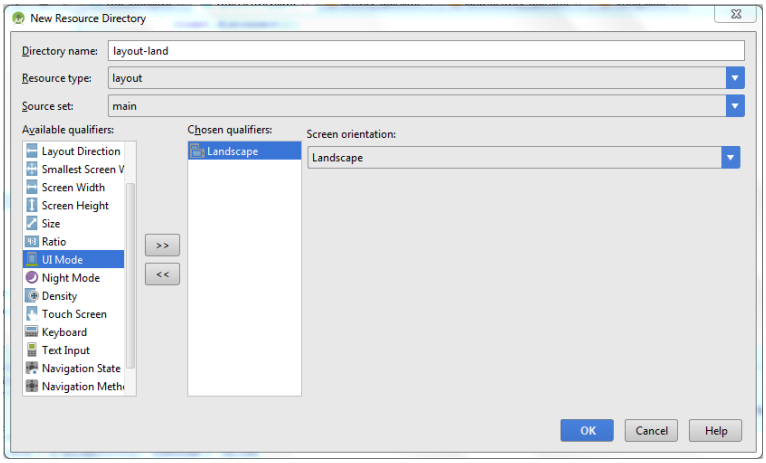

Figure 5.2: Create landscape 2

To create the layout of the landscape mode, just simply copy the XML file from the portrait mode. To do that, we have to switch the perspective from Android to Project. Afterwards copy the file *activity_quiz.xml* to the folder *layout-land*.

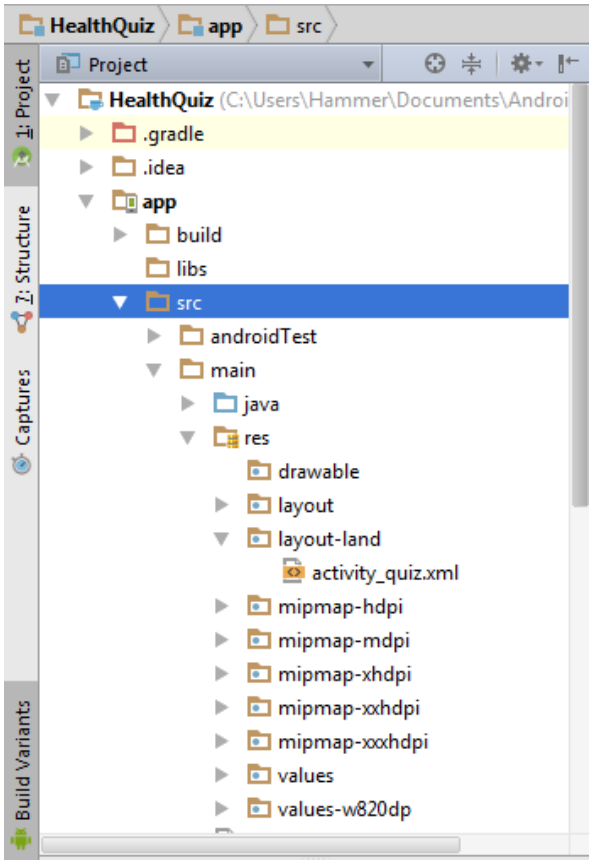

Figure 5.3: Change perspective

6 Saving content

If you add additional logging statements to the *QuizActivity.java* file, you can see that the application will be restarted every time when the app is rotated. To see that during execution, add the following lines to the source code:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
        .
        .
}
@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart() called");
    .
    .
}
@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop() called");
    .
    .
}
@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}
@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume() called");
}
@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}
```

We can not prevent that our application will restart, so we have to store the current important information on the device. Further we have to reload this information every time the application is created.

The important information in our first application is simply the current index of our question pool. Information are typically stored as a pair of the value and a key. Therefore we define an additional string in our class *QuizActivity.java*.

```
private static final String KEY_INDEX = "index";

@Override
protected void onCreate(Bundle savedInstanceState) {
    .
    .
    mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
    if(savedInstanceState != null) {
        mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }
    updateQuestion();
    .
    .
}
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    saveInstanceState.putInt(KEY_INDEX, mCurrentIndex)
}
```

Inside the method *onCreate()* the last state is restored. If no last state is available, the method will return the default value. In this example it returns 0.

To save the content of a variable, the method *onSaveInstanceState()* has to be overwritten. Inside this method we call the method itself from the super class and additionally store our user specific information. In our application is this information the index.

7 Add a second activity

In this chapter we add a second activity to our previous developed application. The functionality behind this second activity implements a cheat mode. This means we have to extend our previous layout with a cheat button. This button will activate our second activity.

Inside the second activity the user should simply be asked if he/she really wants to cheat. If the user says **yes** the solution gets displayed.

Creating a new activity

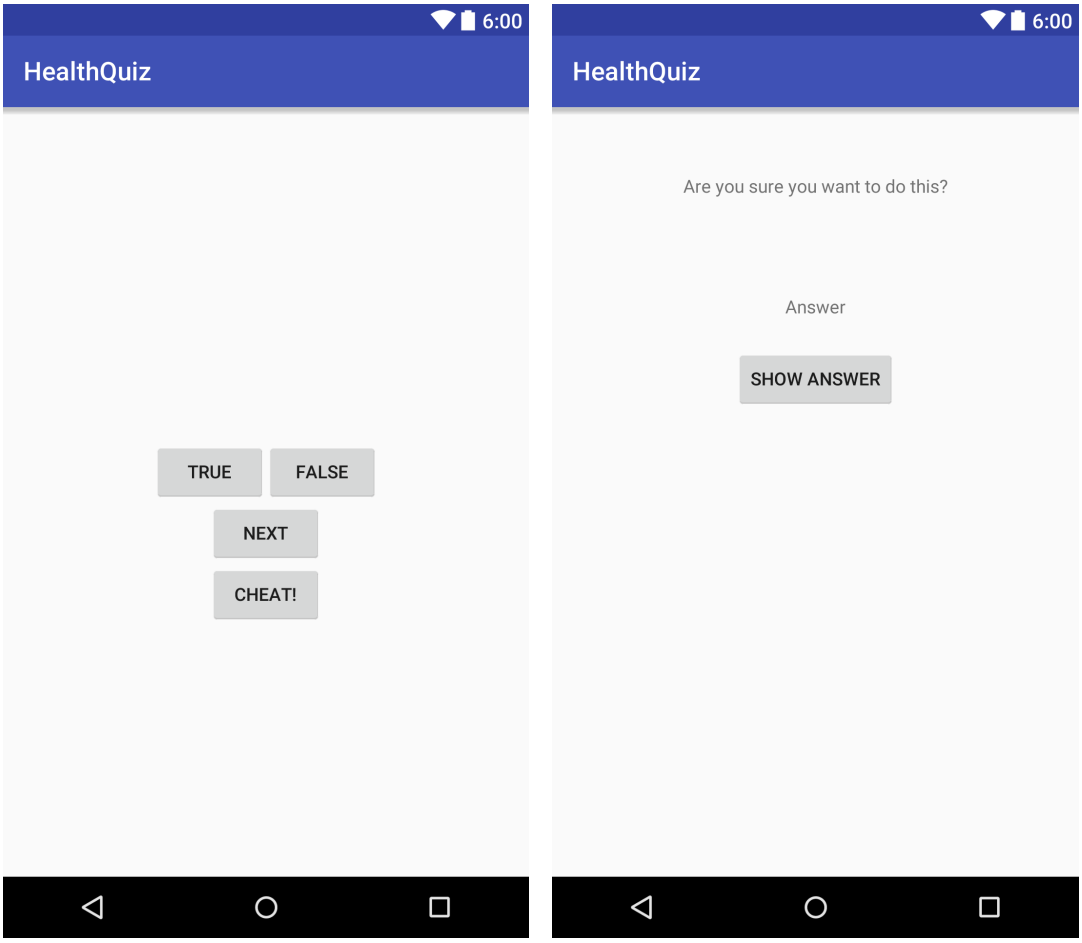

Figure 7.1: Layout with second activity

To create a new activity, right click on *app* → *New* → *Activity* → *Empty Activity* (see figure 7.2).

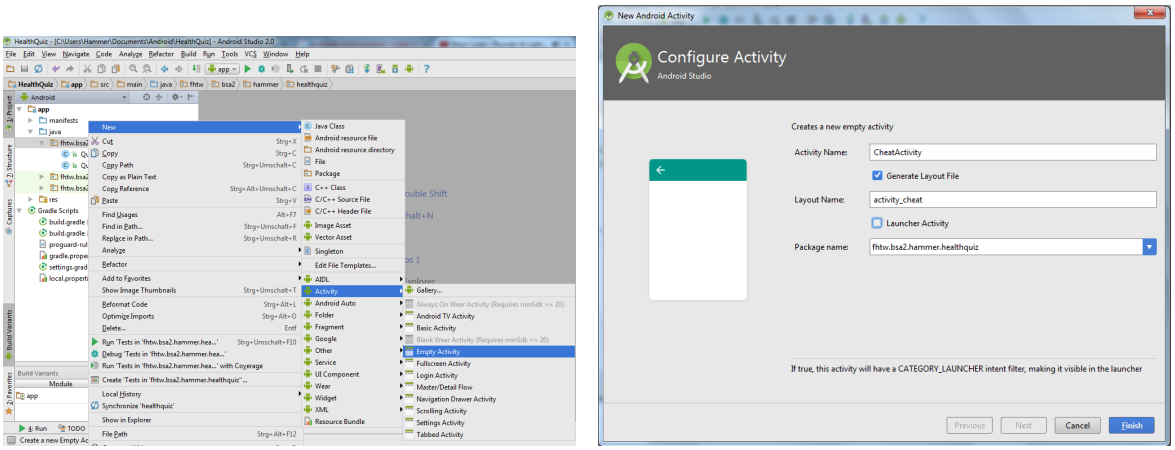

Figure 7.2: Create new activity

Now you can see a new Java Class called *CheatActivity.java* and a new XML file (*activity_cheat.xml*). To change the default layout to our new one, repeat the steps we did before. Respectively you can use the drag and drop functionality from the IDE and change all necessary parameters there.

string.xml

```
<string name="warning_text">Are you sure you want to do this?</string>
<string name="show_answer_button">Show Answer</string>
<string name="cheat_button">Cheat!</string>
<string name="judgment_toast">Cheating is wrong!</string>
<string name="title_activity_cheat">Cheat</string>
```

activity_cheat.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:orientation="vertical"
    tools:context="fhtw.bsa2.hammer.healthquiz.CheatActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/warning_text"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:padding="24pt" />

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Answer"
        android:id="@+id/textView2"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button"
        android:id="@+id/show_answer_button"
        android:layout_gravity="center_horizontal" />
</LinearLayout>
```

Declaring activity

Every activity in an application must be declared on the manifest, so that the OS can access it. The manifest is always stored inside the *app/manifests* directory. To check if the IDE added the new activity, open the *AndroidManifest.xml* file and compare the manifest with the following. If the activity is not declared, some exceptions could occur, if the application wants to start the new unknown activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="fhtw.bsa2.hammer.healthquiz">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".QuizActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

        <activity android:name=".CheatActivity">
        </activity>
    </application>
</manifest>
```

Extend previous App
Now we have to extend the previous app, so that we can trigger the cheat mode. Therefore we add the cheat button to both layouts of the *QuizActivity* (portrait and landscape). Additionally, we are wiring the buttons from the layouts to our source code (*QuizActivity.java*) and implement the corresponding listener, like we did it before.

Starting second activity
To start the second activity when the user presses the *Cheat* button, we have to add the following lines to our *onClick()* method of the cheat button. This two method calls are simply generating a new Intent and afterwards starts them.

```
Intent i = new Intent(QuizActivity.this, CheatActivity.class);
startActivity(i);
```

Parsing data between activities
If you want to pass information from one activity to another, you have to add this functionality manually. In our case we want to send the answer of the current question to the new activity, every time the new activity gets started.
This data is always sent as a so called extra. An extra is structured as a pair of a key and a value (compare saveInstance).

To send the answer from the *QuizActivity* to the *CheatActivity* we must extend the functionality of our *CheatActivity* class.
Inside the *CheatActivity* class, we add a static method, which generates a new Intent. This method basically includes the same functionality as shown in *Starting second activity*, but additionally it puts an Extra to the Intent. Furthermore this Extra, which has to be a pair of a key and a value, is build out of the constant string (key) and a variable from type boolean (value).

Afterwards the "normal" functionality can be implemented for the new Activity. Basically we need to wire up the textView and the button.
Additionally we have to read back the passed answer (Extra) and store this information locally. Inside the listener of the button *mShowAnswer*, we want to display the answer on the screen.

CheatActivity.java

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "fhtw.bsa2.hammer.healthquiz.answer_is_true";

    private boolean mAnswerIsTrue;
    private TextView mAnswerTextView;
    private Button mShowAnswer;

    public static Intent newIntent(Context packageContext,
        boolean answerIsTrue) {
```

```
        Intent i = new Intent(packageContext, CheatActivity.class);
        i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
        return i;
    }

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

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
                false);

        mAnswerTextView = (TextView) findViewById(R.id.answer_text_view);
        mShowAnswer = (Button) findViewById(R.id.show_answer_button);
        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
            }
        });
    }
}
```

QuizActivity.java

```
        .
        .
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
//      Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
        Intent i = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
        startActivity(i);
    }
});
        .
        .
```

Inside the *QuizActivity* class we have to modify the start procedure of a new Intent. Therefore we evaluate the current answer of the question and pass them to the previously implemented method *newIntent()*.

8 Challenge

Extend the previous developed application, that it is possible to see if the user is cheating or not. Therefore modify the two class files to exchange the information in both directions (Intent).
Further display the response message as a Toast onto the screen.