Mobile Application Development

Multiple Activities

In this Lecture, you will learn:

- ➢ Creating new activity and layout
- ➢ Start an activity from another activity
- ➢ Intents
- ➢ Pass data between activities
- ➢ Returning result from an activity

Creating new activities:

Creating a new activity typically involves three files: the Java class file, the XML layout file, and the application manifest file. Creating new activity by yourself can cause some errors, so you can use Android Studio’s New Activity wizard to do this work for you.

To create a new activity, launch the New Activity wizard by right-clicking on your application package in the project window.
Choose New → Activity → Empty Activity, as shown in Figure 1.

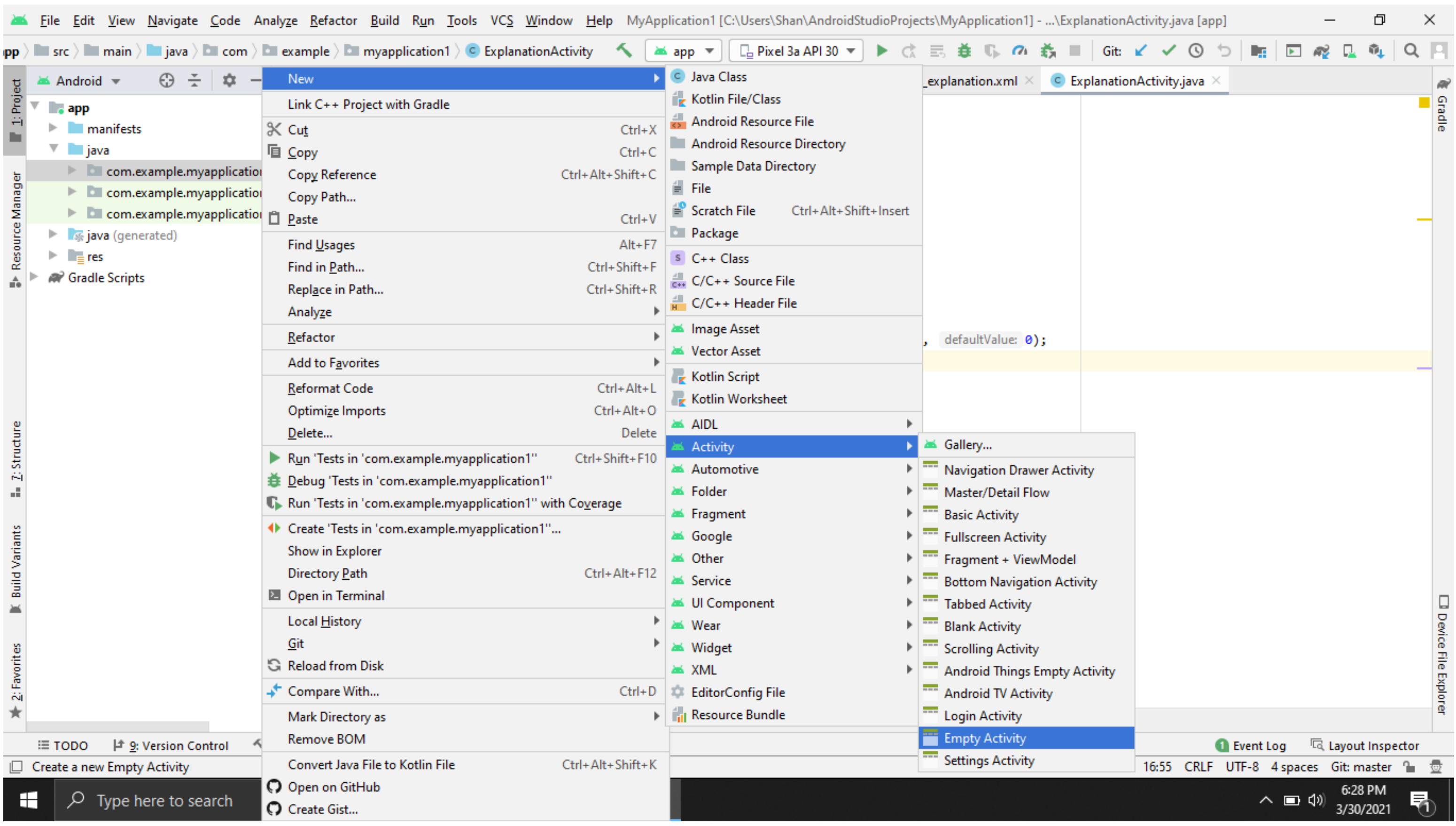
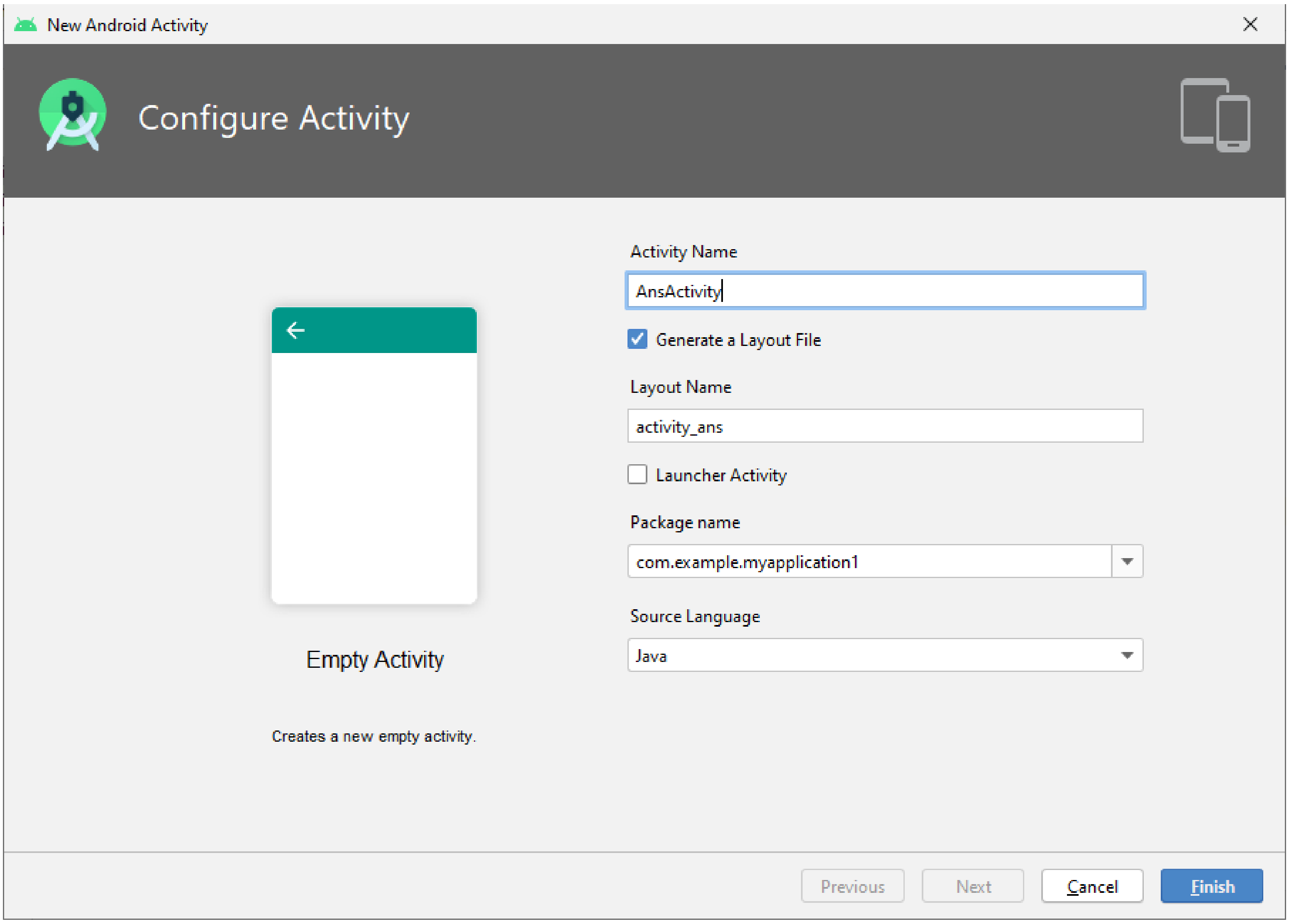


Figure 1: New activity wizard

You should see a dialog like Figure 2. Set Activity Name to AnsActivity. Default Layout Name will be set to activity_ans.


New Android Activity

Configure Activity

Activity Name: AnsActivity

[✓] Generate a Layout File

Layout Name: activity_ans

[ ] Launcher Activity

Package name: com.example.myapplication1

Source Language: Java

Empty Activity

Creates a new empty activity.

Figure 2: New Empty Activity wizard

The default values for the remaining fields are fine, just make sure that the package name is what you expect. Click the Finish button automatically create the new files and register the new activity in the AndroidManifest.xml file.
Now open and update the XML layout file for the new activity. Add a TextView for the answer and a button to return to previous activity.

A new activity subclass

Open the AnsActivity.java file to view AnsActivity class. This class already includes a basic implementation of onCreate(Bundle) that passes the resource ID of the layout defined in activity_ans.xml to setContentView(…). The New Activity wizard also declared AnsActivity in the application’s Manifest file.

Declaring activities in the manifest

The manifest is an XML file containing metadata that describes your application to the Android OS. The file is named AndroidManifest.xml, and is located in the app/manifests directory of project.

In the project tool window open AndroidManifest.xml. You can also use *Android Studio’s Quick Open dialog by pressing Ctrl+Shift+N and starting to type the filename.*

Every activity in an application must be declared in the manifest so that the OS can access it. The entry in the Manifest file should look like below:

<activity android:name=".AnsActivity">
</activity>

The android:name attribute is required. The dot at the start of this attribute’s value tells the OS that this activity’s class is in the package specified in the package attribute in the manifest element at top of the file.

Starting new Activity

public void startActivity(Intent intent)
The startActivity(Intent) is simplest way to start an activity. When an activity calls startActivity(Intent), it is sent to a part of the OS called the ActivityManager. The ActivityManager then creates the Activity instance and calls its onCreate(Bundle) method.

The ActivityManager starts a particular activity which is mentioned in the Intent parameter.

Intents

An intent is an object that a component can use to communicate with the OS. The components apart from activities include services, broadcast receivers, and content providers.

Intents are multipurpose communication tools. Intent class provides different constructors depending on what you are using the intent to do.

Here, the intent tells the ActivityManager which activity to start, so you will use this constructor:
public Intent(Context packageContext, Class<?> cls)

The Class argument specifies the activity class that the ActivityManager should start. The Context argument tells the ActivityManager which application package the activity class can be found in.

In mAnsButton's listener, create an Intent that includes the AnsActivity class. Pass this intent to startActivity(Intent) (Listing 1).

```
mAnsButton = (Button)findViewById(R.id.ans_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this,
                AnsActivity.class);
                startActivity(intent);
        }
});
```

Listing 1 Starting AnsActivity (MainActivity.java)

ActivityManager checks the package's manifest for a declaration with the same name as the specified Class. If it finds a declaration, it starts the activity. If it does not, you get a ActivityNotFoundException, which will crash your app.
Run the application and press the button to open new activity.

Explicit and implicit intents

Creating an Intent with a Context and a Class object, creates an explicit intent. You use explicit intents to start activities within your application.

When an activity in one application starts an activity in another application, it is an implicit intent.

Passing Data Between Activities

The MainActivity will inform the AnsActivity of the answer to the current question when the AnsActivity is started. When user presses the Back button to return to the MainActivity, the AnsActivity will be destroyed. It will send data to the MainActivity about whether the user cheated.

Using intent extras

To inform the MainActivity of the answer to the current question, you will pass it the value of mQuestionBank[mCurrentIndex].isAnswerTrue()

You will send this value as an extra on the Intent that is passed into startActivity(Intent). Extras are arbitrary data that the calling activity can include with an intent. The OS forwards the intent to the recipient activity, which can then access the extras and retrieve the data

An extra is structured as a key-value pair:
public Intent putExtra(String name, boolean value)
Intent.putExtra(…) has many overloads, but it always has two arguments. The first argument is always a String key, and the second argument is the value. It returns the Intent itself, so you can chain multiple if needed.

```
mAnsButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this,
                AnsActivity.class);
                boolean answerIsTrue =
                mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent intent = AnsActivity.newIntent(MainActivity.this,
                answerIsTrue);
                startActivity(intent);
        }
});
```
Listing 2: Launching AnsActivity with an extra (MainActivity.java)

To retrieve the value from the extra, use:
public boolean getBooleanExtra(String name, boolean defaultValue)

The first argument is the name of the extra. The second argument of getBooleanExtra(…) is a default answer if the key is not found.
In AnsActivity, retrieve the value from the extra in onCreate(Bundle) and store it in a member variable.

```
public class AnsActivity extends AppCompatActivity
{
        private static final String EXTRA_ANSWER_IS_TRUE =
        " answer_is_true";
        private boolean mAnswerIsTrue;
        ...
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_cheat);
                mAnswerIsTrue =
                getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
                false);
        }
        ...
}
```
Listing 3: Using an extra (AnsActivity.java)

Activity.getIntent() always returns the Intent that started the activity. This is what you sent when calling startActivity(Intent).

Finally, wire up the answer TextView and the SHOW ANSWER button to use the retrieved value.

```
public class AnsActivity extends AppCompatActivity
{
    ...
    private boolean mAnswerIsTrue;
    private TextView mAnswerTextView;
    private Button mShowAnswerButton;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);
        mAnswerIsTrue =
        getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
        false);
        mAnswerTextView = findViewById(R.id.answer_text_view);
        mShowAnswerButton                                    =
        findViewById(R.id.show_answer_button);
        mShowAnswerButton.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);
            }
        }
        });
    }
}
```

Listing 4: Enabling show answer (AnsActivity.java)

Getting a result back from a child activity

The AnsActivity can tell the MainActivity whether the user chose to view the answer.
To get result back from the child activity, call the following method:
public void startActivityForResult(Intent intent, int requestCode)

The first parameter is the same intent. The second parameter is the request code, which is a user-defined integer that is sent to the child activity and then received back by the parent. It is used when an activity starts more than one type of child activity and needs to know who is reporting back.

In MainActivity, modify mAnsButton's listener to call startActivityForResult(Intent, int).

```
public class MainActivity extends AppCompatActivity {
        private static final String TAG = "MainActivity";
        private static final String KEY_INDEX = "index";
        private static final int REQUEST_CODE_CHEAT = 0;
        ...
        @Override
        protected void onCreate(Bundle savedInstanceState) {
        ...
        mAnsButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                boolean answerIsTrue =
                mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent intent = AnsActivity.newIntent(MainActivity.this,
                answerIsTrue);
                startActivity(intent);
                startActivityForResult(intent, REQUEST_CODE_CHEAT);
        }
});
```

Listing 5 Calling startActivityForResult(…) (MainActivity.java)

Setting a result

There are two overloads to send data back to the parent activity:
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)

Typically, the result code is one of two predefined constants: Activity.RESULT_OK or Activity.RESULT_CANCELED.
You can use another constant, RESULT_FIRST_USER, when defining your own result codes.

Setting result codes is useful when the parent needs to take different action depending on how the child activity finished. For example, if a child activity had an OK button and a Cancel button, the child activity would set a different result code depending on which button was pressed. The parent activity would take a different action depending on the result code.

If setResult(…) is not called and the user presses the Back button, the parent will receive Activity.RESULT_CANCELED.

Sending back an intent

To pass data back to MainActivity, create an Intent, put an extra on it, and then call Activity.setResult(int, Intent) to get that data into MainActivity's. In AnsActivity, add a constant for the extra's key and a private method that does this work. Call this method in the SHOW ANSWER button's listener.

```
public class AnsActivity extends AppCompatActivity
{
        private static final String EXTRA_ANSWER_IS_TRUE
        = "answer_is_true";
        private static final String EXTRA_ANSWER_SHOWN =
        "answer_shown";
        ...
        @Override
        protected void onCreate(Bundle savedInstanceState) {
        ...
        mShowAnswerButton.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);
                }
                setAnswerShownResult(true);
        }
        });
        }
        private void setAnswerShownResult(Boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN,isAnswerShown);
        setResult(RESULT_OK, data);
        }
}
```

Listing 6: Setting a result (AnsActivity.java)

When the user presses the SHOW ANSWER button, the AnsActivity calls setResult(int, Intent).
When the user presses the Back button to return to the MainActivity, the ActivityManager calls the following method on the parent activity:
protected void onActivityResult(int requestCode, int resultCode, Intent data)

Handling a result

In MainActivity.java, add a new member variable to hold the value that AnsActivity is passing back. Override onActivityResult(…) to retrieve it, checking the request code and result code to be sure they are what you expect.

```
public class MainActivity extends AppCompatActivity {
...
private int mCurrentIndex = 0;
private boolean mIsCheater;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
...
}
@Override
protected void onActivityResult(int requestCode,
int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
                return;
        }
        if (requestCode == REQUEST_CODE_CHEAT) {
                if (data == null) {
                        return;
                }
                mIsCheater = AnsActivity.wasAnswerShown(data);
        }
}
...
}
```

Listing 7 Implementing onActivityResult(…) (MainActivity.java)

Modify the checkAnswer(boolean) method in MainActivity to check whether the user cheated and to respond appropriately.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
...
mNextButton =(Button)findViewById(R.id.next_button);
mNextButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1) %
        mQuestionBank.length;
        mIsCheater = false;
        updateQuestion();
}
});
...
} ...
private void checkAnswer(boolean userPressedTrue) {
boolean answerIsTrue =
mQuestionBank[mCurrentIndex].isAnswerTrue();
int messageResId = 0;
if (mIsCheater) {
        messageResId = R.string.judgment_toast;
} else {
        if (userPressedTrue == answerIsTrue) {
                messageResId = R.string.correct_toast;
        } else {
                messageResId = R.string.incorrect_toast;
        }
}
Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
.show();
}
```

Listing 8: Changing toast message (MainActivity.java)

Run the app to observe the output.

Android OS and activities

The OS starts the application’s launcher activity. The MainActivity is the launcher activity in QuizApp.

When the New Project wizard created the QuizApp application and MainActivity, it made MainActivity the launcher activity by default. Launcher activity s specified in the manifest by the intentfilter element in MainActivity’s declaration (Listing 9).

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
... >
<application
... >
<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=".AnsActivity">
</activity>
</application>
</manifest>
```

Listing 5.18 MainActivity declared as launcher activity (AndroidManifest.xml)

Opening AnsActivity creates its instance on top of the MainActivity. These activities exist in a stack.

Pressing the Back button in AnsActivity pops this instance off the stack, and the MainActivity resumes its position at the top.

Calling Activity.finish() in AnsActivity would also pop the AnsActivity off the stack.

Press Back from the MainActivity pops off the stack and you will return to the last screen you were viewing before running QuizApp.

ActivityManager maintains a back stack and that this back stack is not just for your application's activities. Activities for all applications share the back stack, which is one reason the ActivityManager is involved in starting your activities and lives with the OS and not your application. The stack represents the use of the OS and device as a whole rather than the use of a single application.