



Hugbúnaðarverkefni 2 / Software Project 2

7. Android User Interfaces

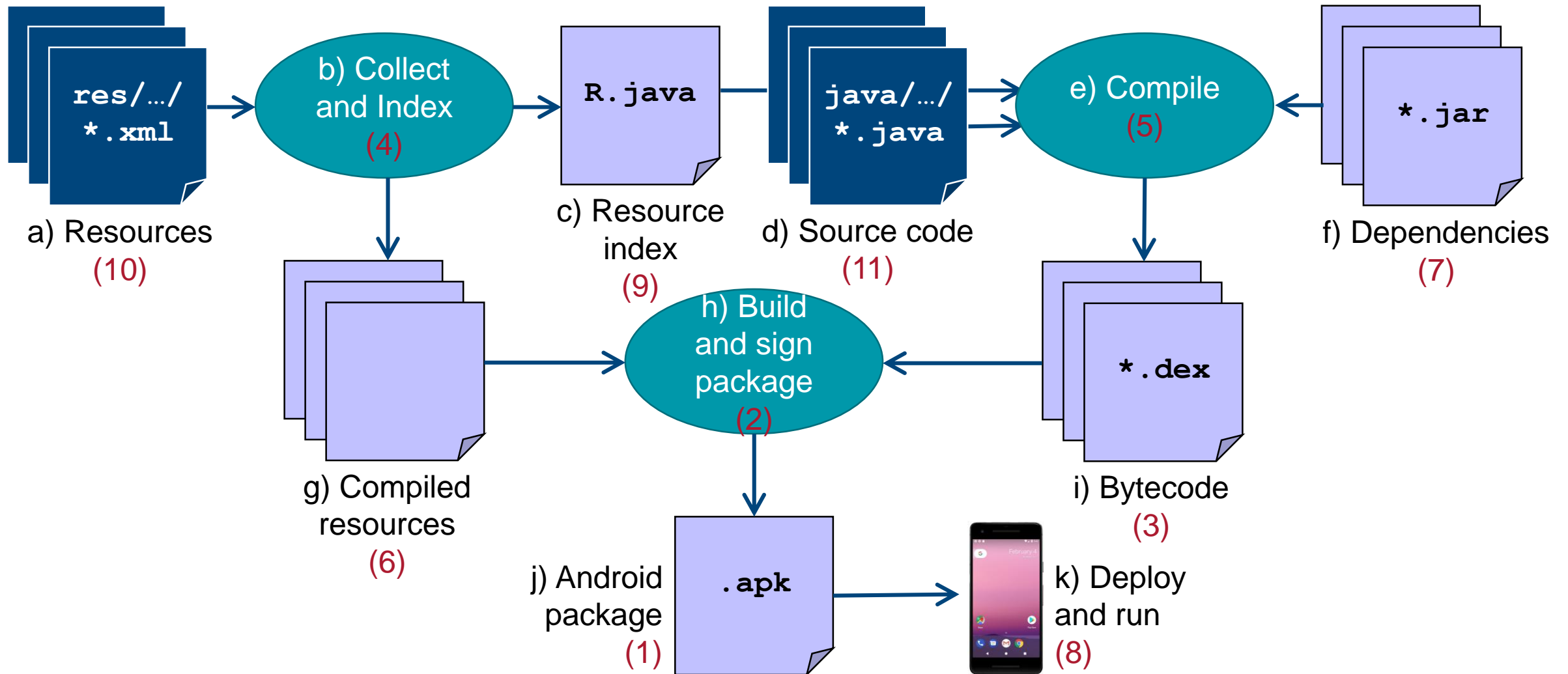
HBV601G – Spring 2019

Matthias Book



HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz #5 Solution



In-Class Quiz 6 Prep

- Please prepare a small scrap of paper with the following information:

ID: _____@hi.is Date: _____

a) _____ d) _____
b) _____ e) _____
c) _____ f) _____

- During class, I'll show you questions that you can answer with a number
- Hand in your scrap at end of class
- All questions in a quiz have same weight
- All quizzes (8-10 throughout semester) have the same weight
 - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam



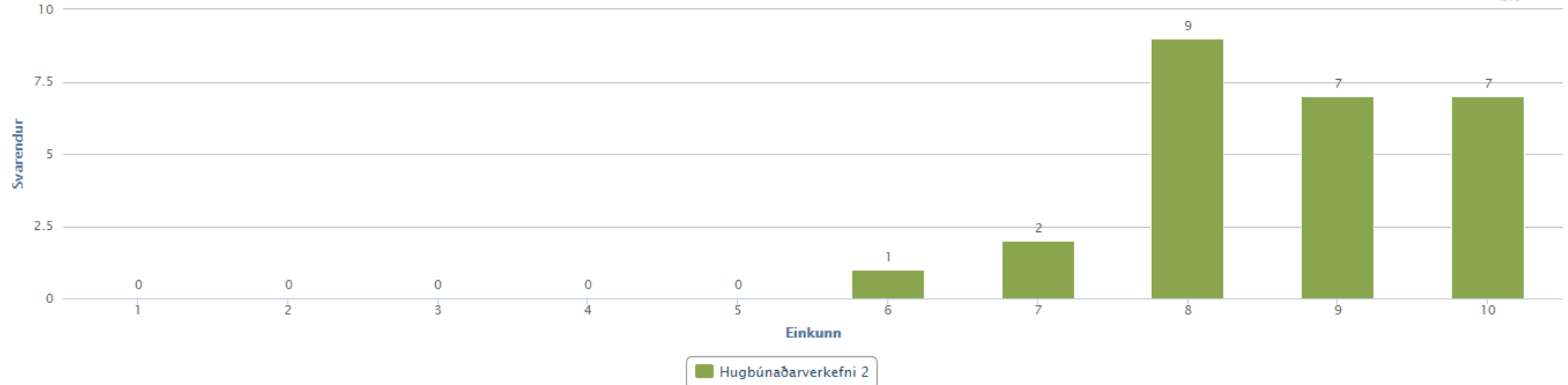
Miðmisseriskönnun

Evaluation Results



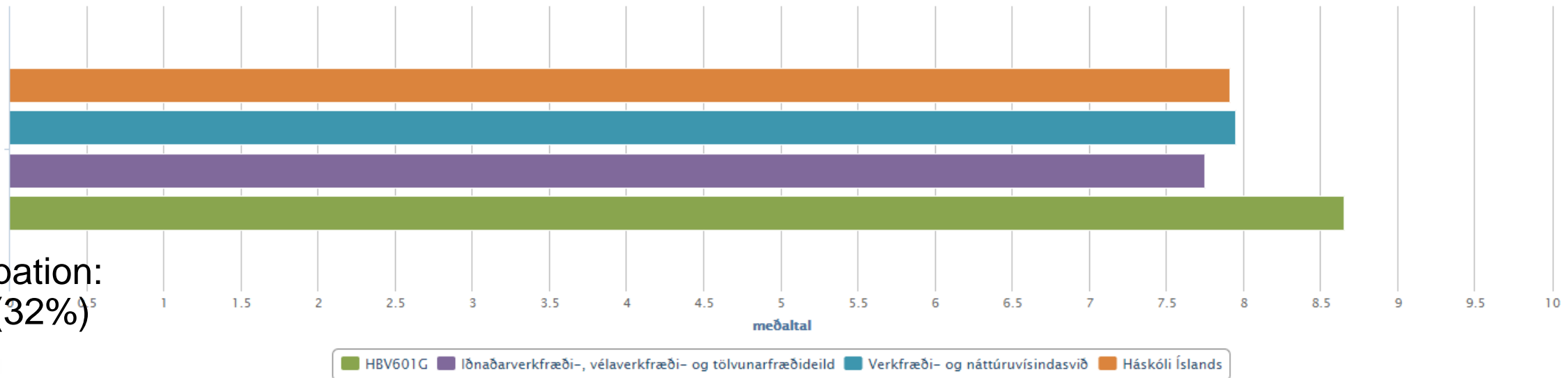
Grades

Gefðu námskeiðinu einkunn:
Dreifing einkunnna



Takk fyrir!

Gefðu námskeiðinu einkunn:
Samanburður



Participation:
26/82 (32%)



HÁSKÓLI ÍSLANDS

Matthias Book: Software Project 2

Feedback (paraphrased)

What people like

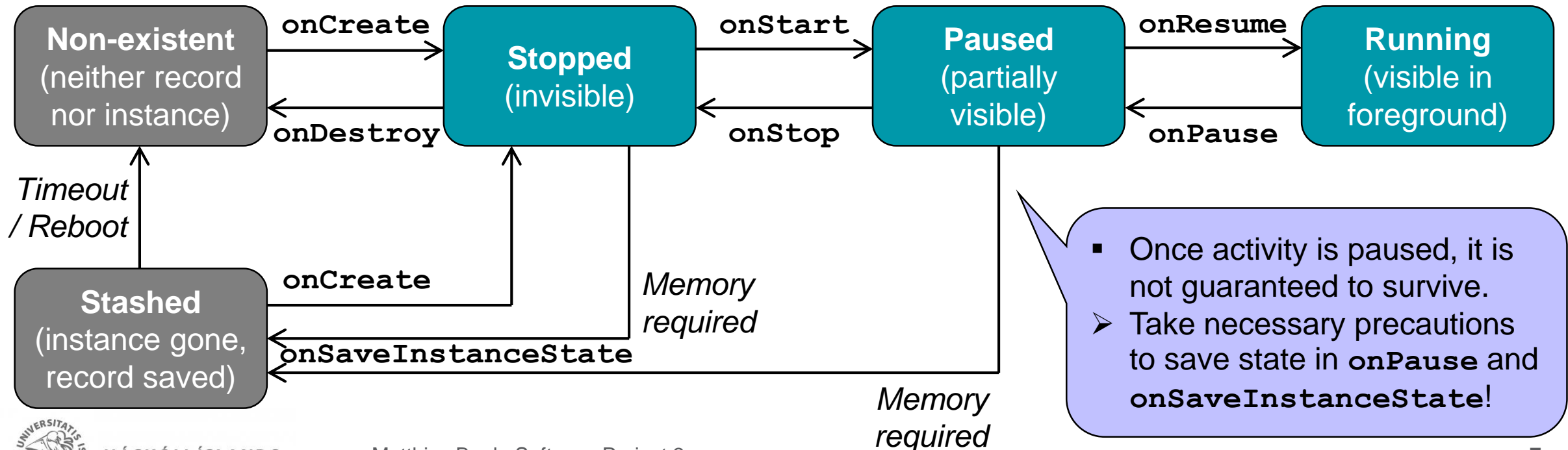
- Informative slides
- Project freedom
- Good teaching

What people would like

- Have necessary knowledge for assignments earlier
 - Will try to teach Android earlier next time
- Documents shouldn't be 80% of the grade while the product is just 20%
 - Your project progress is reflected in your documents, so the project effort is reflected in the grade
- Better coordination between tutors
 - Please ask (on Piazza, in class, by e-mail, etc.) anytime something is unclear

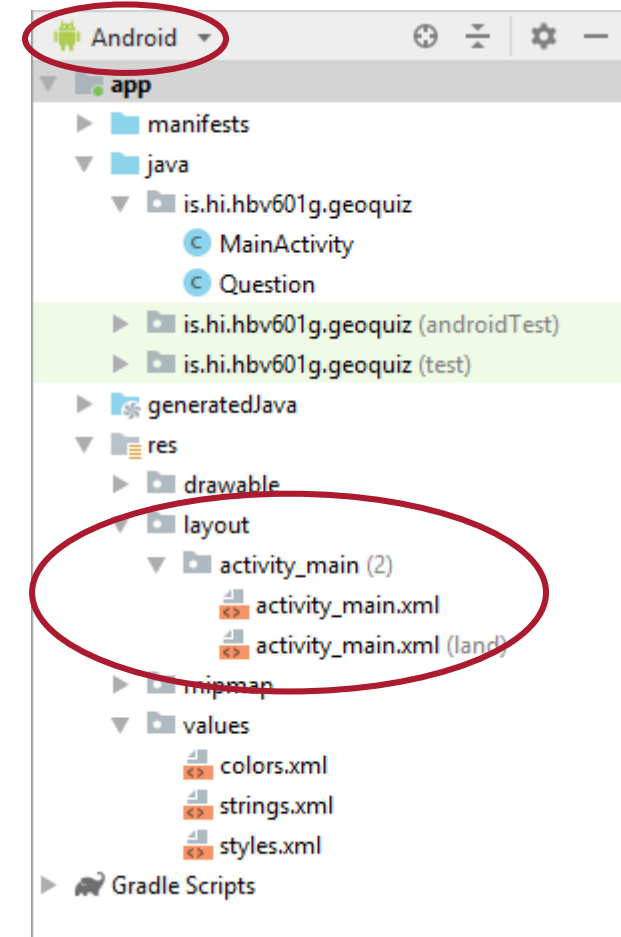
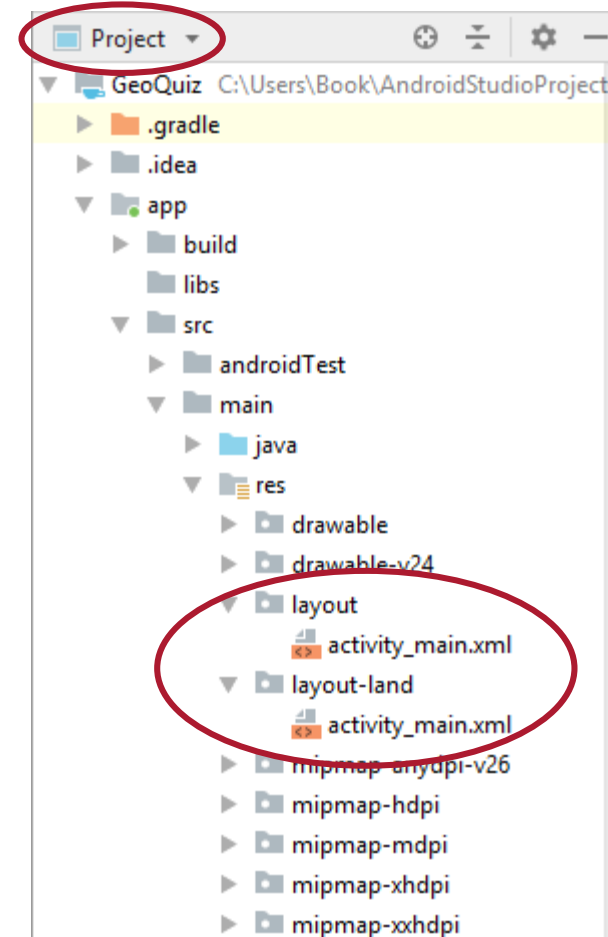
Recap: Storing Activity State in Activity Records

- Android provides a way to save and retrieve activity state at time of transitions:
 - `onSaveInstanceState(Bundle)` saves current instance state in an **activity record**
 - `onCreate(Bundle)` reconstitutes the activity's state from the saved activity record
- An activity record is kept even if the activity instance is removed from memory
 - Can be used to recreate an activity in its previous state



Recap: Configuration Qualifiers

- Portrait layout used so far was stored in **res/layout**
- Android expects landscape layout in **res/layout-land**
- **-land** suffix is a **configuration qualifier** helping Android to find resources that best match a given device configuration
 - Many more folder types and qualifiers possible, e.g. different image resolutions for different screen densities
 - <http://developer.android.com/guide/topics/resources/providing-resources.html>



Communicating With Intents

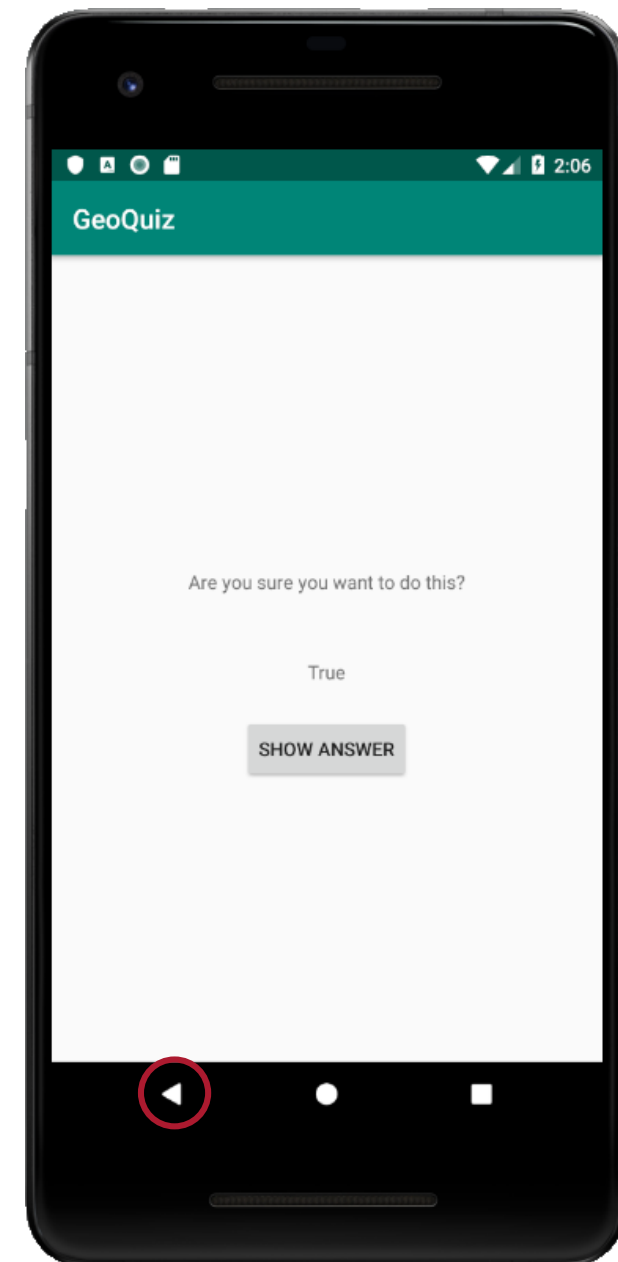
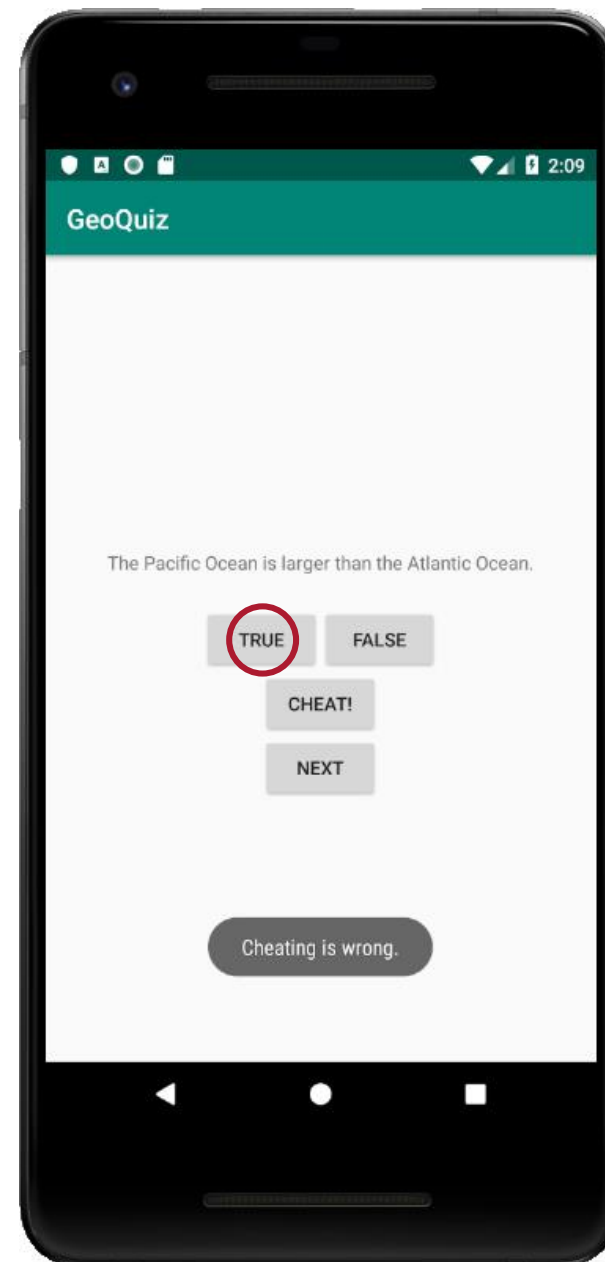
see also:

- Phillips et al.: Android Development, Ch. 5
- <http://developer.android.com/training/basics/firstapp/starting-activity.html>
- <http://developer.android.com/guide/components/intents-filters.html>



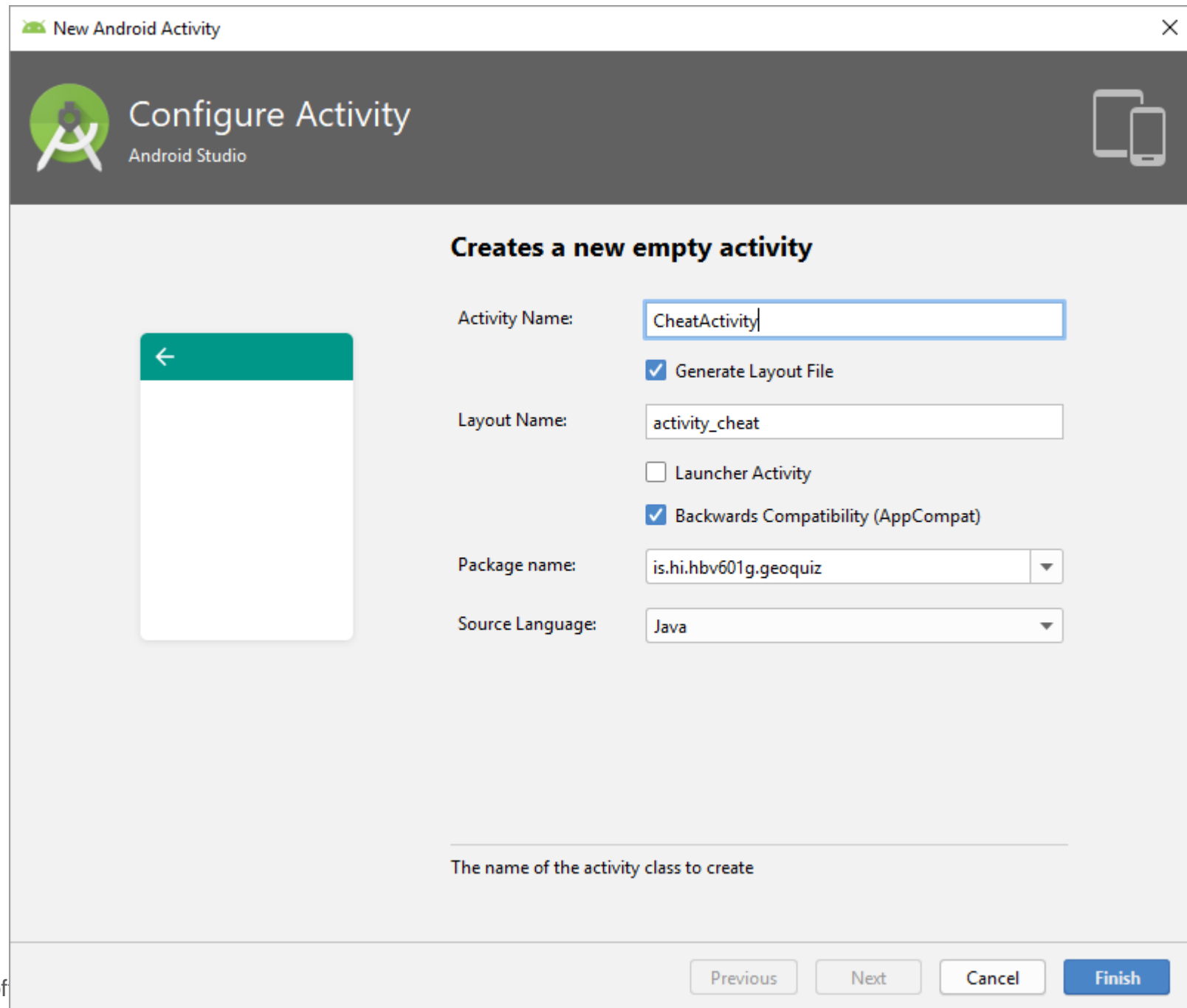
Example Scenario

- On each question screen, we want to give the user the option to “cheat” by looking up the right answer on another screen, before returning to the question screen.
- That means we need to
 - invoke another screen (i.e. activity)
 - and pass information to it;
 - then return to the previous activity
 - and return information to it.
- This is accomplished with **intents**.



Creating a New Activity

- Choose “File > New > Activity > Empty Activity”
- Provide a name for the activity
 - here: CheatActivity for the activity governing the cheat screen’s behavior
- Layout name will be set automatically, following naming convention
 - ...and layout file will be created together with activity file



New Android Activity

Configure Activity
Android Studio

Creates a new empty activity

Activity Name: CheatActivity

☒ Generate Layout File

Layout Name: activity_cheat

☐ Launcher Activity

☒ Backwards Compatibility (AppCompat)

Package name: is.hi.hbv601g.geoquiz

Source Language: Java

The name of the activity class to create

Previous Next Cancel Finish

Explicit Intents

- An activity cannot directly invoke another activity by calling its methods.
- Rather, an activity communicates with the Android Operating System (OS) by exchanging **intents**.
- To invoke another activity within the same app, we use an **explicit intent**:

```
Intent i = new Intent(MainActivity.this, CheatActivity.class);  
i.putExtra("is.hi.hbv601g.geoquiz.answer_is_true", answerIsTrue);  
startActivity(i);
```

Context we are calling from

Activity to invoke

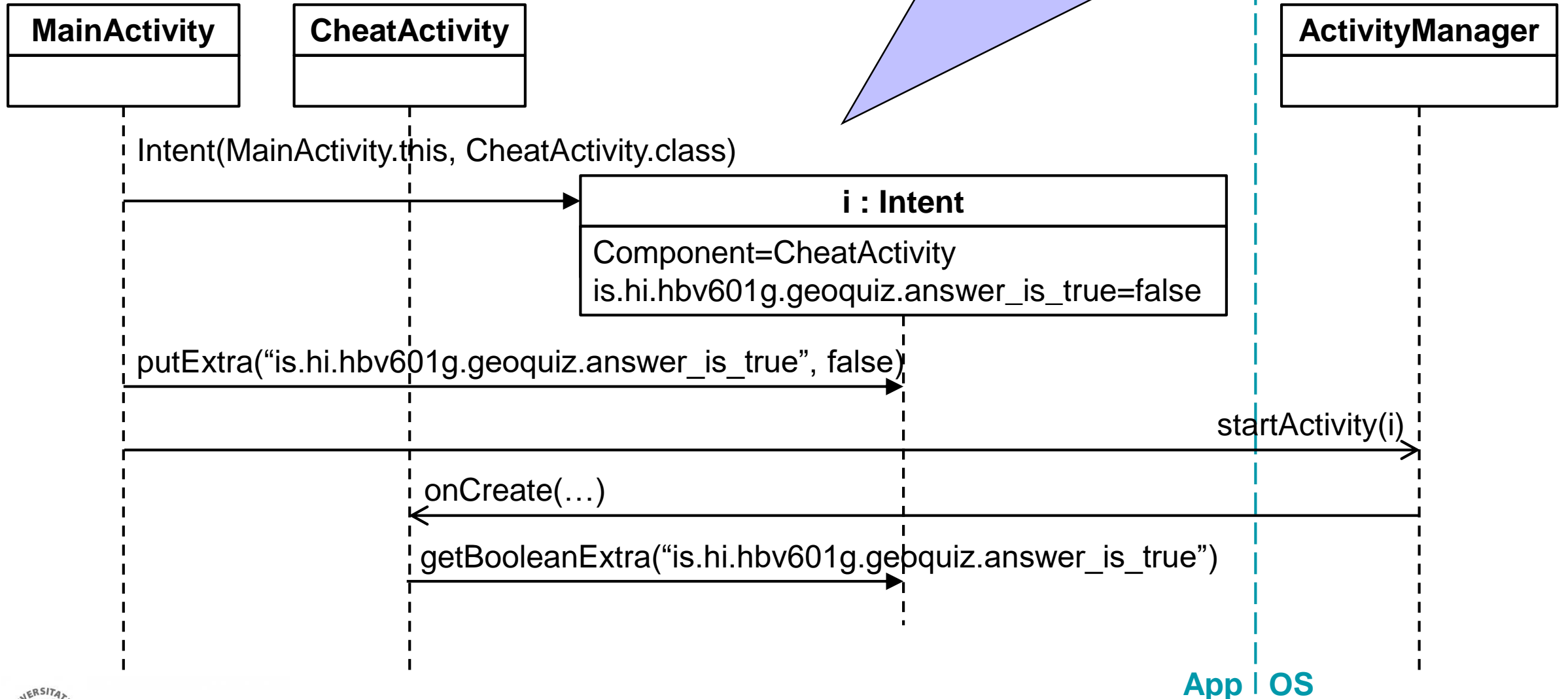
Extra's key

Extra's value

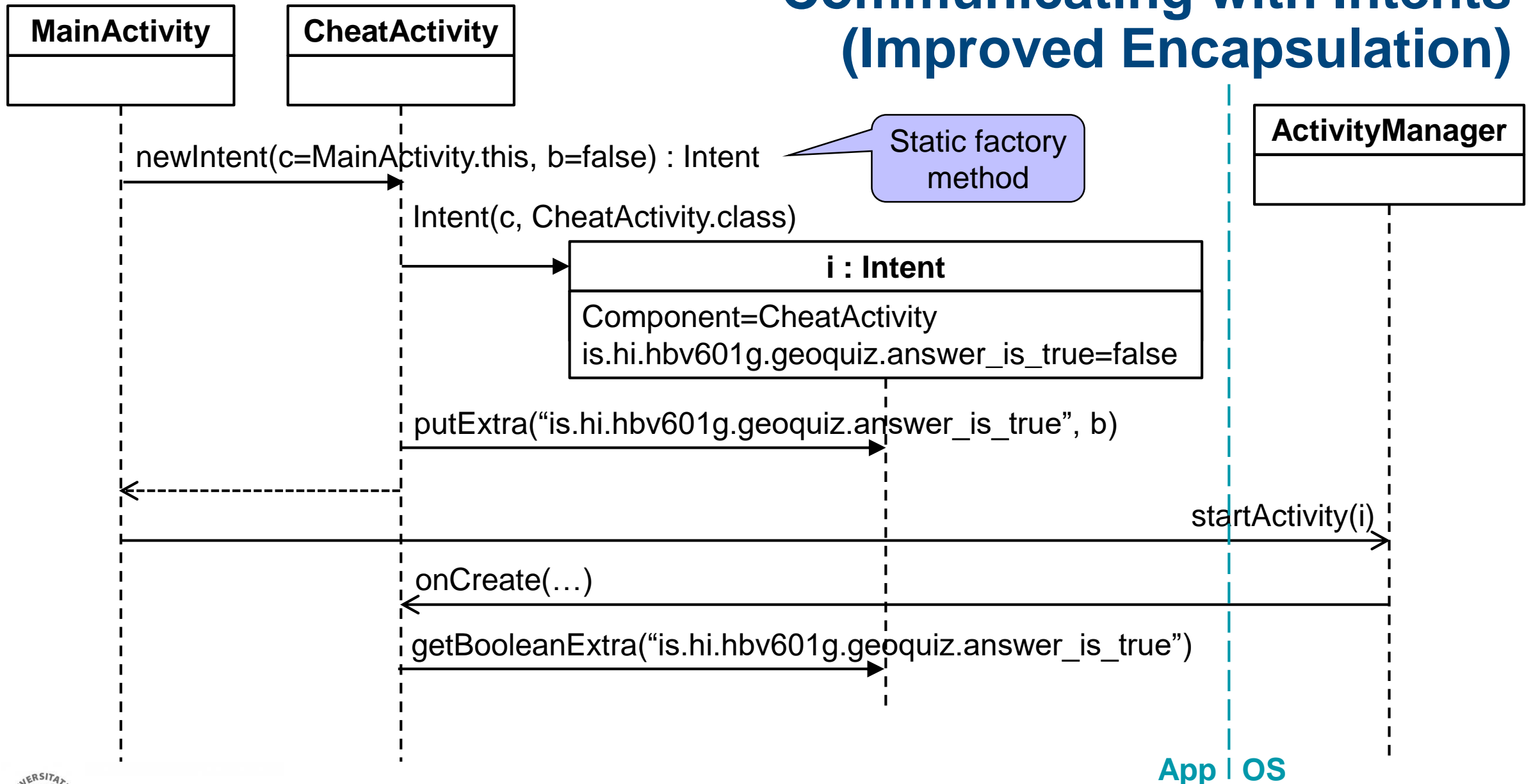
- **Extras** are arbitrary data (key-value pairs) included with the intent.
- We can also invoke activities in other apps using **implicit intents** (*covered later*)

Communicating with Intents

Inelegant: The extra's keys and types constitute an interface of **CheatActivity**! **MainActivity** should not have to be aware of its implementation details.



Communicating with Intents (Improved Encapsulation)



Defining an Intent for Other Activities to Use (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {  
  
    private static final String EXTRA_ANSWER_IS_TRUE =  
        "is.hi.hbv601g.geoquiz.answer_is_true";  
  
    public static Intent newIntent(Context packageContext, boolean answerIsTrue) {  
        Intent i = new Intent(packageContext, CheatActivity.class);  
        i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);  
        return i;  
    }  
  
    // ...  
}
```

Use a constant instead of literals to ensure we use the same key when putting and getting the extra

- **CheatActivity** provides static method **newIntent** that determines what the intents that it expects will look like:
 - a **Context** parameter (the calling activity)
 - a **boolean** parameter (the correct answer)
- No need for other activities to construct intents and extras, or to know what the extra should be called

Invoking a Child Activity in Order to Get a Result Back

- To get a result back, don't invoke an activity with `startActivity(Intent i)` but with `startActivityForResult(Intent i, int requestCode)`
- **requestCode** is an arbitrary integer that your activity will receive back with the result, in order to recognize which request the result refers to
 - since your activity may invoke multiple other activities, and doesn't know when it'll hear back

Using the Intent to Invoke CheatActivity (MainActivity.java)

```
public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_CODE_CHEAT = 0;

    // ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // ...
        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
                Intent i = CheatActivity.newIntent(MainActivity.this, answerIsTrue);
                startActivityForResult(i, REQUEST_CODE_CHEAT);
            }
        });
        // ...
    }
}
```

Accessing the Extras in the Received Intent (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {  
  
    private static final String EXTRA_ANSWER_IS_TRUE =  
        "is.hi.hbv601g.geoquiz.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);  
  
        // ...  
    }  
  
    // ...  
}
```

Retrieve the intent that
started this activity

Retrieve the extra
expected in the intent

Default value if extra is
not defined in intent

Sending a Result Back From a Child Activity in an Intent

- [Not shown before for brevity: Code for revealing the answer]
- To return data to the parent activity, perform these steps in the child activity:
 - Create another intent
 - Put the data you want to return into it as an extra
 - Call the `setResult` method in order to create the intent that will be sent back to the caller:
`public final void setResult(int resultCode, Intent data)`
 - `resultCode`: can be set to `Activity.RESULT_OK` or `Activity.RESULT_CANCELED`
 - `data`: the intent containing your custom result data (optional)
 - The intent will be sent by Android when the user leaves the child activity (i.e. backs out of it)
- Example scenario:
 - We want to inform `MainActivity` about whether the user actually peeked at the answer in `CheatActivity`, so we can react accordingly.

Returning a Result in an Intent (and Accessing it)

(CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {  
  
    private static final String EXTRA_ANSWER_SHOWN =  
        "is.hi.hbv601g.geoquiz.answer_shown";  
  
    private void setAnswerShownResult(boolean isAnswerShown) {  
        Intent data = new Intent();  
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);  
        setResult(RESULT_OK, data);  
    }  
  
    public static boolean wasAnswerShown(Intent result) {  
        return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);  
    }  
  
    // ...  
}
```

Called by the event handler for clicks on the “Show Answer” button

Put an extra in the return intent to indicate whether the answer has been revealed

Static method to help parent activity access the content of the intent’s extras without having to know about their label and type

Receiving the Result Data in the Parent Activity

- Android will call the `onActivityResult` event handler on the parent activity in order to send the result back to it:

```
protected void onActivityResult(int requestCode,  
                                int resultCode,  
                                Intent data)
```

- `requestCode` is the code you set in your original intent invoking the child activity
 - `resultCode` is the code set by the child activity
 - `data` is the intent containing the child activity's custom result data
- Since the structure of the result intent is another interface of the child activity, that activity should provide a method for interpreting the intent's extras
 - So the parent activity doesn't need to take the intent apart and be dependent on [= exposed to changes of] the extras' implementation details

Receiving and Reacting to the Result (MainActivity.java)

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

```
    private boolean mIsCheater;
```

```
    @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 = CheatActivity.wasAnswerShown(data);
```

```
        }
```

```
    }
```

```
    // ...
```

```
}
```

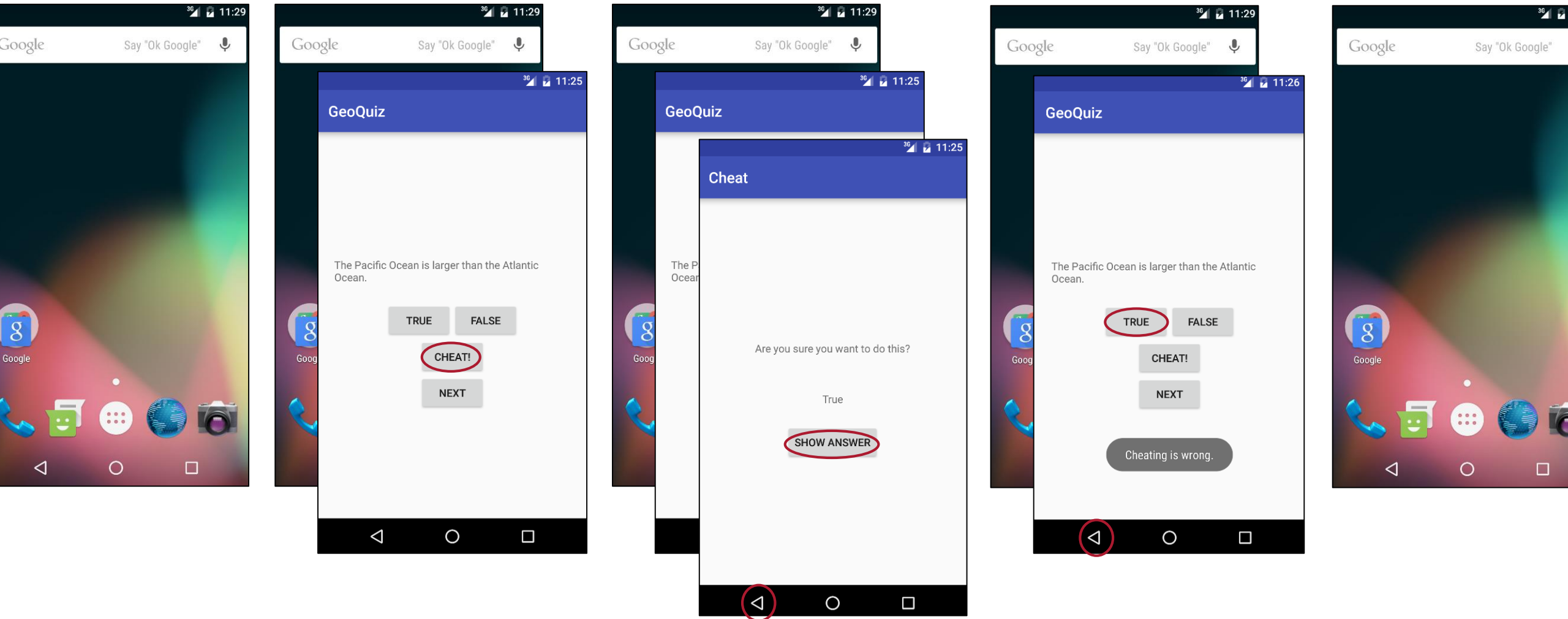
Called by Android when an intent with a result comes in

Check whether this event has the request and result codes we're expecting

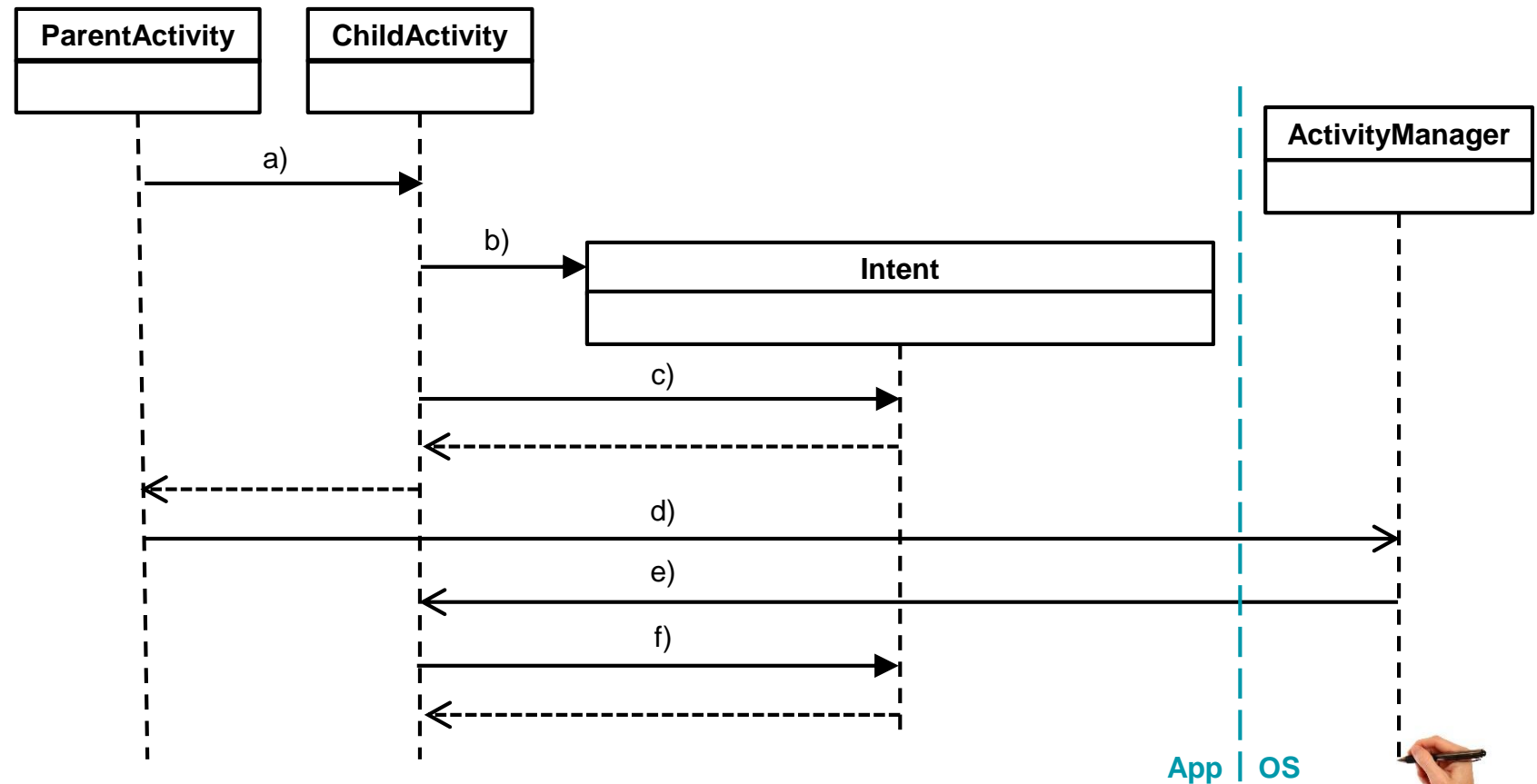
Retrieve result data using the "interpreter" method provided by the child activity

[Not shown for brevity: Code for calling out the user as a cheater]

The Activity Stack



In-Class Quiz #6: Communicating with Intents



Assign the proper labels
to the messages in the
sequence diagram:

1. `getBooleanExtra(String) : boolean`
2. `Intent(Context, Class)`
3. `newIntent(Context, boolean) : Intent`
4. `onCreate(Bundle) : void`
5. `putExtra(String, boolean) : void`
6. `startActivity(Intent) : void`

Fragments

see also:

- Phillips et al.: Android Development, Ch. 7, 9 (2nd ed.) / 8 (3rd ed.), 10, 11, 17
- <http://developer.android.com/guide/components/fragments.html>

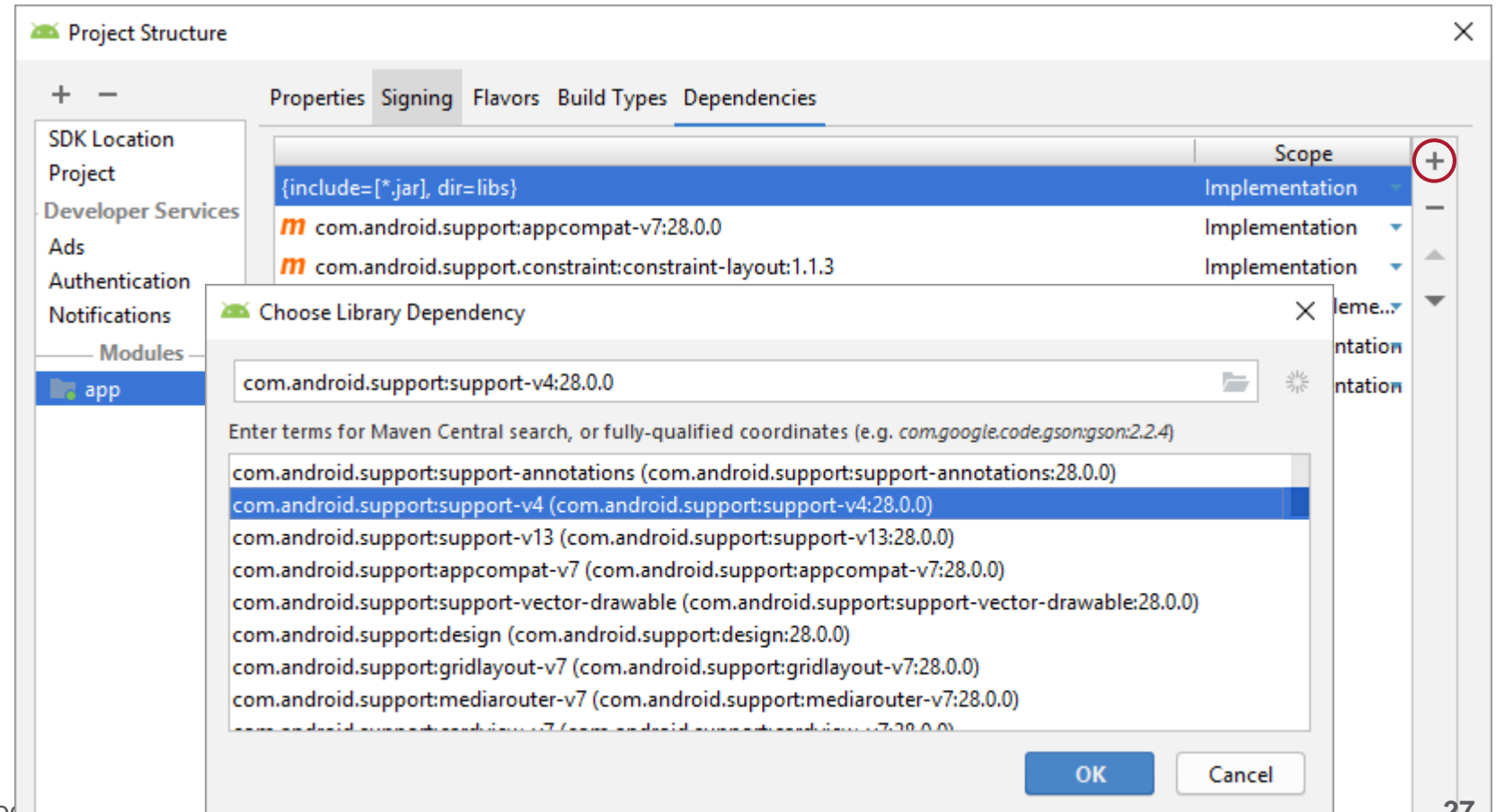


Preparation: The Android Support Library

- Different Android versions on different devices offer different “built-in” features
- By incorporating the Android Support Library, an app can “bring along” API features that may not be built into older Android versions
 - Increase backward compatibility
 - Rely on features only present in the support library
 - Benefit from faster updates of support library than updates rolled out to devices
- The **-vX** suffix of a support library indicates the API level it requires
 - e.g. **support-v4** can be used on devices running API level 4 (i.e. Android 1.6) or higher
- By using **android.support.v4.app.Fragment** and **.FragmentActivity** instead of the built-in **android.app.Fragment** and **.FragmentActivity**, we make sure our app runs on more devices and benefits from updates earlier.

Preparation: Incorporating the Android Support Library

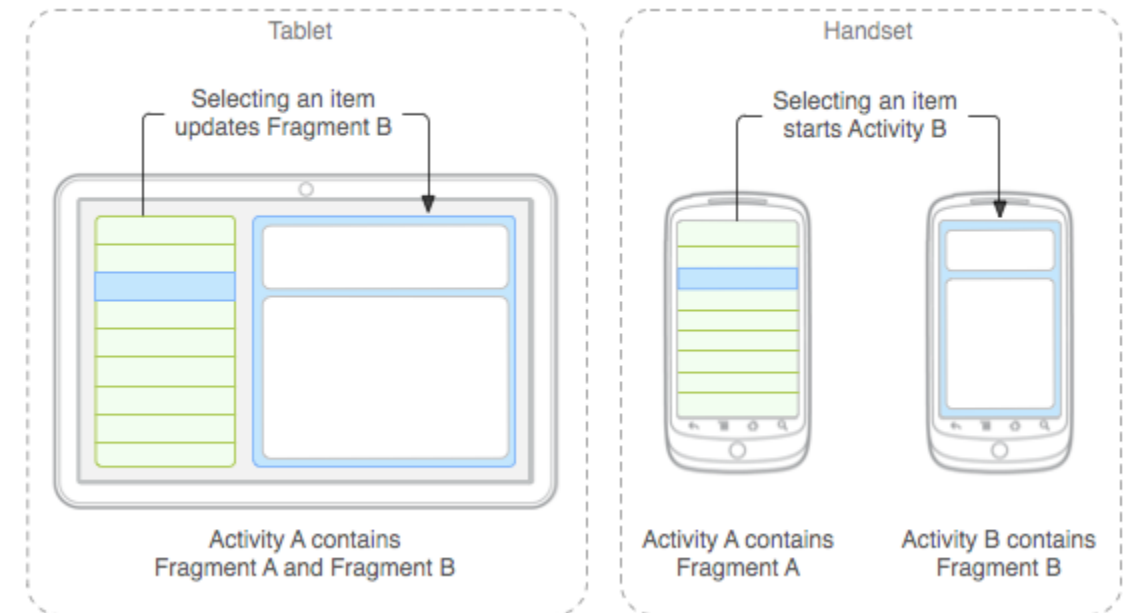
- Open menu “File > Project Structure”; choose “Dependencies” tab under “app”
- Click “+” to add a Library Dependency; pick **support-v4** library



Activities vs. Fragments

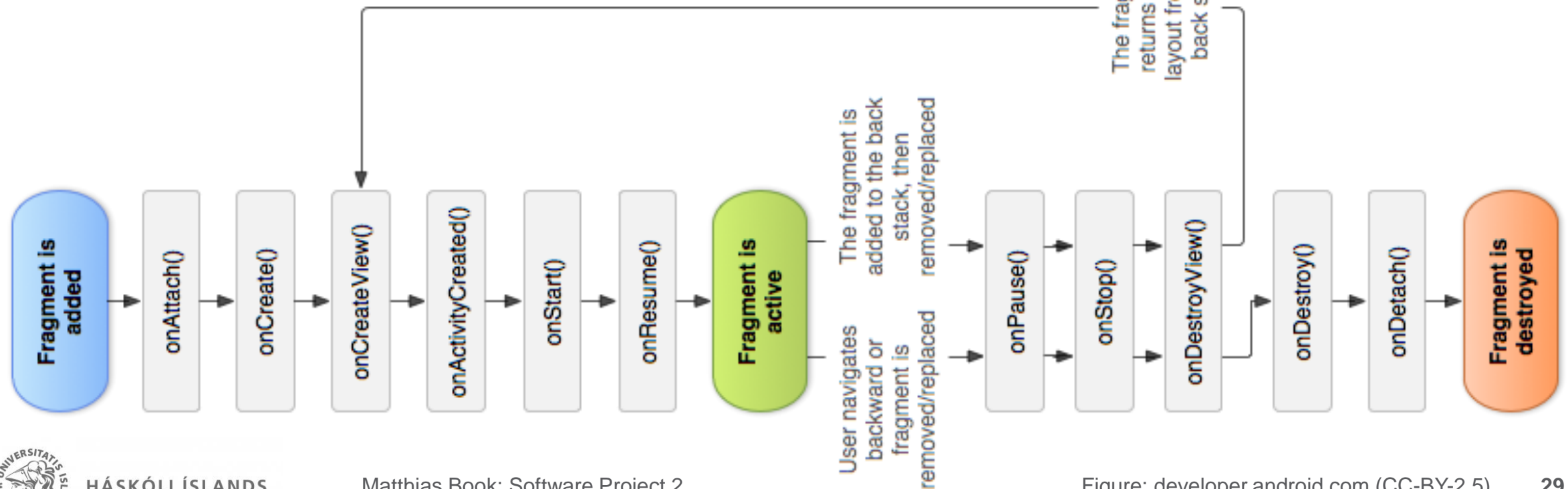
- An **activity** is the controller for one particular screen with a static layout
 - A **fragment** is the controller for a quite independent user interface “module” that
 - is nested into a host activity
 - can be added to or deleted from the activity at runtime
 - has its own layout and behavior
 - receives its own input and lifecycle events
 - can be shown in combination with other fragments inside one activity
 - can be shown as a pop-up or tabbed dialog
- Fragments enable dynamic UI changes.

One activity enabling paging through several fragments



Fragment Lifecycle

- A fragment's lifecycle methods (**onCreate**, **onPause** etc.) mirror an activity's.
- When an activity's lifecycle method is called, the same method is also called on the fragment(s) hosted by the activity.
 - In addition, the methods **onAttach** and **onDetach** are called when a fragment is associated/dissociated with its host activity.

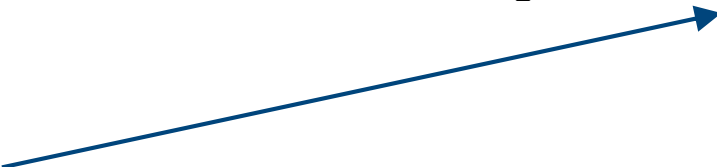


In Principle, It's Easy: Creating a Fragment

- To create a fragment, extend the **Fragment** class and inflate the fragment's layout in its **onCreateView** method:

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```



- The layout for a fragment is constructed in the same way as the layout for an activity (*not shown here; see Lecture 5*)

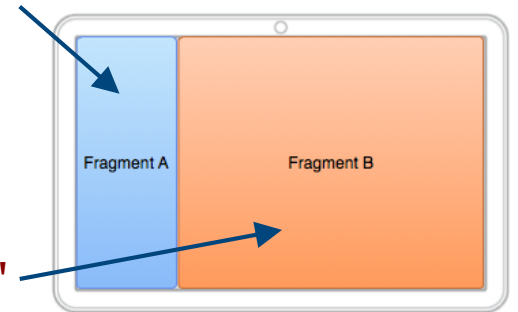
The Easy Way: Including Fragments Statically in Layout

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

```
<fragment android:name="com.example.android.fragments.HeadlinesFragment"
    android:id="@+id/headlines_fragment"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
```

```
<fragment android:name="com.example.android.fragments.ArticleFragment"
    android:id="@+id/article_fragment"
    android:layout_weight="2"
    android:layout_width="0dp"
    android:layout_height="match_parent" />
```

```
</LinearLayout>
```



Note: Fragments added to a layout this way cannot be removed or swapped out at runtime!

The Easy Way: Main Activity Controlling Overall Layout

- The layout we just defined still needs an associated activity:

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }
}
```

Note: Need to inherit from **FragmentActivity** if the layout contains fragments.

Inflate the layout we just defined.

- Note: The activity controls the lifecycle and events of the overall layout, but cannot influence fragments that are defined directly in the layout XML.
 - To have control of the fragments at runtime, we need to *add* them at runtime.
 - This is considerably more complex, but also more flexible.

In Practice, It's More Complex...

Example: Building a List-Detail Interface with Fragments

- **User interface components:**

- One “list” fragment showing a list of several items
- One “detail” fragment showing the details of one item

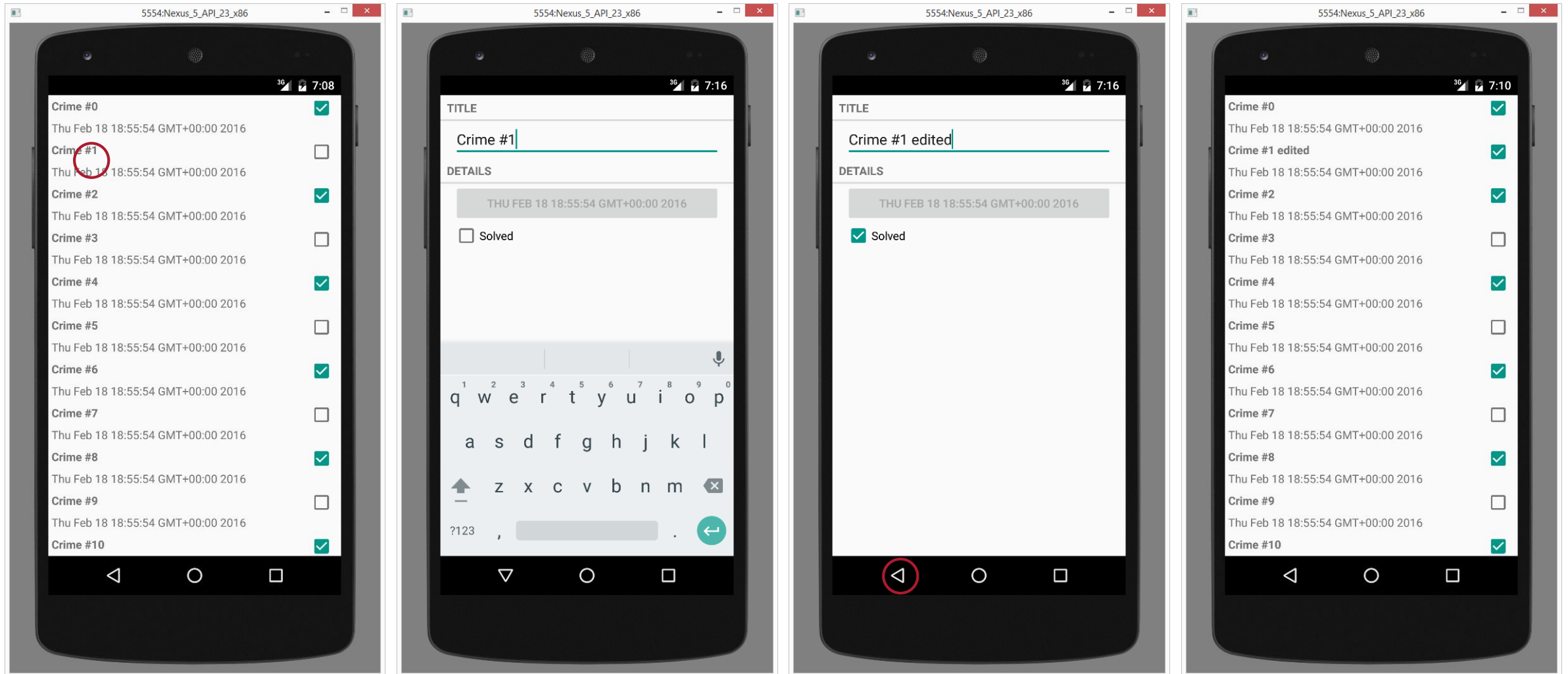
- **Intended behavior:**

- Selecting one item in the “list” fragment should show its details in the “detail” fragment
- Editing information in the “detail” fragment should be reflected in the “list” fragment

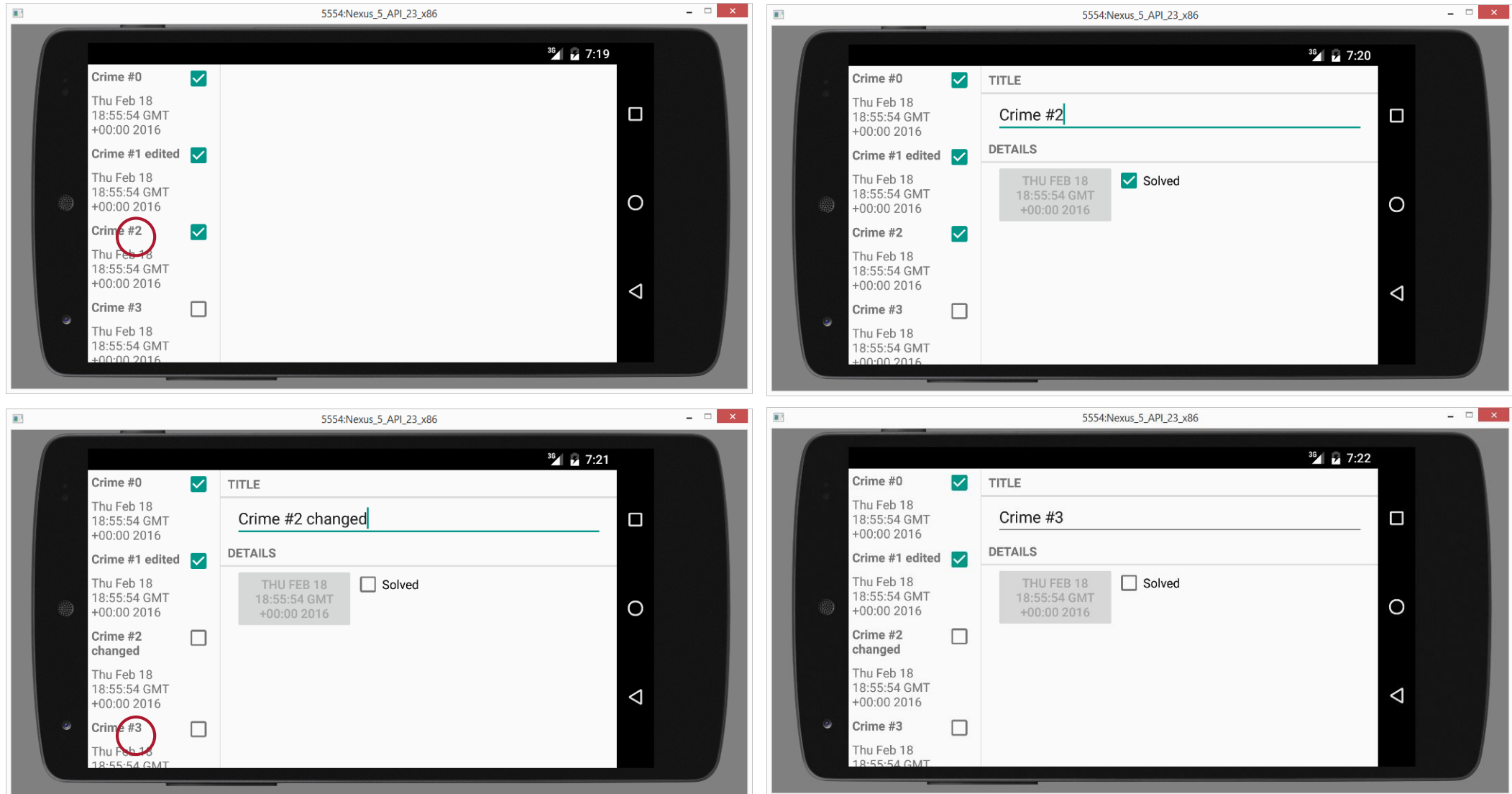
- **Layout flexibility:**

- In portrait orientation, only one of the fragments should be visible at any time
- In landscape orientation, both fragments should be visible side by side

Requirement: Layout and Behavior in Portrait Orientation



Requirement: Layout and Behavior in Landscape Orient.



Solution: Design Model

