
# Hugbúnaðarverkefni 2 / Software Project 2

## 6. Android Control Flow

HBV601G – Spring 2019

**Matthias Book**

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

# Miðmisseriskönnun

**Evaluate this course on Ugla!**

(open until tomorrow)

# In-Class Quiz #5 Prep

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

ID: ____@hi.is  Date: _____

a) ____  e) ____  i) ____
b) ____  f) ____  j) ____
c) ____  g) ____  k) ____
d) ____  h) ____

- During class, I'll show you questions that you can answer very briefly
  - Just numbers or letters, no elaboration
- Hand in your scrap at the end of class

- All questions in a quiz weigh same
- All quizzes (ca. 10-12 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

# In-Class Quiz 4 Solution

- **Fill the blanks in the following sentences with the words (A)ctivity, (L)ayout, (R)esource and (W)idget:**

a) An **activity** manages the user's interaction with the UI defined by a **layout**.

b) A **layout** defines a set of **widgets** and their screen positions.

c) A view hierarchy contains the **widgets** of one **layout**.

d) The **`setContentView`** method of an **activity** inflates the given **layout**.

e) Inflating a **layout** means creating Java objects for all **widgets** declared in it.

f) A **resource** is any piece of an app that is not executable code.

g) The auto-generated **`R`** class contains references to all **resources** and **widgets** defined in XML files under **`app/res/`**.

# Recap: Layout Implementation

- A **layout** defines a set of UI widgets and their screen positions.
- **Widgets** can show text or graphics, interact with the user, or arrange other widgets on screen
- The widgets of each layout form a hierarchy of View objects (the **“View hierarchy”**)

```
<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: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>
```


**/res/layout/activity_main.xml**

# Recap: Activity Implementation

- An **activity** is responsible for managing user interaction with a screen of information
  - i.e. it implements a part of the functionality of your app
- Android apps are event-driven
  - i.e. after starting, they wait ("listen") for a user- or system-initiated event to occur
- Android provides listener interfaces for many events
  - You override the listener with code determining what happens upon the event

```
public class MainActivity extends AppCompatActivity {

  private Button mTrueButton;
  private Button mFalseButton;

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

    mTrueButton = (Button) findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Toast.makeText(MainActivity.this,
          R.string.correct_toast, Toast.LENGTH_SHORT).show();
      }
    });

    mFalseButton = (Button) findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Toast.makeText(MainActivity.this,
          R.string.incorrect_toast, Toast.LENGTH_SHORT).show();
      }
    });
  }
}
```

Event listener in anonymous inner class

/java/is/hi/hbv601g/geoquiz/MainActivity.java

# Recap: Resources

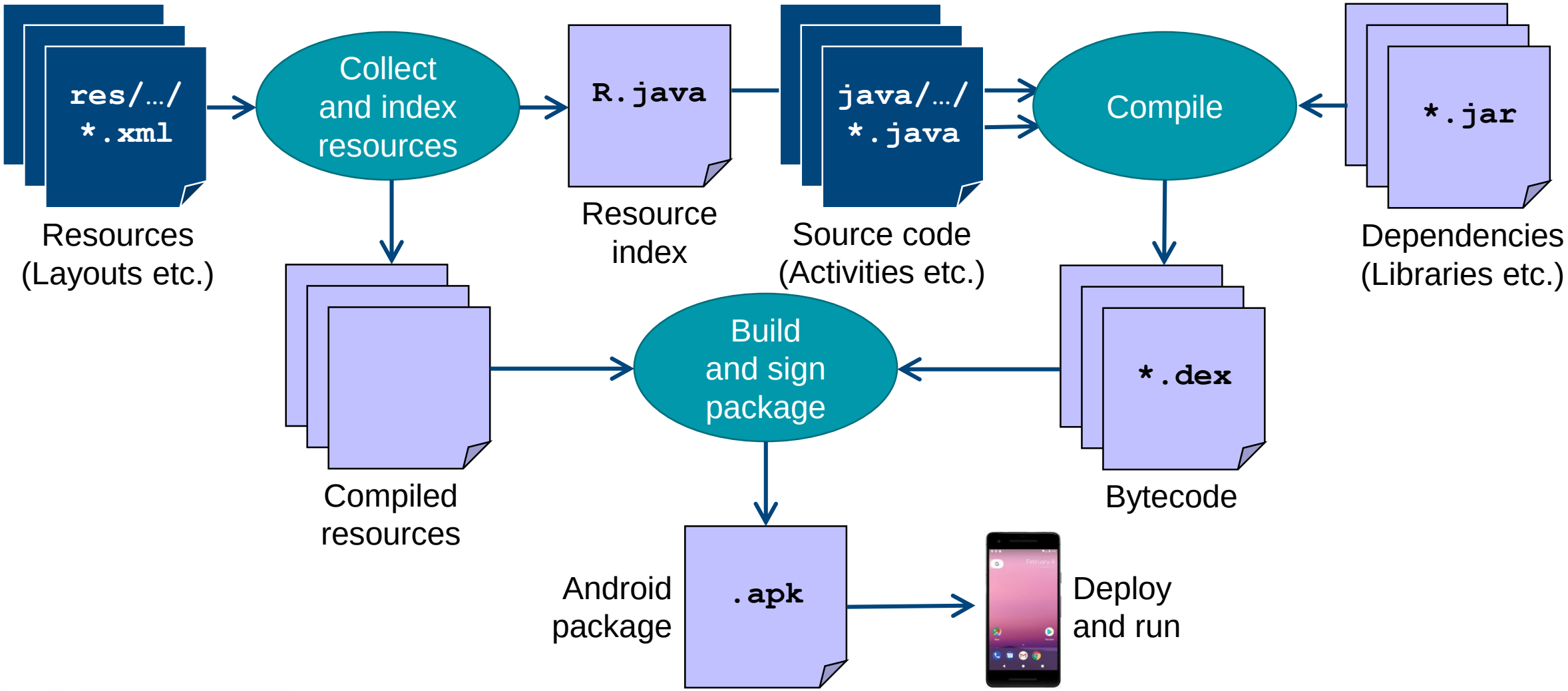
- A resource is any piece of an app that is not source code
  - e.g. images, XML files etc.
- Resources are stored in subdirectories of app/src/main/res, e.g.
  - …/layout/activity_main.xml
  - …/values/strings.xml
- In source code, resources are referenced using constants defined in the automatically-generated class **R**

/res/values/strings.xml

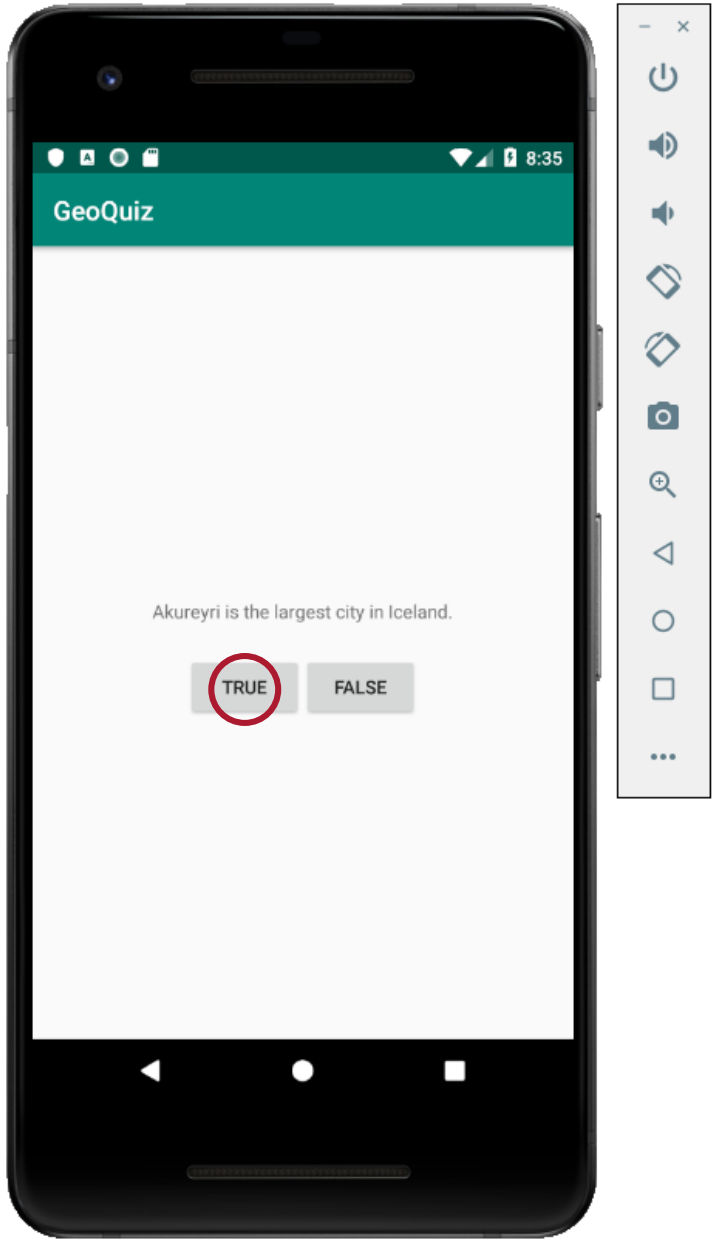
```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">GeoQuiz</string>
    <string name="question_text">
        Akureyri is the largest city in Iceland.
    </string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>

</resources>
```

```java
/* AUTO-GENERATED FILE.  DO NOT MODIFY. */

package is.hi.hbv601g.geoquiz;

public final class R {

  public static final class layout {
    public static final int activity_main=0x7f040019;
    // ...
  }

  public static final class string {
    public static final int false_button=0x7f0a0012;
    public static final int true_button=0x7f0a0015;
    // ...
  }

  public static final class id { ... }

  // ...

}
```

# Under the Hood: The Android Build Process (simplified)



# Recap: Running Application





- …obviously requires some more logic ☺

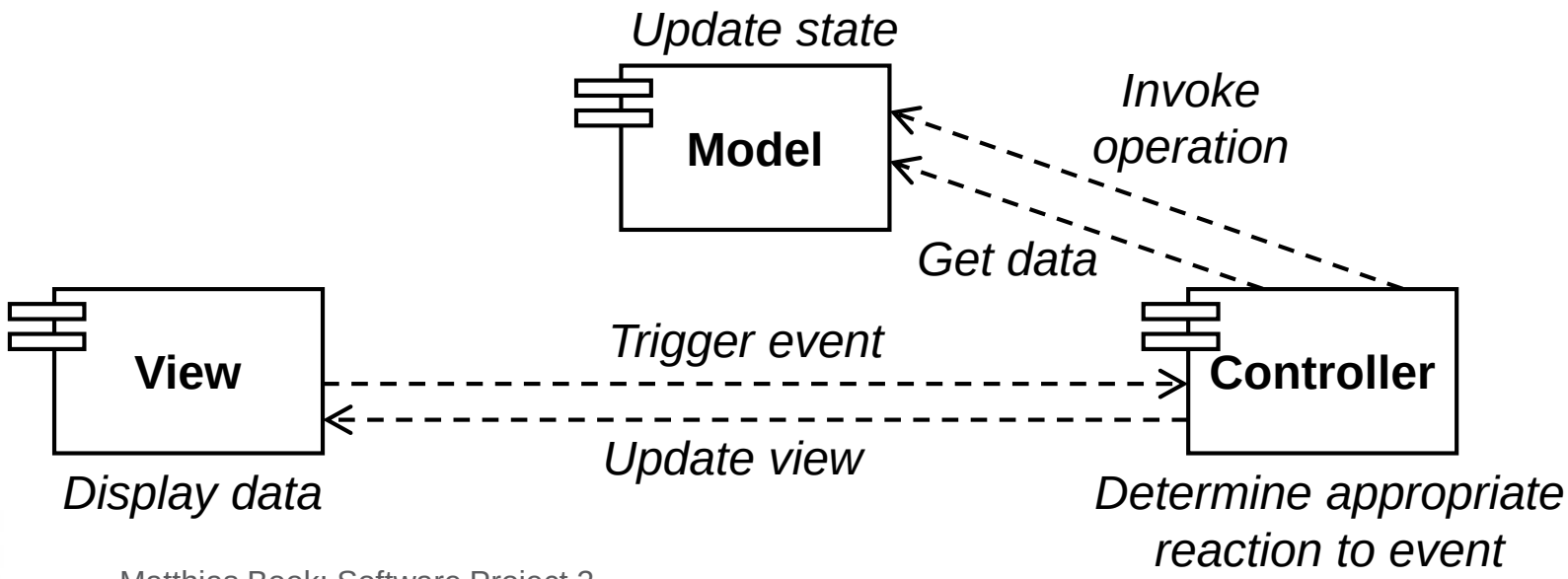
# The Model-View-Controller Pattern in Android Apps

see also:

- Phillips et al.: Android Development, Ch. 2

# Model-View-Controller (MVC) Pattern in Android Apps

- **Model:** Objects responsible for holding the app's data and business logic
- **View:** Objects that know how to draw themselves on the screen and how to respond to user input
- **Controller:** Objects that respond to events triggered by views, invoke logic on the model, and thus manage the data and control flow between model and view

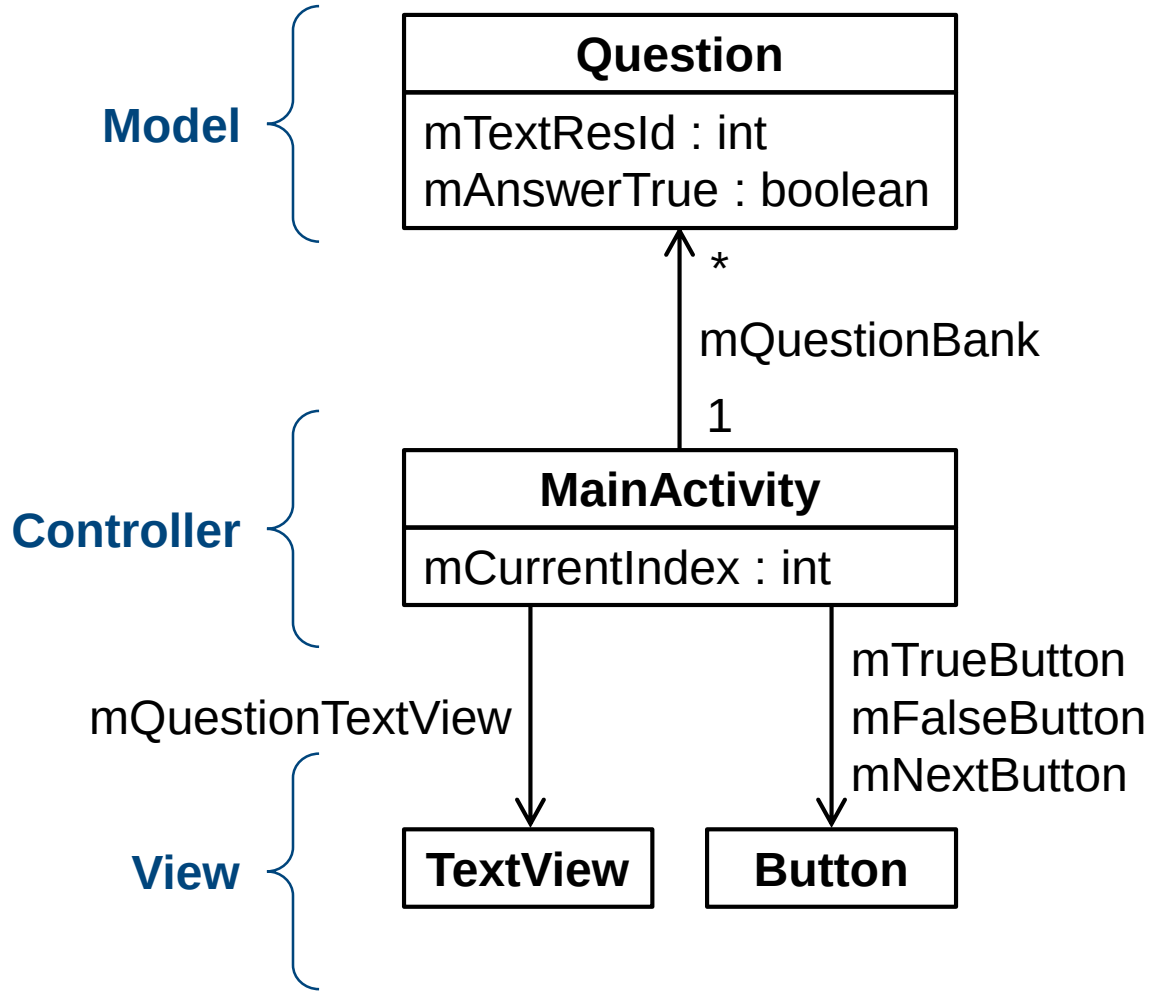


# Example: Managing Questions in the GeoQuiz App

- Instead of just having one static question with a hard-wired answer, the app should let users go through a series of questions.

➢We need:

- **in the Model layer:**
  Objects storing questions and answers
- **in the View layer:**
  Buttons for navigating between questions
- **in the Controller layer:**
  Logic that ties buttons to questions, and creates toasts according to the answers



# Model: `Question.java`

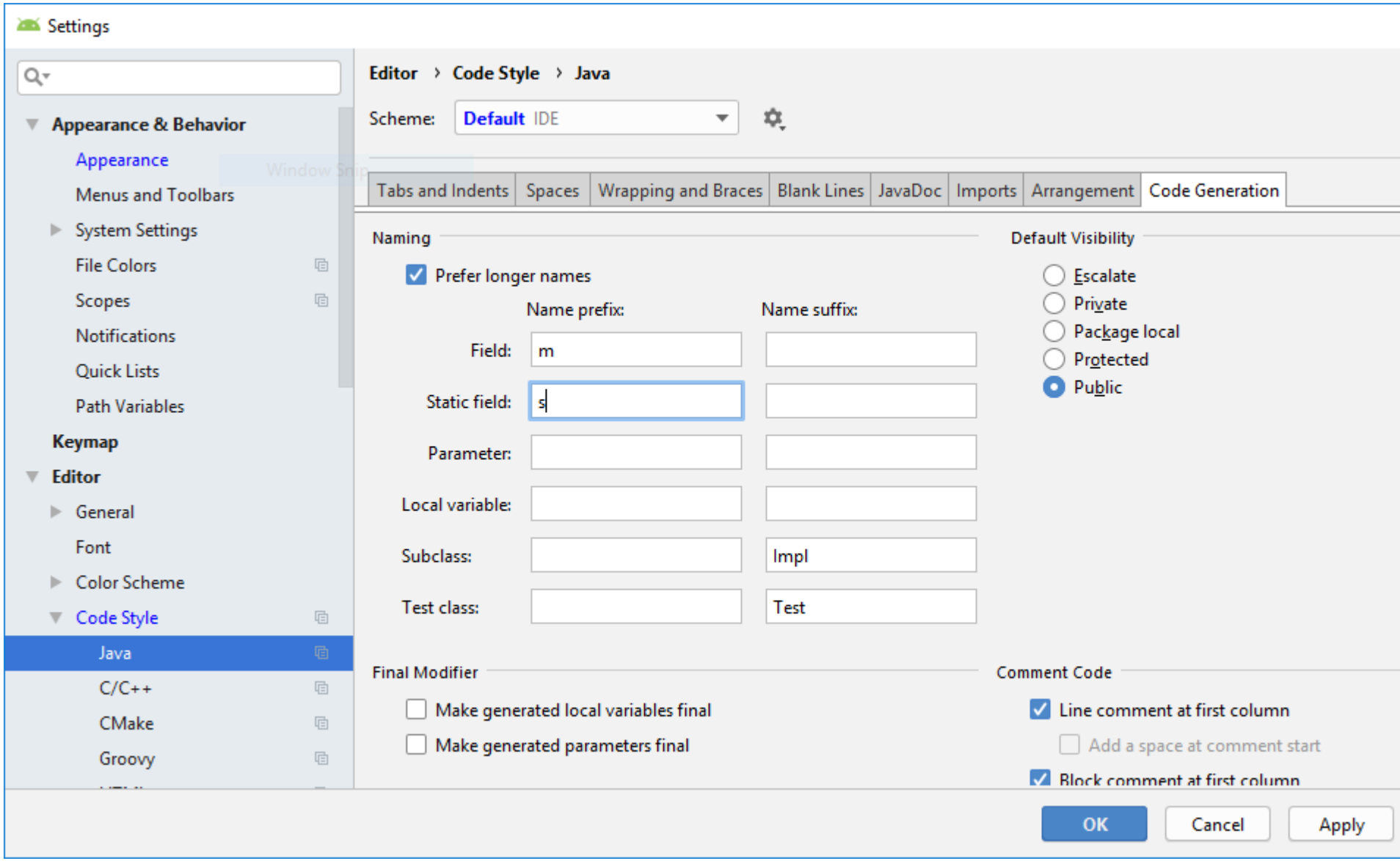
POJO describing a question and its correct answer

Note: We use `int` instead of `String` since we won't store the literal question strings here, but references to the string resource file (to enable easier internationalization)

```
package is.hi.hbv601g.geoquiz;

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) {
        mTextResId = textResId;
    }

    public boolean isAnswerTrue() {
        return mAnswerTrue;
    }

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

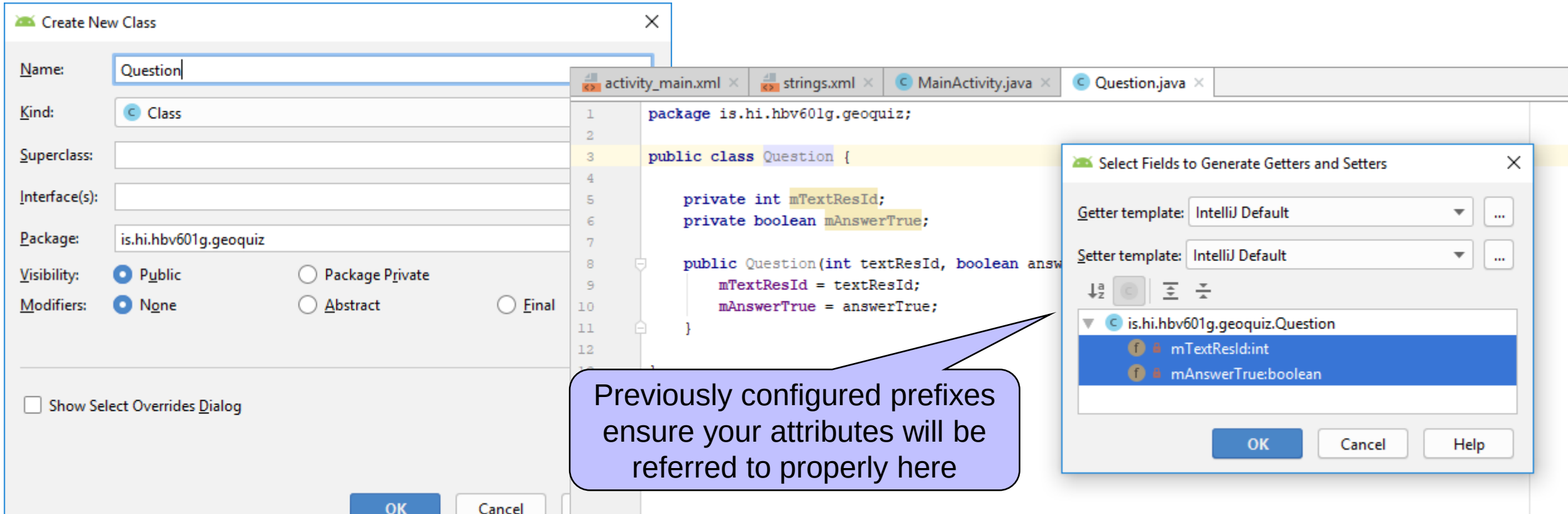
# IDE Tip: Prefixes

- In “File > Settings” menu
- Open the “Editor > Code Style > Java” dialog
- In the “Code Generation” tab, enter `m` and `s` prefixes for fields and static fields
  - to enable proper generation of getters and setters



# IDE Tip: Adding a New Class; Generating Constructor and Accessors

- Right-click on package where you want the class, select “New > Java Class…”
- After coding the attributes, right-click on class name and select “Generate…”
  - Choose attributes that generated code shall apply to
  - Let IDE generate constructor, getters and setters



# View: `activity_main.xml`

Rather than referring to a static string (`@string/question_text`), the **`TextView`** for the question now gets its own ID through which we can modify it

New “Next” button

```
...

<TextView
    android:id="@+id/question_text_view"
    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/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button"/>

    ...

</LinearLayout>

...
```

# View: strings.xml

Additional strings for Next button and questions

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="next_button">Next</string>
    <string name="incorrect_toast">Incorrect!</string>
    <string name="correct_toast">Correct!</string>
    <string name="action_settings">Settings</string>
    <string name="question_oceans">The Pacific Ocean is larger than the Atlantic
        Ocean.</string>
    <string name="question_mideast">The Suez Canal connects the Red Sea and the Indian
        Ocean.</string>
    <string name="question_africa">The source of the Nile River is in Egypt.</string>
    <string name="question_americas">The Amazon River is the longest river in the
        Americas.</string>
    <string name="question_asia">Lake Baikal is the world\'s oldest and deepest freshwater
        lake.</string>
</resources>
```

# Controller: `MainActivity.java`
# Setting up the QuestionBank

```
public class MainActivity extends AppCompatActivity {

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

    private Question[] mQuestionBank = new Question[] {
            new Question(R.string.question_oceans, true),
            new Question(R.string.question_mideast, false),
            new Question(R.string.question_africa, false),
            new Question(R.string.question_americas, true),
            new Question(R.string.question_asia, true)
    };

    private int mCurrentIndex = 0;

    // ...
```

Just a simple array for now – we'll see more scalable solutions for data storage later

# Controller: `MainActivity.java`
# Enabling Updates of the Displayed Question

```
public class MainActivity extends AppCompatActivity {
    // ...
    private TextView mQuestionTextView;

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

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // ...
        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        // ...
    }
    // ...
}
```

2. Retrieve text for current question and update the **`QuestionTextView`** to display it

1. Retrieve the **`QuestionTextView`** object from the view hierarchy

# Controller: `MainActivity.java`
# Handling Clicks on the Next Button

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // ...

        mNextButton = (Button) findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                updateQuestion();
            }
        });

        updateQuestion();
    }
    // ...
}
```

Increment question index (modulo total number of questions) and display new question

After all the setup is complete, display first question

# Controller: `MainActivity.java`
# Checking Answer and Displaying Response

```
public class MainActivity extends AppCompatActivity {

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

        int messageResId = 0;

        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT).show();
    }

    // ...
}
```

Retrieve correct answer for current question

Determine appropriate response

Create and show toast with response

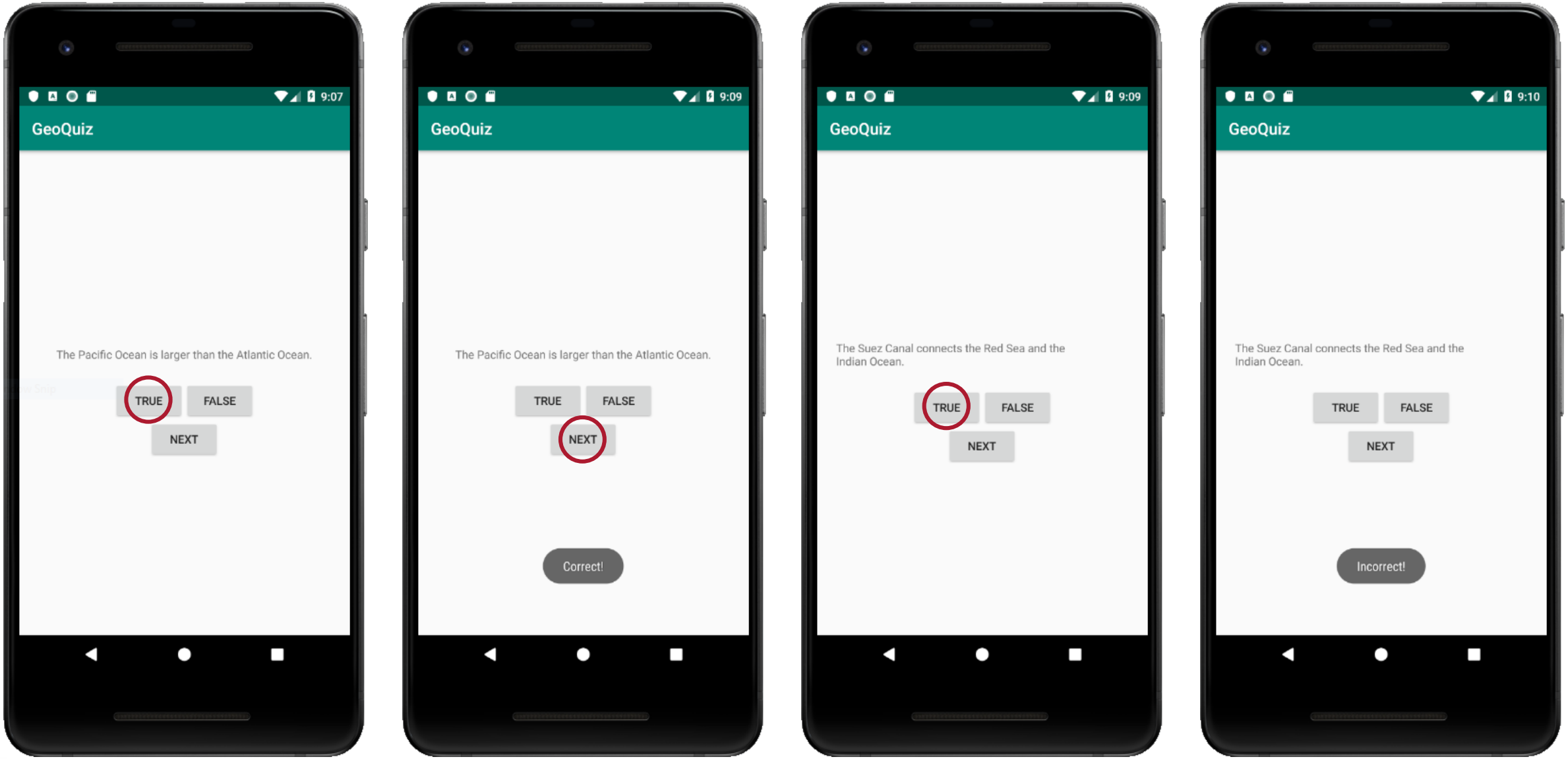
# Controller: `MainActivity.java`
# Connecting the Buttons to the Answer Checking Code

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        mTrueButton = (Button) findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { checkAnswer(true); }
        });

        mFalseButton = (Button) findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { checkAnswer(false); }
        });
        // ...
    }
    // ...
}
```

Check if `true` is correct answer

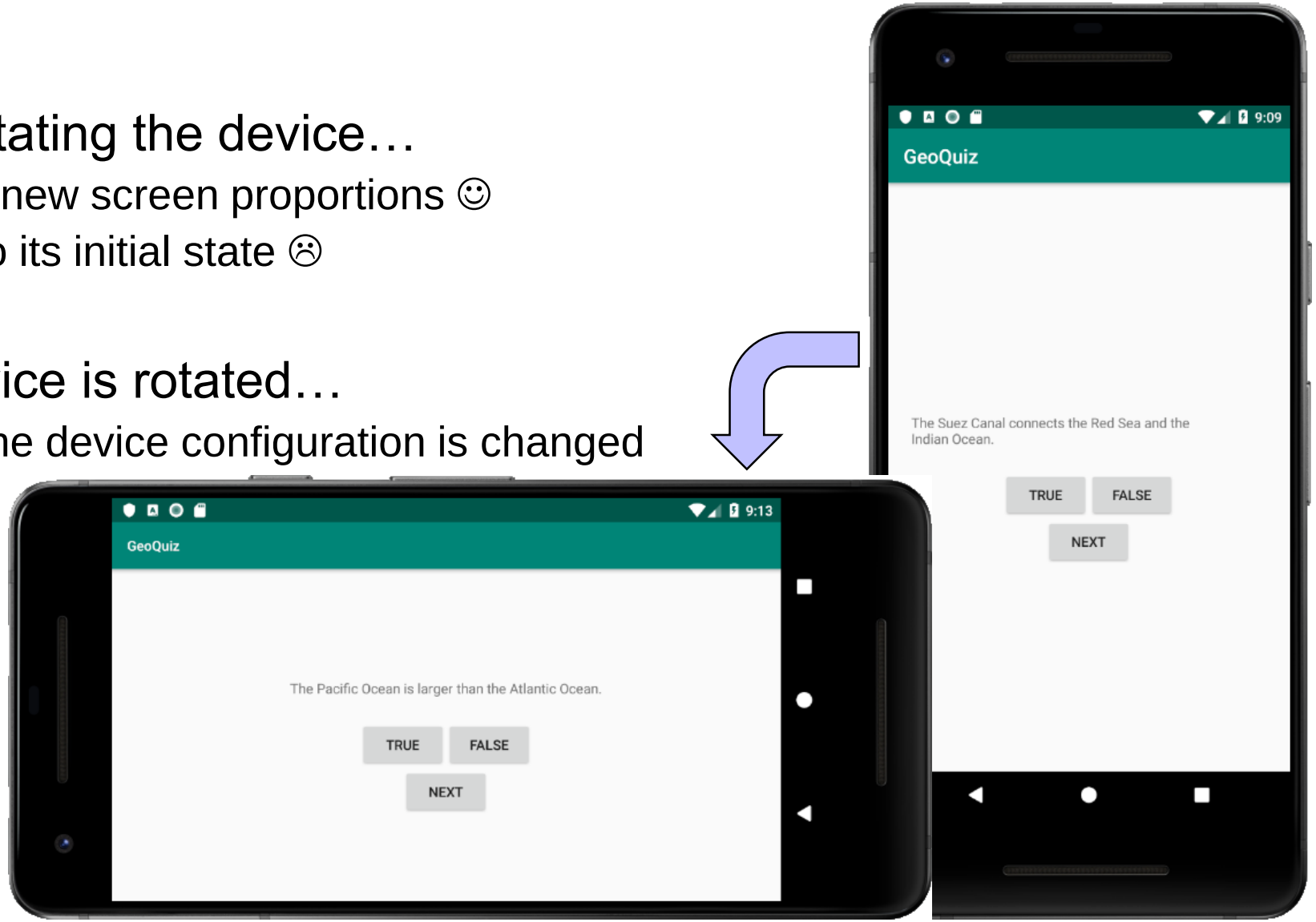
Check if `false` is correct answer

# Running the App Again



# Rotating the Device

- **Observation:** When rotating the device…
  - the layout adapts to the new screen proportions ☺
  - but the activity reverts to its initial state ☹
- **Reason:** When the device is rotated…
  - more generally: When the device configuration is changed
- …the running activity is destroyed and recreated to match the new configuration
  - i.e. **`onCreate`** is called again, which resets our question counter

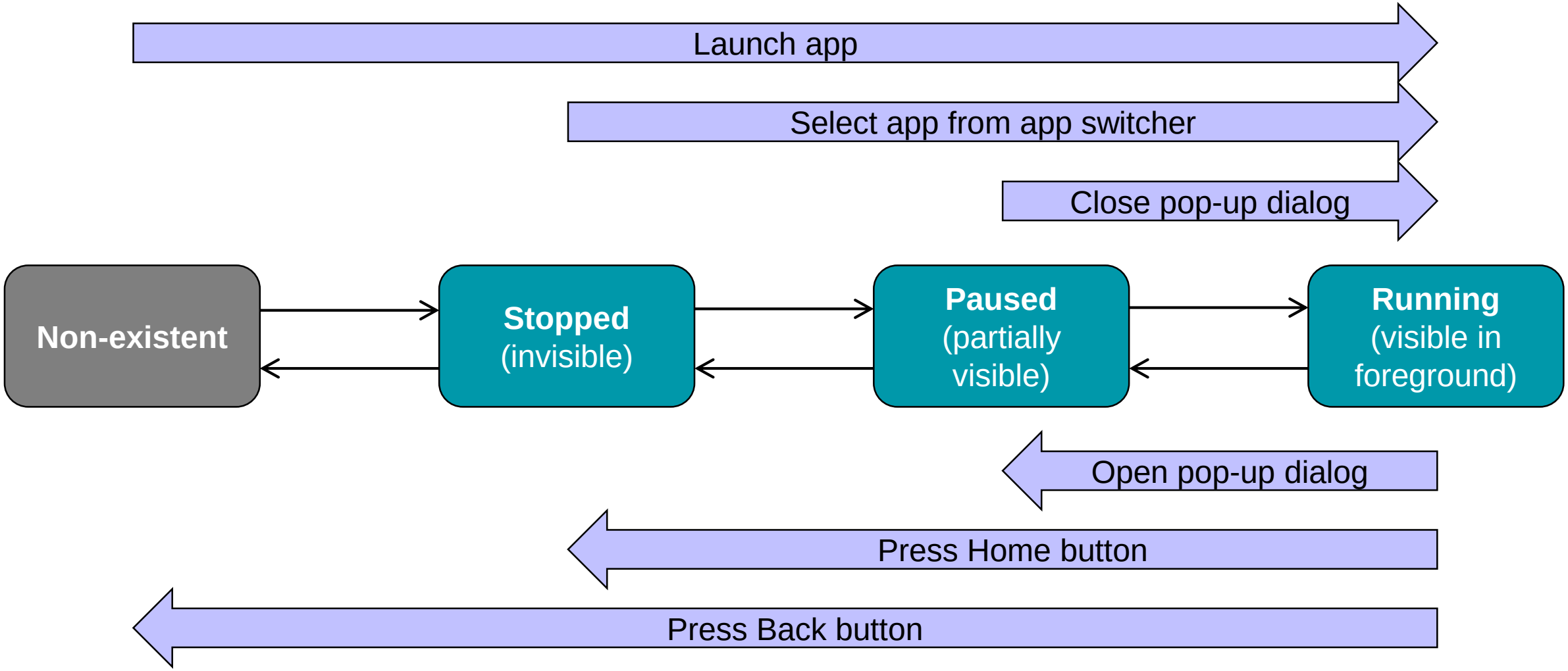


# The Activity Lifecycle

see also:

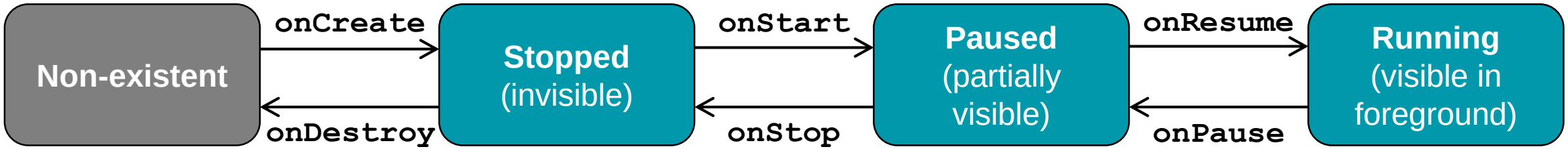
- Phillips et al.: Android Development, Ch. 3
- http://developer.android.com/guide/components/activities.html

# The Activity Lifecycle



# The Activity Lifecycle

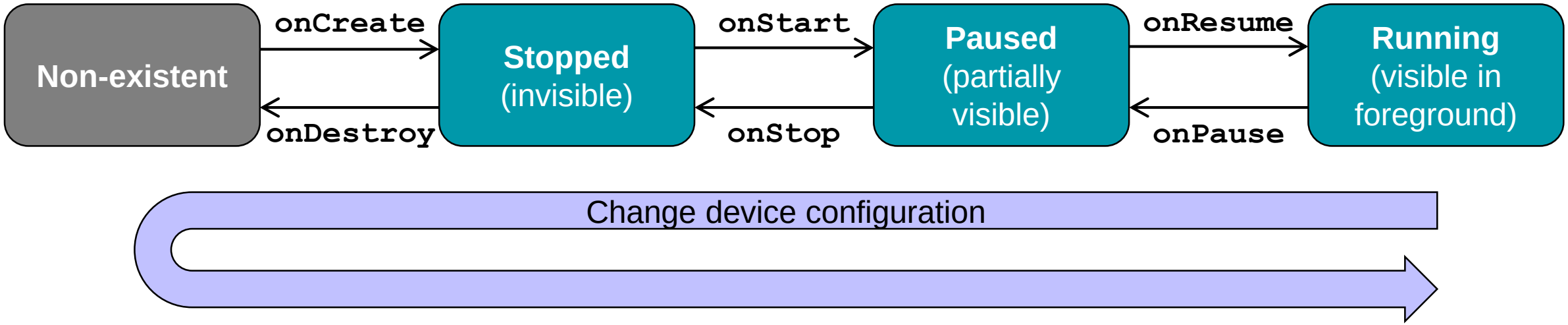
- Activities may transition between three states during their lifecycle
  - e.g. from Non-existent through Stopped and Paused to Running upon launching the app
  - e.g. from Running through Paused to Stopped as a user switches between apps
  - e.g. from Running through Paused and Stopped to Non-existent upon pressing Back button




- Various **`on`**… methods can be overridden to *react* to these transitions
  - The methods are called by the Android Operating System, *not* the activity!
  - When overriding these, always call superclass' method first (e.g. **`super.onPause()`**)

# Reacting to Device Configuration Changes

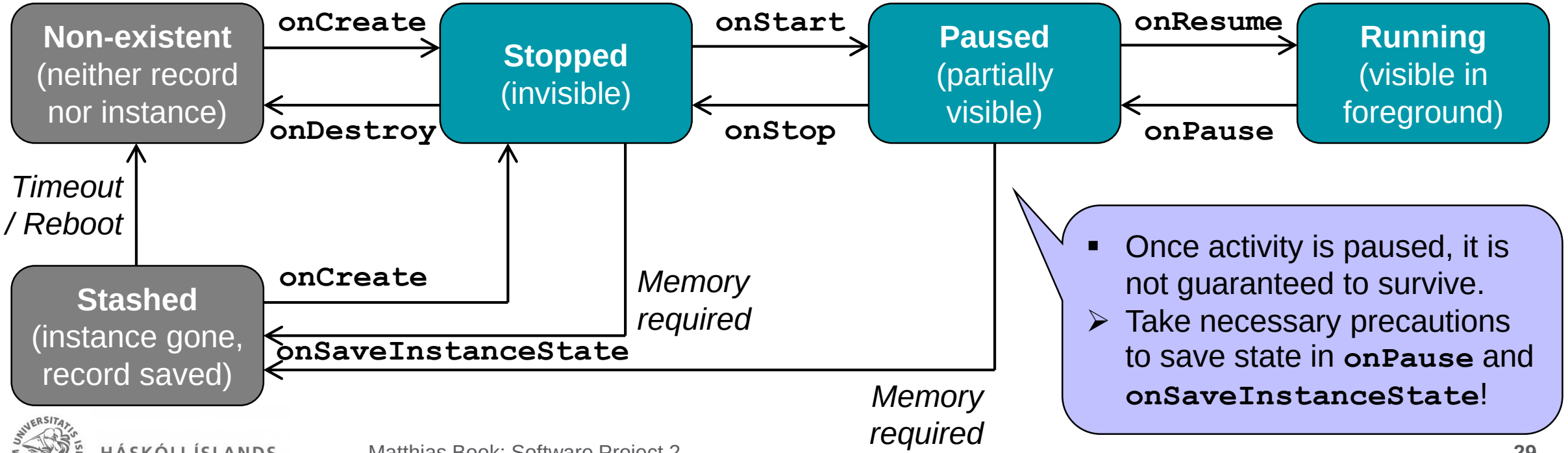
- Some aspects of the device configuration may change at runtime
  - e.g. screen orientation, keyboard type, dock mode, language, etc.
- To work optimally under the changed configuration, an activity may require different resources (e.g. a different layout or different graphics)
  - To re-initialize the resources, the activity depends on **`onCreate`** being called again




➢Call order: **`onPause`**, **`onStop`**, **`onDestroy`**, **`onCreate`**, **`onStart`**, **`onResume`**

# 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



# Caution: Simplified Introduction to Activity Lifecycle

- The preceding introduction has been simplified for the sake of clarity.

- For a precise specification of the circumstances under which the various lifecycle and state-saving methods are called, see
  - http://developer.android.com/training/basics/activity-lifecycle/starting.html
  - http://developer.android.com/reference/android/app/Activity.html

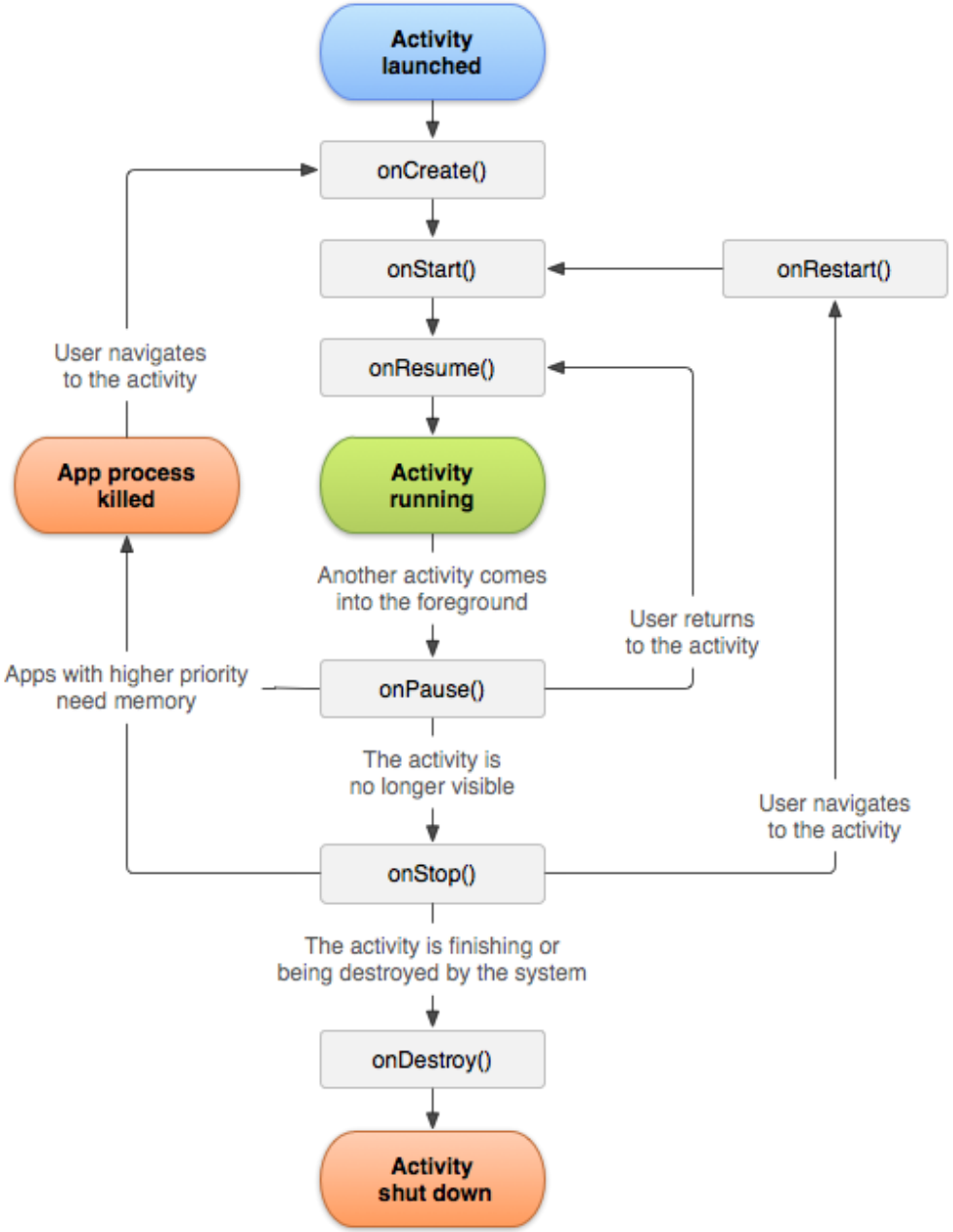




Figure: developer.android.com (CC-BY-2.5)

# Storing Activity State in `onSaveInstanceState`

- Method called by Android when activity becomes killable
  - i.e. when it reaches a state where it may be removed from memory without further notice
  - Default implementation lets all view objects store their state in a **`Bundle`**
- We can extend the method to store additional data in the same **`Bundle`**
- A **`Bundle`** is a data structure mapping keys to values
  - Keys are string constants
  - Values are typically primitive types
    - **`Serializable`** or **`Parcelable`** classes are possible but discouraged

```
public class MainActivity extends AppCompatActivity {

    private static final String KEY_INDEX = "index";

    private int mCurrentIndex = 0;

    // ...
    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        super.onSaveInstanceState(savedInstanceState);
        savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
    }
    // ...
}
```

# Recovering Activity State in `onCreate`

- Android provides a **`Bundle`** containing previously stored activity state (if any)
- We can recover previously stored values from this (if present)

```
public class MainActivity extends AppCompatActivity {

    private static final String KEY_INDEX = "index";
    private int mCurrentIndex = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // ...
        if (savedInstanceState != null) {
            mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
        }
        // ...
    }
    // ...
}
```

Check if we received any state from a previous instance of the activity

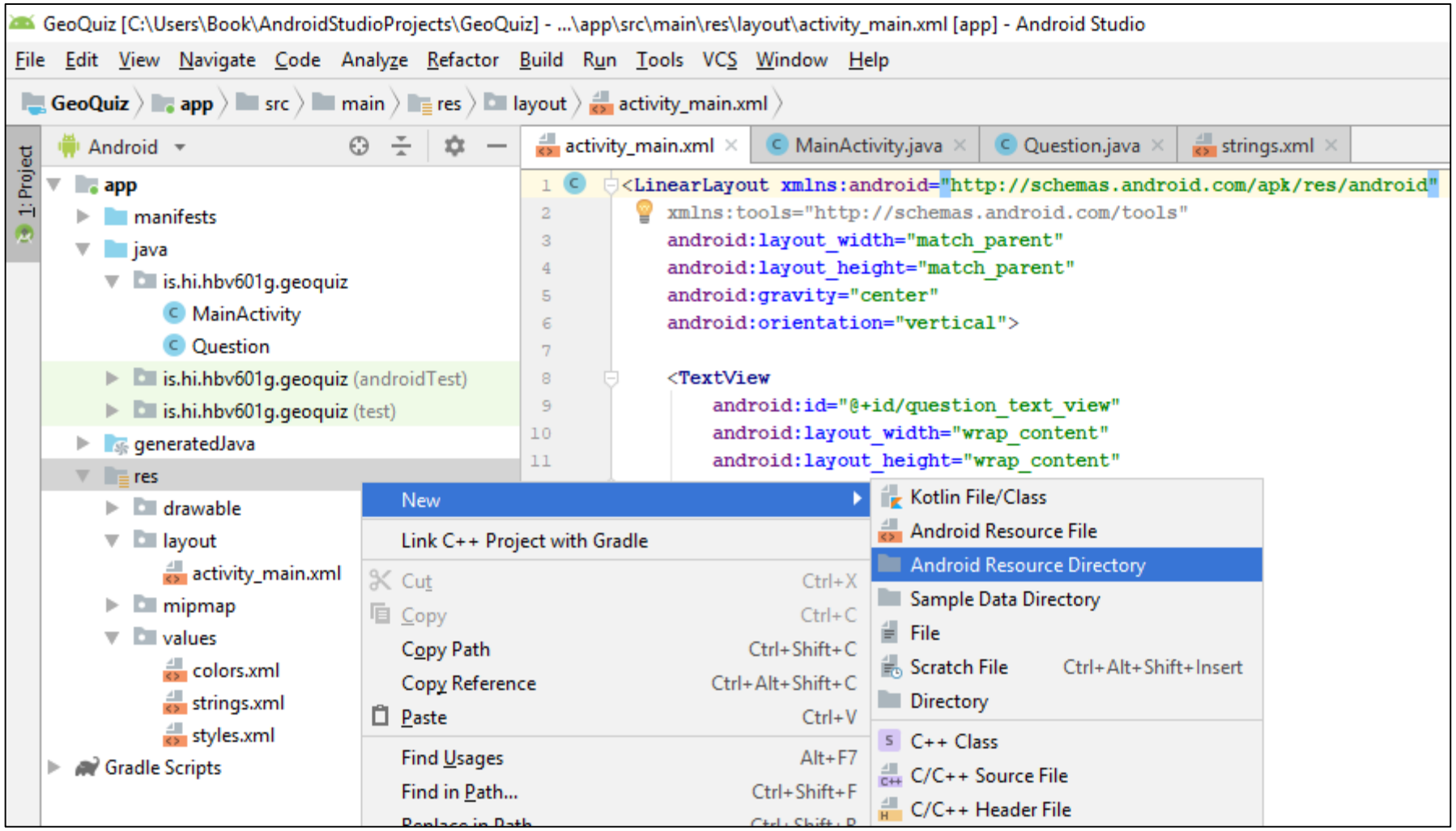
Retrieve question index or use default value of 0 if none was stored in **`Bundle`**

# What to Do Where?

- **`onCreate`** is typically overridden to prepare the activity's user interface:
  - Inflating widgets and putting them on screen (with **`setContentView`**)
  - Getting references to inflated widgets, in order to work with them later
  - Setting listeners on widgets to handle user interaction
  - Retrieving saved instance state
  - Connecting to external model data
- **`onSaveInstanceState`** is typically overridden to store small, transient items
- **`onPause`** is typically overridden for larger save/cleanup tasks upon loss of focus
- What you do in **`onPause`**, **`onStop`**, **`onDestroy`** etc. depends on your activity:
  - How will people probably use it?
  - How will people probably leave it?
  - How might people get interrupted?
  - How might people resume it?
  - How would people expect to find it upon returning?

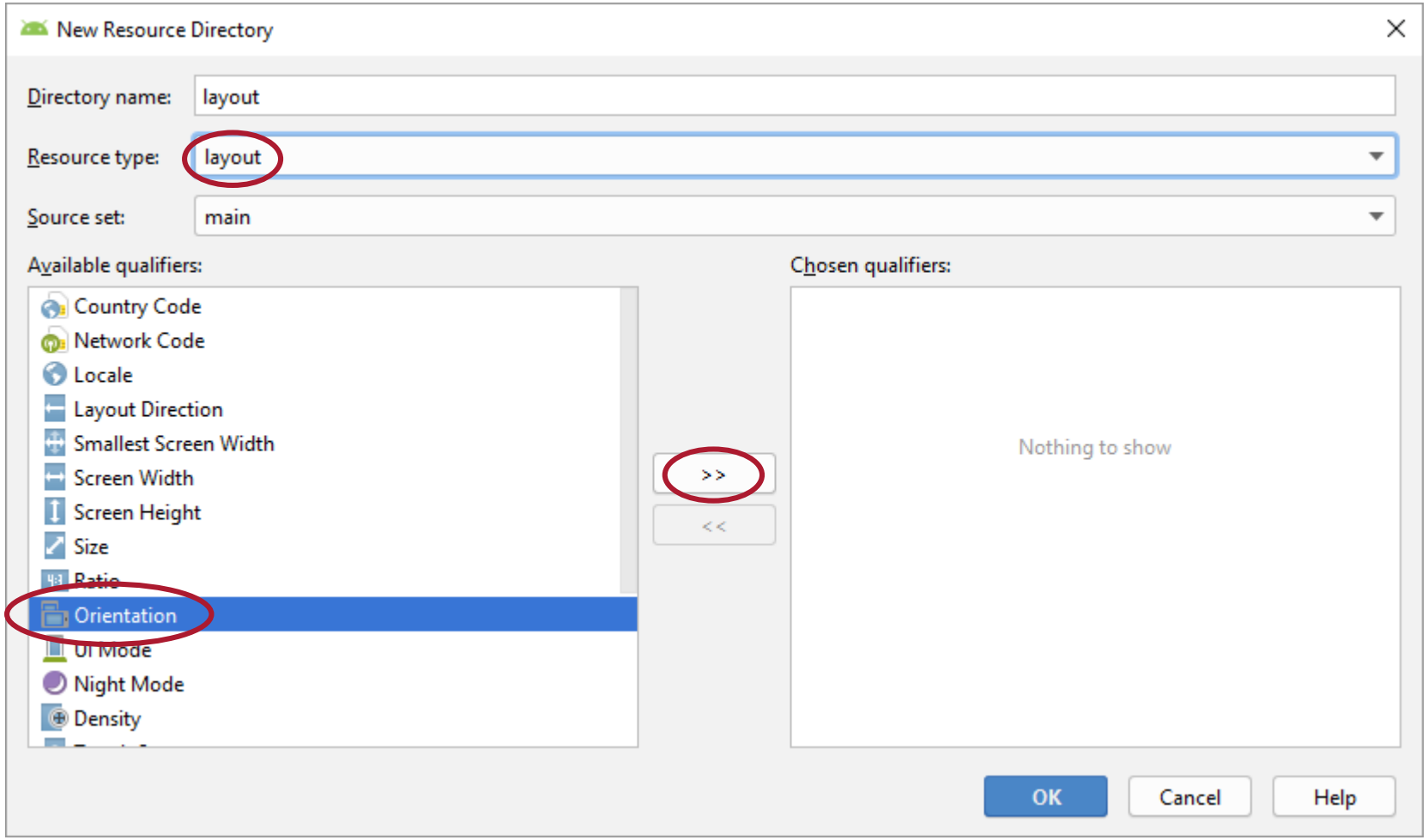
# Adding a Different Layout for Landscape Orientation

- Portrait layout used so far was stored in **`res/layout`**
- Android expects landscape layout in **`res/layout-land`**
- To create, right-click **`res`** folder; choose “New > Android Resource Directory”



# Adding a Different Layout for Landscape Orientation

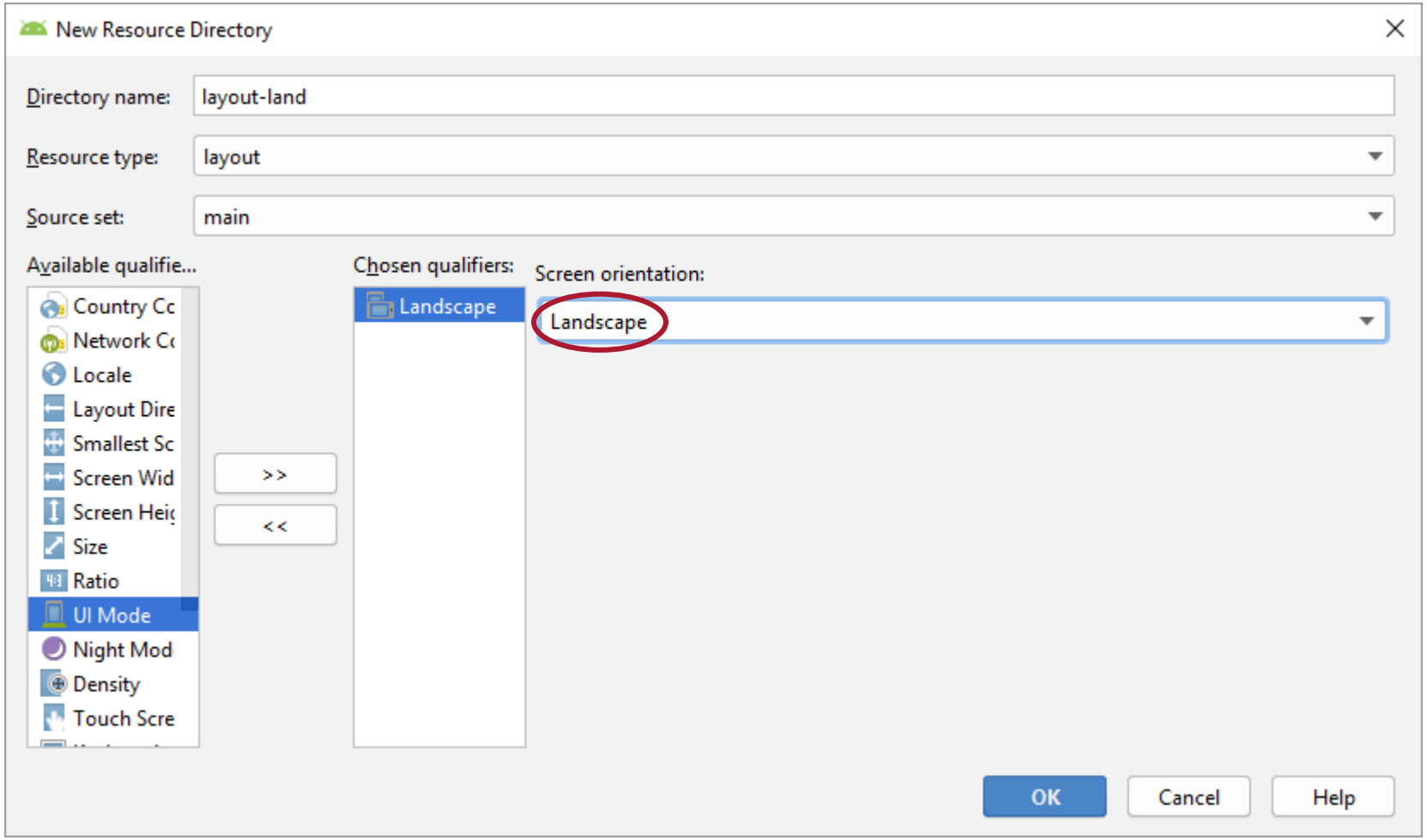
- Choose resource type “layout”
  - Note directory name is updated automatically
- Select qualifier “Orientation” and click “>>” button



# Adding a Different Layout for Landscape Orientation

- Select screen orientation “Landscape”
  - Note directory name is updated accordingly again
- Click OK button

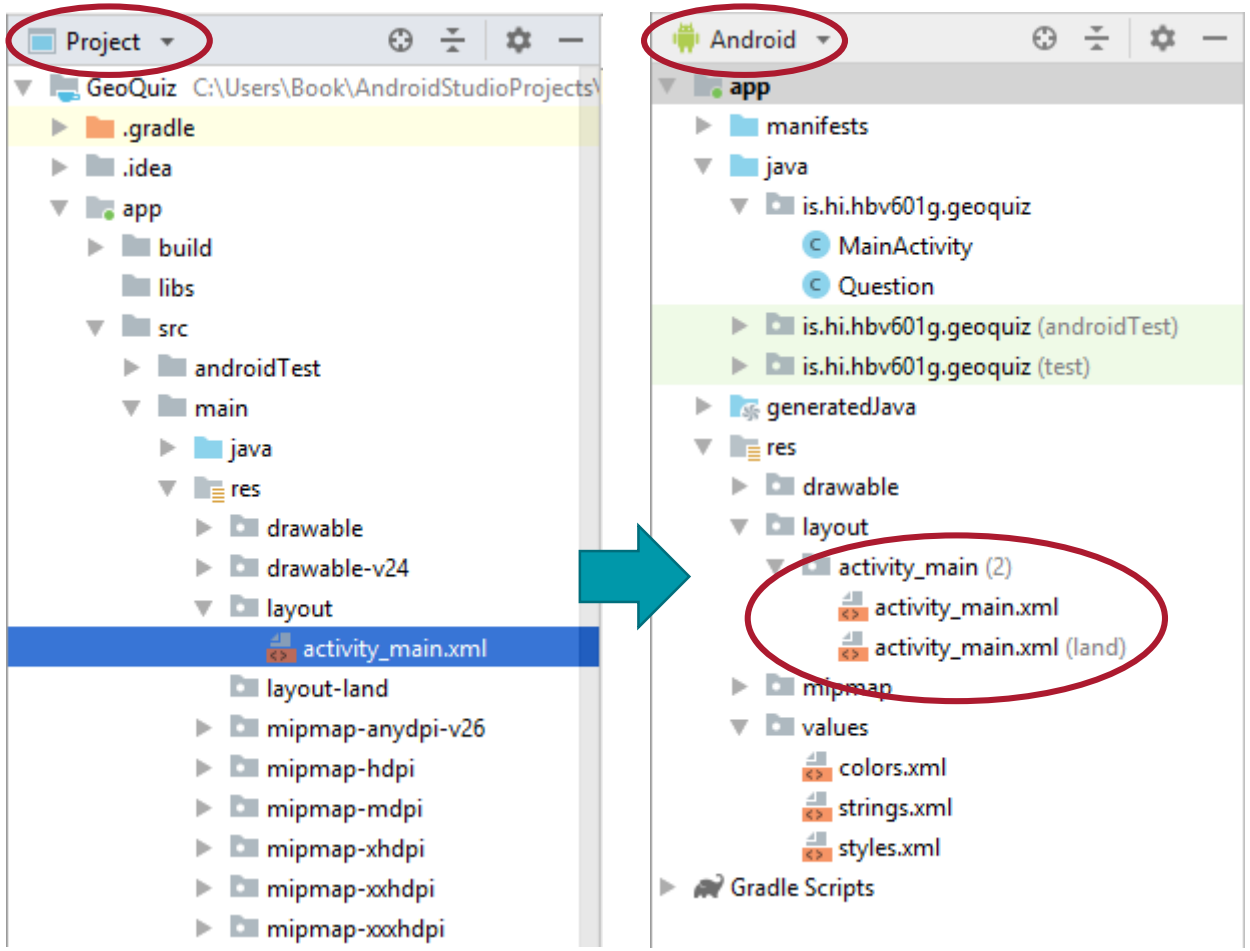
- This will create the additional folder **`res/layout-land`** beside the original **`layout`** resource folder



# Configuration Qualifiers

- **`-land`** suffix is one of many **configuration qualifiers** 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

- Note: New folder doesn't show up in IDE's *Android* perspective while empty
  - Use *Project* perspective to copy layout file **`activity_main.xml`** from **`layout`** folder to **`layout-land`** folder



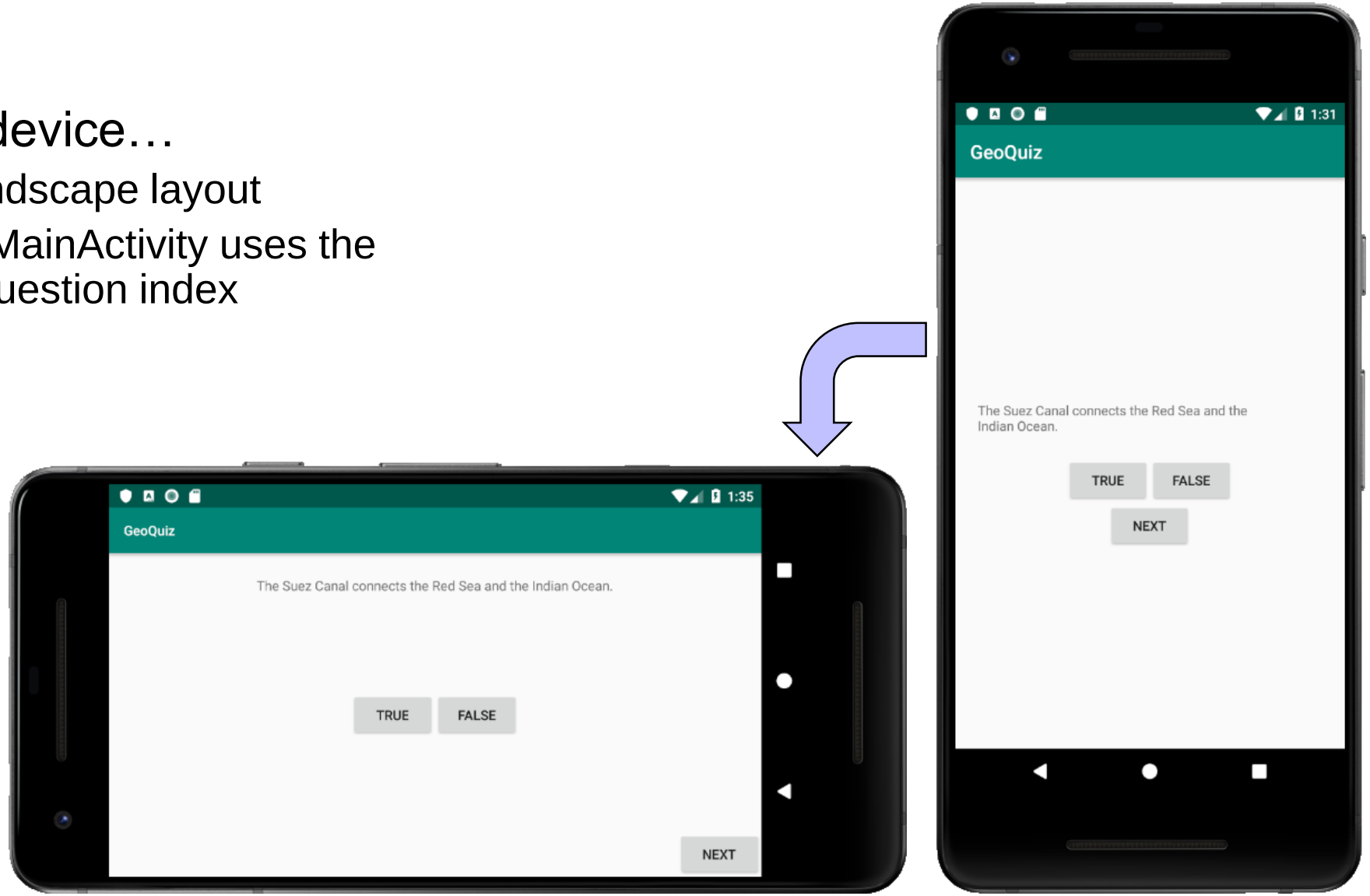
# Defining a Landscape Layout (layout-land/activity_main.xml)

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

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp"/>

    <LinearLayout
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal">

        <Button android:id="@+id/true_button" .../>
        <Button android:id="@+id/false_button" .../>

    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"/>

</FrameLayout>
```

Indicates where this view should be placed in enclosing view

# Testing the Revised Activity and New Layout

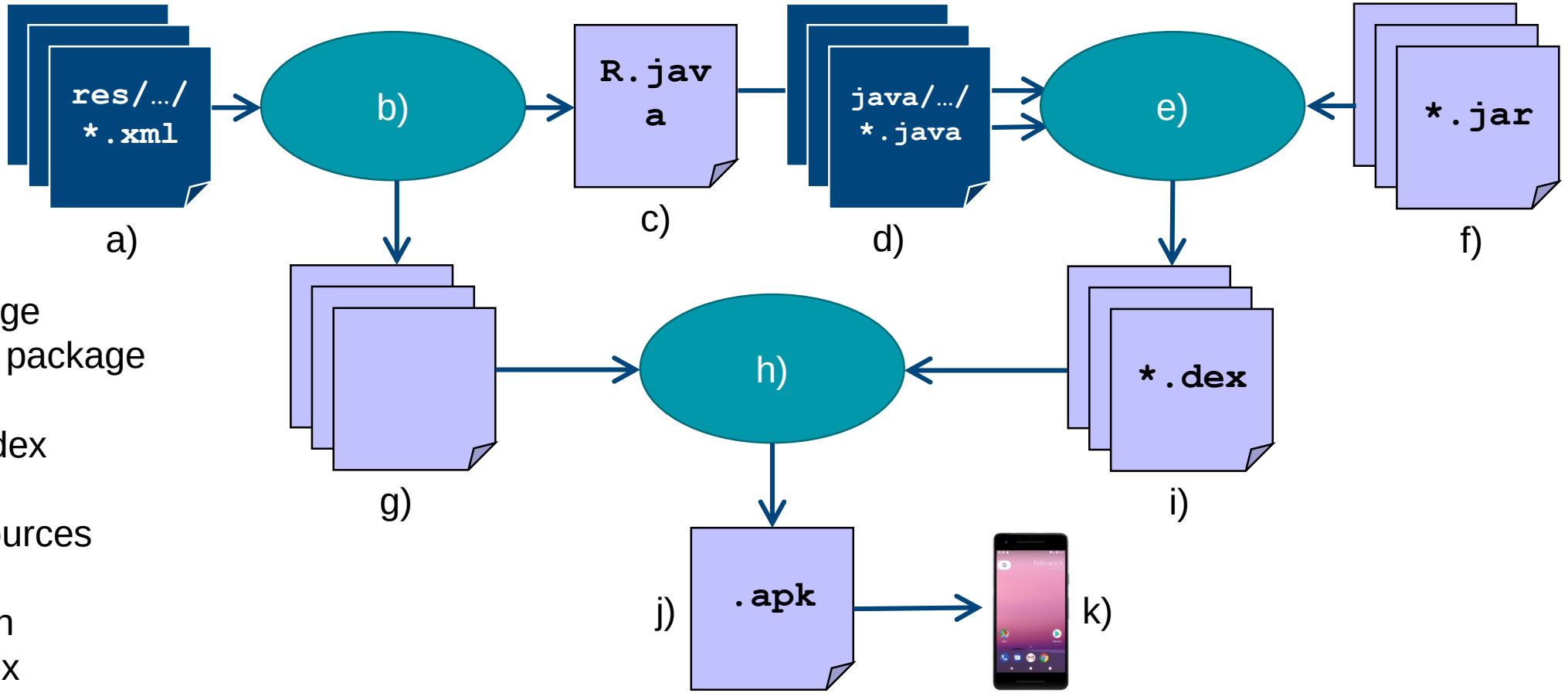
- When rotating the device…
  - ✓ we see the new landscape layout
  - ✓ the newly created MainActivity uses the previously saved question index



# In-Class Quiz #5: The Android Build Process

**Assign the proper labels to the elements of the build process:**

1. Android package
2. Build and sign package
3. Bytecode
4. Collect and index
5. Compile
6. Compiled resources
7. Dependencies
8. Deploy and run
9. Resource index
10. Resources
11. Source code



# Examples and Template Projects

- Most lecture examples are adapted from the book
  - Bill Phillips, Chris Stewart, Kristin Marsicano: Android Programming, 3rd Edition. Big Nerd Ranch, 2016
- Source code of examples is available at
  - https://www.bignerdranch.com/books/android-programming/
    - "Download solutions to the 3rd edition exercises"
- You could use the code examples from the following chapters as templates for your app:
  - Chapter 6 – working with activities, layouts and intents
  - ➢ Chapter 17 – working with fragments, complex views, DBs…
  - Chapter 22 – working with multimedia assets
  - ➢ Chapter 30 – working with web services
  - Chapter 31ff – working with gestures, GPS, animations etc.