# Android Development

## Chapter 2 – Activities, Fragments and Intents

katholieke hogeschool
associatie KU Leuven
vives

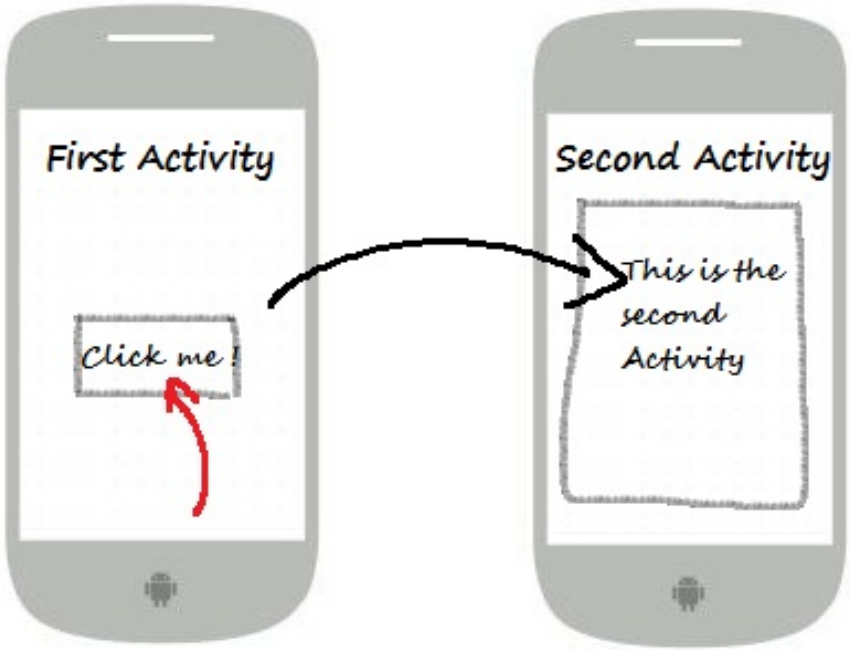
# Android Development
# Chapter 2 – Activities, Fragments and Intents

## Activities

katholieke hogeschool
associatie KU Leuven
vives

# Activities

- An activity is a window that contains the user interface of your application
- An application can have any number of activities
- Main purpose is user interaction



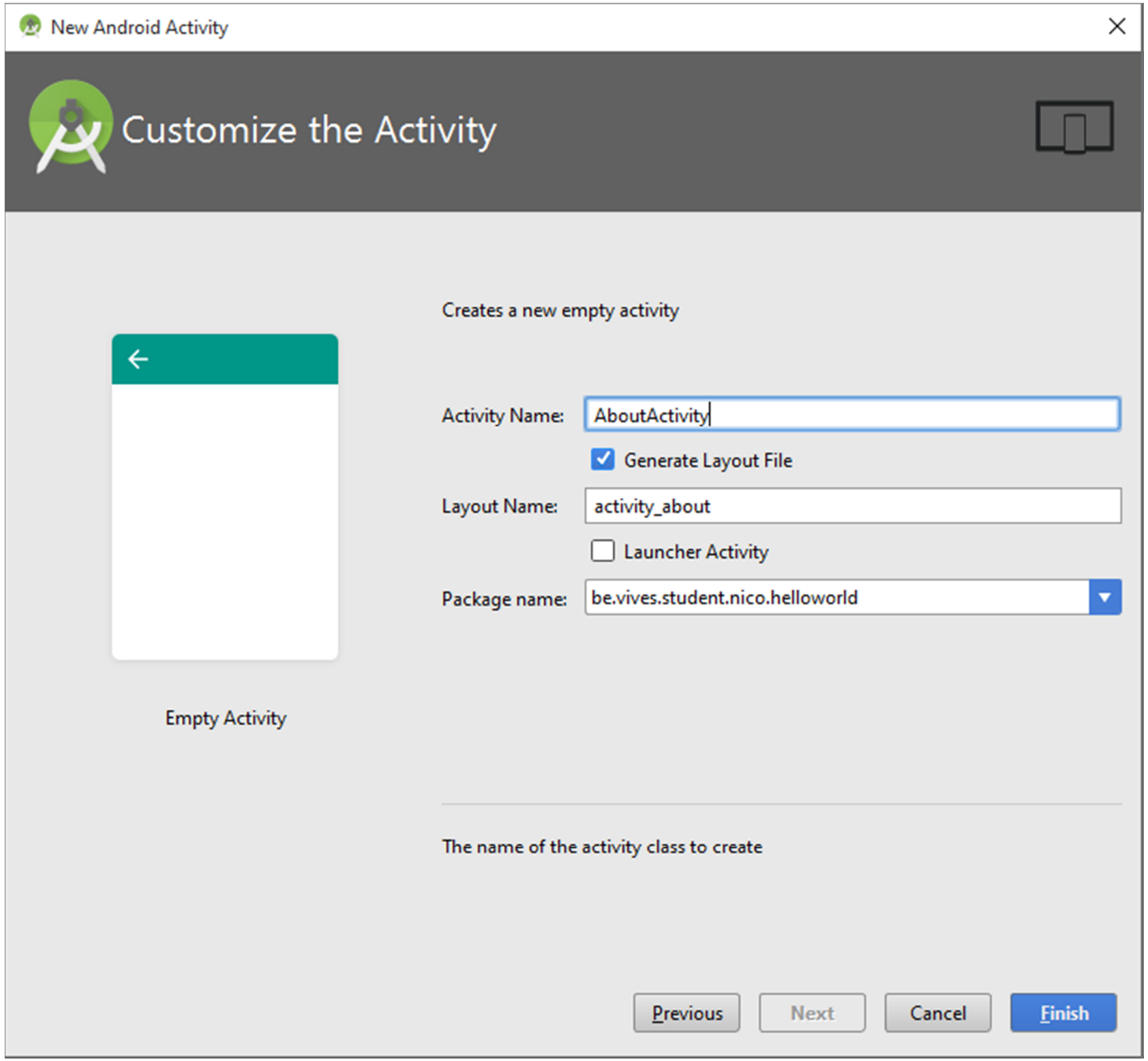
# Creating a new Activity

- To create a new activity in Android Studio all you need to do is go to

```
File => New => Activity => Gallery
```

- Choose one of the templates and give the activity a name
  - For starters always choose the "Empty Activity" or "Blank Activity"
  - Always add the suffix "Activity" to the name

# Creating a new Activity

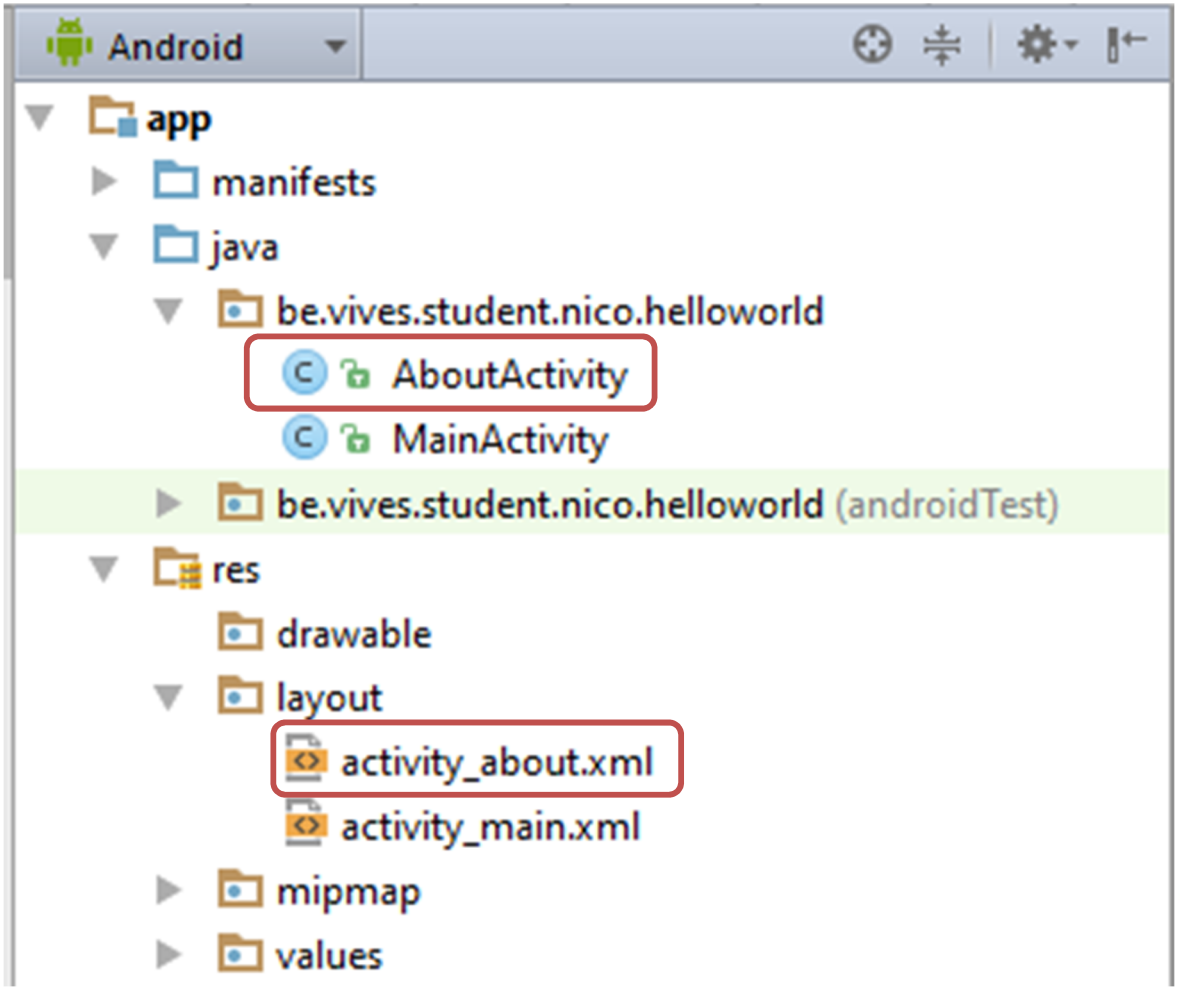


# Creating a new Activity

- Android Studio will automatically create a Java source code file and Layout markup file for you

Java activity source code file

Layout markup file (XML)



# Android Development
# Chapter 3 – Getting to know the Android User Interface

## Application and Activity Templates

katholieke hogeschool
associatie KU Leuven
vives

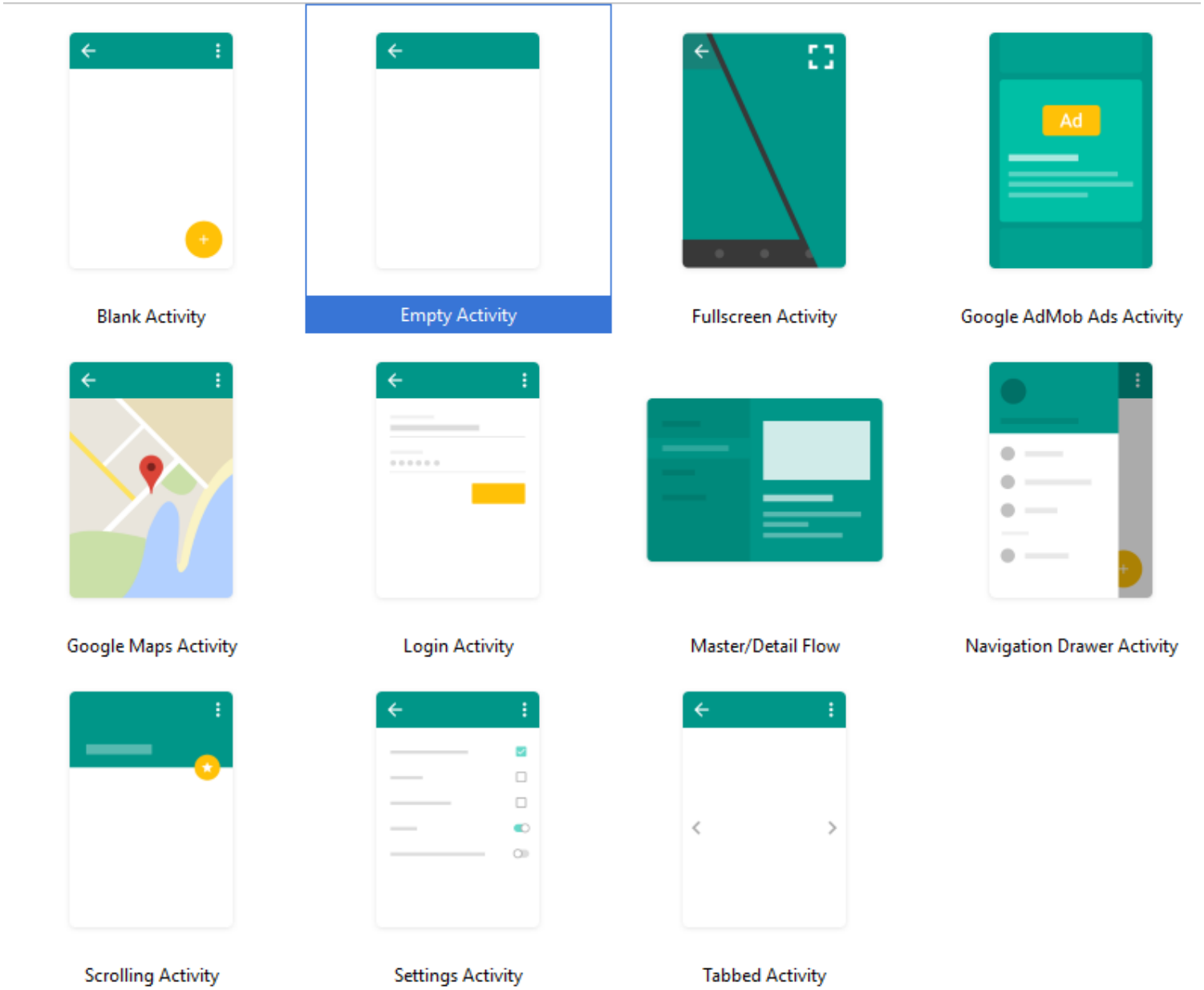
# Application Templates

- Application templates create basic Android applications that you can immediately run and test on your Android device.

- These templates are available when you create a new Android project
- When you add new activities to an existing project you are also able to select an activity template
  - Some pre-generated code to get you started

- We will mostly create an application based on the "Blank Activity" template

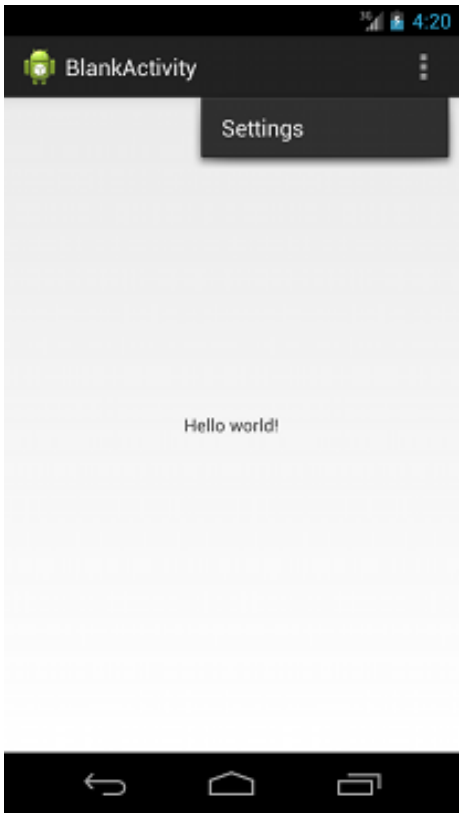
# Application Templates

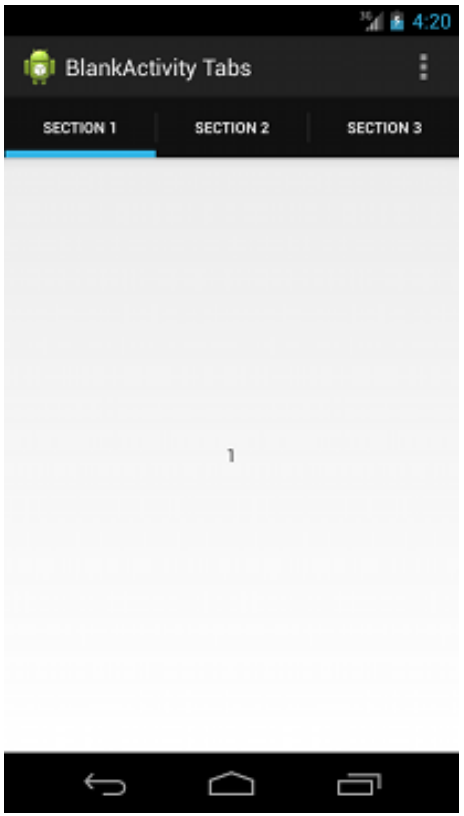



katholieke hogeschool
associatie KU Leuven
vives

# Blank Activity Template

- The Blank Activity template creates a simple application that follows the Android Design guidelines.
  - http://developer.android.com/design/index.html
  - Use this template to create a basic app as a starting point for your project.



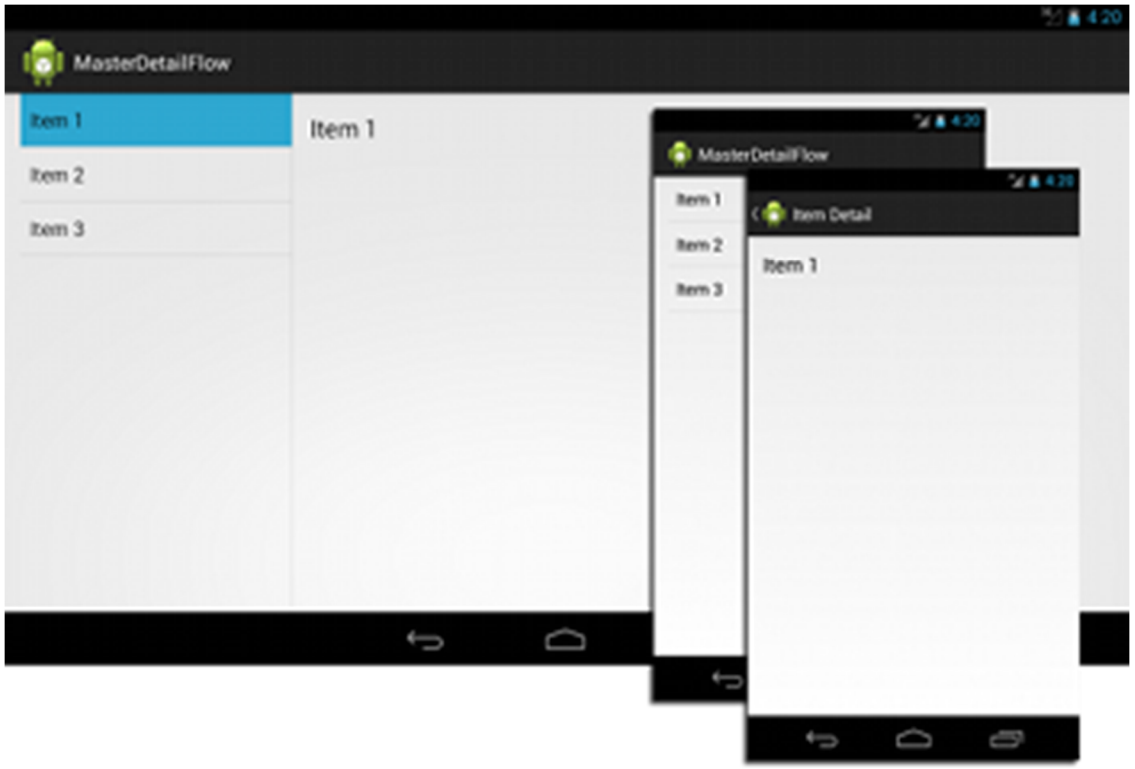




# Full Screen Activity Template

- This template provides an implementation of an activity which alternates between a primary, full screen view and a view with standard user interface controls, including the notification bar and application title bar.
  - The full screen view is the default and a user can activate the standard view by touching the device screen.

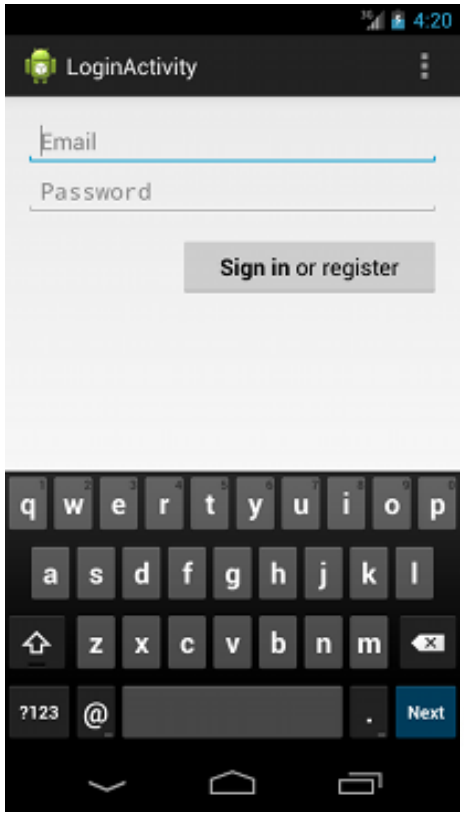
# Master Detail Flow Template

- This template creates an adaptive layout for a set of items and associated details.
  - On a tablet device, the item list and item details are displayed on the same screen.
  - On a smaller device, the list and details are displayed on separate screens.



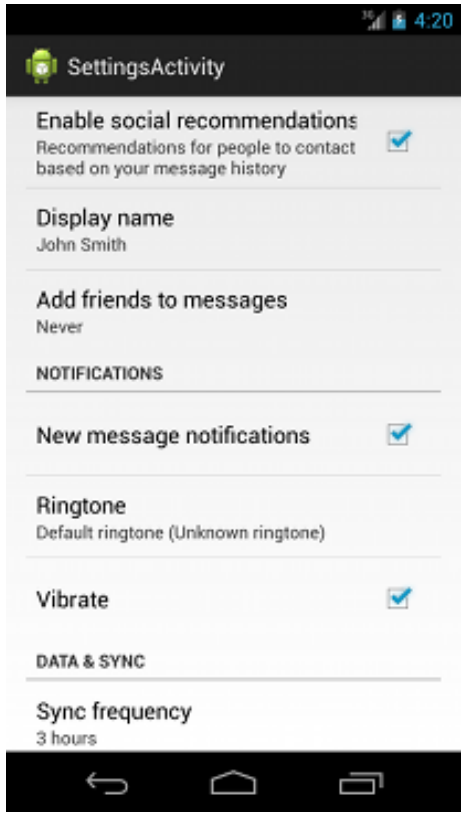
# Login Activity Template

- The Login Activity template provides input fields and a sample implementation of an AsyncTask that asks users to login or register with their credentials.
  - In this case the implementation of AsyncTask handles network operations separately from the main user interface thread
    - Keeps the UI responsive



# Settings Activity Template

- The Settings Activity template extends the PreferenceActivity class and uses an XML file to create preference settings.
- This template also demonstrates how to implement several data types for settings.



# Android Development
# Chapter 2 – Activities, Fragments and Intents

## Blank Activity Boilerplate Code

katholieke hogeschool
associatie KU Leuven
vives

# Boilerplate Code

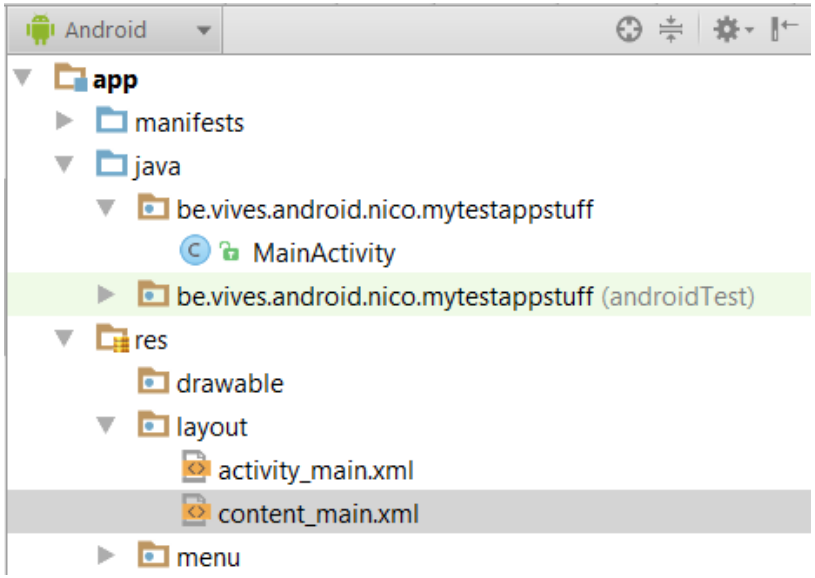
- When creating a new activity most of the start code is generated by the IDE

```
package be.vives.android.nico.testapp1;

import ...

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // ...
    }
}
```

- Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View).
  - The activity loads its UI component from the XML layout files

# Boilerplate Code

- The Layout can be found as two files under res/layout
  - content_main.xml
    - Which contains our actual layout we want to display to the user
  - activity_main.xml
    - Which acts a container for the content and some other things like a toolbar
    - It also defines the general layout of the activity



# Boilerplate Code

## content_main.xml

- In the content_main.xml layout XML file you will find the following starting code
  - It shows a TextView component inside a Layout container
    - More on layout containers later …

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main" tools:context=".MainActivity">

    <TextView
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

# Boilerplate Code

## activity_main.xml

- activity_main.xml acts as a container for the content and some other things like a toolbar
  - For the moment this is less important for us

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

    <android.support.design.widget.AppBarLayout ... >

        <android.support.v7.widget.Toolbar ... />

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main" />

    <android.support.design.widget.FloatingActionButton ... />

</android.support.design.widget.CoordinatorLayout>
```

# Boilerplate Code

## AndroidManifest.xml

- The activity also needs to be defined as an XML node inside the Manifest file

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

    <application
        ... >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

*The name of the class that implements the activity, a subclass of Activity. The attribute value should be a fully qualified class name (such as, "com.example.project.MyActivity"). However, as a shorthand, if the first character of the name is a period (for example, ".MyActivity"), it is appended to the package name specified in the <manifest>.*

# Boilerplate Code

## AndroidManifest.xml

- Notice the @string/app_name reference
  - This mechanism allows us to put ALL strings inside an XML file
    - Can be found in res/values/strings.xml
    - Easy to change and reuse text
    - Also faster and easier to translate application

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

    <application
        ... >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar" >
            <intent-filter>
                ...
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Let's Try It Out


- Let's create a new application called "CoolBrowser" based on the "Blank Activity" template

- Create a new activity called "AboutActivity" (also based on the "Blank Activity" template)
  - Let's add an about title to it
    - Make sure to use a string-value reference
  - Let's also add a small text giving some general info about our app

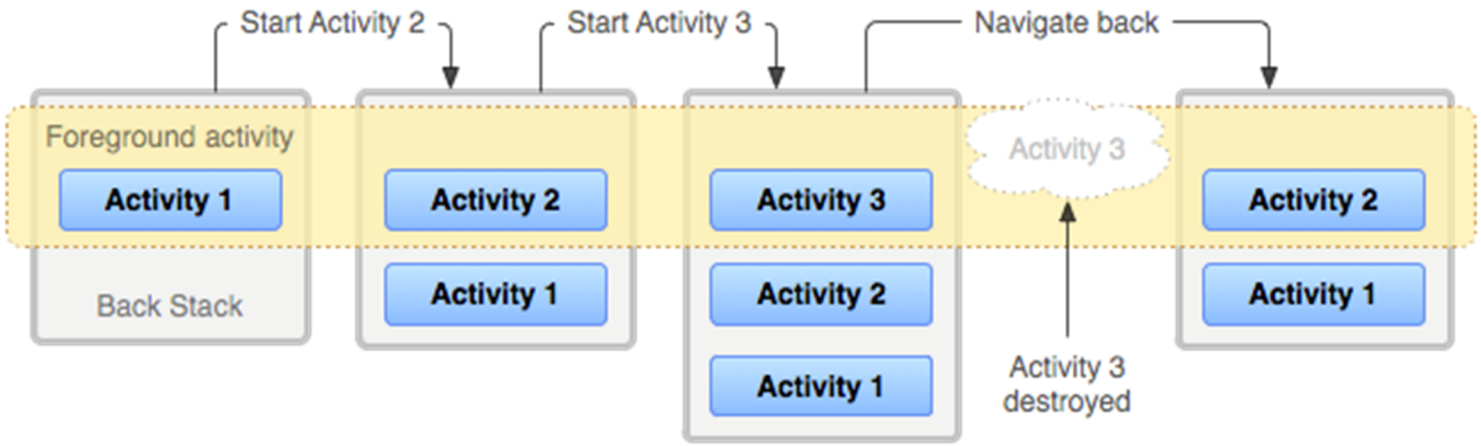
# Android Development
# Chapter 2 – Activities, Fragments and Intents

## Activity's lifecycle

katholieke hogeschool
associatie KU Leuven
vives

# Activity's lifecycle

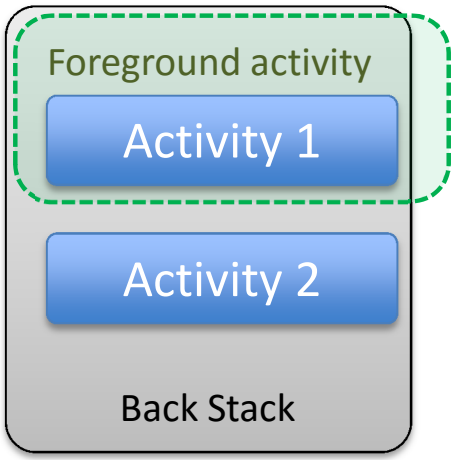
- Activities in the system are managed as an activity stack.
- When a new activity is started, it is placed on the top of the stack and becomes the running activity
  - The previous activity remains below it in the stack
  - It will not come to the foreground again until the new activity exits




  - When you press the back button the current activity is destroyed
    - The stack's top activity is shown

# Activity's lifecycle

- An activity has essentially four states
  - If an activity in the foreground of the screen (at the top of the stack), it is active or running.



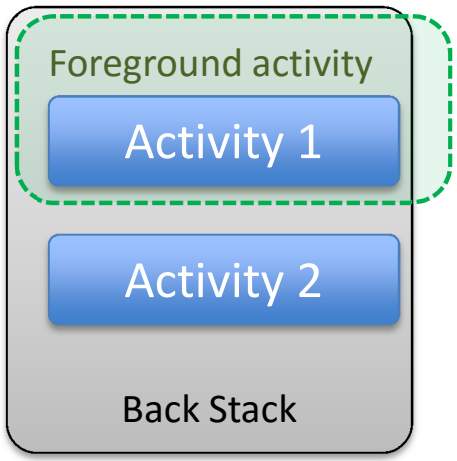

  - In this example "Activity 1" is running / alive

# Activity's lifecycle

- An activity has essentially four states
  - If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused.
    - A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager)
    - However it can be killed by the system in extreme low memory situations.

# Activity's lifecycle

- An activity has essentially four states
  - If an activity is completely obscured by another activity, it is stopped.
    - It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.



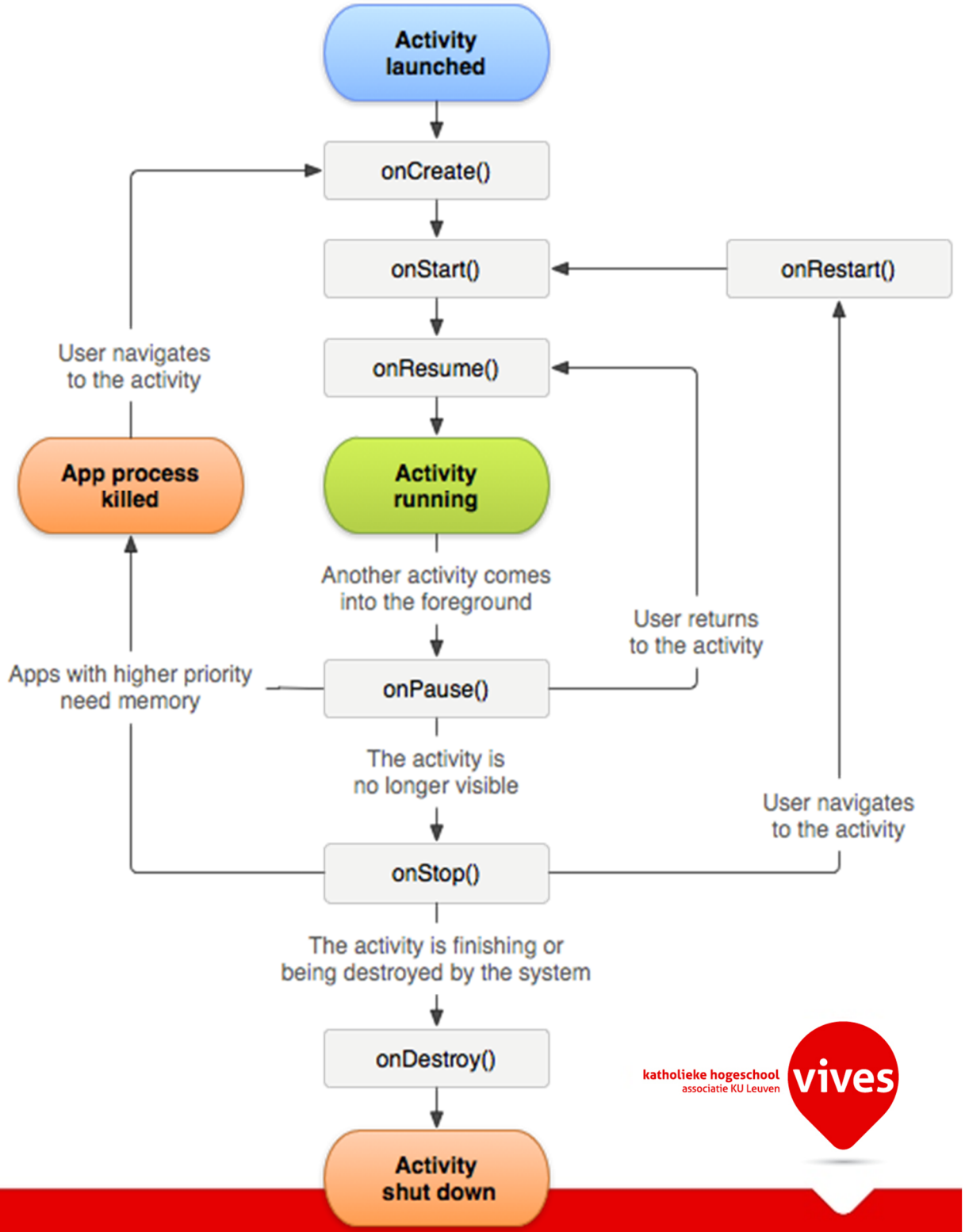

  - In this example "Activity 2" is stopped

# Activity's lifecycle

- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process.
  - When it is displayed again to the user, it must be completely restarted and restored to its previous state.
  - It is your task as a programmer to save the state of the activity before it is killed
    - Android provides callback methods which can be used for this purpose

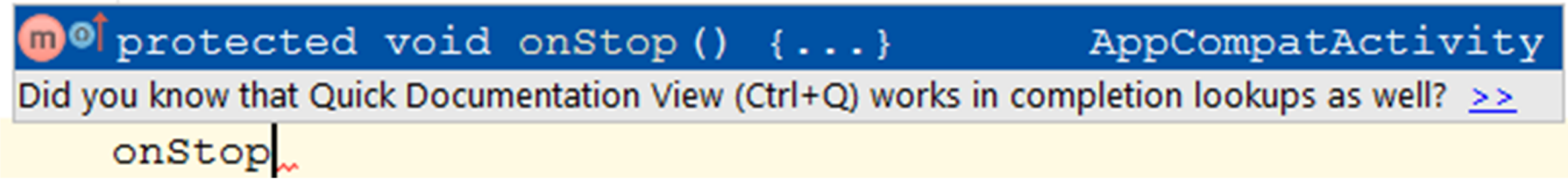
# State Paths of an Activity

- The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states.
- The colored ovals are major states the Activity can be in.



# State Paths of an Activity

- A simple application to learn to know all the stages of an activity's lifecycle is just overriding all the lifecycle methods and outputting a message to the log
  - We will do this in the lab
  - When using Android Studio as an IDE just go to the activities java file and type the name of the method (*onStop* for example)
    - Code completion will suggest to override the base method
    - Select the correct one and hit TAB or ENTER to insert the snippet




- If code completion is lost or does not kick in try to instantiate it using the CTRL-SPACE

# Activity's lifecycle

- Crucial to know is that an Activity can be destroyed by the OS or the user at any time the activity loses focus
  - Hence you will need to provide the necessary code to save the state of your application when it is destroyed.

- General guidelines
  - Use onCreate() to create and instantiate objects that your application will be using
  - Use onResume() to start any services or code that needs to run while your activity is in the foreground
  - Use onPause() to stop any services or code that does not need to run when your application is in the background
  - Use onDestroy() to free up resources before your activity is destroyed

# Let's Try It Out


- Let's add a console message to the onCreate() method of our MainActivity of CoolBrowser

```
Log.v("TAG", "My message");    // Verbose
Log.w("TAG", "My message");    // Warning
Log.e("TAG", "My message");    // Error
Log.d("TAG", "My message");    // Debug
```

  - Create filter to catch the messages with our tag

- Override the onPause() method and also log message to console

- Try messing around with the app and check when the onPause() and onCreate() methods are called

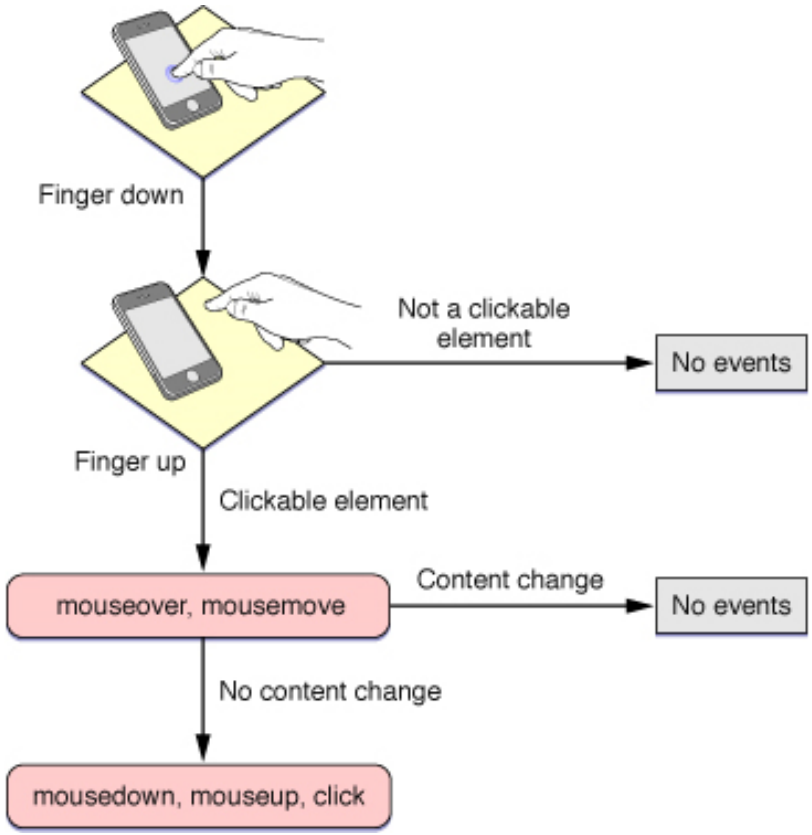
# Android Development
# Chapter 2 – Activities, Fragments and Intents

## Event Handlers

katholieke hogeschool
associatie KU Leuven
vives

# Event Handlers

- An event handler is method that is called when a particular event takes place
- Most typical example is the method that is called when a certain button is pressed



# Event Handlers

- An example of a button click event handler could be a Toast (small message)

```
public void onAboutClick(View view) {
    Toast.makeText(this, "Should launch the about Activity",
        Toast.LENGTH_SHORT).show();
}
```

  - This code will show a small message at the bottom (called a Toast in Android) when the button is clicked.
  - This is also called an event handler
    - Note the argument which is the View (graphical element) that caused the event handler to be fired
    - Always public and void (no return value)
  - You can use the ALT-ENTER shortcut key to import libraries when necessary

# Event Handlers

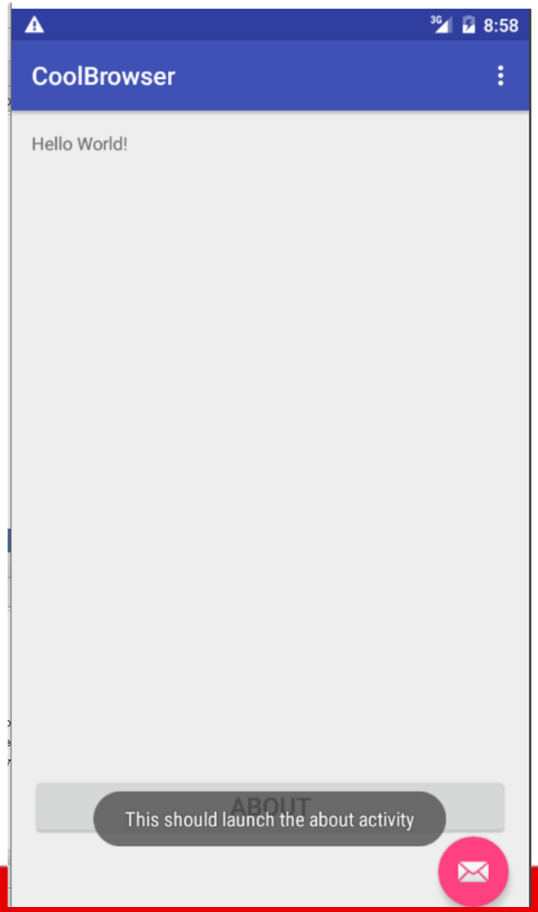
- While we did create a method with code to be executed when the button is pressed there is no connection between the method and the actual button
  - This needs to be done in the markup code of the button (in the XML file)
  - We call this registering an event handler

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="About"
    android:id="@+id/btnAbout"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:onClick="onAboutClick"/>
```

  - Notice that we need to specify the name of the method for the "onClick" property (case sensitive !)
    - Tab completion will be your best friend if you allow it to be !!!!

# Let's Try It Out


- Add a button to the Main Activity of the application with the text "About"
- Add an event handler to show a toast message
- Register the event handler with the about button on the main activity



# Android Development
# Chapter 2 – Activities, Fragments and Intents

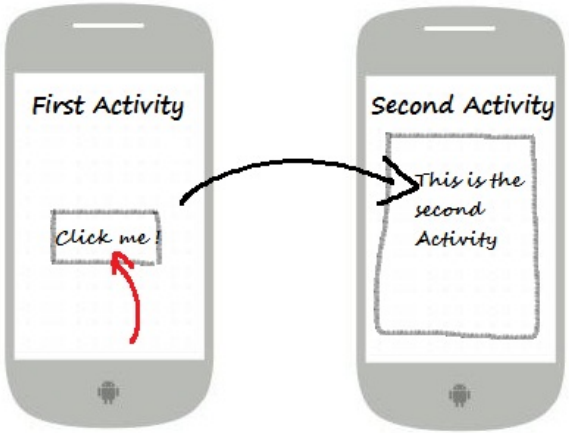
## Intents

katholieke hogeschool
associatie KU Leuven
vives

# Intents

- An Intent is a messaging object you can use to request an action from another app component.

- There are three fundamental use-cases
  - To start an activity:
    - An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity().
    - The Intent describes the activity to start and carries any necessary data.



# Intents

- To start a service:
  - A Service is a component that performs operations in the background without a user interface.
  - You can start a service to perform a one-time operation (such as download a file) by passing an Intent to startService().
  - The Intent describes the service to start and carries any necessary data.

- To deliver a broadcast:
  - A broadcast is a message that any app can receive.
  - The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.
  - You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

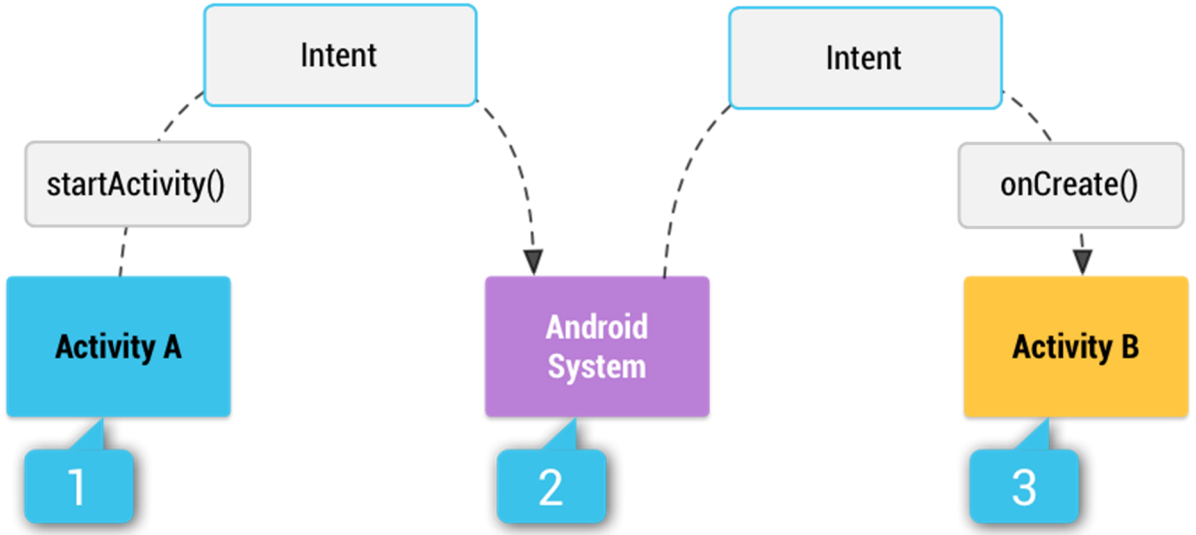
# Intent Types

- There are two types of intents:
  - Explicit intents
    - Specify the component to start by name (the fully-qualified class name).
    - You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.
    - When you create an explicit intent to start an activity or service, the system immediately starts the app component specified in the Intent object.
  - Implicit intents
    - Do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
    - For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

# Intent Types

- Implicit intents
  - When you create an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device.
  - If the intent matches an intent filter, the system starts that component and delivers it the Intent object.
  - If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

# Intent Types – Implicit Intents




- Example
  - Activity A creates an Intent with an action description and passes it to startActivity().
  - The Android System searches all apps for an intent filter that matches the intent.
  - When a match is found, the system starts the matching activity (Activity B) by invoking its onCreate() method and passing it the Intent.

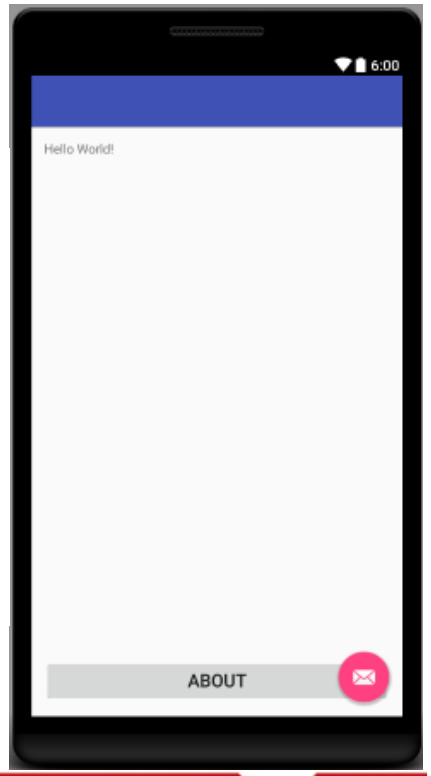
# Linking Activities using Intents

- An android device can have any number of activities.
- When your application has more than one, you most likely want to navigate from one activity to another.
  - This can be achieved using intents.
- Suppose a simple application with a MainActivity and an AboutActivity

- Let's say we have an about button

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="About"
    android:id="@+id/btnAbout"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:onClick="onAboutClick"/>
```

Method name of event handler



# Linking Activities using Intents

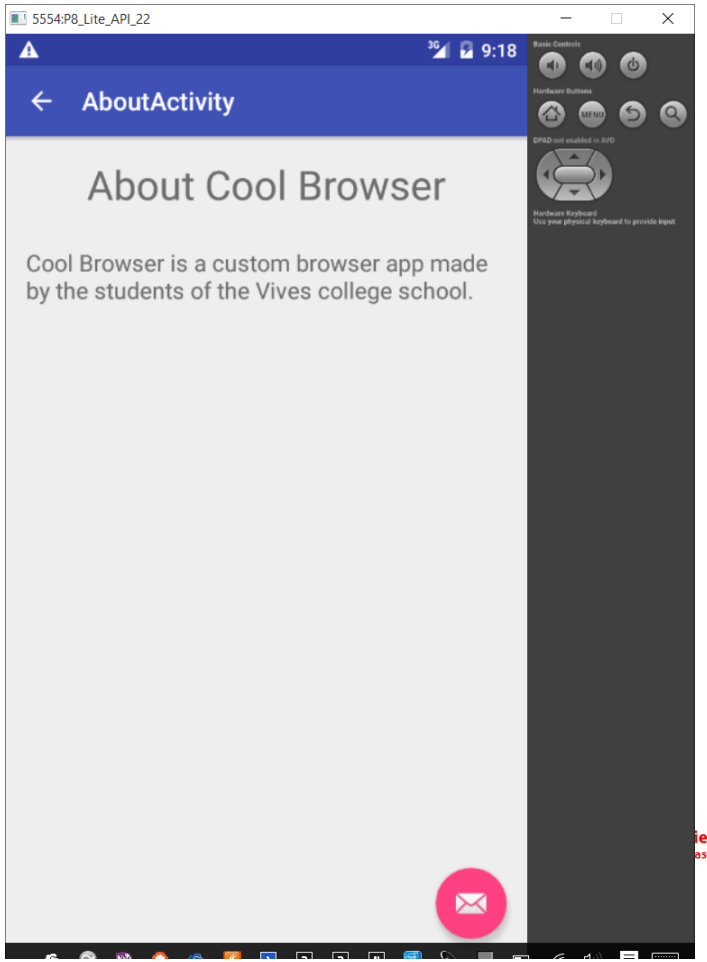
- If you do not specify any intent filters for your activity then it cannot be invoked using implicit intents
  - In this case you will need to use explicit intents to start the activity inside your own application
  - As the about activity should only be launched from inside our application we do not need to create an intent filter

```
// Launch activity by creating an explicit intent
Intent about = new Intent(this, AboutActivity.class);
startActivity(about);
```

  - To start the activity we use the startActivity(Intent intent) method

# Let's Try It Out


- Launch the AboutActivity when the user presses the about button

# Android Development
# Chapter 2 – Activities, Fragments and Intents

## Responding to Implicit Intents

katholieke hogeschool
associatie KU Leuven
vives

# Responding to Implicit Intents

- Earlier, you saw how an activity can invoke another activity using the Intent object.
- In order for other activities to invoke your activity, you need to specify the action and category within the <intent-filter> element in the AndroidManifest.xml file
- Let's create an activity called BrowserActivity and the following intent-filter

```
<activity
    android:name=".BrowserActivity"
    android:label="@string/title_activity_browser"
    android:parentActivityName=".MainActivity"
    android:theme="@style/AppTheme.NoActionBar" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="be.vives.android.examples.nico.coolbrowser.MainActivity" />
    <intent-filter>
        <action android:name="be.vives.android.examples.nico.coolbrowser.MyBrowser" />
        <action android:name="android.intent.action.view" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

# Responding to Implicit Intents

- An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
- For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent.
- Likewise, if you do not declare any intent filters for an activity, then it can be started only with an explicit intent

# Responding to Implicit Intents

- The following allows your activity to respond to a general ACTION_VIEW intent

```
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
```

- The data node states that the intent data should adhere to the http-scheme

```
<data android:scheme="http" />
```

  - In other words, the data attached to the intent object should start with “http://”

- To allow this to work we need to give the application permission to access the internet
  - Place the following permission node on the same level as the application node in the manifest file

```
<uses-permission android:name="android.permission.INTERNET"/>
```

katholieke hogeschool associatie KU Leuven vives

# Responding to Implicit Intents

- Add a Webview component to the BrowserActivity GUI

```
<WebView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/webView"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentEnd="true" />
```

# Responding to Implicit Intents

- We also need to extend the WebViewClient class and override the shouldOverrideUrlLoading() method

```
public class BrowserActivity extends AppCompatActivity {

    private class WebViewCallback extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            // Allows the app to interfere with loading of a url
            return false;
            // return true means the host application handles the url
            // return false means the current WebView handles the url
        }
    }
...
```

- This is a private inner class inside BrowserActivity

# Responding to Implicit Intents

- In the onCreate() method of BrowserActivity we need to get the data (the url of the site) from the intent

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

    // Get the URL
    Uri url = getIntent().getData();

    // Get the WebView
    WebView webview = (WebView)findViewById(R.id.webView);

    // Specify a WebViewClient
    webview.setWebViewClient(new WebViewCallback());

    // Load up the URL
    webview.loadUrl(url.toString());
}
```

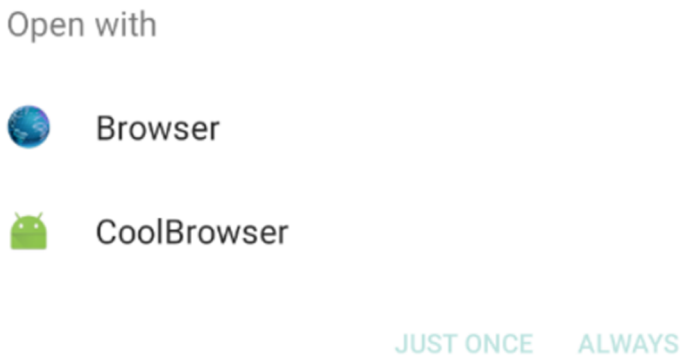
*More info on http://developer.android.com/reference/android/webkit/WebViewClient.html*

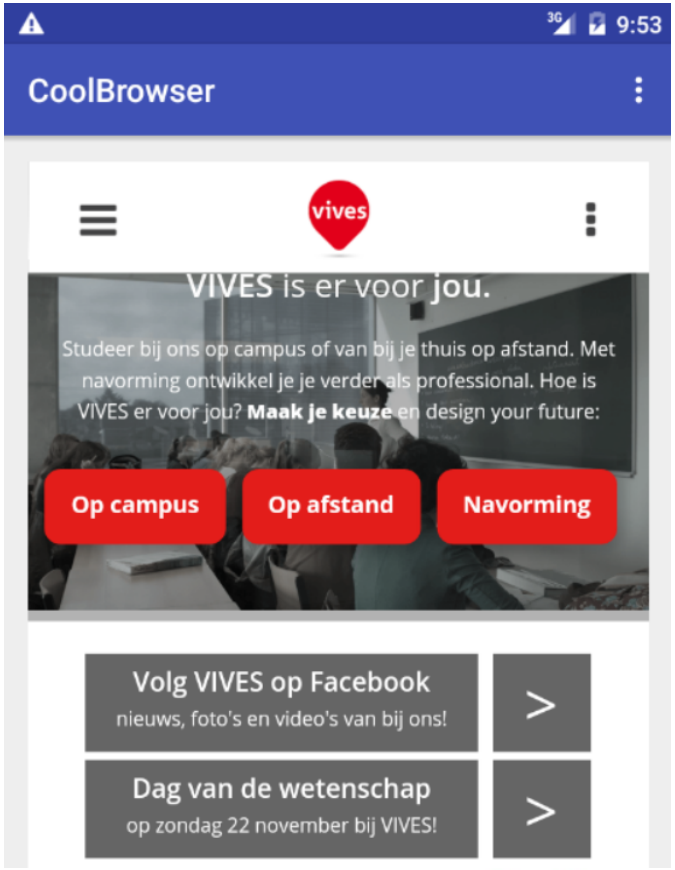
# Let's Try It Out


- Create a blank activity called BrowserActivity
- Add an intent filter to the BrowserActivity in the manifest file
- Also add permission for internet use to the manifest file (same level as application)
- Add a webview client to the GUI
- Create inner class for callback
- When the browser activity is created we need to fetch the url from the intent and give it to the webview

# Let's Try It Out


- Now how do we test this ??
- Quickest solution is to send sms message to invalid number with url in the message
  - You will be able to click it and you should get the opportunity to launch your browser



# Android Development
# Chapter 2 – Activities, Fragments and Intents

Passing Data to an Activity using an Intent Object

&

Sending an Implicit Intent

katholieke hogeschool
associatie KU Leuven
vives

# Passing Data using an Intent Object and Generating an Implicit Intent

- It is common to pass data to an activity.
- This can be done by using the Intent object to pass the data to the target activity.
  - Very similar to the way we pass data back from an Activity

- Data can also be added and retrieved from an intent using the setData() and getData() methods

```
// Create the Intent object
String url = "http://www.google.be";
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse(url));

// Verify that the intent will resolve to an activity
if (intent.resolveActivity(getPackageManager()) != null) {
    startActivity(intent);
}
```

# Let's Try It Out


- Add a Plain Text element to the MainActivity UI
- Add a Button element to the MainActivity UI
- Create an event handler for the button and register it with the UI element
  - Get the URL from the Plain Text
  - Create intent
  - Add URL to intent
  - Start the activity