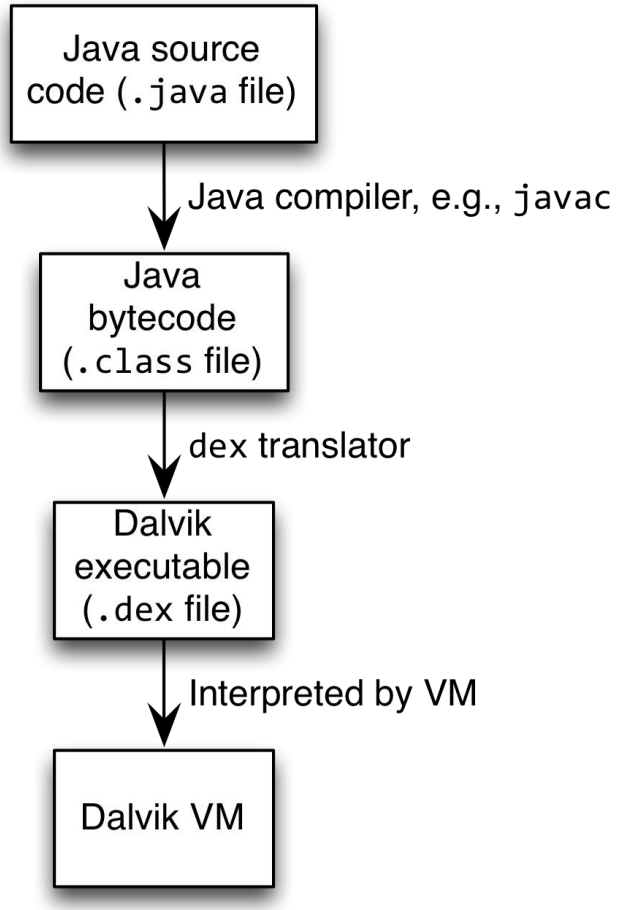
Android Apps

*Important for Class Project

Based on material from Adam Champion, Xinfeng Li, C. Horstmann [1], J. Bloch [2], C. Collins et al. [4], M.L. Sichitiu (NCSU), V. Janjic (Imperial College London), CSE 2221 (OSU), and other sources

Android Highlights (1)

- Android apps execute on Dalvik VM, a "clean-room" implementation of JVM
  - Dalvik optimized for efficient execution
  - Dalvik: register-based VM, unlike Oracle's stack-based JVM
  - Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets
  - Apps developed for Dalvik should work when running with ART. However, some techniques that work on Dalvik do not work on ART.



Android Highlights (2)

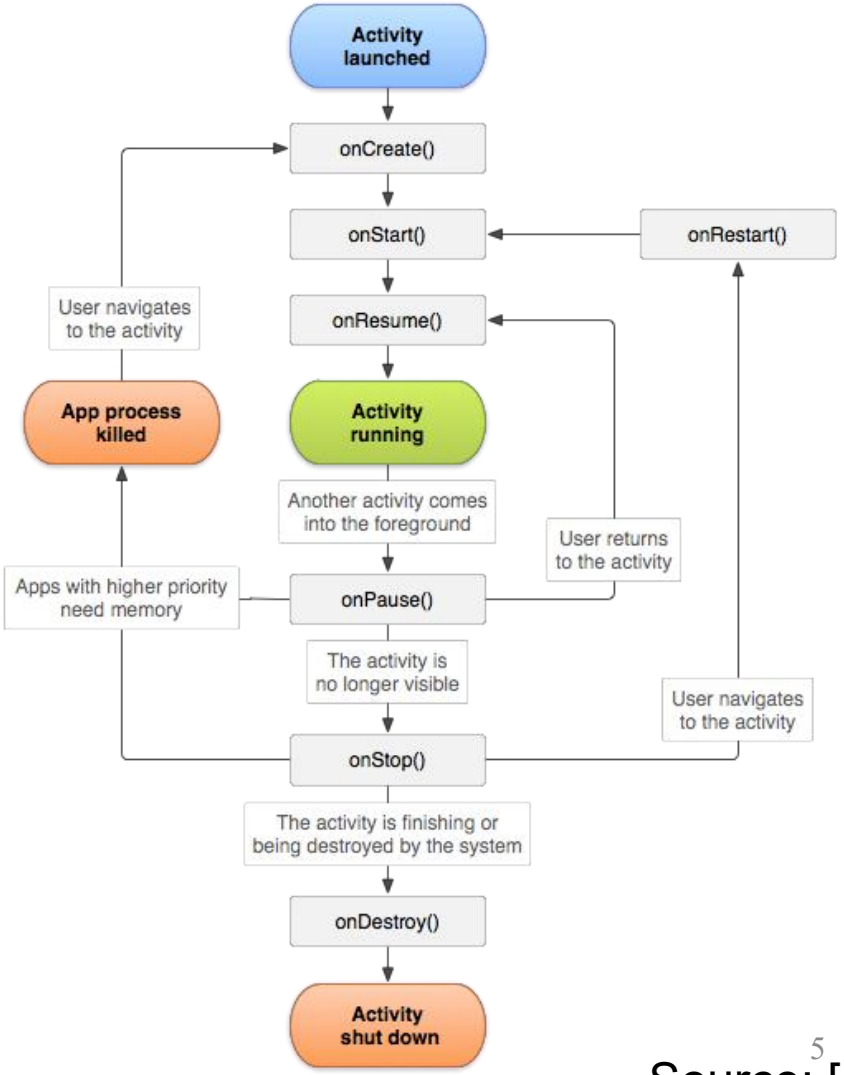
- Android apps written in Java 5, 6 (most Java 7 syntax should work)
  - Actually, a Java dialect (Apache Harmony)
  - Everything we've learned still holds
- Apps use four main components:
  - `Activity`: A "single screen" that's visible to user
  - `Service`: Long-running background "part" of app (*not* separate process or thread)
  - `ContentProvider`: Manages app data (usually stored in database) and data access for queries
  - `BroadcastReceiver`: Component that listens for particular Android system "events", e.g., "found wireless device", and responds accordingly

App Manifest

- Every Android app must include an `AndroidManifest.xml` file describing functionality
- The manifest specifies:
  - App's Activities, Services, etc.
  - Permissions requested by app
  - Minimum API required
  - Hardware features required, e.g., camera with autofocus
  - External libraries to which app is linked, e.g., Google Maps library

Activity Lifecycle

- `Activity`: key building block of Android apps
- Extend `Activity` class, override `onCreate()`, `onPause()`, `onResume()` methods
- Dalvik VM can stop any `Activity` without warning, so saving state is important!
- Activities need to be "responsive", otherwise Android shows user "App Not Responsive" warning:
  - Place lengthy operations in `Runnable Threads`, `AsyncTasks`



Source: [12]

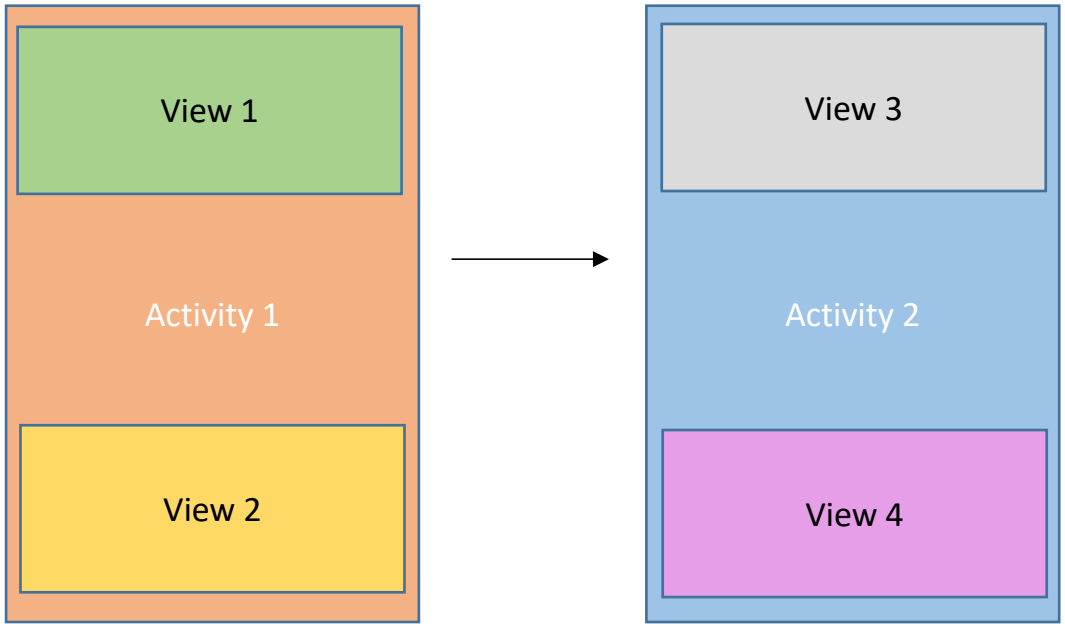
Android Fragment

Slides from **Luca Bedogni & Marco Di Felice**

Dipartimento di Scienze dell'Informazione

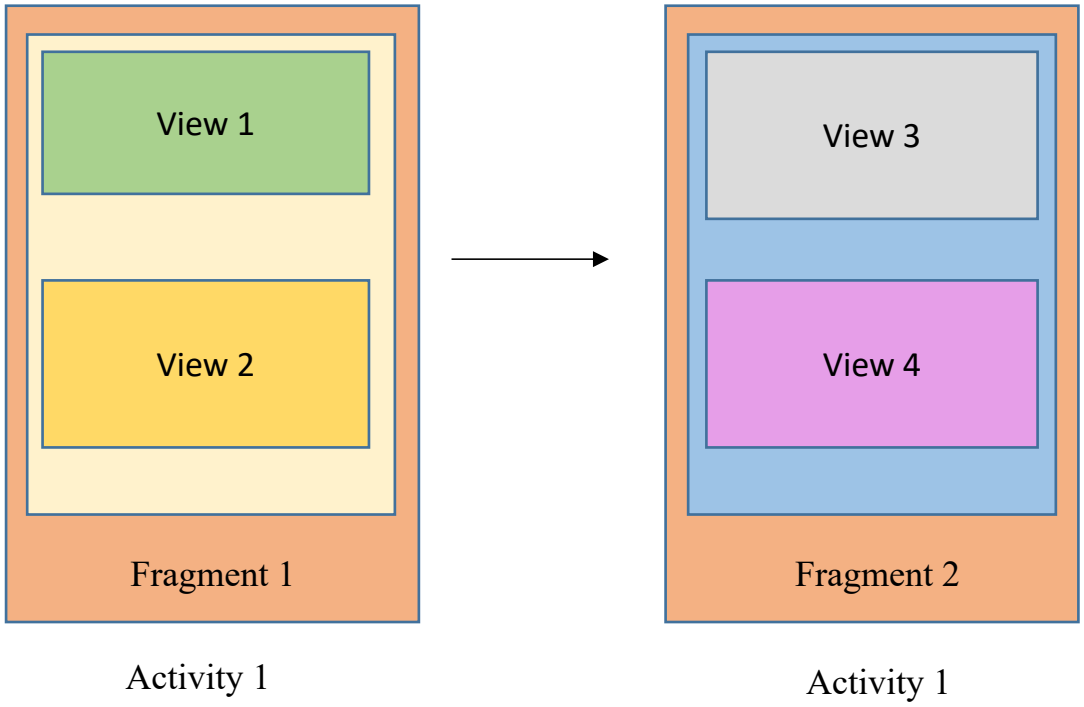
Università di Bologna

What is a Fragment?



Before Android 3.0 ,
If you have to show
View 3 and View 4
you have to
create new activity

What is a Fragment?



Now, just insert fragment or delete it and insert another fragment.

Fragments are so much customizable.

Android: Fragments

Fragment → A portion of the user interface in an Activity.

Introduced from **Android 3.0** (API Level 11)

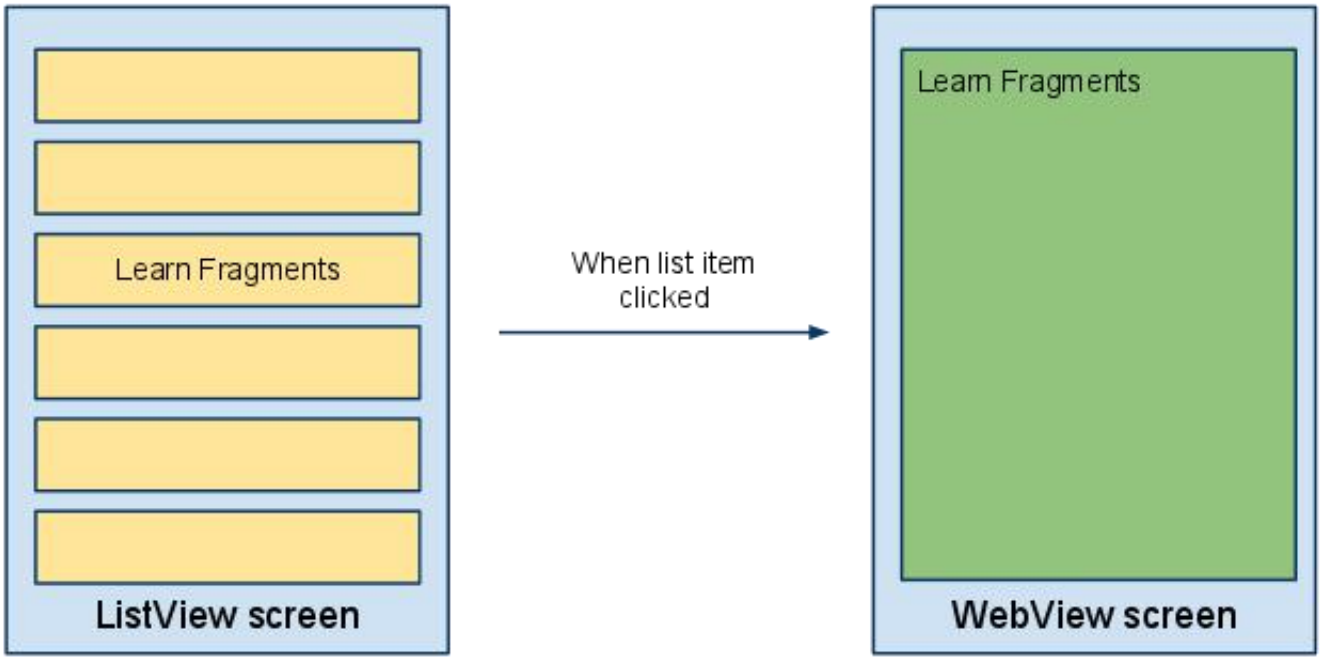
Practically, a Fragment is a modular section of an Activity.

DESIGN PHILOSOPHY

- **Structure** an Activity as a collection of Fragments.
- **Reuse** a Fragment on different Activities …

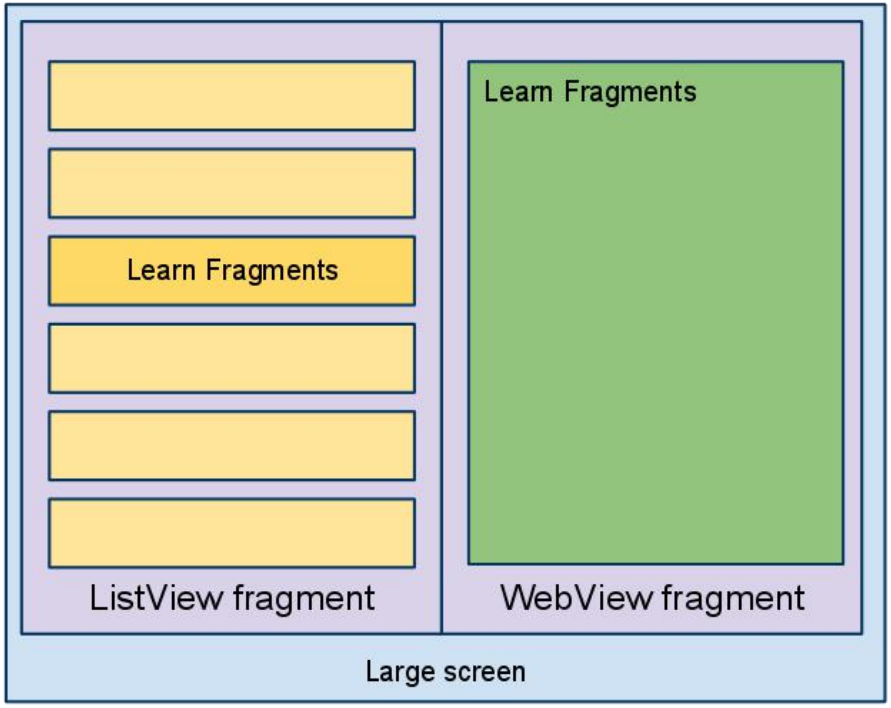
Android: Fragments Design Philosophy

EXAMPLE: Structuring an Application using multiple Activities.



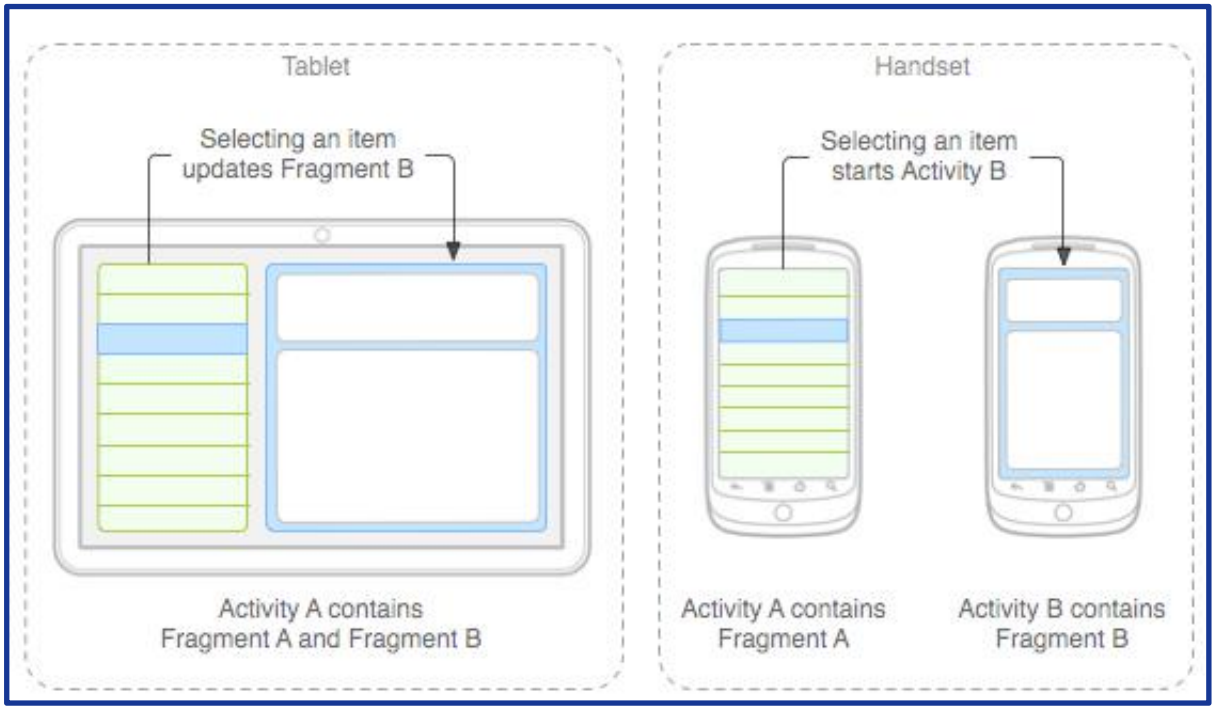
Android: Fragments Design Philosophy

EXAMPLE: Structuring an Application using 1 Activity and 2 Fragments.



Android: Fragment Transactions

EXAMPLE: Using Fragments on Different Devices (Smartphone/Tab)



Android: Fragment Creation

To define a new Fragment → create a subclass of Fragment.

```
public class MyFragment extends Fragment { …}
```

PROPERTY of a Fragment:

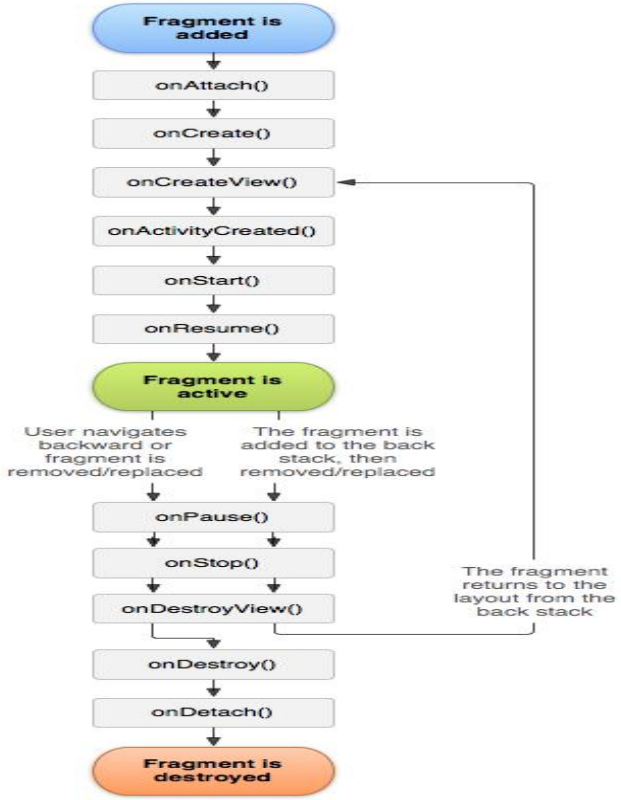
- Has its own **lifecycle** (partially connected with the Activity lifecyle)
- Has its own **layout** (or may have)
- Can receive its own **input events**
- Can be added or removed while the Activity is running.

Android: Adding a Fragment to the UI

Specify layout properties for the Fragment as it were a View.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
   <fragment android:name="it.cs.android30.FragmentOne"
       android:id="@+id/f1"
       android:layout_width="wrap_content"
       android:layout_height="fill_parent"
       />
   <fragment android:name="it.cs.android30.FragmentTwo"
       android:id="@+id/f2"
       android:layout_width="wrap_content"
       android:layout_height="fill_parent"
       />
</LinearLayout>
```

Android: Fragment Lifecycle



Several **callback methods** to handle various stages of a Fragment lifecycle:

onCreate() → called when creating the Fragment.

onCreateView() → called when it is time for the Fragment to draw the user interface the first time.

onPause() → called when the user is leaving the Fragment.

Android: Fragment Creation

onCreateView() → must return the **View** associated to the UI of the Fragment (if any) …

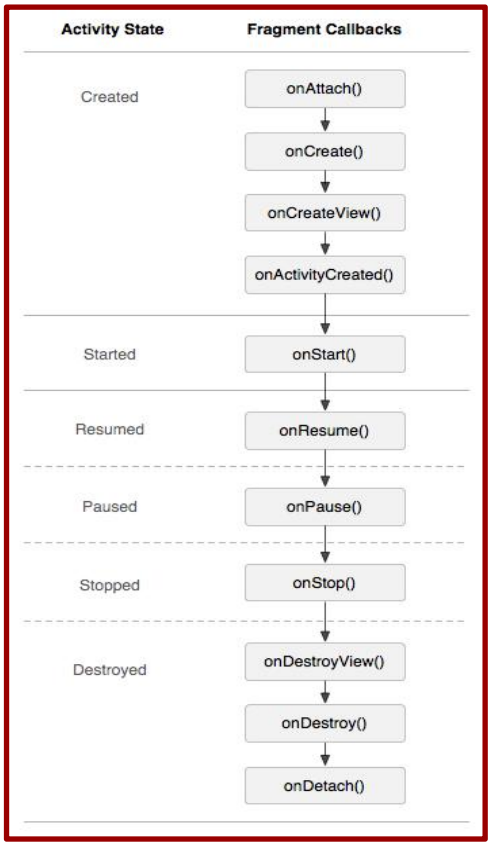
```
  public class ExampleFragment extends Fragment {

 @Override
 public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle saved) {

  return inflater.inflate(R.layout.example_fragment,
container, false);

 }
```

Android: Fragment Lifecycle



The lifecycle of the Activity in which the Fragment lives directly affects the lifecycle of the Fragment.

onPause (Activity) → **onPause** (Fragment)

onStart (Activity) → **onStart** (Fragment)

onDestroy (Activity) → **onDestroy** (Fragment)

Fragments have also extra lifecycle callbacks to enable runtime creation/destroy.

Android: Managing Fragments

A **Fragment** can get a reference to the Activity …

```
Activity getActivity()
```

An **Activity** can get a reference to the Fragment …

```
ExampleFragment fragment=(ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_f
ragment)
```

The **FragmentManager** manages the Fragment associated to the current Activity.

Android: Managing Fragments

In some cases, a Fragment must share an event with the Activity … how to do it?

1. Define a **callback** interface inside the Fragment

```
public interface OnArticleSelectedListener {
    public void onArticleSelected(Uri uri);
    …
}
```

1. Require that the host Activity implements it

Android: Fragment Transactions

- Fragments can be added/removed/replaced while the Activity is running …
- Each set of changes to the Activity is called a **Transaction**.
- **Transaction** can be saved in order to allow a user to navigate backward among Fragments when he clicks on the “Back” button.

Android: Fragment Transactions

1. **ACQUIRE** an instance of the FRAGMENT MANAGER

```
FragmentManager man=getFragmentManager();
FragmentTransaction transaction=man.beginTransaction();
```

2. **CREATE** new Fragment and Transaction

```
FragmentExample newFragment=new FragmentExample();
transaction.replace(R.id.fragment_container, newFragment);
```

3. **SAVE** to backStack and **COMMIT**

```
transaction.addToBackState(null);
```

```
transaction.commit();
```

Android: Fragment Transactions

- A Transaction is not performed till the **commit** …
- If **addToBackStack()** is not invoked → the Fragment is destroyed and it is not possible to navigate back.
- If **addToBackStack()** is invoked → the Fragment is stopped and it is possible to resume it when the user navigates back.
- **popBackStack**() → simulate a Back from the user.

Why do we need to learn about fragment?

- Design perspective: Fragment is easier to customize than Activity
- Testing perspective: Potential new failures due to:
  - Fragment Lifecycle
  - Activity-Fragment Coordination
  - Fragment Transaction

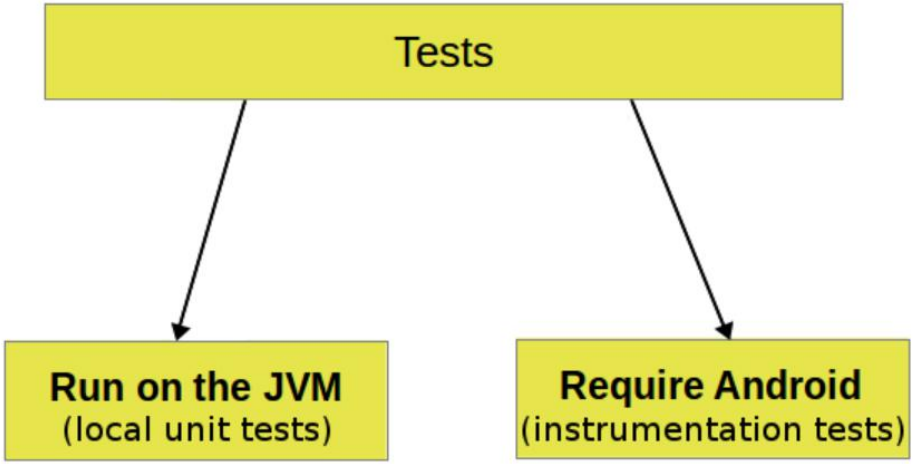
Android Programming Notes

- Android apps have multiple points of entry: no main() method
  - Cannot “sleep” in Android
  - During each entrance, certain Objects may be null
  - Defensive programming is very useful to avoid crashes, e.g., if (!(myObj == null)) { // do something } (Potential Fault!)
- Java concurrency techniques are required
  - Don’t block the “main” thread in Activities
  - Implement long-running tasks such as network connections asynchronously, e.g., as AsyncTasks
  - Recommendation: read [4]; chapter 20 [10]; [11]
- Logging state via android.util.Log throughout app is essential when debugging (finding root causes)
- Better to have “too many” permissions than too few
  - Otherwise, app crashes due to security exceptions!
  - Remove “unnecessary” permissions before releasing app to public
- Event handling in Android GUIs entails many listener Objects

Android Testing

Slides from https://www.vogella.com/tutorials/AndroidTesting/article.html

Categories of Android tests



Local unit tests run much faster compared to the time required to deploy and run tests on an Android device. Prefer writing local unit tests and only run tests on Android, if you require a real Android system.

If you write local unit test and have dependencies to Android API, you need to replace them, e.g., via a mocking framework like Mockito.

1.4. Android project organization for tests

The following is the default directory structure for your application and test code:

- `app/src/main/java` - for your source code of your main application build
- `app/src/test/java` - for any unit test which can run on the JVM
- `app/src/androidTest/java` - for any test which should run on an Android device

If you follow this conversion, the Android build system runs your tests on the correct target (JVM or Android device).

Solving the "error duplicate files in path" error

If you receive the following error message: "error duplicate files in path. Path in archive: LICENSE.txt" you can add the following to your `app/gradle.build` file.

COPY TEXT

```
android {
    packagingOptions {
    exclude 'LICENSE.txt'
    }
}
```

2. Android unit testing

2.1. Unit testing in Android

A unit test verifies in isolation the functionality of a certain component. For example, assume a button in an Android activity is used to start another activity. A unit test would determine, if the corresponding intent was issued, not if the second activity was started

A unit tests for an Android application can be:

- local unit test - which runs on the JVM
- Android unit test - which runs on the Android runtime

If they run on the JVM, they are executed against a modified version of the `android.jar` Android library. In this version all final modifiers have been stripped off. This makes it easier to use mocking libraries, like Mockito.

The local unit tests of an Android project should be located in the `app/src/test` folder.

2.2. Required dependencies in the Gradle build file

To use JUnit tests for your Android application, you need to add it as dependency to your Gradle build file.

```
dependencies {
    // Unit testing dependencies
    testCompile 'junit:junit:4.12'
    // Set this dependency if you want to use the Hamcrest matcher library
    testCompile 'org.hamcrest:hamcrest-library:1.3'
    // more stuff, e.g., Mockito
}
```

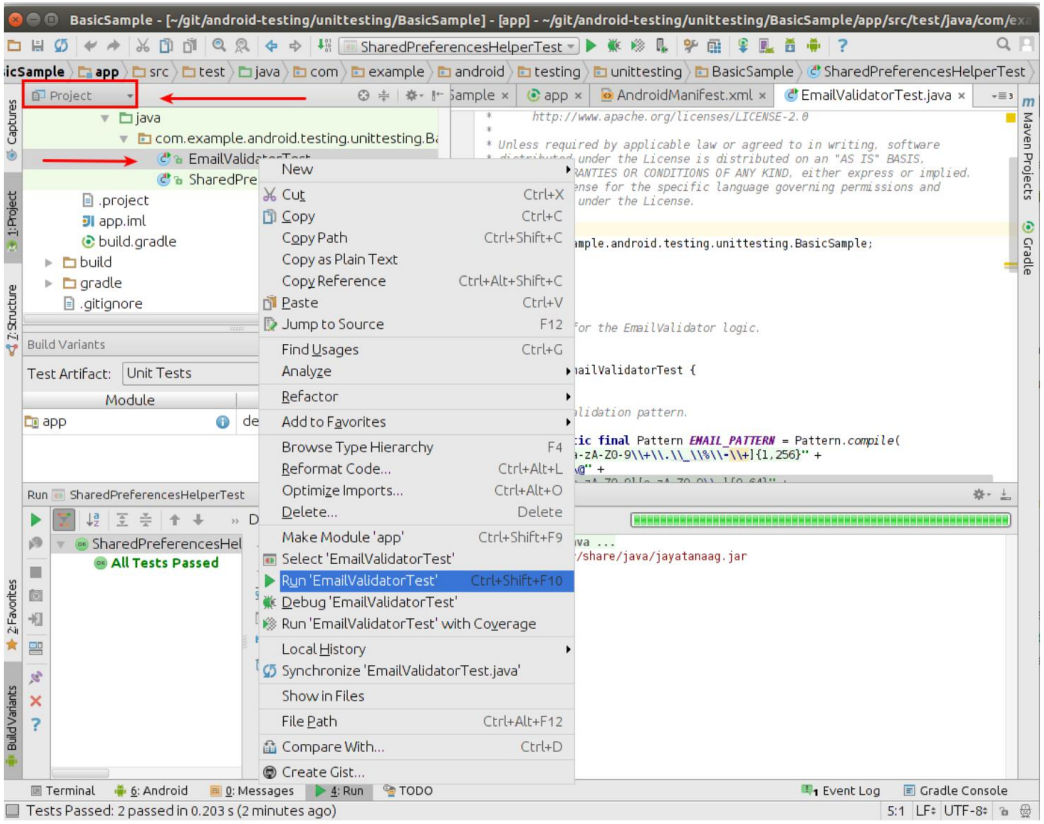
2.3. Running the unit tests

2.3.1. Using Gradle

Run your unit tests with the `gradlew test` command.

2.3.2. Using Android Studio

To run a unit test, right-click on your test class in the *Project* window and select *Run*.



2.4. Location of test reports

The Test reports are created in the `app/build/reports/tests/debug/` directory. The `index.html` gives an overview and links to the individual test pages.

Android UI Testing

Slides from https://www.vogella.com/tutorials/AndroidTestingEspresso/article.html

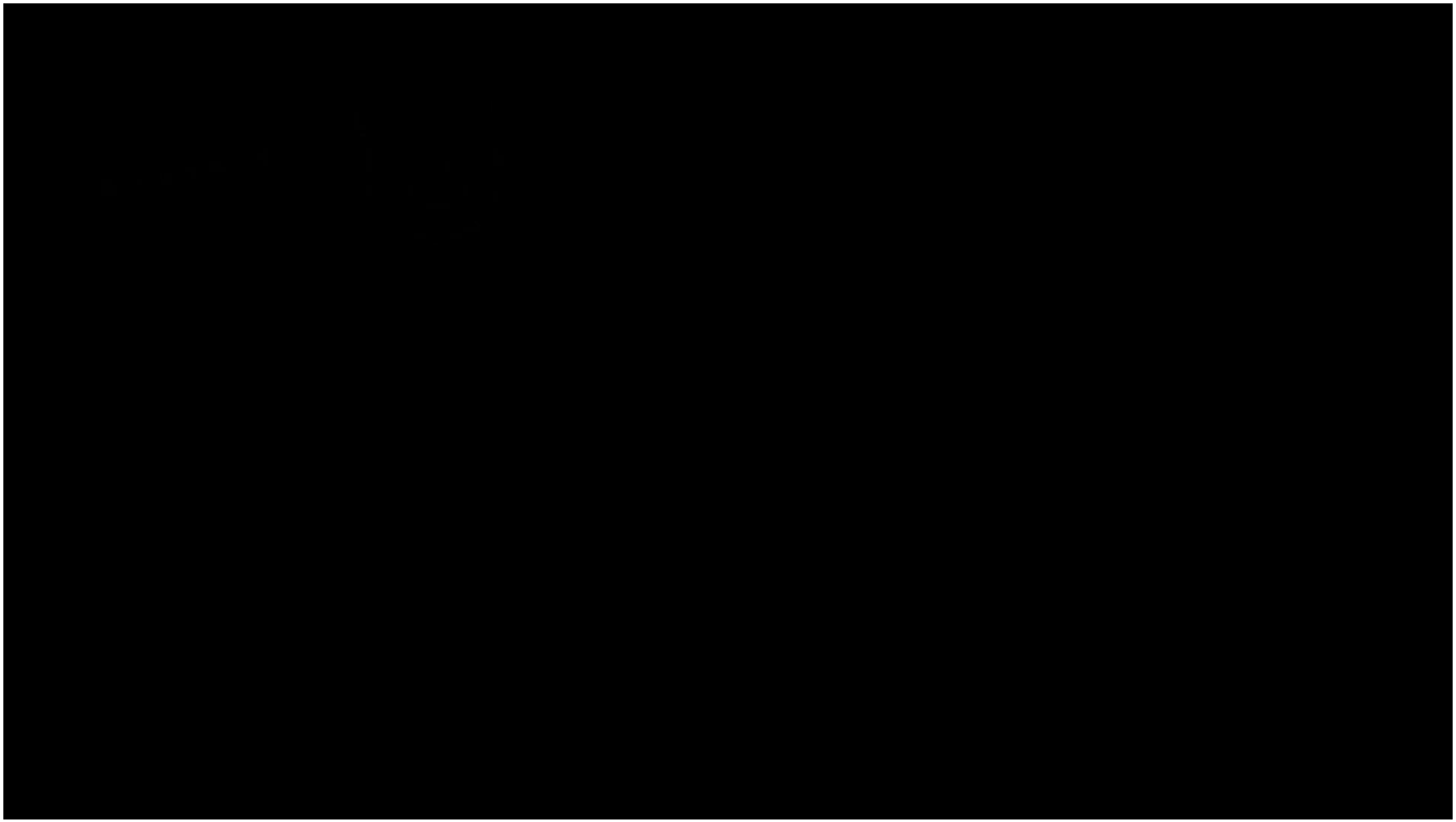
The Espresso test framework

- A testing framework for Android to make it easy to write reliable user interface tests.
- automatically synchronizes your test actions with the UI of your app.
- ensures that your activity is started before the tests run & let the test wait until all observed background activities have finished.
- Used for testing a single app but can also be used to test across apps. If used for testing outside your app, you can only perform black box testing, as you cannot access the classes outside of your app.

Record an Espresso test

- Espresso tests consist of 2 components:
  - UI interactions
    - tap and type actions used to interact with your app
  - Assertions on View elements
    - Verify the existence or contents of visual elements on the screen.
  - Example: an Espresso test for the Notes testing app might include UI interactions for clicking on a button and writing a new note but would use assertions to verify the existence of the button and the contents of the note.

Record UI test with Espresso Test Recorder



From: https://developer.android.com/studio/test/espresso-test-recorder

Record an Espresso test

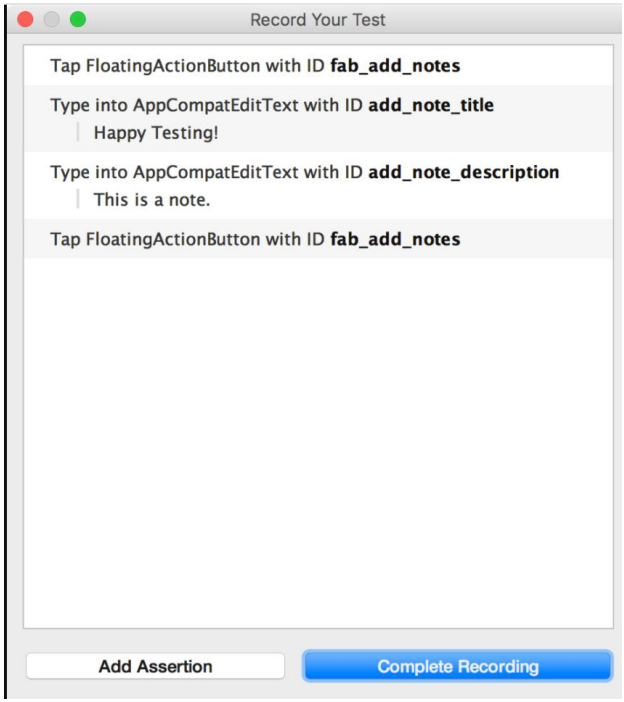
Record UI interactions

To start recording a test with Espresso Test Recorder, proceed as follows:

1. Click **Run** > **Record Espresso Test**.
2. In the **Select Deployment Target** window, choose the device on which you want to record the test. If necessary, create a new Android Virtual Device. Click **OK**.
3. Espresso Test Recorder triggers a build of your project, and the app must install and launch before Espresso Test Recorder allows you to interact with it. The **Record Your Test** window appears after the app launches, and since you have not interacted with the device yet, the main panel reads "No events recorded yet." Interact with your device to start logging events such as "tap" and "type" actions.

Assertions

- Assertions verify the existence or contents of a View element through three main types:
  - **text is**: Checks the text content of the selected View element
  - **exists**: Checks that the View element is present in the current View hierarchy visible on the screen
  - **does not exist**: Checks that the View element is not present in the current View hierarchy



Add Assertions

- To add an assertion to your test, proceed as follows:
- Click **Add Assertion.** A Screen Capture dialog appears while Espresso gets the UI hierarchy and other information about the current app state. The dialog closes automatically once Espresso has captured the screenshot.
- A layout of the current screen appears in a panel on the right of the **Record Your Test** window. To select a View element on which to create an assertion, click on the element in the screenshot or use the first drop-down menu in the **Edit assertion** box at the bottom of the window. The selected View object is highlighted in a red box.
- Select the assertion you want to use from the second drop-down menu in the **Edit assertion** box. Espresso populates the menu with valid assertions for the selected View element.
- If you choose the "text is" assertion, Espresso automatically inserts the text currently inside the selected View element. You can edit the text to match your desired assertion using the text field in the **Edit assertion** box.
- Click **Save and Add Another** to create another assertion or click **Save Assertion** to close the assertion panels.

What to do for this lab?

- Record an Espresso test for your selected app for your project!
  - Record 1 test using the “text is” assertion
  - Record 1 test using the “exists” assertion
  - Record 1 test using the “does not exist” assertion
  - Do not select the app if you couldn't record tests!

What to submit for this lab?

- Reminder:
  - Project proposal due soon on October 16, 11.59pm!
- Nothing to submit for this lab!

CS409 Project: Project Proposal *(Total: 20 points)*
Deadline: October 16, 11.59pm

The class project (Invitation links:https://classroom.github.com/g/nJLg4_dw) is a group-based project where students will be divided into groups of three (Only two groups in the entire class could have four members). The objective of the project is to allow student to apply the knowledge that they learned in the class to testing real-world applications. Each group need to choose three apps (one app per student) across different categories from the list of open-source Android apps at: https://github.com/topics/android-apps or https://github.com/pcqpcq/open-source-android-apps or https://github.com/Mybridge/amazing-android-apps or https://apt.izzysoft.de/fdroid/index.php/?repo=main

Note that the class will not provide any hardware/devices so it is not advisable to select the apps that require particular devices (e.g., apps from the "Android Wears" categories). All group members should carefully discuss and agree upon 3 (or 4) Android apps based on the criteria listed below:.

- Ease of use: The project could be compiled successfully without errors. The app could be executed in a device. *All selected projects should fulfill this criteria.*

4. Why do your group select this project? Explain the reason in terms of the criteria below: *(Total: 13 points)*

- Ease of use: Have you successfully compiled and executed the app? (2 points)

  - Attached screenshots to show that you can build and execute the app by showing the recorded Espresso tests in previous slide!

References (1)

1. C. Horstmann, *Big Java Late Objects*, Wiley, 2012. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/–/9781118087886
2. J. Bloch, *Effective Java*, 2nd ed., Addison–Wesley, 2008. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/java/9780137150021
3. S.B. Zakhour, S. Kannan, and R. Gallardo, *The Java® Tutorial: A Short Course on the Basics*, 5th ed., Addison–Wesley, 2013. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/java/9780132761987
4. C. Collins, M. Galpin, and M. Kaeppler, *Android in Practice*, Manning, 2011. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/android/9781935182924
5. M.L. Sichitiu, 2011, http://www.ece.ncsu.edu/wireless/MadeInWALAN/AndroidTutorial/PPTs/javaReview.ppt
6. Oracle, http://docs.oracle.com/javase/1.5.0/docs/api/index.html
7. Wikipedia, https://en.wikipedia.org/wiki/Vehicle_Identification_Number
8. Nielsen Co., “Smartphone Milestone: Half of Mobile Subscribers Ages 55+ Own Smartphones”, 22 Apr. 2014, http://www.nielsen.com/us/en/insights/news/2014/smartphone-milestone-half-of-americans-ages-55-own-smartphones.html
9. Android Open Source Project, http://www.android.com

References (2)

10. http://bcs.wiley.com/he-bcs/Books?action=index&itemId=1118087887&bcsId=7006
11. B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, Java Concurrency in Practice, Addison-Wesley, 2006, online at http://proquest.safaribooksonline.com/book/programming/java/0321349601
12. https://developer.android.com/guide/components/activities.html
13. https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts
14. https://cloud.genymotion.com/page/doc/#collapse4
15. http://blog.zeezonline.com/2013/11/install-google-play-on-genymotion-2-0/