

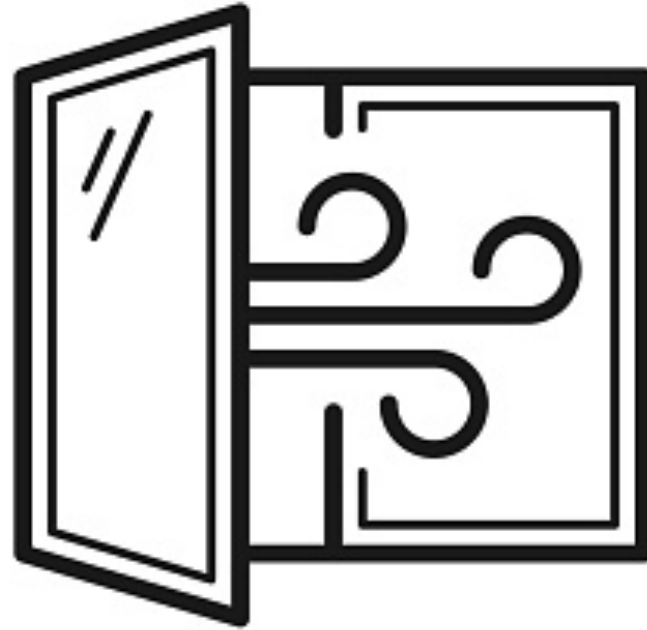
Android App Programming

Lecture 1: Introduction, Activities
October 4, 2021

Anastasia Bezerianos
Université Paris-Saclay

parts of the course based on that of Thomas Nowak

in class



Course organization

- Classes Tuesday 2:30-5:30 pm in room I105-107. Labs every week, right after lecture.
- Moodle for uploading lab (TD) code, Quizzes, Project ...
Register !!!
<https://cours.iut-orsay.fr/course/view.php?id=262>
- Slides, etc. on <https://www.lri.fr/~anab/teaching/Android/>
- Contact: forum *Moodle*, or email first.last@universite-paris-saclay.fr add [Android] to the title

Grading

1. Reading quizzes: 10%
2. Practical TD/exam 1: 20%
3. Practical TD/exam 2: 20%
4. Final project: 50%

Reading quizzes

- Advance reading before each class
- Short quiz before lecture on reading material
- ~5 minutes on Moodle
- A few multiple choice questions each time

Exams

- Two practical exams (marked TD)
- ~2 hours
- Create a small app
- Checks your (initial/rough) understanding of the content of the preceding unit

Course overview

- Unit 1:
 - Activities
 - Intents
 - Web services
 - Asynchronous tasks
 - Peripherals
 - Databases

Course overview

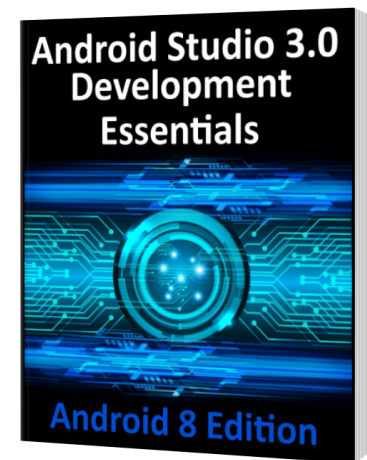
- Unit 2:
 - Android Runtime
 - User Interface Design
 - RESTful services
 - Dealing with disconnectivity
 - Testing and profiling

Course overview

- Unit 3 (with Chantal Keller and Thomas Nowak)
 - Design and develop your own app
 - Groups of 4
 - Version control via git
 - Final presentations and demos

Resources

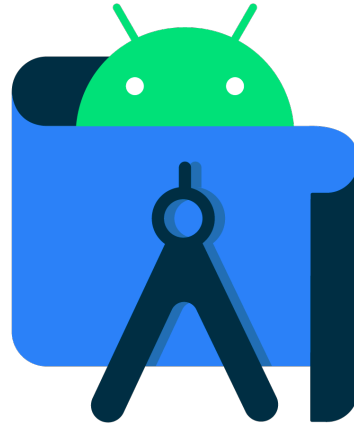
- Android and Java APIs and documentation
- Book: Android Studio 3.0 (or 4.0) Development Essentials by Neil Smyth
- Lecture slides on course website
- Lots of online articles and tutorials



Introduction

- Getting started with Android Studio
- Create first app
- Single screen app: single Activity

Android Studio

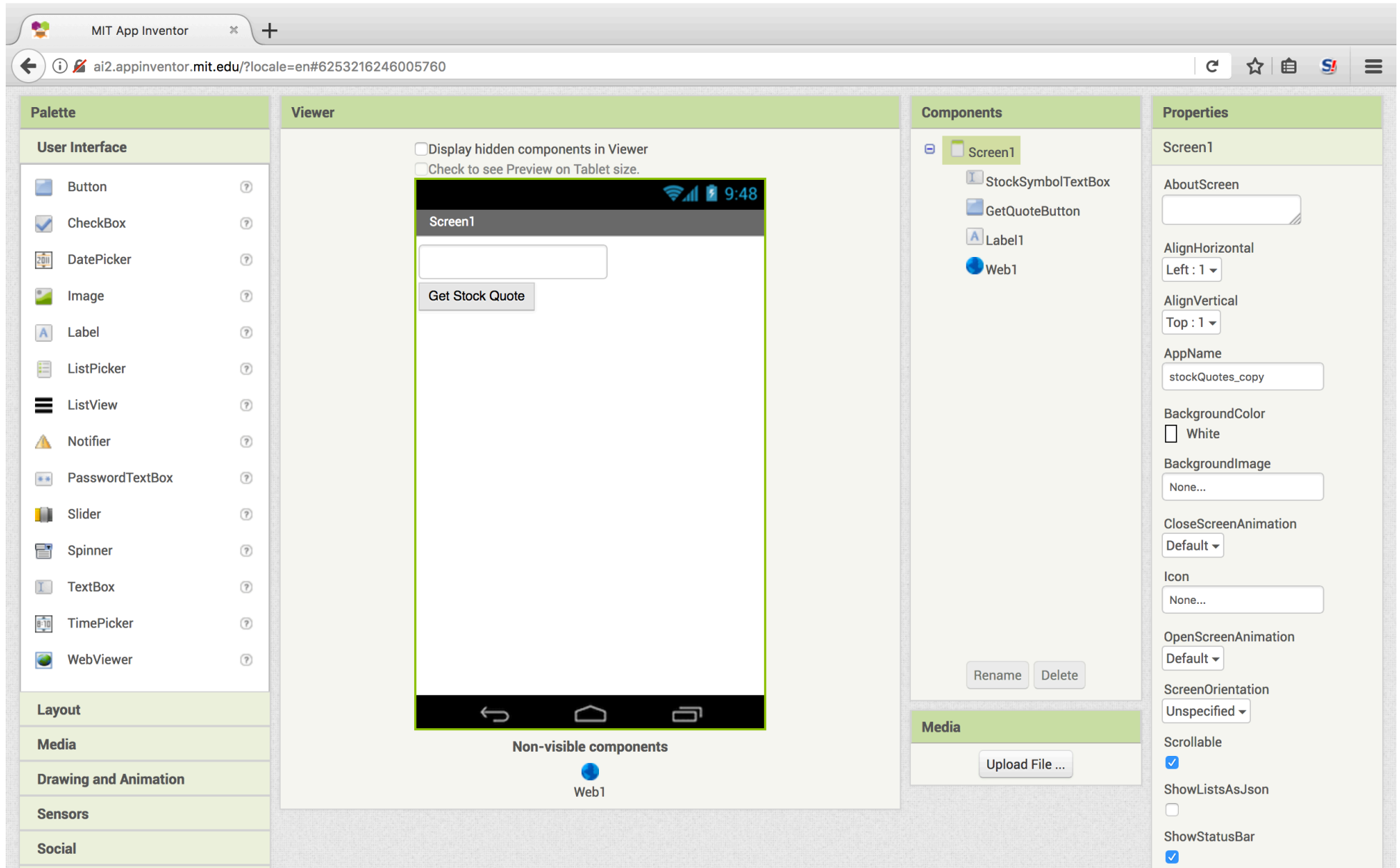


- Official integrated development environment (IDE) for Google's Android operating system
- It is built on the IntelliJ IDEA software and designed specifically for Android development

(aside) App Inventor 2

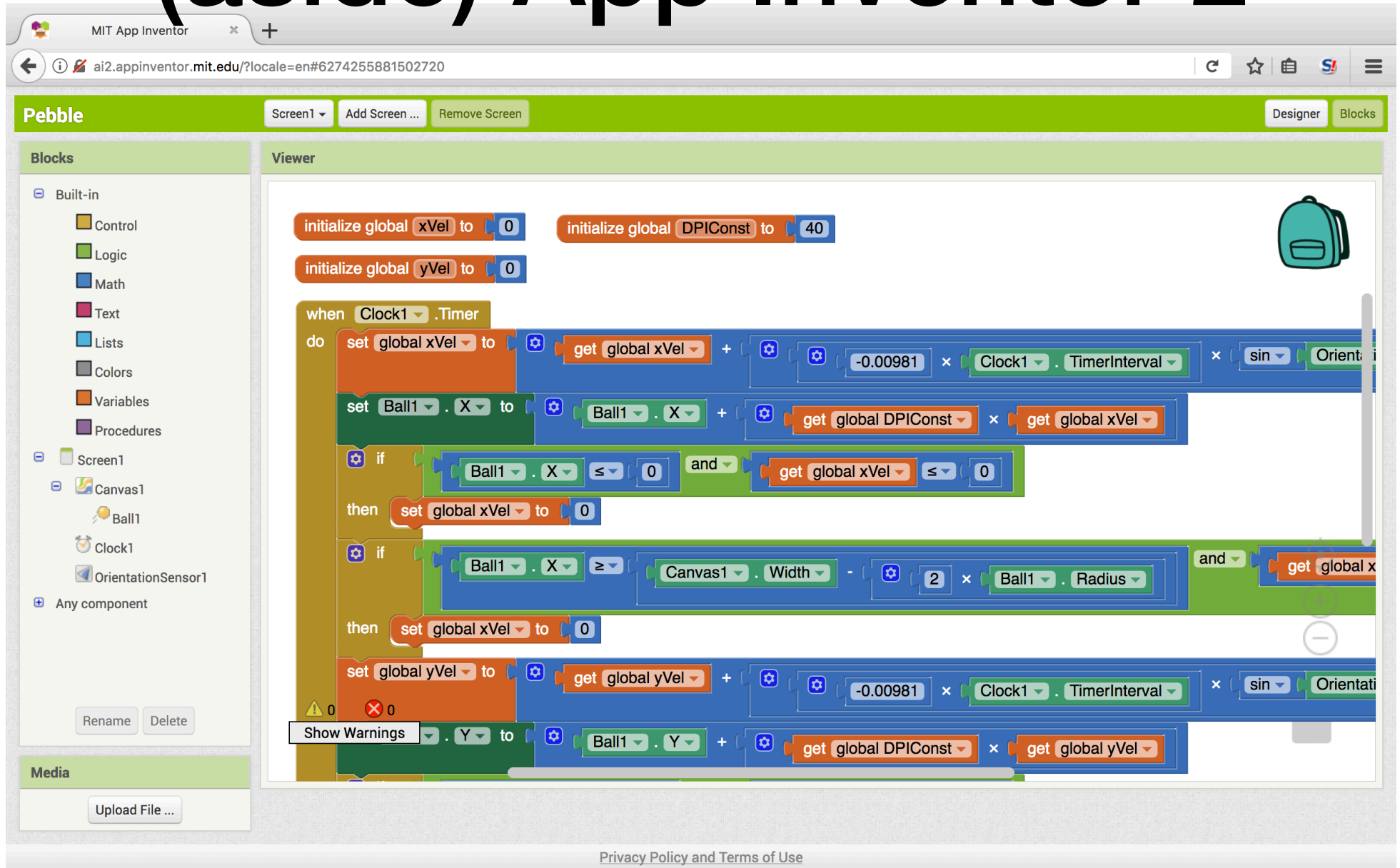
- Online app creation tool
- Generates APKs
- Good for fast prototyping
- Graphical programming language
- BUT can become cumbersome for larger projects

(aside) App Inventor 2



- Graphical editor

(aside) App Inventor 2



- Scratch based visual programming language

Java crash course (1)

- Interfaces/classes:

```
class MainActivity extends AppCompatActivity
```

```
class AwesomeButtonClick implements View.OnClickListener
```


Java crash course (2)

- Anonymous inner class:

```
mybutton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        ...;  
    }  
});
```

Anonymous inner class

Anonymous Inner classes

```
new <class-name> () { <body> };
```

This construction does 2 things:

- creates a new class without name, that is a subclass of <class-name> defined by <body>
- creates a (unique) instance/object of this new class and returns its value

This (inner) class has access to variables and methods of the class inside which it is defined

Java crash course (3)

- Callbacks / Listeners:

```
boolean onOptionsItemSelected(MenuItem item)
```

Callbacks

- Mechanism in OOP for an app to handle subscribed events, that appear at runtime
- This is done through an interface (listener interface)
- You provide an implementation of the interface abstract methods (ie how to handle the subscribed events)
- When the subscribed event is triggered, your listener code will execute

Callbacks/Listeners

- Mechanism in OOP for an app to handle subscribed events, that appear at runtime
- This is done through an interface (listener interface)
- You provide an implementation of the interface abstract methods (ie how to handle the subscribed events)
- When the subscribed event is triggered, your listener code will execute

Callbacks/Listeners

- Define a listener class

```
public class MyListener implements View.OnClickListener{  
    @Override  
    public void onClick(View view) {  
        Log.d("Msg", "button clicked");  
    }  
};
```

define a new class MyListener that follows the pattern of the interface View.OnClickListener (predefined in Java).

It creates a callback that listens for Click events.

We need to add the code for the method onClick (callback method).

- bind the listener to a widget

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        MyListener myl = new MyListener();  
  
        Button b = findViewById(R.id.button);  
        b.setOnClickListener(myl);  
    }  
}
```

Create an object myl of the class MyListener.

Register (subscribe, assign) the listener object myl to button b.

Callbacks/Listeners

- Example

```
View.OnClickListener my1 = new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Log.d("Msg", "button clicked");  
    }  
};  
Button b = findViewById(R.id.button);  
b.setOnClickListener(my1);
```

Assign the listener object my1 to button b

Use of an anonymous inner class to create a listener object my1.

Create an object my1 (notice the keyword **new**) that uses the pattern of the interface View.OnClickListener (predefined in Java).

We need to add the code for the method onClick.

Callbacks/Listeners

- ... same as

```
Button b = findViewById(R.id.button);
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Log.d("Msg", "button clicked");
    }
});
```


Callbacks/Listeners

```
Call<List<PlaceholderPost>> call = myPlaceholderApi.getPosts();

call.enqueue(new Callback<List<PlaceholderPost>>() {
    @Override
    public void onResponse(Call<List<PlaceholderPost>> call,
                          Response<List<PlaceholderPost>> response) {

        if (response.isSuccessful()) {
            ...
        } else {
            ...
        }
    }

    @Override
    public void onFailure(Call<List<PlaceholderPost>> call, Throwable t) {
        textView.setText(t.getMessage());
    }
});
```

Another example of a callback from web services with Retrofit2.

enqueue assigns to our list `call` a listener that waits for data to be added into the list.

The pattern of code that the listener class follows is that of `Callback<data I am waiting for>`.

We provide code for it in the methods `onResponse()` and `onFailure()`

Java crash course (4)

- Lambda expressions:

```
view -> System.out.println(view)
```

Lambda expressions

- A short block of code which takes in parameters and returns a value.
- Similar to methods, but they do not need a name and they can be implemented right in the body of a method

`parameter -> expression`

`(parameter1, parameter2) -> { code ... }`

Lambda expressions

`view -> System.out.println(view)`

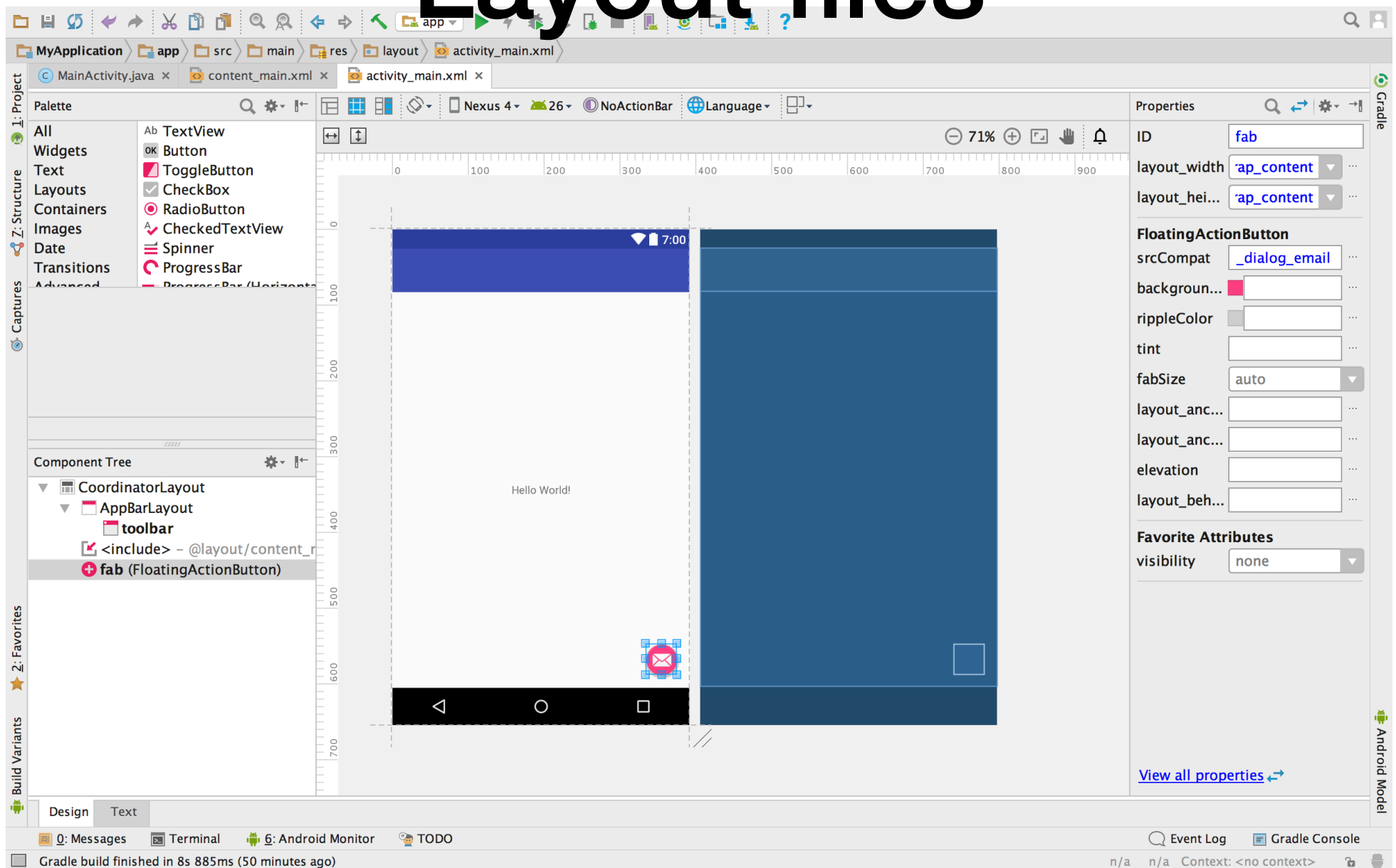
- is the same as

```
void foo (String s){  
    System.out.println(s)  
}  
foo(view);
```

Android Studio

- Comes with latest version of Java OpenJDK embedded and fetches Android SDK
- Includes management of virtual devices (emulator), using the Android Virtual Device (AVD) manager
- Includes Android Debug Bridge (ADB) to connect with virtual and physical devices
- Tools for testing, profiling, packaging, shipping
- Lots of tutorials on the web

Layout files



Layout files

- All user interface (UI) elements can be created in Java code
- Most of the time they aren't (in XML instead) ...
- Description in XML format
- UI Elements created at startup

Activities

- Single block of user interaction
- In the simplest case, one UI screen
- Can be a window or part of the screen
- Activities are by default independent from each other
- To display something, set its content:
- `setContentView(View)`

Views

- Basic UI component
- E.g., Button, TextView, ImageView, menus, etc.
- Has a set of mutable (changeable) properties (e.g., text size)
- Can register listeners:
- `button.setOnClickListener(...);`

R

- Dynamically generated class during compilation
- Links XML layout files to Java
- Contains all assets of the application (layout files, images, strings, etc.)
- Field names defined in layout editor
- Ex: `R.id.toolbar`

more details

- If you want to learn more about the structure of Android programs ...
- Ch. 10 The Anatomy of an Android Application [Android Studio 3.0 (or 4.0) Development Essentials]

Lab 1

1. Setup Android Studio on your machine and a device
2. Create a simple Activity and explore the layout editor
3. Add simple interactivity (a button)
4. Create an app with a more complex layout

Upload your code (parts 2-3) in a zip file

Advance reading

Next week's topic:

- Intents (a.k.a. switching Activities) - 5min **test next week**
- Android Studio 3.0 Development Essentials (*)
 - Ch. 11.4-11.8 (11.4 Activity Lifecycle, 11.5 Activity Stack, 11.6 Activity States, 11.7 Configuration Changes 11.8 Handling State Change)
 - Ch. 42.1-42.2 (42.1 An Overview of Intents, 42.2 Explicit Intents)

(*) numbering of sections may differ in Android Studio 4.0 book

Questions