Tutorial: Programming in Java for Android Development

Adam C. Champion and Dong Xuan

CSE 4471: Information Security

Autumn 2013

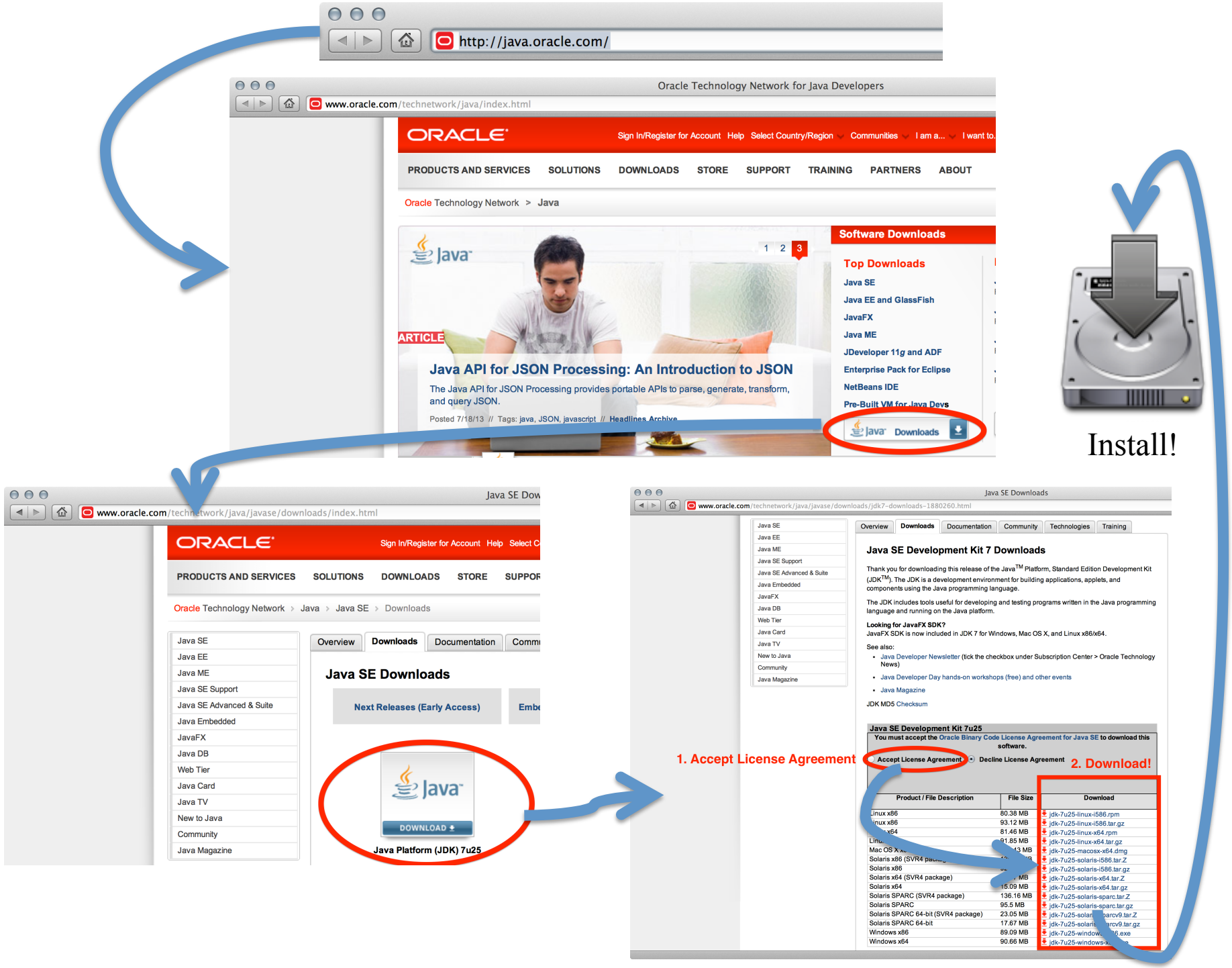
Based on material from C. Horstmann [1], J. Bloch [2], C. Collins et al. [4],
M.L. Sichitiu (NCSU), V. Janjic (Imperial College London), CSE 2221 (OSU), and other sources

Outline

- **Getting Started**
- Java: The Basics
- Java: Object–Oriented Programming
- Android Programming

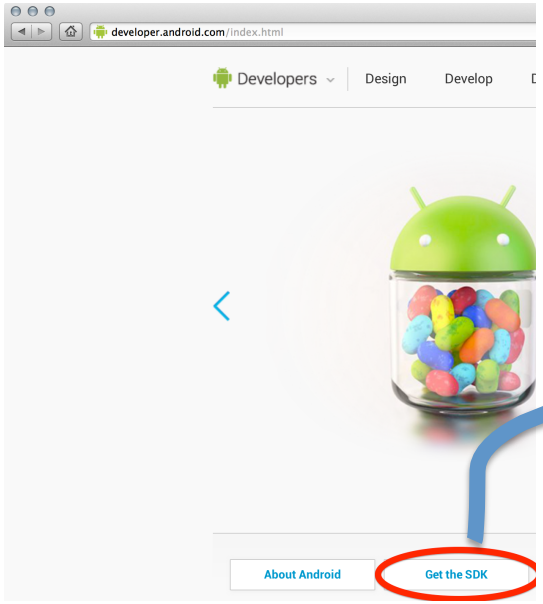
Getting Started (1)

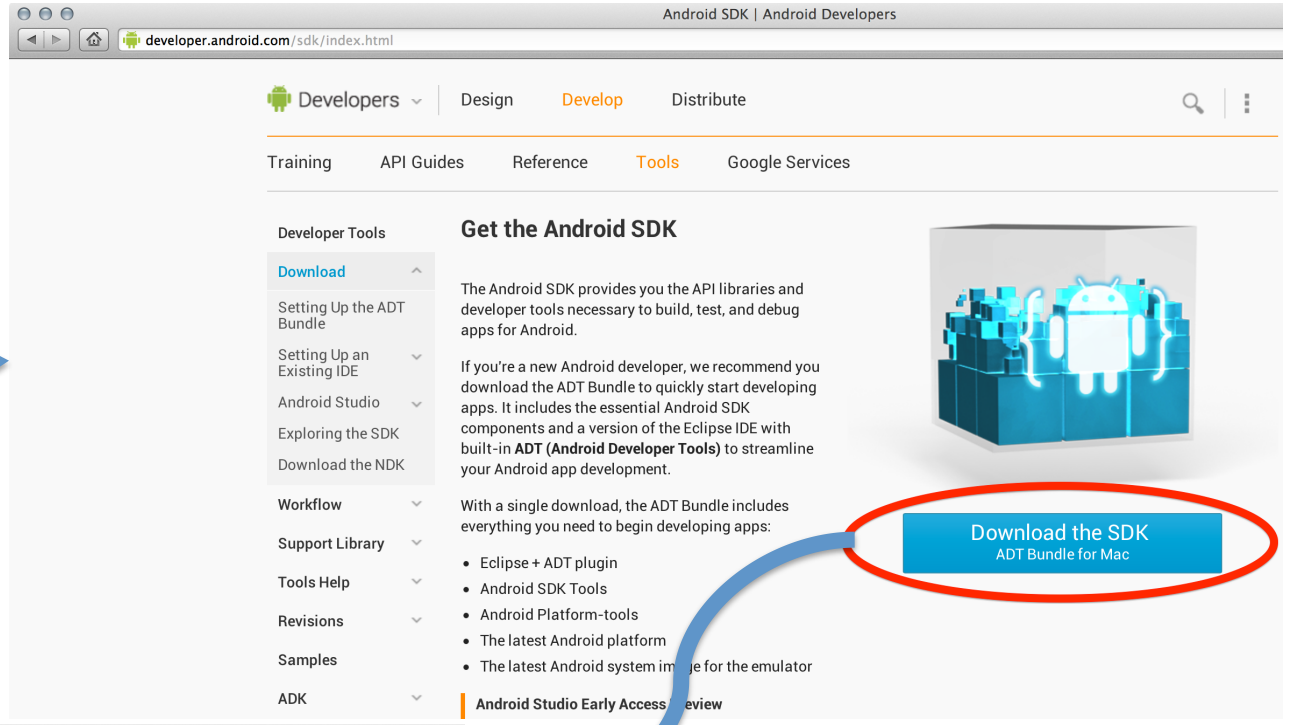
- Need to install Java Development Kit (JDK) to write Java (and Android) programs
  - **Do not** install Java Runtime Environment (JRE); JDK and JRE are different!
- Can download the JDK for your OS at http://java.oracle.com
- Alternatively, for OS X, Linux:
  - OS X:
    - Open `/Applications/Utilities/Terminal.app`
    - Type `javac` at command line
    - Install Java when prompt appears
  - Linux:
    - Type `sudo apt-get install default-jdk` at command line (Debian, Ubuntu)
    - Other distributions: consult distribution's documentation

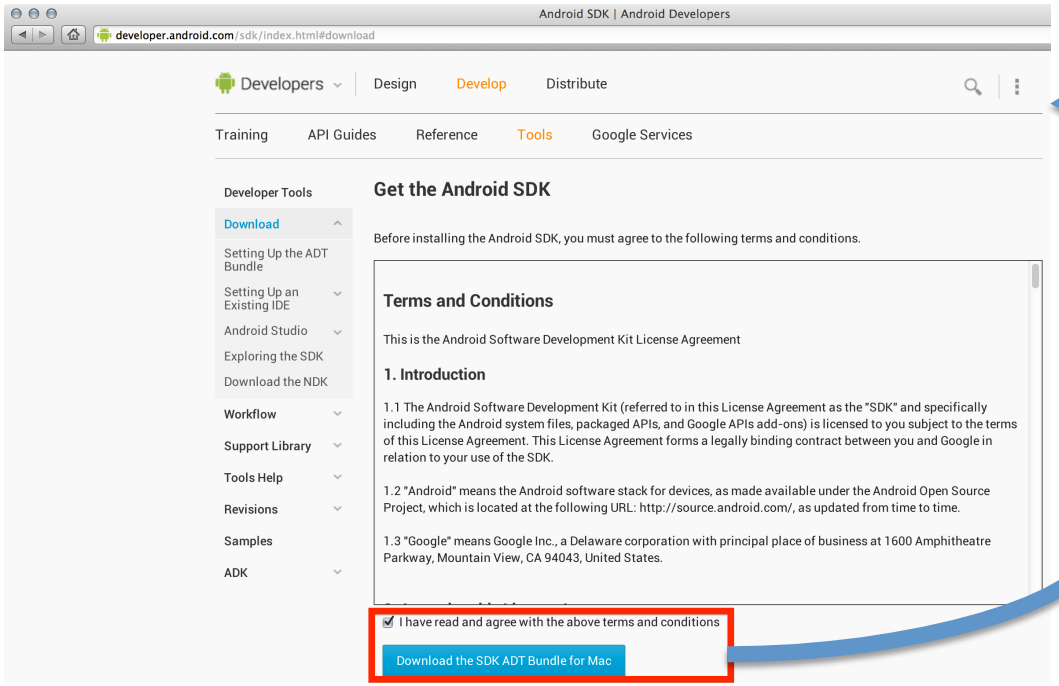

http://java.oracle.com/
Oracle Technology Network for Java Developers
www.oracle.com/technetwork/java/index.html
ORACLE
Sign In/Register for Account Help Select Country/Region Communities I am a... I want to...
PRODUCTS AND SERVICES SOLUTIONS DOWNLOADS STORE SUPPORT TRAINING PARTNERS ABOUT
Oracle Technology Network > Java
Software Downloads
Top Downloads
Java SE
Java EE and GlassFish
JavaFX
Java ME
JDeveloper 11g and ADF
Enterprise Pack for Eclipse
NetBeans IDE
Pre-Built VM for Java Devs
Java Downloads
ARTICLE
Java API for JSON Processing: An Introduction to JSON
The Java API for JSON Processing provides portable APIs to parse, generate, transform, and query JSON.
Posted 7/18/13 // Tags: java, JSON, javascript // Headlines Archive
Install!
Java SE Dow
www.oracle.com/technetwork/java/javase/downloads/index.html
Oracle Technology Network > Java > Java SE > Downloads
Java SE
Java EE
Java ME
Java SE Support
Java SE Advanced & Suite
Java Embedded
JavaFX
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine
Overview
Downloads
Documentation
Java SE Downloads
Next Releases (Early Access)
DOWNLOAD
Java Platform (JDK) 7u25
Java SE Downloads
www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
Overview
Downloads
Documentation
Community
Technologies
Training
Java SE Development Kit 7 Downloads
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.
The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.
Looking for JavaFX SDK?
JavaFX SDK is now included in JDK 7 for Windows, Mac OS X, and Linux x86/x64.
See also:
Java Developer Newsletter (tick the checkbox under Subscription Center > Oracle Technology News)
Java Developer Day hands-on workshops (free) and other events
Java Magazine
JDK MD5 Checksum
Java SE Development Kit 7u25
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Accept License Agreement
Decline License Agreement
1. Accept License Agreement
2. Download!
Product / File Description
File Size
Download
Linux x86 80.38 MB jdk-7u25-linux-i586.rpm
Linux x86 93.12 MB jdk-7u25-linux-i586.tar.gz
Linux x64 81.46 MB jdk-7u25-linux-x64.rpm
Linux x64 91.85 MB jdk-7u25-linux-x64.tar.gz
Mac OS X 143 MB jdk-7u25-macosx-x64.dmg
Solaris x86 (SVR4 package) jdk-7u25-solaris-i586.tar.Z
Solaris x86 jdk-7u25-solaris-i586.tar.gz
Solaris x64 (SVR4 package) jdk-7u25-solaris-x64.tar.Z
Solaris x64 15.09 MB jdk-7u25-solaris-x64.tar.gz
Solaris SPARC (SVR4 package) 136.16 MB jdk-7u25-solaris-sparc.tar.Z
Solaris SPARC 95.5 MB jdk-7u25-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package) 23.05 MB jdk-7u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit 17.67 MB jdk-7u25-solaris-sparcv9.tar.gz
Windows x86 89.09 MB jdk-7u25-windows-i586.exe
Windows x64 90.66 MB jdk-7u25-windows-x64.exe

Getting Started (2)

- After installing JDK, download Android SDK from http://developer.android.com
- Simplest: download and install Android Developer Tools (ADT) bundle (including Android SDK) for your OS
- Alternatives:
  - Download/install Android Studio (beta) from this site (based on IntelliJ IDEA)
  - Install Android SDK tools by themselves, then install ADT for Eclipse separately (from this site)
- We'll use ADT with SDK (easiest)


developer.android.com/index.html
Developers
Design
Develop
About Android
Get the SDK

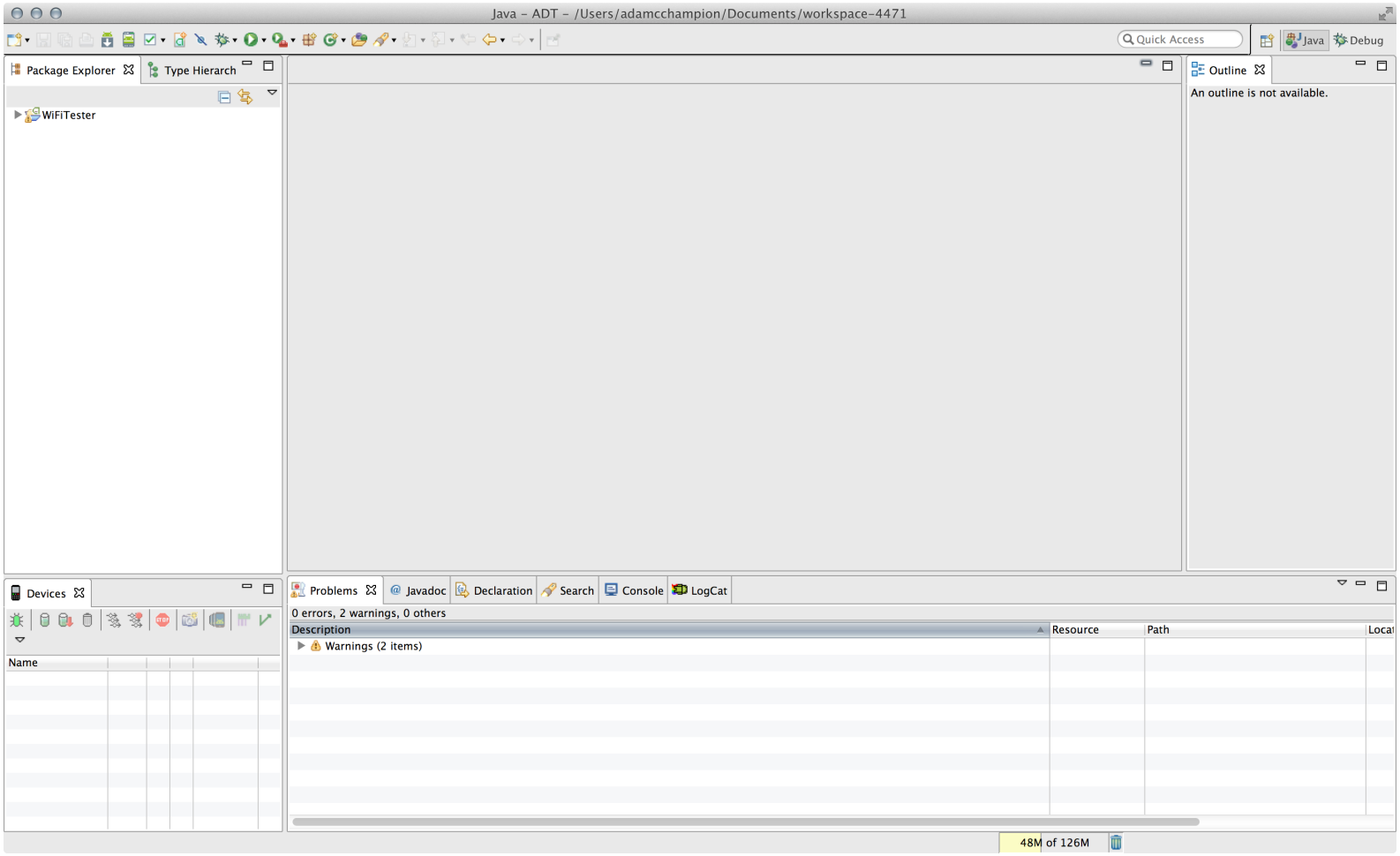

Android SDK | Android Developers
developer.android.com/sdk/index.html
Developers
Design
Develop
Distribute
Training
API Guides
Reference
Tools
Google Services
Developer Tools
Download
Setting Up the ADT Bundle
Setting Up an Existing IDE
Android Studio
Exploring the SDK
Download the NDK
Workflow
Support Library
Tools Help
Revisions
Samples
ADK
Get the Android SDK
The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.
If you're a new Android developer, we recommend you download the ADT Bundle to quickly start developing apps. It includes the essential Android SDK components and a version of the Eclipse IDE with built-in ADT (Android Developer Tools) to streamline your Android app development.
With a single download, the ADT Bundle includes everything you need to begin developing apps:
Eclipse + ADT plugin
Android SDK Tools
Android Platform-tools
The latest Android platform
The latest Android system image for the emulator
Download the SDK
ADT Bundle for Mac
Android Studio Early Access Preview


Android SDK | Android Developers
developer.android.com/sdk/index.html#download
Developers
Design
Develop
Distribute
Training
API Guides
Reference
Tools
Google Services
Developer Tools
Download
Setting Up the ADT Bundle
Setting Up an Existing IDE
Android Studio
Exploring the SDK
Download the NDK
Workflow
Support Library
Tools Help
Revisions
Samples
ADK
Get the Android SDK
Before installing the Android SDK, you must agree to the following terms and conditions.
Terms and Conditions
This is the Android Software Development Kit License Agreement
1. Introduction
1.1 The Android Software Development Kit (referred to in this License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of this License Agreement. This License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.
1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.
1.3 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States.
I have read and agree with the above terms and conditions
Download the SDK ADT Bundle for Mac

Install!

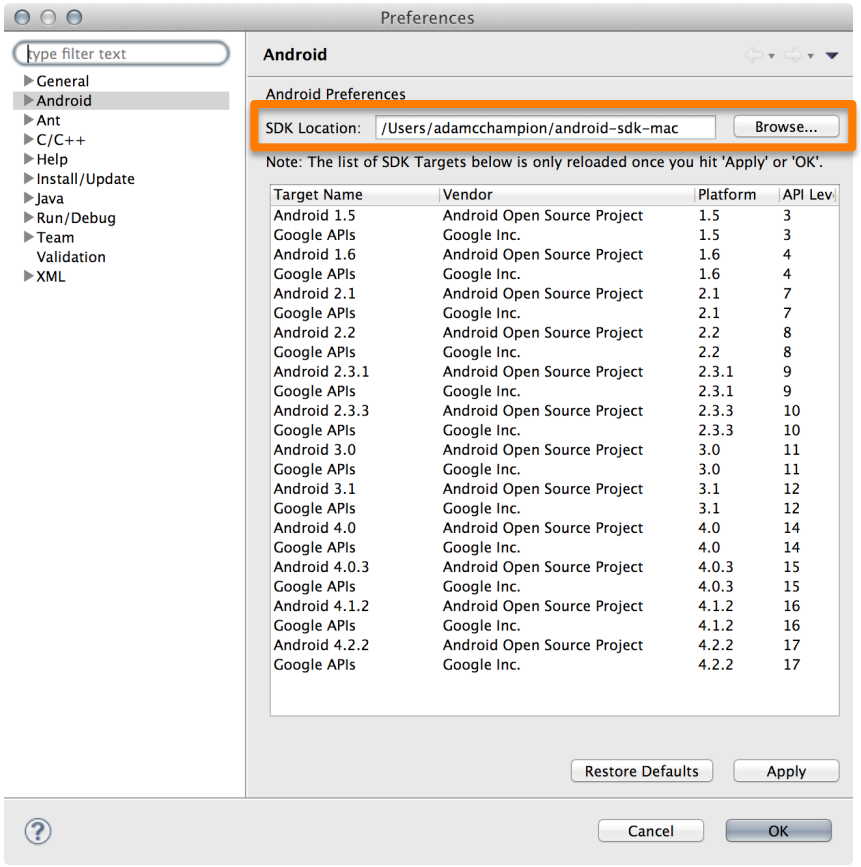
Getting Started (3)

- Unzip ADT package to directory `<adt-bundle-os>`, then run `<adt-bundle-os>/eclipse/Eclipse` app. You should see this:



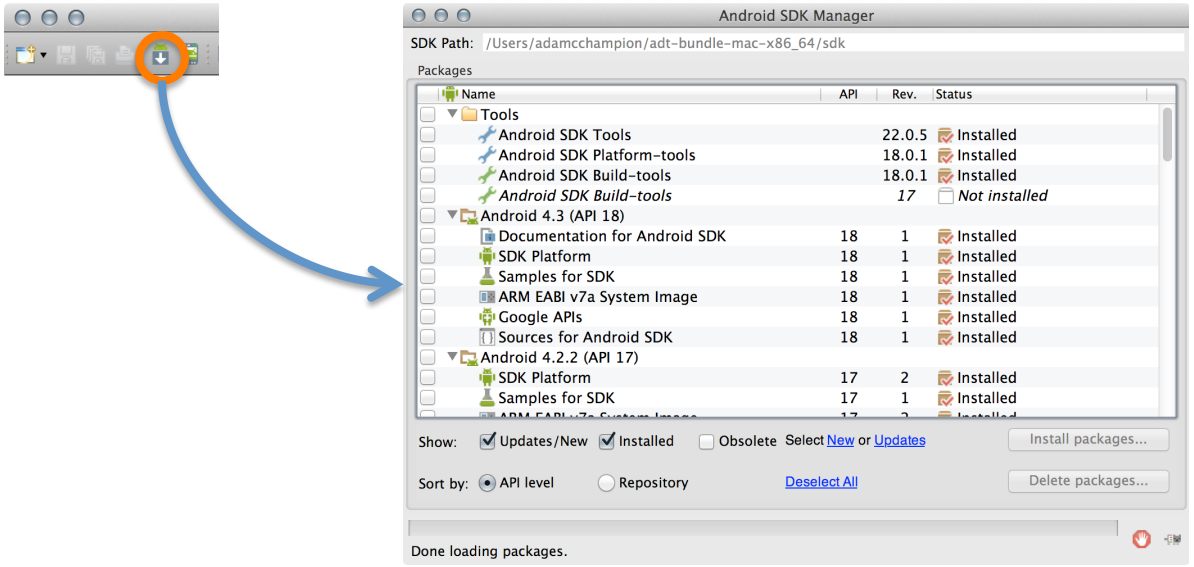
Getting Started (4)

- Go to Eclipse preferences (*Window→Preferences* or *ADT→Preferences*), select Android tab
- Make sure the Android SDK path is correct (`<adt-bundle-os>/sdk`)
- Strongly recommend testing with real Android device
  - Android emulator: *very* slow
  - Install USB drivers for your Android device



Getting Started (5)

- Bring up the Android SDK Manager
  - Recommended: Install Android 2.2, 2.3.3 APIs and 4.x API
  - Do not worry about Intel x86 Atom, MIPS system images



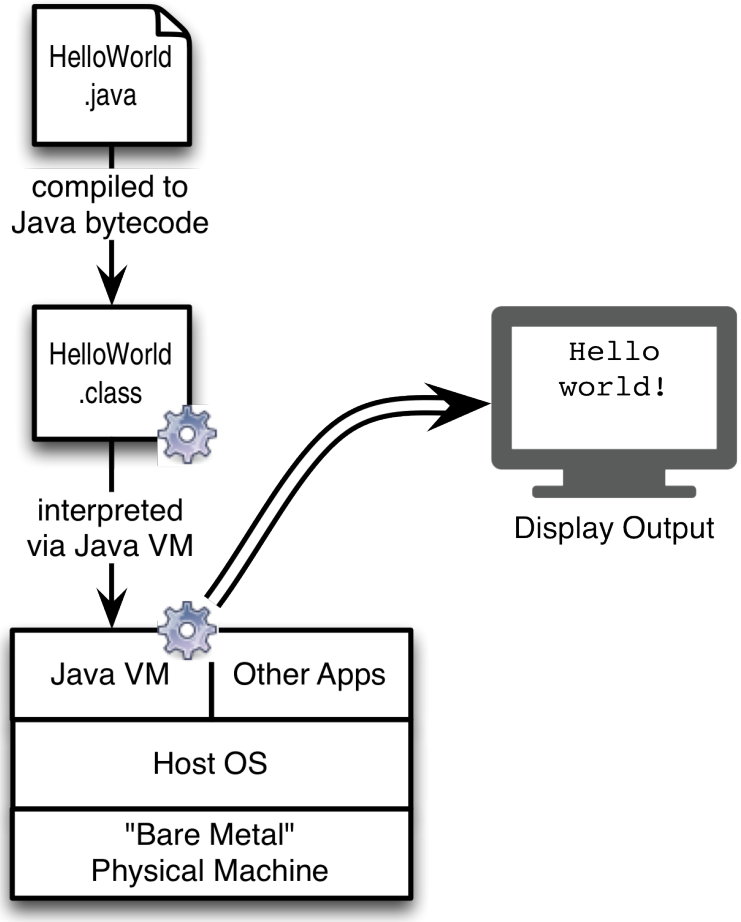
- In Eclipse, click *Window→Show View→Other*… and select views *Android→Devices*, *Android→LogCat*
- Now you're ready for Android development!

Outline

- Getting Started
- **Java: The Basics**
- Java: Object–Oriented Programming
- Android Programming

Java Programming Language

- Java: general–purpose language designed so developers write code once, it runs anywhere
- The key: Java Virtual Machine (JVM)
  - Program code compiled to JVM bytecode
  - JVM bytecode interpreted on JVM
- We'll focus on Java 5 (Android uses this). See chapters 1–7 in [1].

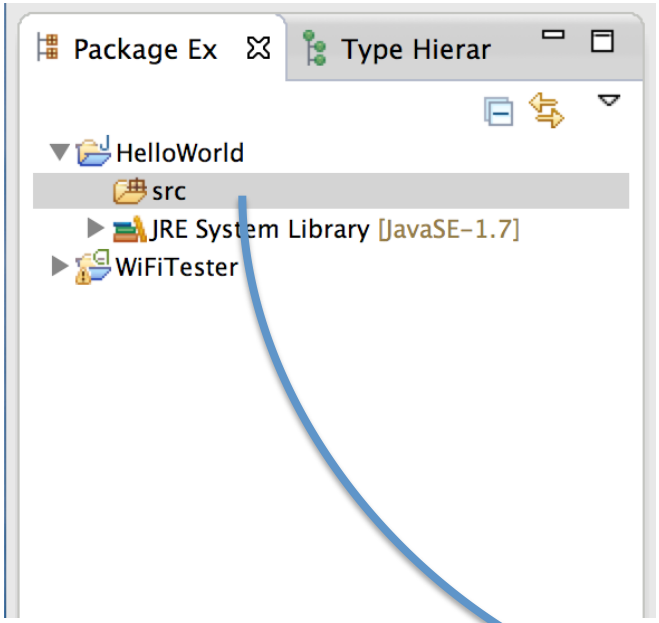


Our First Java Program (1)

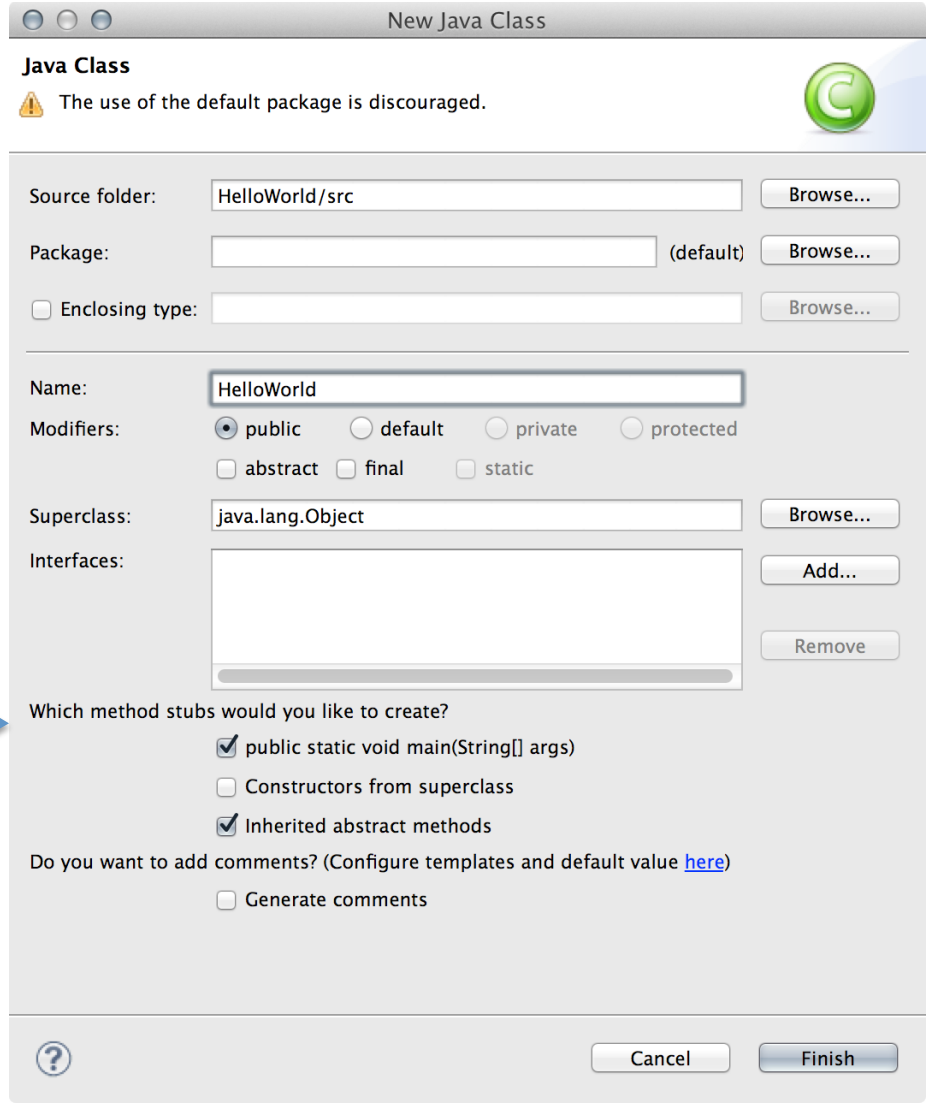
```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println(“Hello world!”);
    }
}
```

- Don’t forget to match curly braces {,} or semicolon at the end!
- In Eclipse: *File→New→Java Project*, name project “HelloWorld” (no quotes), click Finish. Open HelloWorld in Package Explorer. Right–click src directory, select *New→Class*, and create a class HelloWorld with a main() method. To run the program, right–click HelloWorld project, click Run As

Our First Java Program (2)



Right–click,
New→Class



Explaining the Program

- Every .java source file contains one class
  - We create a class `HelloWorld` that greets user
  - The class `HelloWorld` must have the same name as the source file `HelloWorld.java`
  - Our class has `public` scope, so other classes can "see" it
  - We'll talk more about classes and objects later
- Every Java program has a *method* `main()` that executes the program
  - Method "signature" must be exactly `public static void main(String[] args) { ... }`
  - This means: (1) `main()` is "visible" to other methods; (2) there is "only one" `main()` method in the class; and (3) `main()` has one argument (`args`, an array of `String` variables)
  - Java "thinks" `main()`, `Main()`, `miAN()` are different methods
- Every Java method has curly braces {,} surrounding its code
- Every statement in Java ends with a semicolon, e.g., `System.out.println("Hello world!");`
- Program prints "`Hello world!`" to the console, then quits

Basic Data Types (1)

- Java variables are instances of mathematical "types"
  - Variables can store (almost) any value their type can have
  - Example: the value of a boolean variable can be either `true` or `false` because any (mathematical) boolean value is *true* or *false*
  - Caveats for integer, floating–point variables: their values are subsets of values of mathematical integers, real numbers. Cannot assign *mathematical* 2^{500} to integer variable (limited range) or *mathematical* $\sqrt{2}$ to a floating–point variable (limited precision; irrational number).
  - Variable names must start with lowercase letter, contain only letters, numbers, _
- Variable *declaration*: `boolean b = true;`
- Later in the program, we might *assign* `false` to `b`: `b = false;`
- Java strongly suggests that variables be initialized at the time of declaration, e.g., `boolean b;` gives a compiler warning (`null` pointer)
- Constants defined using `final` keyword, e.g.,
  `final boolean falseBool = FALSE;`

Basic Data Types (2)

- Java's primitive data types: [5]

Primitive type	Size	Minimum	Maximum	Wrapper type
`boolean`	1–bit	N/A	N/A	`Boolean`
`char`	16–bit	Unicode 0	Unicode $2^{16} - 1$	`Character`
`byte`	8–bit	–128	+127	`Byte`
`short`	16–bit	-2^{15}	$+2^{15} - 1$	`Short`
`int`	32–bit	-2^{31}	$+2^{31} - 1$	`Integer`
`long`	64–bit	-2^{63}	$+2^{63} - 1$	`Long`
`float`	32–bit	IEEE 754	IEEE 754	`Float`
`double`	64–bit	IEEE 754	IEEE 754	`Double`

Note: All these types are signed, except `char`.

Basic Data Types (3)

- Sometimes variables need to be *cast* to another type, e.g., if finding average of integers:

```
int intOne = 1, intTwo = 2, intThree = 3, numInts = 2;
double doubOne = (double)intOne, doubTwo = (double)myIntTwo, doubThree =
(double)intThree;
double avg = (doubOne + doubTwo + doubThree)/(double)numInts;
```

- `Math` library has math operations like `sqrt()`, `pow()`, etc.
- `String`: immutable type for sequence of characters
  - Every Java variable can be converted to `String` via `toString()`
  - The `+` operation concatenates `String`s with other variables
  - Let `str` be a `String`. We can find `str`'s length (`str.length()`), substrings of `str` (`str.substring()`), and so on [6]

Basic Data Types (4)

- A literal is a "fixed" value of a variable type
  - `TRUE`, `FALSE` are `boolean` literals
  - '`A`', '`\t`', '`\"`', and '`\u03c0`' are `char` literals (escaped tab, quote characters, Unicode value for \pi)
  - `-1`, `0`, `035`, `0x1a` are `int` literals (last two are octal and hexadecimal)
  - `0.5`, `1.0`, `1E6`, `6.023E23` are `double` literals
  - "`At OSU`", "`Hello world!`" are `String` literals
- Comments:
  - Single-line: `// some comment to end of line`
  - Multi-line: `/* comments span multiple lines */`

Common Operators in Java

String	boolean	char	int	double
	`!`		`++ --`	
`+`	`||`		`+ -`	`+ -`
	`&&`		`* / %`	`* /`

		`< >`
		`<= >=`
		`== !=`
			`< >`
			`<= >=`
			`== !=`
				`< >`

Notes:

- Compare `String` objects using the `equals()` method, not `==` or `!=`
- `&&` and `||` use *short-circuit evaluation*. To see this, say `boolean canPigsFly = FALSE` and we evaluate (`canPigsFly && <some Boolean expression>`). Since `canPigsFly` is `FALSE`, the second part of the expression won't be evaluated.
- The second operand of `%` (integer modulus) must be positive.
- Don't compare `doubles` for equality. Instead, define a constant like so:

```
final double EPSILON = 1E-6; // or some other threshold
…     // check if Math.abs(double1 – double2) < EPSILON
```

Control Structures: Decision (1)

- Programs don't always follow "straight line" execution; they "branch" based on certain conditions
- Java decision idioms: if-then-else, switch
- if-then-else idiom:

```
if (<some Boolean expression>)
{
    // take some action
}
else if (<some other Boolean expression)
{
    // take some other action
}
else
{
    // do something else
}
```

Control Structures: Decision (2)

- Example:

```
final double OLD_DROID = 2.0, final double NEW_DROID = 4.0;
double myDroid = 4.1;
if (myDroid < OLD_DROID)
{
    System.out.println(“Antique!”);
}
else if (myDroid > NEW_DROID)
{
    System.out.println(“Very modern!”);
}
else
{
    System.out.println(“Your device: barely supported.”);
}
```

- Code prints “`Very modern!`” to the screen.
- What if `myDroid == 1.1`? `myDroid == 2.3`?

Control Structures: Decision (3)

- Example two:

```
final double JELLY_BEAN = 4.1, final double ICE_CREAM = 4.0;
final double EPSILON = 1E-6;
double myDroid = 4.1;
if (myDroid > ICE_CREAM)
{
    if (Math.abs(myDroid – ICE_CREAM) < EPSILON)
    {
        System.out.println(“Ice Cream Sandwich”);
    }
    else
    {
        System.out.println(“Jelly Bean”);
    }
}
else
{
    System.out.println(“Old version”);
}
```

- Code prints “`Jelly Bean`” to screen. Note nested if-then-else, `EPSILON` usage.

Control Structures: Decision (4)

- Other idiom: switch
- Only works when comparing an int or boolean variable against a fixed set of alternatives
- Example:

```
int api = 10;
switch (api)
{
    case 3:  System.out.println(“Cupcake”); break;
    case 4:  System.out.println(“Donut”); break;
    case 7:  System.out.println(“Éclair”); break;
    case 8:  System.out.println(“Froyo”); break;
    case 10: System.out.println(“Gingerbread”); break;
    case 11: System.out.println(“Honeycomb”); break;
    case 15: System.out.println(“Ice Cream Sandwich”); break;
    case 16: System.out.println(“Jelly Bean”); break;
    default: System.out.println(“Other”); break;
}
```

Control Structures: Iteration (1)

- Often, blocks of code should loop while a condition holds (or fixed # of times)
- Java iteration idioms: while, do-while, for
- While loop: execute loop as long as condition is true (checked each iteration)
- Example:

```
String str = “aaaaa”;
int minLength = 10;

while (str.length() < minLength)
{
    str = str + “a”;
}

System.out.println(str);
```

- Loop executes 5 times; code terminates when `str = “aaaaaaaaaa”`
- Notice: if the length of `str` was `minLength`, the while loop would not execute

Control Structures: Iteration (2)

While Loop

```
String str = “aaaaaaaaaa”;
int minLength = 10;

while (str.length() <
minLength)
{
    str = str + “a”;
}

System.out.println(str);
```

Do-While Loop

```
String str = “aaaaaaaaaa”;
int minLength = 10;

do
{
    str = str + “a”;
}
while (str.length() <
minLength)

System.out.println(str);
```

Unlike the while loop, the do-while loop executes at least once so long as condition is true. The while loop prints “aaaaaaaaaa” whereas the do-while loop prints “aaaaaaaaaaa” (11 as)

Control Structures: Iteration (3)

- The for loop has the following structure:

```
for (<expression1>; <expression2>; <expression3>)
{
    . . .
}
```

- Semantics:
  - `<expression1>` is loop initialization (run once)
  - `<expression2>` is loop execution condition (checked every iteration)
  - `<expression3>` is loop update (run every iteration)
- Example:

```
int i;
for (i = 0; i < 10; i++)
{
    System.out.println("i = " + i);
}
System.out.println("i = " + i);
```

- What do you think this code does?

Methods and Design-by-Contract (1)

- Design your own methods to perform specific, well-defined tasks
- Each method has a *signature*:

```
public static ReturnType method(paramType1 param1, … paramTypeN paramN)
{
    // perform certain task
}
```

- Example: a method to compute area of rectangle:

```
public static double findRectArea(double length, double width)
{
     return length * width;
}
```

- Each method has a precondition and a postcondition
  - Precondition: constraints method's caller must satisfy to call method
  - Postcondition: guarantees method provides if preconditions are met
- For our example:
  - Precondition: `length > 0.0`, `width > 0.0`
  - Postcondition: returns `length × width` (area of rectangle)

Methods and Design-by-Contract (2)

- In practice, methods are annotated via JavaDoc, e.g.,

```
/**
    Compute area of rectangle.

    @param length Length of rectangle
    @param width Width of rectangle
    @return Area of rectangle
*/
```

- Methods called from `main()` (which is `static`) need to be defined `static` too
- Some methods may not return anything (`void`)

Array Data Structure

- Array: fixed-length sequence of variable types; cannot change length at run-time
  Examples:
```
final int NUMSTUDENTS = 10;
String[] students; // Declaration
String[] students = new String[NUMSTUDENTS];
    // Declaration and initialization
String[] moreStudents = { “Alice”, “Bob”, “Rohit”, “Wei”};
    // Declaration and explicit initialization
System.out.println(moreStudents.length) // Prints 4
```
- Enhanced `for` loop: executed for each element in array
  Example:
```
for (String student: moreStudents)
{
    System.out.println(student + “, ”);
}
```
- Prints “`Alice, Bob, Rohit, Wei,`” to screen
- Array indices are numbered 0, …, *N*–1; watch for off-by-one errors! `moreStudents[0]` is “`Alice`”; `moreStudents[3]` is “`Wei`”

Two-Dimensional Arrays

- We can have two-dimensional arrays.
  Example:

```
final int ROWS = 3; final int COLUMNS = 3;
char[][] ticTacToe = new char[ROWS][COLUMNS]; // declare
for (int i = 0; i < ROWS; i++)
{
    for (int j = 0; j < COLUMNS; j++)
    {
        ticTacToe[i][j] = '_'; // Initialize to 'blank'
    }
}
// Tic-tac-toe logic goes here (with 'X's, 'O's)
```

- `ticTacToe.length` returns number of rows;
  `ticTacToe[0].length` returns number of columns
- Higher-dimensional arrays are possible too

Parameterized Data Structures

- We can define data structures in terms of an arbitrary variable type (call it Item).
- `ArrayList<Item>`, a variable-length array that can be modified at run-time. Examples:

```
ArrayList<String> arrStrings = new ArrayList<String>();
ArrayList<Double> arrDoubles = new ArrayList<Double>();
arrStrings.add("Alice"); arrStrings.add("Bob"); arrStrings.add("Rohit");
arrStrings.add("Wei");
String str = arrStrings.get(1); // str becomes "Bob"
arrStrings.set(2, "Raj"); // "Raj" replaces "Rohit"
System.out.println(arrStrings.size()); // prints 4
```

- Notice:
  - Need to call `import java.util.ArrayList;` at beginning of program
  - Off-by-one indexing: cannot call `arrStrings.get(4);`
  - *Auto-boxing:* we cannot create an `ArrayList` of `doubles`. We need to replace `double` with *wrapper class* `Double`. (Recall the "primitive data types" table)
- Other parameterized data types include `Lists`, `Sets`, `Maps`, `Stacks`, `Queues`, `Trees` (see chapters 14–16 in [1])

Exception Handling (1)

- If we had called `arrStrings.get(4)`, we would have an error condition
  - The JVM throws an `IndexOutOfBounds` exception, halts execution

ArrayException.java

```
1  import java.util.ArrayList;
2
3
4  public class ArrayException
5  {
6
7      /**
8       * @param args
9       */
10     public static void main(String[] args)
11     {
12         // TODO Auto-generated method stub
13         ArrayList<String> arrStrings = new ArrayList<String>();
14         arrStrings.add("Alice");
15         arrStrings.add("Bob");
16         arrStrings.add("Rohit");
17         arrStrings.add("Wei");
18         int size = arrStrings.size();
19         arrStrings.get(size);
20     }
21
22 }
23
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 4, Size: 4
        at java.util.ArrayList.rangeCheck(ArrayList.java:604)
        at java.util.ArrayList.get(ArrayList.java:382)
        at ArrayException.main(ArrayException.java:19)
```

Exception Handling (2)

- We handle exceptions using the try-catch-finally structure:

```
try
{
    // Code that could trigger an exception
}
catch (IndexOutOfBoundsException e) // Or another Exception
{
    // Code that “responds” to exception, e.g.,
    e.printStackTrace();
}
finally
{
    // Code that executes regardless of whether exception occurs
}
```

- There can be many `catch` blocks for different `Exceptions`, but there is only one `try` block and one (optional) `finally` block. (See Section 7.4 in [1] for the full “hierarchy” of `Exceptions`)
- Exceptions always need to be caught and “reported”, especially in Android

Outline

- Getting Started
- Java: The Basics
- **Java: Object–Oriented Programming**
- Android Programming

Objects and Classes (1)

- *Classes* serve as "blueprints" that describe the states and behaviors of *objects*, which are actual "instances" of classes
- For example, a Vehicle class describes a motor vehicle's blueprint:
  - States: "on/off", driver in seat, fuel in tank, speed, etc.
  - Behaviors: startup, shutdown, drive "forward", shift transmission, etc.
- There are many possible Vehicles, e.g., Honda Accord, Mack truck, etc. These are *instances* of the Vehicle blueprint
- Many Vehicle states are specific to each Vehicle object, e.g., on/off, driver in seat, fuel remaining. Other states are specific to the class of Vehicles, not any particular Vehicle (e.g., keeping track of the "last" Vehicle ID # assigned). These correspond to *instance fields* and *static fields* in a class.
- Notice: we can operate a vehicle without knowing its implementation "under the hood". Similarly, a class makes public *instance methods* by which objects of this class can be manipulated. Other methods apply to the set of all Vehicles (e.g., set min. fuel economy). These correspond to *static methods* in a class

Objects and Classes (2)

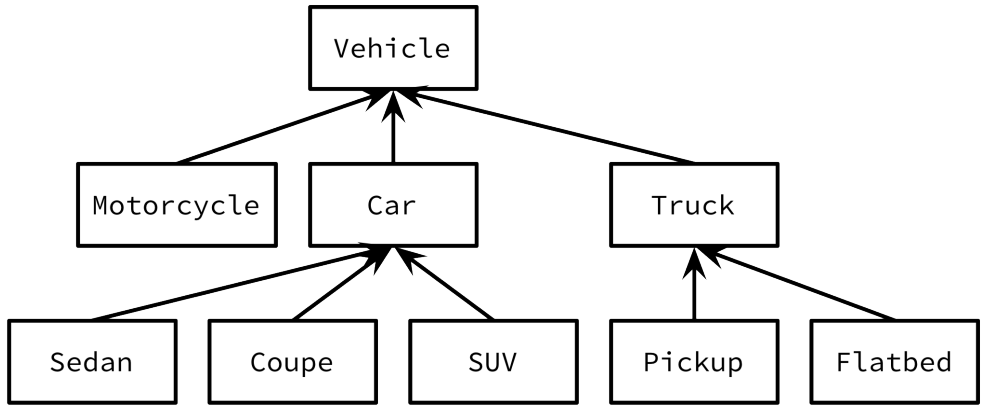
```
public class Vehicle
{
    // Instance fields (some omitted for brevity)
    private boolean isOn = false;
    private boolean isDriverInSeat = false;
    private double fuelInTank = 10.0;
    private double speed = 0.0;

    // Static fields
    private static String lastVin = “4A4AP3AU*DE999998”;

    // Instance methods (some omitted for brevity)
    public Vehicle() { … } // Constructor
    public void startUp() { … }
    public void shutOff() { … }
    public void getIsDriverInSeat() { … }  // getter, setter methods
    public void setIsDriverInSeat() { … }
    private void manageMotor() { … }    // More private methods …

    // Static methods
    public static void setVin(String newVin) { … }
}
```

Objects and Classes (3)

- How to use the `Vehicle` class:
  - First, create a new object via constructor `Vehicle()`, e.g., `Vehicle myCar = new Vehicle();`
  - Change Vehicle states, e.g., `startUp()` or `shutOff()` the `Vehicle`
  - You can imagine other use cases
  - Mark a new `Vehicle`'s ID number (VIN) as "taken" by calling `Vehicle.setVin(…)`
  - Caveat: VINs more complex than this (simple) implementation [7]
- Notes:
  - Aliasing: If we set `Vehicle myTruck = myCar`, both `myCar` and `myTruck` "point" to the same variable. Better to perform "deep copy" of `myCar` and store the copy in `myTruck`
  - `null` reference: refers to no object, cannot invoke methods on `null`
  - Implicit parameter and the `this` reference
- Access control: public, protected, private

Inheritance (1)

- Types of `Vehicles`: `Motorcycle`, `Car`, `Truck`, etc. Types of `Cars`: `Sedan`, `Coupe`, `SUV`. Types of `Trucks`: `Pickup`, `Flatbed`.
- Induces inheritance hierarchy
- Subclasses inherit fields/methods from superclasses.
- Subclasses can add new fields/methods, override those of parent classes
- For example, `Motorcycle`'s `driveForward()` method differs from `Truck`'s `driveForward()` method



Inheritance (2)

- Inheritance denoted via `extends` keyword

```
public class Vehicle
{
    …
    public void driveForward
(double speed)
    {
        // Base class method
    }
}
```

```
public class Motorcycle
extends Vehicle
{
    …
    public void driveForward
(double speed)
    {
        // Apply power…
    }
}
```

Inheritance (3)

```
public class Truck extends Vehicle
{
    private boolean useAwd = true;
…
    public Truck(boolean useAwd) { this.useAwd = useAwd; }
…
    public void driveForward(double speed)
    {
        if (useAwd)
        {
            // Apply power to all wheels…
        }
        else
        {
            // Apply power to only front/back wheels…
        }
    }
}
```

Polymorphism

- Suppose we create `Vehicles` and invoke the `driveForward()` method:

```
Vehicle vehicle = new Vehicle();
Vehicle motorcycle = new Motorcycle();
Truck truck1 = new Truck(true);
Vehicle truck2 = new Truck(false);
// Code here to start vehicles…
vehicle.driveForward(5.0);
motorcycle.driveForward(10.0);
truck1.driveForward(15.0);
truck2.driveForward(10.0);
```

- For `vehicle`, `Vehicle`'s `driveForward()` method is invoked
- For `motorcycle`, `Motorcycle`'s `driveForward()` method is invoked
- With `truck1` and `truck2`, `Truck`'s `driveForward()` function is invoked (with all-wheel drive for `truck1`, not for `truck2`).
- Dynamic method lookup: Java looks at objects' actual types in determining which method to invoke
- Polymorphism: feature where objects of different subclasses can be treated the same way. All `Vehicles` `driveForward()` regardless of (sub)class.

The `Object` Class

- *Every* class in Java is a subclass of `Object`
- Important methods in `Object`:
  - `toString()`: Converts the `Object` into a `String` representation
  - `equals()`: Compares "contents" of `Objects` to see if they're the same
  - `hashCode()`: Hashes the `Object` to a fixed-length `String`, useful for data structures like `HashMap`, `HashSet`
- If you create your own class, you should override `toString()` and `hashCode()`

Interfaces

- Java interfaces abstractly specify methods to be implemented
- Intuition: decouple method definitions from implementations (clean design)
- Interfaces, implementations denoted by `interface`, `implements` keywords
- Example:

```
public interface Driveable
{
    public void driveForward(double speed);
}

public class Vehicle implements Driveable
{
    public void driveForward(double speed) { // implementation }
}

public class Motorcycle extends Vehicle implements Driveable
{
    public void driveForward(double speed) { // implementation }
}
```

The `Comparable` Interface

- Comparing `Objects` is important, e.g., sorting in data structures
- The `Comparable` interface compares two `Objects`, e.g., `a` and `b`:

```
public interface Comparable
{
    int compareTo(Object otherObject);
}
```

- `a.compareTo(b)` returns negative integer if `a` "comes before" `b`, `0` if `a` is the same as `b`, and a positive integer otherwise
- In your classes, you should implement `Comparable` to facilitate `Object` comparison

Object-Oriented Design Principles

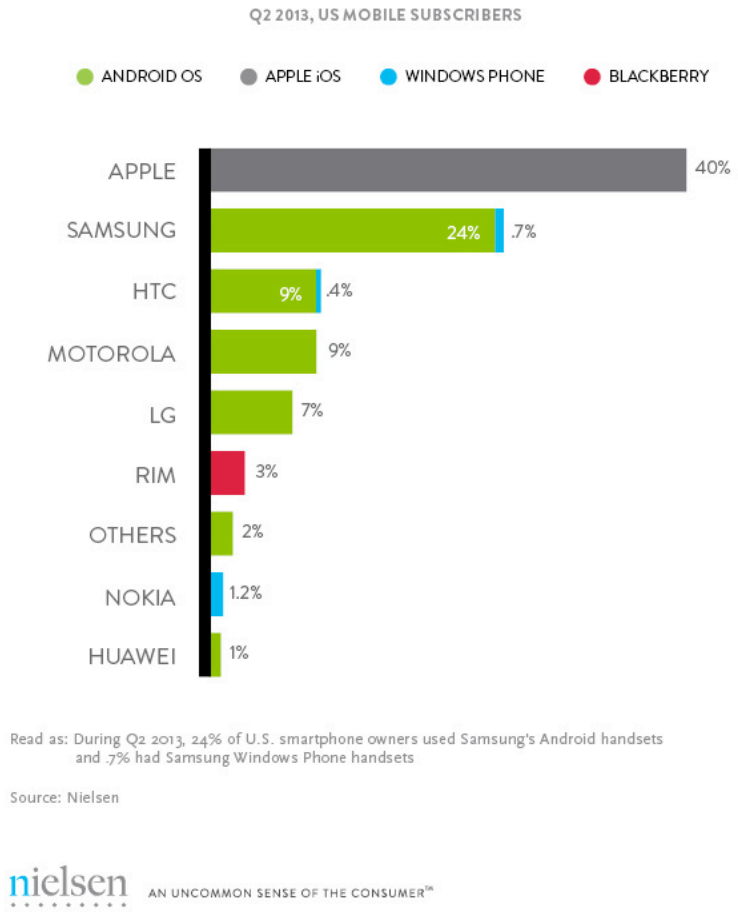
- Each class should represent a single concept
  - Don't try to fit all functionality into a single class
  - Consider a class per "noun" in problem description
  - Factor functionality into classes, interfaces, etc. that express the functionality with minimal coupling
- For software projects, start from use cases (how customers will use software: high level)
  - Then identify classes of interest
  - In each class, identify fields and methods
  - Class relationships should be identified: is-a (inheritance), has-a (aggregation), implements interface, etc.
- Packages provide class organization mechanism
  - Examples: java.lang.*, java.util.*, etc.
  - Critical for organizing large numbers of classes!
  - All classes in a package can "see" each other (scope)

Outline

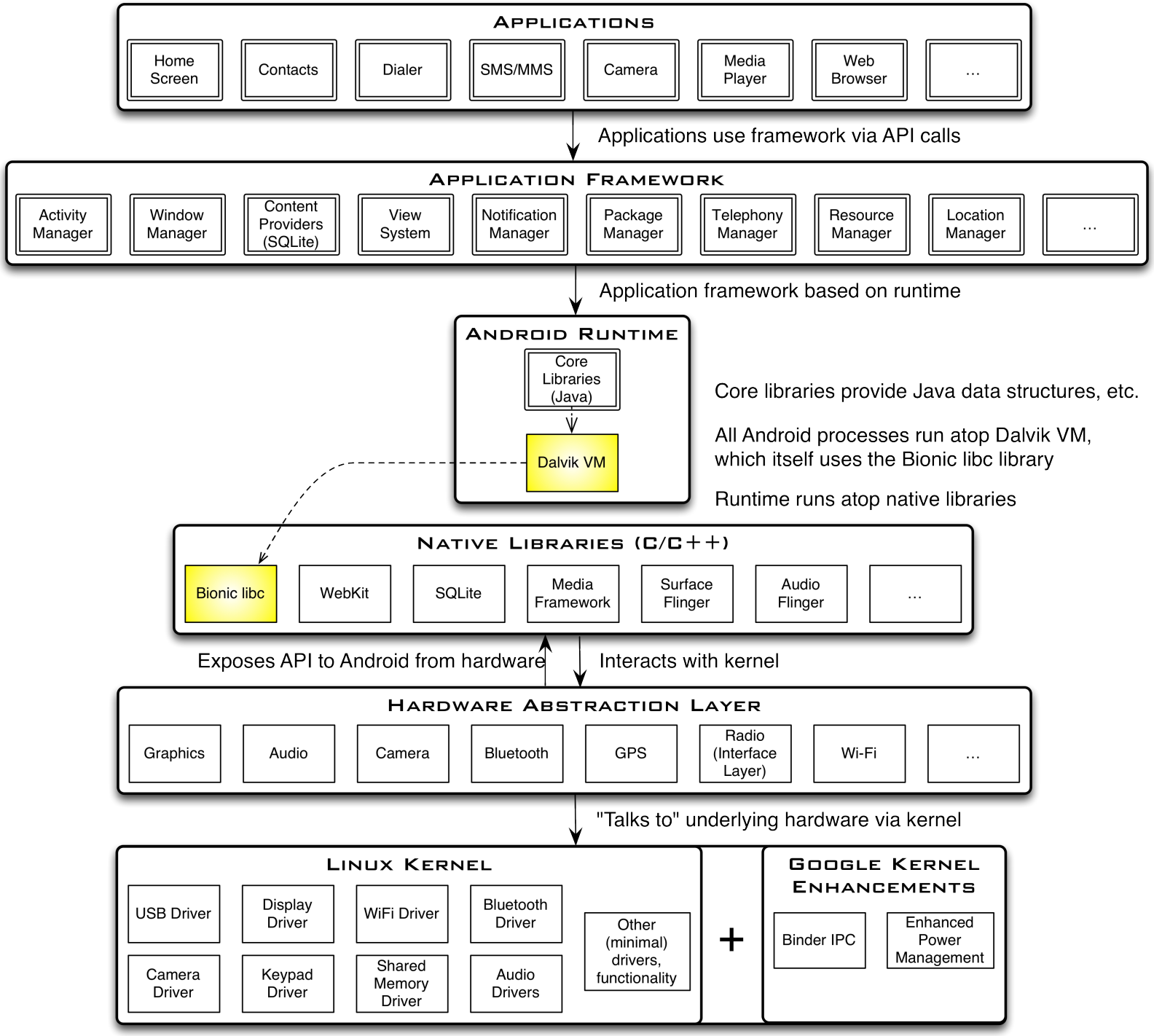
- Getting Started
- Java: The Basics
- Java: Object–Oriented Programming
- **Android Programming**

Introduction to Android

- Popular mobile device OS: 52% of U.S. smartphone market [8]
- Developed by Open Handset Alliance, led by Google
- Google claims 900,000 Android device activations [9]

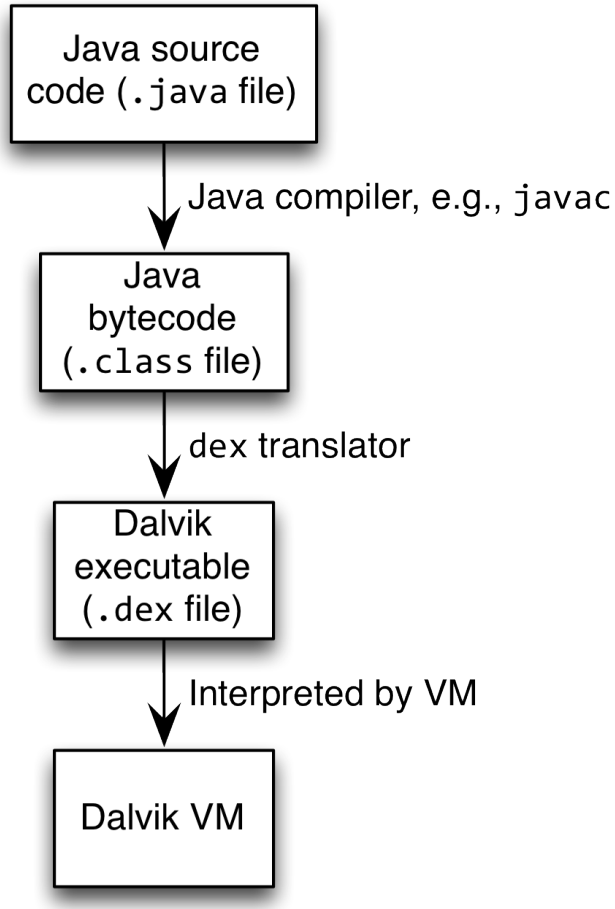


Source: [8]


Applications
Home Screen
Contacts
Dialer
SMS/MMS
Camera
Media Player
Web Browser
…
Applications use framework via API calls
Application Framework
Activity Manager
Window Manager
Content Providers (SQLite)
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
…
Application framework based on runtime
Android Runtime
Core Libraries (Java)
Dalvik VM
Core libraries provide Java data structures, etc.
All Android processes run atop Dalvik VM, which itself uses the Bionic libc library
Runtime runs atop native libraries
Native Libraries (C/C++)
Bionic libc
WebKit
SQLite
Media Framework
Surface Flinger
Audio Flinger
…
Exposes API to Android from hardware
Interacts with kernel
Hardware Abstraction Layer
Graphics
Audio
Camera
Bluetooth
GPS
Radio (Interface Layer)
Wi-Fi
…
"Talks to" underlying hardware via kernel
Linux Kernel
USB Driver
Display Driver
WiFi Driver
Bluetooth Driver
Camera Driver
Keypad Driver
Shared Memory Driver
Audio Drivers
Other (minimal) drivers, functionality
+
Google Kernel Enhancements
Binder IPC
Enhanced Power Management

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



Android Highlights (2)

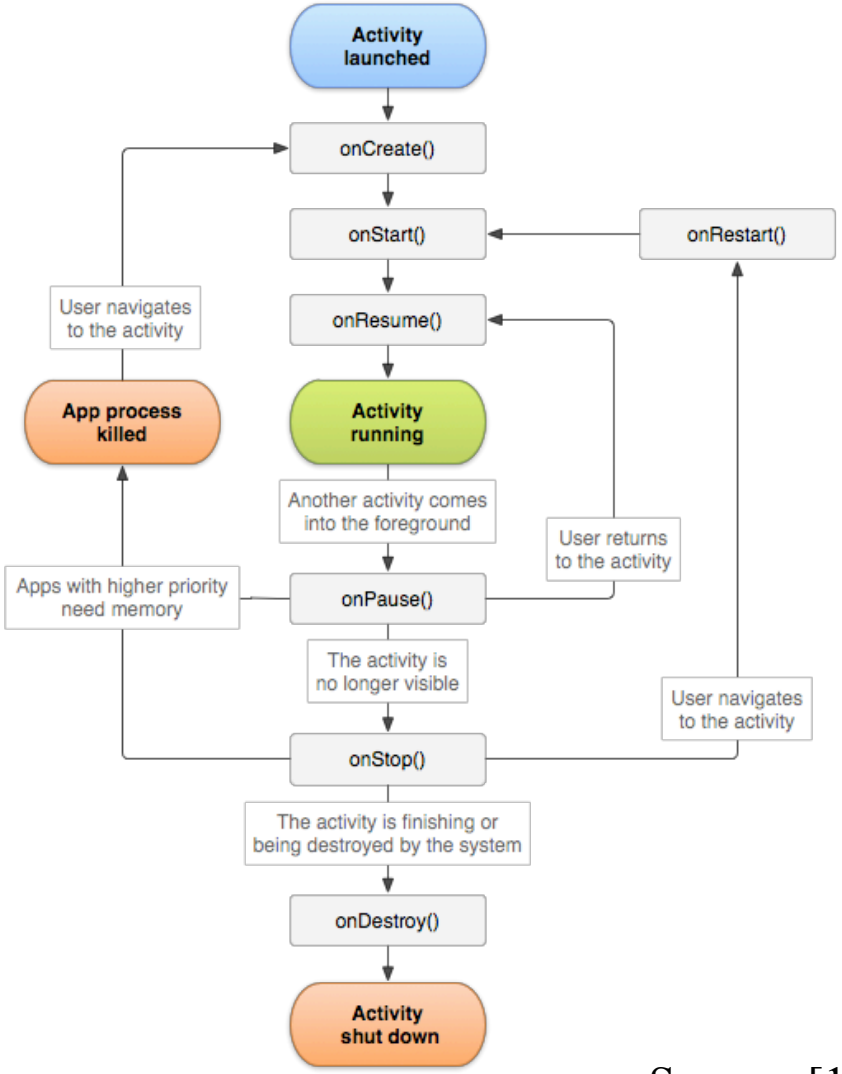
- Android apps written in Java 5
  - Actually, a Java dialect (Apache Harmony)
  - Everything we've learned still holds
- Apps use four main components:
  - `Activity`: A "single screen" that's visible to user
  - `Service`: Long-running background "part" of app (*not* separate process or thread)
  - `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 `Threads`, `AsyncTasks`



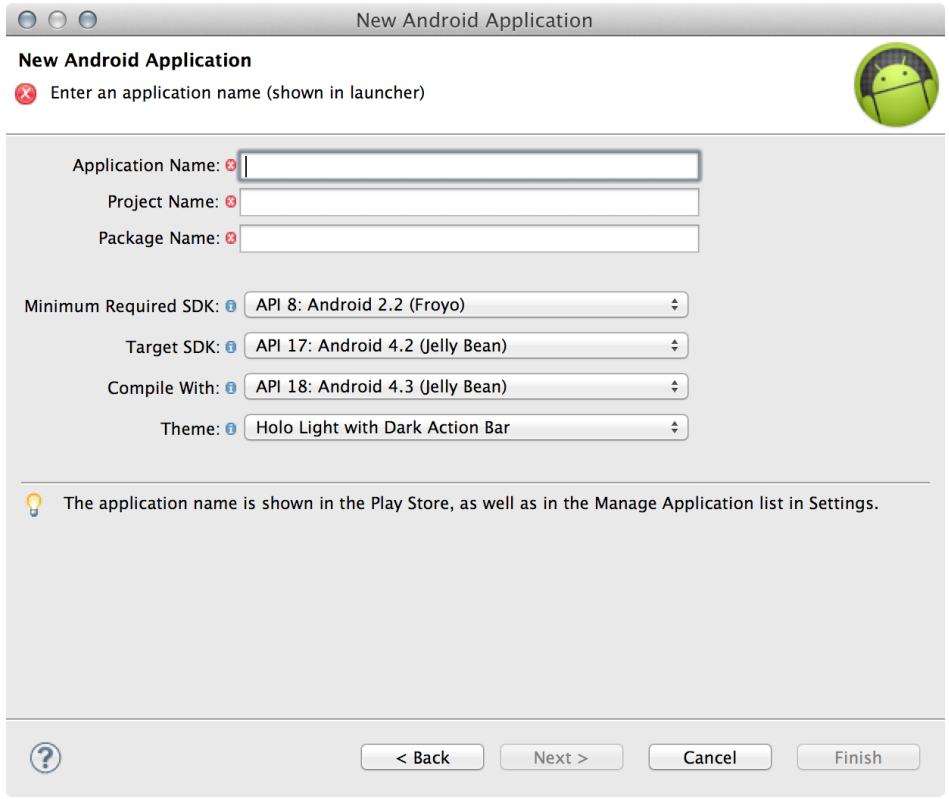
Source: [12]

App Creation Checklist

- If you own an Android device:
  - Ensure drivers are installed
  - Enable developer options on device under *Settings*, specifically *USB Debugging*
    - Android 4.2, 4.3: Go to *Settings→About phone*, press *Build number* 7 times to enable developer options
- For Eclipse:
  - Make sure you've enabled LogCat, Devices views. Click a connected device in Devices to see its log
  - Programs should log states via `android.util.Log`'s `Log.d(APP_TAG_STR, "debug")`, where `APP_TAG_STR` is a `final String` tag denoting your app
  - Other commands: `Log.e()` (error); `Log.i()` (info); `Log.w()` (warning); `Log.v()` (verbose) – same parameters

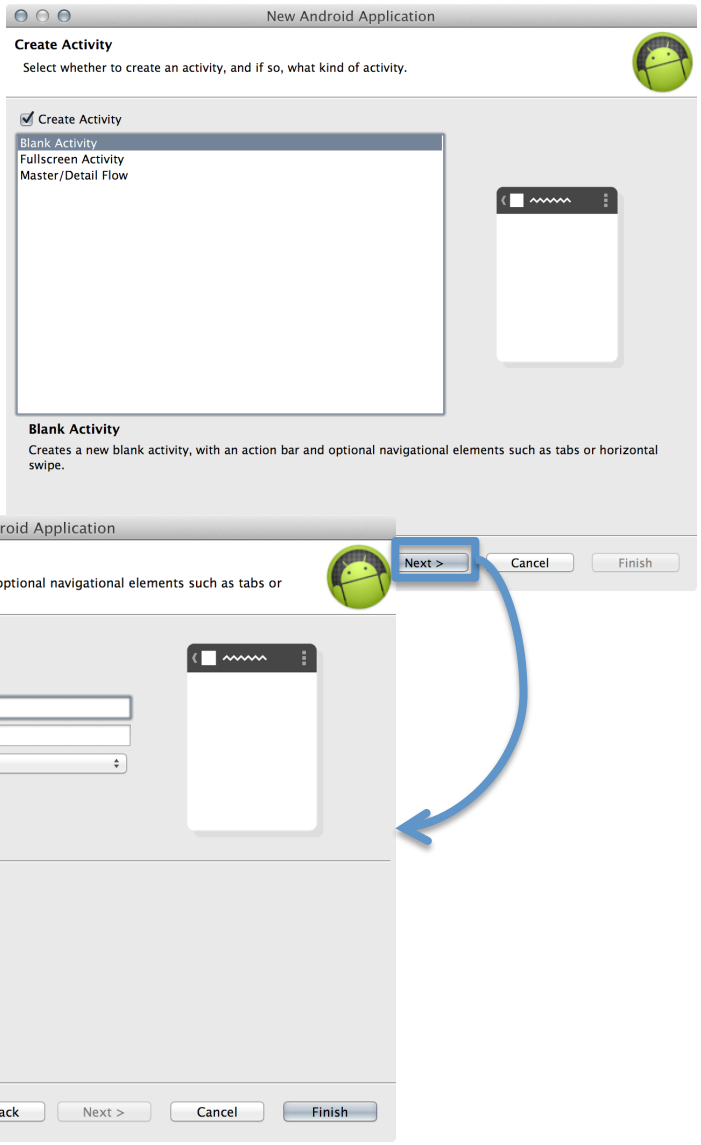
Creating Android App (1)

- Creating Android app project in Eclipse:
  - Go to *File→New→Other…*, select Android Application Project (Android folder), click Next
  - Enter app, project name
  - Choose package name using "reverse URL" notation, e.g., `edu.osu.myapp`
  - Select APIs for app, then click Next



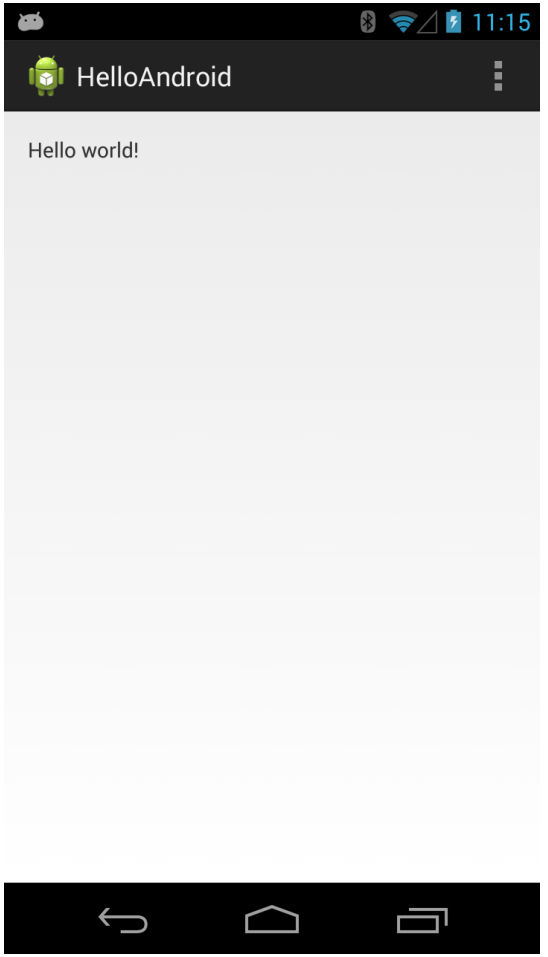
Creating Android App (2)

- Click Next for the next two windows
- You'll see the "Create Activity" window. Select "Blank Activity" and click Next.
- Enter Activity's name and click Finish
- This automatically creates "Hello World" app



Deploying the App

- Two choices for deployment:
  - Real Android device
  - Android virtual device
- Plug in your real device; otherwise, create an Android virtual device
- Emulator is slow. Try Intel accelerated version, or perhaps `http://www.genymotion.com/`
- Run the app: right click project name, select *Run As→Android Application*



Underlying Source Code

src/…/MainActivity.java

```
package edu.osu.helloandroid;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

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

     @Override
     public boolean onCreateOptionsMenu(Menu menu)
     {
          // Inflate the menu; this adds items to the action bar if it is present.
          getMenuInflater().inflate(R.menu.main, menu);
          return true;
     }
}
```

Underlying GUI Code

res/layout/activity_main.xml

```
<RelativeLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

– `RelativeLayouts` are quite complicated. See [13] for details

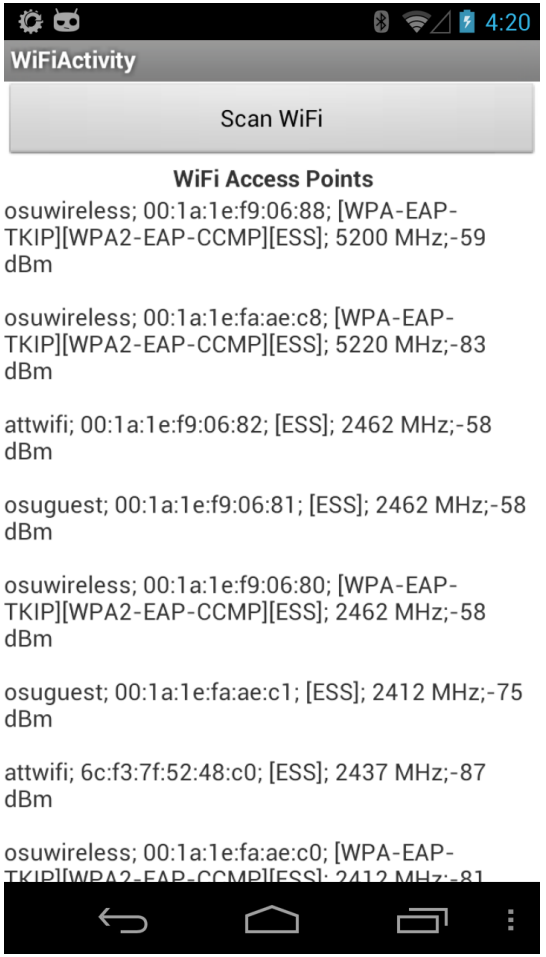
The App Manifest

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.osu.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="edu.osu.helloandroid.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

A More Interesting App

- We'll now examine an app with more features: WiFi Tester (code on class website)
- Press a button, scan for WiFi access points (APs), display them



Underlying Source Code (1)

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

	// Set up WifiManager.
	mWifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);

	// Create listener object for Button. When Button is pressed, scan for
	// APs nearby.
	Button button = (Button) findViewById(R.id.button);
	button.setOnClickListener(new View.OnClickListener()
	{
		public void onClick(View v)
		{
			boolean scanStarted = mWifiManager.startScan();

			// If the scan failed, log it.
			if (!scanStarted) Log.e(TAG, "WiFi scan failed...");
		}
	});

	// Set up IntentFilter for "WiFi scan results available" Intent.
	mIntentFilter = new IntentFilter();
	mIntentFilter.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
}
```

Underlying Source Code (2)

- Code much more complex
- First get system `WifiManager`
- Create listener `Object` for button that performs scans
- We register Broadcast Receiver, `mReceiver`, to listen for `WifiManager`'s "finished scan" system event (expressed as `Intent WifiManager.SCAN_RESULTS_AVAILABLE_ACTION`)
- Unregister Broadcast Receiver when leaving `Activity`

```
@Override
protected void onResume()
{
    super.onResume();
    registerReceiver(mReceiver,
mIntentFilter);
}

@Override
protected void onPause()
{
    super.onPause();
unregisterReceiver(mReceiver);
}
```

The Broadcast Receiver

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver()
{
      @Override
      public void onReceive(Context context, Intent intent)
      {
            String action = intent.getAction();
            if (WifiManager.SCAN_RESULTS_AVAILABLE_ACTION.equals(action))
            {
                  Log.e(TAG, "Scan results available");
                  List<ScanResult> scanResults = mWifiManager.getScanResults();
                  mApStr = "";
                  for (ScanResult result : scanResults)
                  {
                        mApStr = mApStr + result.SSID + "; ";
                        mApStr = mApStr + result.BSSID + "; ";
                        mApStr = mApStr + result.capabilities + "; ";
                        mApStr = mApStr + result.frequency + " MHz;";
                        mApStr = mApStr + result.level + " dBm\n\n";
                  }
                  // Update UI to show all this information.
                  setTextView(mApStr);
            }
      }
};
```

User Interface

Updating UI in code

```
private void setTextView(String str)
{
    TextView tv = (TextView)
findViewById(R.id.textview);
    tv.setMovementMethod(new
ScrollingMovementMethod());
    tv.setText(str);
}
```

- This code simply has the UI display all collected WiFi APs, makes the text information scrollable

UI Layout (XML)

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:text="@string/button_text"/>

    <TextView
        android:id="@+id/header"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/ap_list"
        tools:context=".WiFiActivity"
        android:textStyle="bold"
        android:gravity="center">
    </TextView>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        tools:context=".WiFiActivity"
        android:id="@+id/textview"
         android:scrollbars="vertical">
    </TextView>
</LinearLayout>
```

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 }`
- 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`

Concurrency: Threads (1)

- Thread: program unit (within process) executing independently
- Basic idea: create class that implements Runnable interface
  - Runnable has one method, run(), that contains code to be executed
  - Example:

```
public class OurRunnable implements Runnable
{
    public void run()
    {
        // run code
    }
}
```

- Create a Thread object from Runnable and start() Thread, e.g.,

```
Runnable r = new OurRunnable();
Thread t = new Thread(r);
t.start();
```

- Problem: this is cumbersome unless Thread code is reused

Concurrency: Threads (2)

- Easier approach: anonymous inner classes, e.g.,

```
Thread t = new Thread(new Runnable(
    {
        public void run()
        {
            // code to run
        }
    });
t.start();
```

- Idiom essential for one-time network connections in Activities
- However, Threads can be difficult to synchronize, especially with UI thread in Activity. AsyncTasks are better suited for this

Concurrency: AsyncTasks

- `AsyncTask` encapsulates asynchronous task that interacts with UI thread in `Activity`:

```
public class AsyncTask<Params, Progress, Result>
{
    protected Result doInBackground(ParamType param)
    {
        // code to run in background
        publishProgress(ProgressType progress); // UI
        …
        return Result;
    }

    protected void onProgressUpdate(ProgressType progress)
    {
        // invoke method in Activity to update UI
    }
}
```

- Extend `AsyncTask` with your own class
- Documentation at http://developer.android.com

Thank You

Any questions?

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., “Who’s Winning the U.S. Smartphone Market?”, 6 Aug. 2013, http://www.nielsen.com/us/en/newswire/2013/whos-winning-the-u-s-smartphone-market-.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