Google Android Development

Lesson #2

Getting Familiar with Java

Java is the programming language that we are going to be using to build our Android applications. It is important to understand its structure, syntax, and ways of doing things in order to be able to write applications.

Luckily, Java is a relatively easy programming language to start working with, and there are countless resources on the web that can help you solve problems that you run into during development.

Java 101

Java is a high-level programming language developed by Sun Microsystems, now owned by Oracle.

The Java programming language is a high-level language that can be characterized in the following way:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Multithreaded
- Robust
- Dynamic
- Secure

http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Java 101

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the java compiler. In our case, we're working with Eclipse, which provides us with the editor, and compiler environment in one environment.

A .class file does not contain code that is native to your processor; it instead contains bytecode — the machine language of the Java Virtual Machine (Java VM).

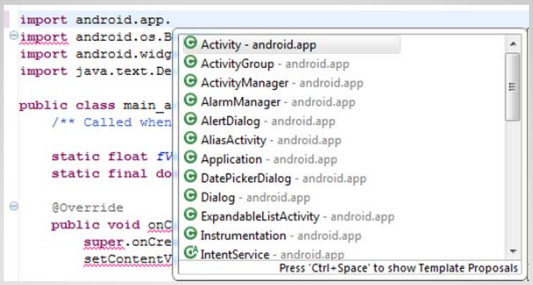
Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, Linux, Mac OS, Android phones, and many other operating systems.

http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html

Syntax

Its important to note that Java is a case-sensitive language. This means that typing in a statement and not using the correct case can easily produce errors.

Java also uses the dot notation. The dot notation provides access to all members of a class within a specific class path.



Syntax

Each statement needs to be ended with a semi-colon. There are cases when this is not necessary, however, to keep things simple, I suggest always adding a semi-colon at the end of each statement.

Statements can be broken down into multiple lines, with the semi-colon added only to the last line of the statement.

Ex:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);

final TextView tOut = (TextView) findViewById(R.id.txtOut);

myPC newPC = new myPC();
newPC.speed = 1000;
newPC.hd_size = 500;
newPC.screen_size = "1280x720";
```

Syntax

Groups of statements, typically known as code blocks, are always surrounded by the **{** and **}** braces.

Example:

```
public class Notebook extends myPC {

    String form_factor = "Notebook";

    String printAttribs() {
        String sRtn = "";
        sRtn += "Speed (in Mhz): " + this.speed + "\n";
        sRtn += "Hard Drive (in GB): " + this.hd_size + "\n";
        sRtn += "Screen Res: " + this.screen_size + "\n";
        sRtn += "Form Factor: " + form_factor + "\n";
        return sRtn;
    }
}
```

Comments

You can create single line, or multi-line comments in Java. Comments are important, because they help the programmer understand their code. You should make an effort to add comments to each section of your code so you can refer back to it at a later time, and clearly understand what you did and why.

JavaDoc is a built-in tool of Java that lets you generate class level documentation for you application based on the JavaDoc comments that are found in your code. The generated documetation is in HTML format.

Examples:

```
/* This is a multi-line comment
     The comment continues until… */

// This is a single-line comment

/** This is a JavaDoc comment
 *  This is used for documenting your work
 */
```

Object Oriented Model

Java is an object oriented programming language. An easy way to understand this, is to look at a real-life object, and how that could be broken down into Java's terms.

Consider a phone – yes, its an object.

A phone may have some of the following characteristics:

- Carrier (AT&T, Verizon, T-Mobile, etc.)
- Type (smartphone, flip phone, etc.)
- Screen size
- Type of input (keyboard, touch, etc.)

A phone is likely to perform one or more of these tasks:

- Make a call
- Send/receive text messages
- Send/receive email
- Keep track of your appointments

Object Oriented Programming

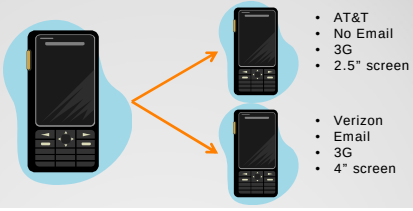
To Java, the phone's characteristics would be represented by **state** (or properties), while the actions the phone performs are known as **behavior** (or methods).

So if you were to set out to program a phone using Java, you would create a generic "object" that has the characteristics and performs the actions we described in the previous slide.

Then, you could create new instances of that object, and modify each property to be different.

Properties:
- Carrier
- Supports Email
- Service Type
- Screen Size

Methods:
- Make Call



- AT&T
- No Email
- 3G
- 2.5" screen

- Verizon
- Email
- 3G
- 4" screen

Object Oriented Programming

Objects make your programming very modular, allowing you to concentrate on a particular task on its own. Because the objects you create may be re-usable, making changes to your original object will automatically let other instances of that object to take advantage of your changes.

In Java, an object is a class. A class can consist of one or more sub-classes. A sub-class occurs when one class extends another, thereby providing its methods and properties to the new class. This is known as inheritance, and you will frequently see this within your Android applications. A class always starts with a capital letter. This is one way to easily identify classes while working with Java.

For example, in our Hello World project, you'll see a class called "hello_world" which extended from a class called "Activity"

```
public class HelloWorld extends Activity {
```

Packages

Java will keep your class files in a package. It's a way for Java to organize and keep track of your application. For the most part, your application will have one package and several class files, but it can contain hundreds of packages, and thousands of class files (although, again, very unlikely on a phone).

Java provides a number of pre-compiled packages as part of its class library. In addition to that, Google provides a number of packages specifically related to Android.

You will typically add these to your project using the **import** statement found at the top of your class file. In many cases, Eclipse will recommend additional classes to add for you.

```
import java.text.DecimalFormat;
import android.app.Activity;
import android.os.Bundle;
```

Anatomy Of A Class

```
package tushinsky.alex.HelloWorld;

import android.app.Activity;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

- Class name and java filename must be the same. (ex: class HelloWorld is in a file called HelloWorld.java)
- First line defines the package the class is in.
- Import statement(s) allow you to leverage other packages. For example, importing android.app.Activity, allows you to use the Activity methods within your class.
- After the import statements, follows your class declaration.
- Inside the class declaration, you'll create one or more methods that your class needs. This represents the functionality of your class.

  You can also define additional classes within an existing class.

Variables

Java stores information in variables. Each variable can contain one piece of information of a particular data type (such as string, integer, or decimal). Depending on where you create your variables within your class determines their scope. Variables can be defined as final (never change throughout execution), or as variables.

```
public class hello_world extends Activity {
  int iGlobalVar = 0;  //GLOBAL VARIABLE
  static final int iConst = 123;  //CONSTANT


  @Override                  //Parameter
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      string sText = "Hello!";  //LOCAL VARIABLE
      final Button btn1 = (Button) findViewById(R.id.btnOK);  //object var
   }

}
```

Variables

Variables are case-sensitive, and must start with a letter. Subsequent characters can be letters or numbers, or the underscore symbol. Variable names can not use Java keywords and statement names.

You can implement a naming convention for your variables. For example, I tend to use the Hungarian notation. Each variable is pre-fixed with a letter (or up to three letters) which tells me what type of variable it is. Examples:

- **s**Text (**S**tring)
- **b**Test (**b**oolean – true / false)
- **i**Value (**i**nteger)
- **l**Value (**l**ong)

Data Types

Java supports the following data types for variables:

- **byte**: -128 to 127.
- **short**: -32,768 to 32,767.
- **int**: -2,147,483,648 to 2,147,483,647.
- **long**: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Suffix with an L. Example: 1234567890123454**L**.
- **float**: a decimal number having up to 6 decimal places. You can tell Java that you're using a float data type when specifying a number by ending it in an F. Example: 1234.345**F**.
- **double**: a decimal with 15 decimal places.
- **boolean**: true or false.
- **char**: one character. Always surrounded by single quotes.
- **String**: one or more text characters. Although this is technically a class (java.lang.String), we tend to think of it more as a variable. Must be surrounded by double-quotes.

Java Upcast

Java tends to up convert variables in order to run mathematical operations on them. For example:

```
long LongNumber = 10;
float floatNumber = 10.1;
float result = LongNumber * floatNumber
```

In the above example, Java automatically converts LongNumber to float in order to multiply it by floatNumber.

The upcast happens in the following order, and only in one direction:

byte → short → int → long → float → double

What that means is that you won't be able to do something like this:

```
long Num1 = 10;
int Num2 = Num1;  //generates a compile error.
```

Declaring Variables

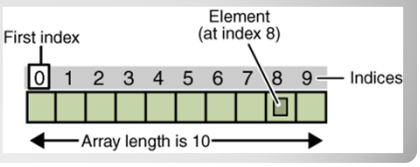
```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
String stext = "This is a string of text";

int iNumber;
iNumber = 100;
```

Note the single quotes around the char variable, and the double-quotes around a text variable. Numeric variables do not need single or double-quotes. Variables can be initialized right away, or their values can be defined later within the code.

Arrays

An Array is a variable that has many values. Each value within this array is identified by an index. When you first create an array, you need to define the maximum number of elements the array is allowed to contain.

```
iArray = new int[10];    // allocates memory for 10 integers
         iArray[0] = 100; // initialize first element
         iArray[1] = 200; // initialize second element
         iArray[2] = 300; // etc.
         iArray[3] = 400;
         iArray[4] = 500;
         iArray[5] = 600;
         iArray[6] = 700;
         iArray[7] = 800;
         iArray[8] = 900;
         iArray[9] = 1000;
```



Using Variables

You create variables for a reason – to store some piece of information that you want to work with. Lets look at ways that we can use to assign values to variables, and perform some basic math operations on these variables.

You've already seen that we can declare a variable and give it a value at the same time: Ex: int iValue = 100;

Once the variable is created, I can use it in my code. For example:

```
int iValue = 1;
int iOut = iValue + 100;
final TextView txtOut = (TextView) findViewById(R.id.txtOut);
txtOut.setText("100 added to iValue = " + iOut);
                          // 100 added to iValue = 101
```

Operators / Assignments

+	Add
-	Subtract
*	Multiple
/	Divide
%	Modulus
++	Increment
--	Decrement

=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a – b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

Increment / Decrement

There is a big difference between where you add the increment / decrement operator. Let's take a quick look at some code:

```
int a = 10;
txtOut.setText(a++);  // 10 is the output
txtOut.setText(a);    // 11 is the output
txtOut.setText(++a);  // 12 is the output
```

So the thing to be vigilant about here is that a++ will output your current value first, then add one to it. ++a, on the other hand will perform the addition first, then output the value.

Concatenation

Text variables can be added together to produce a result.

For example:

```
String a = "Hello";
String b = "World";
String c = a + " " + b;
```

Would produce "Hello World" in the message box.

Casting and Converting

At times it is necessary to convert one type of variable type to another. For example, maybe you start out with a float variable, but in the end would like to treat it like an integer. You may also have text content that you would like to see as a real number.

To cast a variable into a new format, precede it with the new data type wrapped in parenthesis. For example, if you wanted to convert a String to an int, you would use:

```
String sNum = "1010";
int iNum = (int) sNum;
```

To convert an int to a double, you would use:

```
int iNum = 1010;
double dNum = (double) iNum;
```

Certain conversions may not be possible – for example "Text" to int, or boolean to any other data type would fail with an error. Some conversions, although possible, should never be done – for example, converting 1000 to a byte (-127 to 128). White it doesn't produce an error, the value becomes meaningless.

Casting and Converting

Similarly to variables, you can create instances of objects using this method:

```
final TextView tOut = (TextView) findViewById(R.id.txtOut);
tOut.setText("Hello World!!!");
```

Escape Sequences

As you already know, string information in Java is surrounded by double-quotes. So what do you do if you want to include a double-quote as part of your string? What if you wanted to insert a tab into your string? Simply enough, you would just escape the double-quote, or use the escape sequence for a tab within your string. Below is a list of special characters that can be used:

\'	Single quote
\"	Double quote
\\	Backslash
\t	Tab
\b	Backspace
\r	Carriage return
\f	Formfeed
\n	Newline

Examples:

"\"Java is great\" - \n\tAlex T."