Introduction to Android Development

Introduction

https://www.oracle.com/java/

1. Java is a popular programming language you will use to write programs to run on Android™ devices. Java gained massive popularity during the growth of the World Wide Web. It became popular because it let any Java program run on any computer in the world. This may sound like a trivial thing to do, but in fact it is not. Different operating systems on different computers often required different code, and programmers had to constantly modify their programs to get them to work. But with a special piece of software called the *Java Virtual Machine, or JVM*, all lower-level issues with the operating system are taken care of. Any computer with a JVM can run any Java program. For the Android operating system, the JVM is built into it. In this way, the JVM hides, or uses *abstraction,* to conceal low-level details of the operating system from the Java program and therefore, from the programmer, too.

Materials

- Computer with BlueJ and Android™ Studio
- Android™ tablet and USB cable, or a device emulator

Resources

- Lesson 1.1 Reference Card for Basic Constructs

Procedure

2. To begin learning Java, you will use an *Integrated Development Environment*, or IDE, called BlueJ. An IDE does the following work for you:
   - Provides an editor to edit the files in your project.
   - Organizes project files.
   - ***Compiles*** your code, converting it to a machine language the computer can process.
   - Runs your program.

Part I: Hello World

The first program a programmer writes is often one that simply says “hello” to the world. In this part of the activity, you will write a simple “hello world” Java program to help you get familiar with programming in Java and BlueJ.

1. Before you start writing a program, create a place to store the program files. Create a folder called *BlueJProjects*. Your teacher will tell you where to create this folder.
2. Start the BlueJ IDE. Your teacher will tell you how to do this.
3. When you first start BlueJ, you will not have a project or any files to look at. But you can set up a useful feature to number the lines of your source code. From the project window, select **Tools** > **Preferences**. Check the box **Display line numbers** and click **Okay**.

4. To begin your first project, select **Project** > **New Project**. The **New Project** dialog appears.

5. In the New Project dialog, navigate to your *BlueJProjects* folder. All BlueJ projects are organized by folders. Name this project and this folder "HelloWorld": in the **Folder Name** text box, enter **HelloWorld**.

Note: Notice there are no spaces in this project name and the first letters of each word are capitalized. This is called camelcase because the uppercase letters seem to form the humps of a camel. Using camelcase is a common programming practice and one you will use throughout this course.

6. To create your new project, click **Create**. An empty project is created in BlueJ.

7. To start writing code, click **New Class**. A *class* is a small collection of code that serves a common purpose (much more on this topic in the next activity). Name your class **HelloWorld**. It is common to name your first class in a program after the project name itself.

3. A window titled HelloWorld appears in BlueJ IDE.

8. To open the editor for the class, either double-click it or select the right-click menu item **Open Editor**.

4. An editor window opens with your *HelloWorld.java* file. It contains default code that BlueJ automatically provides. The first few lines, surrounded by /* and */ are *comments*, helpful information about your program.

5. The first "real" or executable line of code in *HelloWorld.java* is public class HelloWorld. It is followed by an **open curly brace ({).** This curly brace and the **closed curly brace (})** at the very end of the file will contain all of the code you will write. BlueJ created default code in your class, but you will not be using it for this project.

9. In the comments, replace (your name) with your name. Replace (a version number or a date) with today's date.

10. Delete all of the code after the first open curly brace up to, but not including, the closed curly brace. Your code should now look similar to this:

```
1:  /**
2:   * Write a description of class HelloWorld here.
3:   *
4:   * @author CKinnard
5:   * @version 4/20/16
6:   */
7:  public class HelloWorld
8:  {
9:
10: }
```

11. Write the first few lines of your program. Add the code on lines 2–5 bellow, between the open curly brace and closed curly brace of your HelloWorld class definition:

```
1:  {
2:      public static void main()
3:      {
4:
5:      }
6:  }
```

***Note**: The line numbers above will not match your line numbers and are used only to reference lines of code in this and other documents.*

6. The statement with the *main* keyword tells the JVM where in your code your program will begin to execute. Don't worry about the syntax public static void. For now, you just need to know it is required to launch your program. You will learn what these words mean in the coming weeks.

12. Next, enter the code on line 5 below, between the open and closed curly braces of main:

```
1:  public class HelloWorld
2:  {
3:      public static void main()
4:      {
5:          System.out.println("Hello World!");
6:      }
7:  }
```

This line of code takes whatever you write between the double quotes and prints it to BlueJ's console window. The line of code should be automatically indented to show that it is inside and part of main. If it does not automatically indent, use the tab key to indent it.

13. In the editor, click Compile. If there are no errors, you should see a message in the bottom window: "Class compiled – no syntax errors". If not, carefully compare what you entered to the code presented above.

14. Close the editor window. The window that remains is your project window and you will use it to run your program. Right-click on the HelloWorld window and notice some new menu items. Select the menu item **void main()**. Your program is sent to the JVM, and the JVM attempts to run it. If it is successful, a BlueJ Terminal Window appears with the text Hello World!.

15. Run your program again, and you now see two lines of Hello World! If you want to clear your output every time you run a program, select **Options** > **Clear screen at method call**. Re-run your program to confirm that the clear occurs.

Part II: Hello Bugs

With any programming language, errors or "bugs" can occur. In Part 2, you will intentionally create some bugs to get used to some of the ways that Java reports errors.

16. Remove the semicolon at the end of the System.out.println line and compile your code. What is the error that appears?

    It says: ';' expected

17. Remove the period between `out` and `println` and compile. What error appears? Fix this bug.

    It says: cannot find symbol - method...

18. Remove the last curly brace at the end of the file and compile. What error appears? Fix this bug.

    It says: reached end of file while parsing.

19. Remove the last double-quote after World and compile. What error appears when you attempt to compile your program? Fix this bug.

    It throws three errors: unclosed string literal, ';' expected and reached end of file while parsing.

20. Remove the first double quote mark (before Hello) and compile. What error appears?

    This throws around the same errors as the previous step, just starting in a different place.

> ***Note**: Notice that this error message is not as helpful as the others. The compiler tries to guess what you are trying to do in your code, and sometimes it guesses wrong. In these cases, you must carefully check the syntax of the code you are writing. Fix this bug.*

21. Delete the spaces before the four lines of code in your HelloWorld class. Compile and notice that there are no errors. Java ignores all spaces, tabs, and new lines, unless they separate instructions (as in `public class`) or are inside double quotes (as in `"Hello World!"`). In fact, your entire program could look like this:

```
1:  public class HelloWorld{public static void main(){System.out.println("Hello World");}}
```

7. Because this is difficult to read and understand, it is strongly discouraged.

22. To restore your program to a respectable and friendly state, select **Edit** > **Auto-layout**.

Part III: Hello Android

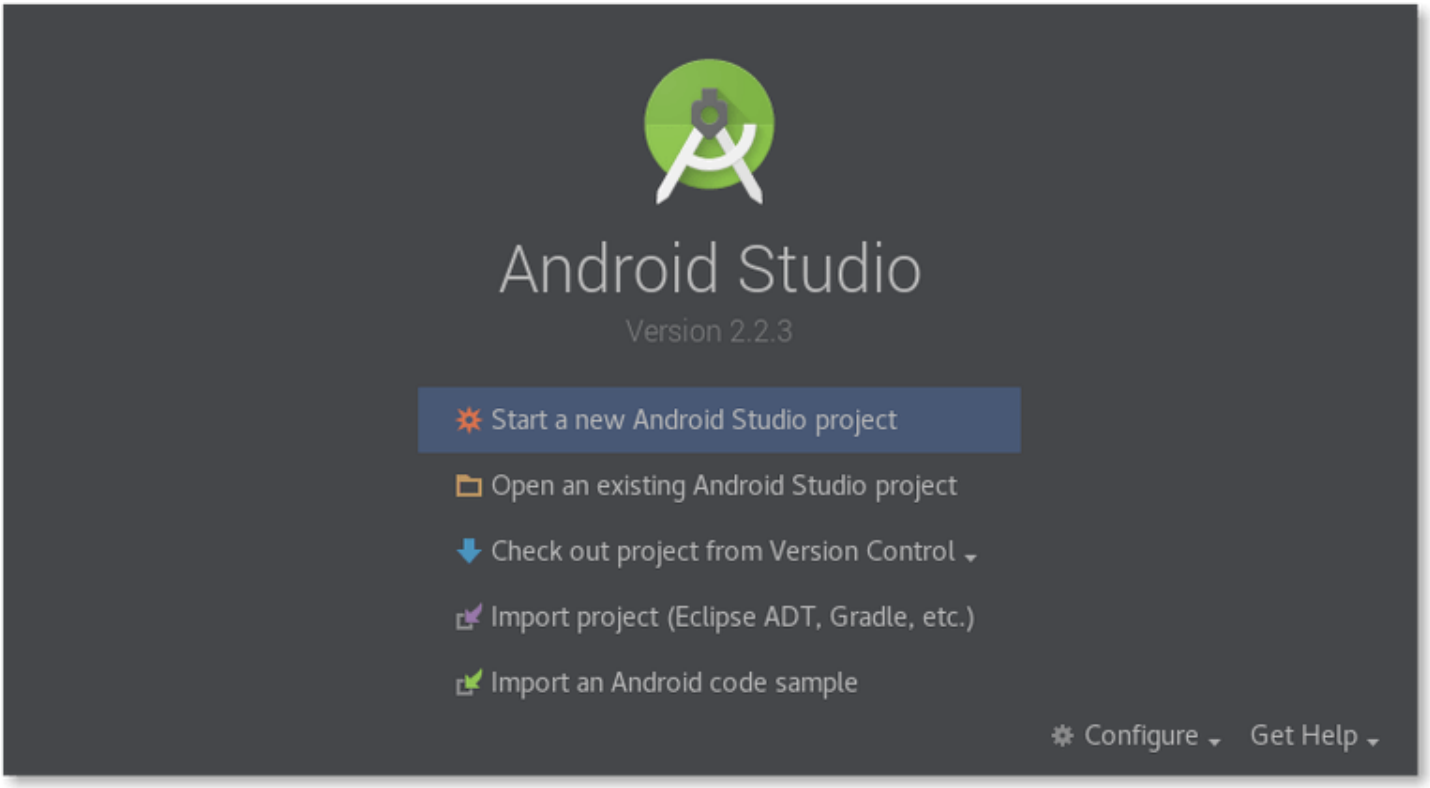
You will use Java to write programs called ***apps*** that can run on Android™ mobile devices, such as phones, tablets, and even wearables. To create these apps, you will use another IDE called Android Studio. You won't be writing code in Android Studio quite yet; you need to master a few Java fundamentals in BlueJ first. For now, in this activity you will explore Android Studio to see what a "real world" IDE looks like.

23. To begin, review the slideshow.

24. Similar to your BlueJProjects folder, create an AndroidProjects folder. Note there are no spaces in the folder name.

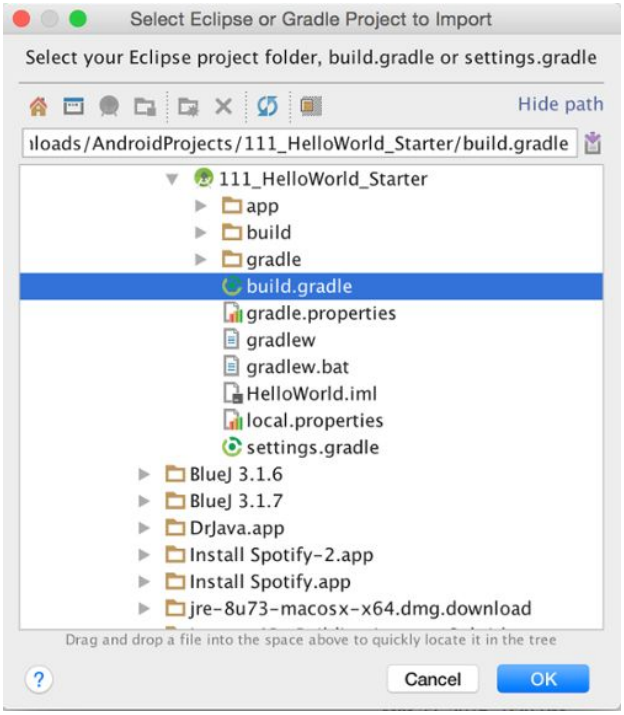
25. Get a copy of the 1.1.1HelloWorldApp Android project from your teacher. Copy or extract the files to a HelloWorldApp folder in your AndroidProjects folder.

26. Launch Android Studio. The first time you run Android Studio, it behaves differently than other times. You will see a welcome screen that asks you if you want to start a new project, open an existing project, etc.



27. Select Open an existing Android Studio project.

28. A dialog appears showing your file structure. Navigate to your *AndroidProjects* folder and then navigate to the location where you copied or extracted the HelloWorld project files. In the *HelloWorld* file structure, select the file named *build.gradle*. It will be in the *HelloWorld* folder, not in a subfolder.



29. Click **OK**.

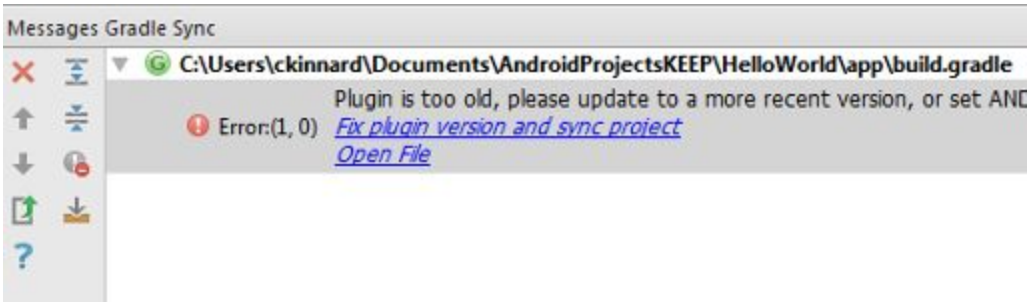
30. You will see the Android Studio IDE window. You may see a dialog that states

> *The path to '\Users\...' does not belong to a directory.*
> *Android Studio will use this Android SDK instead: '\Users...'*

8. The *ellipsis* (...) indicates unnecessary information is not included. Click **OK**.

31. You will also see a "Tips" dialog. The Tips dialog is an optional feature that helps you learn the IDE. Feel free to read through some tips or dismiss the dialog. You may also choose to disable it for future sessions.

9. Gradle is a feature built in to Android Studio. It manages and organizes your project files and other files that are necessary for app development. Depending on your version of Android Studio, you may see one or more errors or warnings from Gradle in the Message Gradle Sync panel at the bottom of the Android Studio window, as shown here.



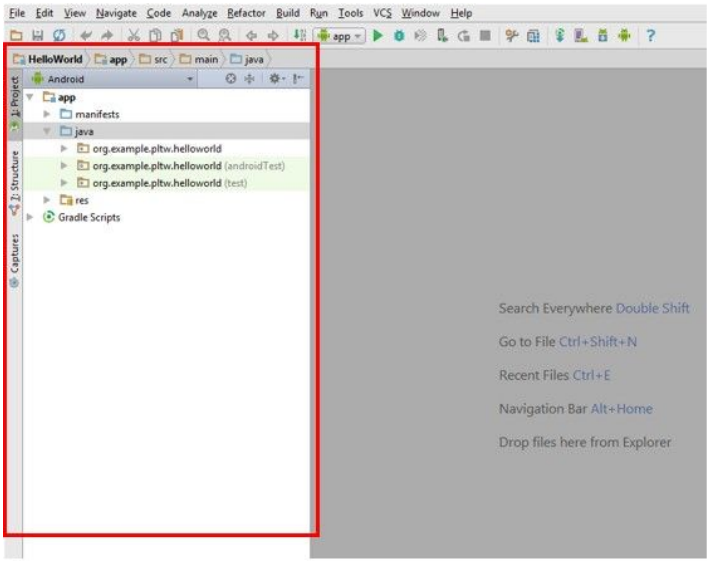
32. The specific errors and warnings along with their fixes are:

    a. "Plugin is too old..." Click "Fix plugin version and sync project" and follow the steps that are displayed.

b. “Failed to find target…" Click “Install missing platform(s) and sync project”. You will be asked to accept a license agreement, do so.

c. “Failed to find Build Tools….” Click “Install build Tools [*buildnum*] and sync project”. (Note the syntax [*buildnum*] means that a specific number will be displayed in place of [*buildnum*].)

33. Your HelloWorld project should be loaded and the Project panel should be showing:



10. If you do not see the Project panel (outlined in red above), **select View > Tool Windows > Project**.

Part IV: Observe Hello World

Running the Hello World app in Android Studio will help you become familiar with projects in Android Studio. After running the app and interacting with it, you will view the main activity file and the Java code it contains.

34. Start the emulator that your teacher has instructed you to use.

35. To run the app, select **Run** > **Run app** or click the green run arrow in the toolbar as shown.

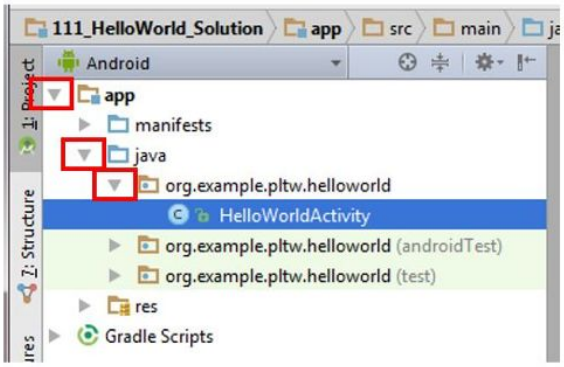
11. 

12. Note that it may take a few minutes for your app to load and run.

36. Enter your name. Click **Say Hello** and see the response.

Unlike your BlueJ program, an Android app does not have a `main` and therefore does not have a single entry point where the app begins. It does have a *default* entry point called a main *activity*. Android apps are based on activities, and you can view the main activity for HelloWorld called *`HelloWorldActiviy.java`*.

37. In the Android Studio Project panel, use the gray arrows to the left of *`app`* and the other folder names to expand the file structure and navigate to the project files.

13. 

14. Navigate to **org.example.pltw.helloworld**. Note: Do not navigate to the other structures **org.example.pltw.helloworld (androidTest)** or **org.example.pltw.helloworld (test)**.

38. To open the HelloWorldActivity.java file, double-click the filename. The code you see is much more complex than the code for your BlueJ project. For now, just note any similarity in overall structure.

Conclusion

1. What are some similarities you found between *`HelloWorld.java`* and *`MainActivty.java`*?

   They both call on functions with "void" preceded by "public" or something else, they use similar structure, they call on comments like / * * * **/, they both connect statements in functions with "."s and they utilize parentheses to call on inputs or changing functions. They also both start by defining "public class".**