TINKER ACADEMY

Programming Using Java

Handout 2: Java Fundamentals

Note your Student ID. You will need to use it throughout the Course.

Connecting to the Network

1. Select “Cupertino Community Center” if your Student ID is divisible by 2. Else choose “Cupertino Community Center 3”. Alternatively choose “AndroidAP” (password “tinker2014”).
2. Open a browser (preferably Chrome, Safari or Firefox)
3. Type in “www.google.com” (without the quotes). Type in *exactly* as indicated. If nothing shows up, check again, did you include the 2 “dots”? Did you spell www google and com correctly.
4. You should be taken to a Cupertino Parks and Recreation. Accept the agreement and click the required buttons to activate the network.

Launching the Virtual Machine in Full Screen

1. Use the VirtualBox menu View->Switch to Fullscreen to switch the VM to fullscreen mode
2. Use the same VirtualBox menu View->Switch to Fullscreen to switch the VM back out of fullscreen mode

Shutting Down the Virtual Machine

1. Click on the red close window button (to the top left on a Mac, top right in Windows).
2. You will prompted with a confirmation message asking if you want to “Power Off” the machine. Click the button to confirm power off.
3. In a few minutes the VM will shut down and you should see the VirtualBox side panel with the “Tinker academy” VM indicating “Powered Off”

Restarting the Virtual Machine

1. Start VirtualBox
2. Click on the VM “TinkerAcademy” in the VirtualBox side panel
3. Click on the Start Button (Green Arrow) in the VirtualBox Toolbar. This will start the VM.
4. Once the VM startup you will be presented with a login screen.

Right Click in VM on Mac

1. Open System Preferences, Trackpad

2. Enable “Secondary Click”, Toggle the small arrow to the right and select “Click with two fingers”.

Updating the Course

1. Make sure you are on WiFi.
2. Click the “Setup” folder in “Nautilus” under “Bookmarks”
3. Double click “Course Update”. Choose “Run”. Notify an Instructor if you see a window popup with the message “update course failed”. *You messed up. No, just kidding :). We’ll fix it for you.*
   If you are doing this after class hours:
   Hop onto Skype, and request help in the class chat group.
2. Click the “Courses” folder under “Bookmarks”. Navigate to the TA-JAV-1 and locate the “Quiz0.odt” under “quiz0” (which is under “quiz”). Select the file.
3. Double click Quiz0.odt to open it in LibreOffice 3
4. Answer the 5 questions in the Quiz. Once you are done, navigate to the top to see the menu and select File->Save. Alternatively use Ctrl S

Submitting Homework

1. Make sure you are on WiFi.
2. Click the “Setup” folder in “Nautilus” under “Bookmarks”
3. Double click “Course Submit”. Choose “Run”. Notify an Instructor if you see a window popup with the message “submit course failed”.

Getting Ready to Program

Import StarterPack2.zip
We will use StarterPack2.zip to write a complete Java program

Click on StarterPack2.zip under “Courses”. Right Click. Select “Extract Here”
Start Eclipse from Desktop. Right Click over Package explorer. Select “Import”. Browse and select the extracted folder “StarterPack”. Click “Finish” to complete the Import.

Structure of StaterPack2.zip
StarterPack2 is an Eclipse Java Project. The Project has

1. A Java Source File called “HelloWorld.java”
2. A Java Source File called “Example1.java”
3. Project references to the Java Runtime Environment 1.6 System Library.

Run the Program
Right Click over project name “StarterPack2”. Select “Run As…”. Select “Java Application”.
The text “Hello World” should be displayed in the Console window (bottom)

If the “Console” window is not visible click on the “Window” Toolbar (top), select “Show View”, select “Console”. If all else fails, click on the “Window” Toolbar (top), select “Reset Perspective…” click “OK”. This will put Eclipse back into the default Factory state. You can now use “Show View” to view the “Console”.

Structure of a Java Program

A Java Program is made up of a collection of Java Source Files. “StarterPack2” has 1 Java source file called “HelloWorld.java”. The Java source file is called a compilation unit.

When you ran the program using “Run As...Java Application”, Eclipse

1. Compiled the Java source file and created a new Java Class File called “HelloWorld.class” using a special program called the “Java Compiler”. This process is called compilation. The Java Class File is called the compiled file. It contains the byte code that the Java Virtual Machine needs to run the Program.
2. Started the Java Virtual Machine (JVM) and provided the JVM with the byte code in the Java Class File. The JVM ran the program printing the text “Hello World” to the console.

The Java Compiler requires that the all source files use the “.java” extension.

Every Java Source File is a text file that contains one or more class definitions.

The “HelloWorld.java” source file contains the class definition for a class called HelloWorld. That is not a coincidence. The name of the class should match the name of the source file (without the “.java” extension) exactly. The class name (and Java Source File name) should not start with a number or contain any spaces.

Capitalization matters in Java.

HelloWorld is not the same as helloWorld or HeLLoWoRLD

Here is the content of the HelloWorld.java source file

Line #	HelloWorld.java
1	/**
2	* HelloWorld.java
3	*
4	* Your First Java Program
5	*/
6	
7	/**
8	* The HelloWorld class will be used to understand the structure of a Java program.

```
 9  *
10  * @author student
11  *
12  */
13  public class HelloWorld {
14
15          /**
16           * This is the first method that will be executed by the Java Virtual Machine.
17           *
18           * It accepts some input and then prints the text "Hello World" to the console.
19           *
20           * @param args
21           */
22          public static void main(String[] args) {
23                  System.out.println("Hello World");
24          }
25
26  }
27
28
29
```

Lines 1 - 5 represent a File level comment. A File level comment describes the contents of the File. A comment can span multiple lines and must start with

/**

and end with

**/

Comments are ignored by the Java Compiler and are used by programmers to document the program.

Line 6 is a blank line that will be ignored by the compiler.

Lines 7 - 12 represent a Class level comment. A Class level comment describes the purpose of the Class.

Lines 13 - 26 is a class definition.
Line 13 uses the keyword class to indicate that a class called HelloWorld is being defined. The class definition begins with the opening { and ends with a closing } on Line 26.

Line 13 defines the signature of the class

```
public class HelloWorld
```

The line uses the keyword “class” to indicate that a new class is being defined.

The name of the class is “HelloWorld”. By convention, class names have the first letter capitalized. Since the name of the class name is the same as that of the Java Source File, java source filenames also need to have their first letter capitalized.

The keyword “public” is very important. You can see that the keyword is used in the class signature and in the signature of the main method.

The keyword “public” is called an access modifier. The access modifier indicates who can access this class.

Line 26 The class definition ends with the closing }.

Lines 14 - 25 contain all the content for the class.

Let’s take a look at the content in Lines 14 - 25.

The class HelloWorld defines one method called “main”. All method definitions in Java are contained within a class definition.

Line 14 is a blank line that will be ignored by the compiler.

Lines 15 - 21 represent a method level comment. A method level comment describes the purpose of the method, its input and expected output if any. It is used here to document the method “main”.

Lines 22 - 24 contain the code for the “main” method. The method definition begins with the opening { and ends with the closing }

Let’s go over the method “main” in greater detail.

The name of the method is “main”. This is not a coincidence. When the JVM starts the HelloWorld program, it will search for a method named “main” in the the byte code. If it successfully finds the method, it will begin running the program by “executing” the method. If it does not find the method, it will fail and Eclipse will display the message

“Selection does not contain a main type”

If you see such an error, it would mean that you asked Eclipse to run a Java Program without defining the main method correctly in any of the Source Files.

It is not enough that the name of the method is “main”. The method should also have the correct signature.

Line 22 defines the signature of the method

```
public static void main(String[] args)
```

If you define the “main” method in your Java Source File, you will need to ensure that it has **exactly** the same signature as the code above. Capitalization matters in Java.

The keyword “public” is very important. You can see that the keyword is used in the class signature and in the signature of the main method.

The keyword “public” is called an access modifier. The access modifier indicates who can access this class.

The keyword “public” in the method signature indicates that all other parts of the program can access the method.

The other access keywords are “private” “protected” and an implicit access called “package” access.

As you write more complex Java programs, you will see that allowing “public” access to a class or to all its methods is not really a good idea. We will cover access modifiers in greater detail in a later lesson.

By now you know the following

1. The name of the method is “main”
2. The method has “public” access which means that all other parts of the program can access this method

The remaining parts of the signature are the keywords “static” and “void” and the special pattern (String[] args).

The keyword “static” in a method signature means that the method can be called by the JVM before an object of the class has been created.

Let’s see how static methods can be invoked.

The term “invoke” a method is a special term use in Java (and other object oriented languages). It means to request the JVM to execute the method.

Double click to open the file Example1. As you can see, it has a public class called “Example1” and a static method called “staticMethod1”.

Add the following code after Line 23. Place the cursor at the end of Line 23 and hit Enter.

```
Example1.staticMethod1();
```

(Remember to include the semi colon ;)

The code invokes the static method “staticMethod1” in the class Example1.

Run the Program
Right Click over project name “StarterPack2”. Select “Run As…”. Select “Java Application”.

The Console window (bottom) should display the following

```
Hello World
Example1.staticMethod1 invoked successfully.
```

The keyword “void” in the method signature indicates that the method does not return any value to the invoker. The invoker is the part of the program that invoked the method.

The last part of the method signature is something that looks like this

(String[] args)

The content between (and) is the input to the method. The main method takes 1 input. The name of the input is “args”. Aargh!. . Not to worry, No one is in pain! The name “args” comes from the name used in the corresponding main method in the C language.

String[] indicates that “args” represents a Java Type. We will cover String[] and other Java types in the next class. For now, all you need to understand is that String[] is a Java Type.

Line 23 is a method invocation

```
System.out.println("Hello World");
```

The term “invoke” a method is a special term use in Java (and other object oriented languages). It means to request the JVM to execute the method.

Line 23 invokes the println method.

println is a very useful method that you used in your homework.
println is used to print something to the console. Here we are printing the text “Hello World”.

Here we are using the class “System” from the Java’s Software Development Kit (SDK). When you installed Java, it also installed the SDK for you. These classes are predefined for you and contain a wide range of very useful classes and methods that you can use in your program.

There are more than 4,000 classes in the JDK!. But don’t worry, we will be covering the important classes and methods you need to know in order to write useful Java programs.

Handling Syntax Errors

As you know by now from homework1, it is quite easy to miss a semicolon or accidently type something incorrectly when entering code into Eclipse.

Such errors are called syntax errors.

Fortunately, Eclipse will detect syntax errors and display syntax errors messages.

Eclipse uses the **Problems View** to display any errors it encounters. The errors might seem cryptic at first, but as you gain experience programming in Java, they will be a no-brainer :)

Click on the Problems View tab. The tab should be visible in the same tab bar as that for the Console view.

If the views are not showing up, navigate to the Eclipse menu bar, select “Windows”, “Reset Perspective”, confirm reset to reset the perspective to the default. Next navigate to the Eclipse menu bar, select “Windows”, “Show View” and select “Problems” to get the “Problems” view to show up and “Console” to get the “Console” view to show up.

You might see some warnings in the “Problems” view. You can ignore them for now.

Creating a new Java Class

Right Click over project name “StarterPack2”.

Select “New”, “Class”.
This will open up a window to create a new Java Class.
Type in the Java class name MyFirstJavaClass in **Name:**

Remember the naming convention

Every Java Class name starts with a capital letter and also follows the CamelCase naming convention.
So the class name file would be named MyFirstJavaClass and not myfirstjavaclass or my_first_java_class or myfirstJAVACLASS.

Click the **Generate comments** checkbox.
Click the Finish button. Eclipse will create a new Java Source File called MyFirstJavaClass.java under src, (default package)

Double click to open the file if it not already open.

Eclipse has already created the structure of the class. It has also added placeholders for adding comments.

Add the following method definition to the class.

```
public static void staticMethod1() {
        System.out.println("MyFirstJavaClass.staticMethod1 invoked successfully.");
}
```

The method definition creates a new static method called “staticMethod1” for the class MyFirstJavaClass.

Now add the code in HelloWorld.java to invoke staticMethod1

```
MyFirstJavaClass.staticMethod1();
```

(Remember to include the semi colon ;)

The code invokes the static method “staticMethod1” in the class MyFirstJavaClass.

Run the Program
Right Click over project name “StarterPack2”. Select “Run As…”. Select “Java Application”.

The Console window (bottom) should display the following

```
Hello World
Example1.staticMethod1 invoked successfully.
MyFirstJavaClass.staticMethod1 invoked successfully.
```

Refactoring Code

Eclipse can be used to refactor code. Refactoring code means to make change to the code, either in the names or the structure.

Refactor the name of the class MyFirstJavaClass to MyJavaClass1

1. Right click over MyFirstJavaClass.java in PackageExplorer
2. Select “Refactor”, “Rename…”
3. The Rename Compilation Unit will pop up. Type in MyJavaClass1 under **New name:**
4. Click Finish
5. Eclipse with refactor your code to handle the rename
6. Open HelloWorld.java. Notice that the main method now invokes MyJavaClass1.staticMethod1().

Refactor the name of the method staticMethod1 to myStaticMethod.

1. Toggle open MyJavaClass1 until it displays staticMethod1. Select staticMethod1. Right click over staticMehod1.
2. Select “Refactor”, “Rename…”
3. The Rename Compilation Unit will pop up. Type in myStaticMethod. under **New name:**
4. Click OK
5. Eclipse will refactor your code to handle the rename.
6. Open HelloWorld.java. Notice that the main method now invokes MyJavaClass1.myStaticMethod().
7. Open MyJavaClass1. Change the output text from “MyFirstJavaClass.staticMethod1 invoked successfully” to “MyJavaClass1.myStaticMethod invoked successfully”.

Run the Program

Right Click over project name “StarterPack2”. Select “Run As…”. Select “Java Application”.

The Console window (bottom) should display the following

```
Hello World
Example1.staticMethod1 invoked successfully.
MyJavaClass1.myStaticMethod invoked successfully.
```

Viewing Line Numbers

Eclipse can display the line number of each line of code in your program.

1. Open HelloWorld.java
2. Right click the left pane and select “Show Line Numbers”.

Debug Code

Eclipse can be used to debug code. Debug code will let you examine your program as it is running to spot logic errors in your program. This will prove very useful as you write more complex Java programs.

Breakpoint
To debug code in Eclipse, you would need to create a Breakpoint. A Breakpoint is a line number in a Source Java File before which Eclipse should stop executing the program and just wait until it is allowed to proceed.

We are going to add a new Breakpoint within the “main” method in HelloWorld.java.
Create a new Breakpoint as follows

DoubleClick just before Line 23 in HelloWorld.java. You might have to click slightly lower than the intended location. Note that this line number has code that is part of the “main” method. A green marker should appear just before the line if your breakpoint has been created successfully.

DoubleClick again to dismiss the marker. The marker represents a Breakpoint and it indicates that the program will stop executing when it is being debugged.

Debug the Program
Right Click over project name “StarterPack2”. Select “Debug As…”. Select “Java Application”.

The code should start execution as usual.

You will be presented with a “Confirm Perspective Switch” window. Click Yes to confirm to go to the Debug Perspective. This will open up the Debug window with the program paused just before Line 23 in HelloWorld.java. The line will be marked with a green highlighter indicating that the statement at Line 23 is the next statement that will be executed.

You should see a green (Resume) arrow above the code.

Click the green arrow to resume execution of the program.

The program should now proceed normally and print the expected text to the console.

Switch back to the Java perspective.
Use the menu bar. Select “Window”, “Open Perspective””, “Java” to go back to the familiar Java Perspective.

That was a LOT we covered!

You made it this far! Awesome!

We will cover Java Classes, Types Methods and Objects in later class. For now you need to make sure you have a good conceptual understand of all parts of the “HelloWorld.java” program.

Quiz 2: Java Fundamentals

Open the Quiz

Make sure you are on WiFi.
Follow the instructions in “Updating the Course” in this Handout.
Open Quiz2.odt under “Courses” “TA-JAV-1” “quiz” “quiz2”

Complete the Quiz

1. Attempt each question. Type in the answers in the “Answer:” box.
2. Save the file using File->Save or Ctrl-S

Submit the Quiz

Make sure you are on WiFi.
Follow the instructions in “Submitting Homework” in this Handout.

Homework 2: Java Fundamentals

Overview

Open the Homework

Make sure you are on WiFi.
Follow the instructions in “Updating the Course” in this Handout.
Open Homework2.sb2 under “Courses” “TA-JAV-1” “homework” “homework2”

Complete the Homework

You will need to refer to this handout (Handout2).

Homework Part 1
Open the Java Source File “MyExample.java”.

Add a static method called printClassName.

The method should use the following signature

public static void printClassName()

Remember to use { and } to begin and end the method definition.

The method should print the text “MyExample” on the Console.

Homework Part 2
Invoke the static method “printClassName” in the class from the static main method in the class “HelloWorld”.

If your code is correct, the program will print the following to the console

HelloWorld
MyExample

Make sure you save your program.

Test your program. If your program does not run successfully you will not get any credit.

Submit the Homework

Make sure you are on WiFi.
Follow the instructions in “Submitting Homework” in this Handout.