

Java Basics

Session 1

Intended Audience

As a basic course, this course is intended for

- Those with no programming background at all
- Those with limited background in languages similar to Java
 - C, C++, C#, JavaScript
- Those with a moderate background in other procedural or Object-Oriented languages
 - Python, Ruby

Where is this deck?

<https://github.com/AcademyGaffney/JavaBasicsJuly2020>

Intended Audience

More experienced programmers may find the course moves slower and covers less than you would like.

That isn't to say it won't be valuable if you are new to Java but otherwise experienced, just that you'll have to be patient with the pace.

Course objective

This course is intended to provide novice programmers with a functional understanding of the core components of the Java language as well as related underlying programming principles.

Think of this course as the first sprint in an agile project of learning Java. It's far from the complete project, but it contains the backlog items with the most value.

Overall Schedule

- Session 1: Java installation, key programming concepts, overview of Java, structure of a Java program
- Session 2: Eclipse IDE, Variables, data types, mathematical operations, Strings
- Session 3: Decisions and control flow
- Session 4: Arrays and methods

Overall Schedule

- Session 5: Classes, objects, and inheritance
- Session 6: Interfaces, the List and Comparable Interfaces

Session 1

Session Topics

- Installation of Java and Eclipse
- The anatomy of a simple Java program
- Key programming concepts as they relate to Java, with examples
 - This will be a big picture overview of the rest of the class with simple examples to provide a conceptual framework.
 - It is not intended to properly introduce you to the code you see and copy.

Java Installation

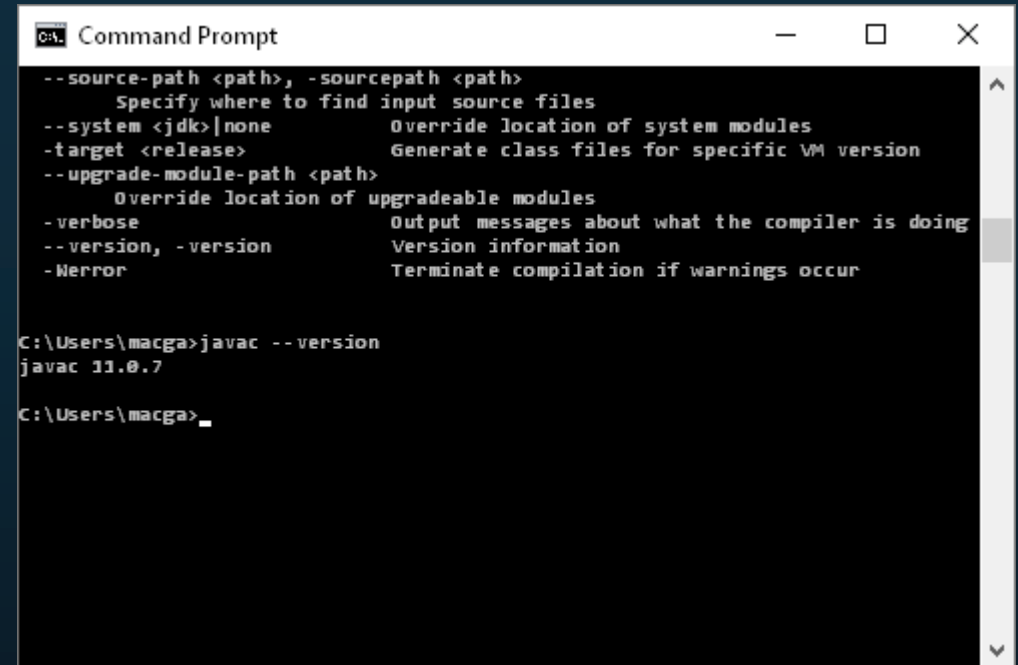
Check for installation

First, you want to make sure you need to install Java.

- Type “cmd” into the windows search bar to open a command prompt
- At the command prompt, type “javac --version”
- If it displays “javac 11.x.x” you’re set.

Check for installation

- Next, search for “eclipse” on the windows search bar. If you have any version, you’re set.
- You’re welcome to work in a different IDE if you’re familiar with one.



```
Command Prompt

--source-path <path>, -sourcepath <path>
    Specify where to find input source files
--system <jdk>|none      Override location of system modules
--target <release>      Generate class files for specific VM version
--upgrade-module-path <path>
    Override location of upgradeable modules
-verbose                Output messages about what the compiler is doing
--version, -version     Version information
-Xerror                 Terminate compilation if warnings occur

C:\Users\macga>javac --version
javac 11.0.7

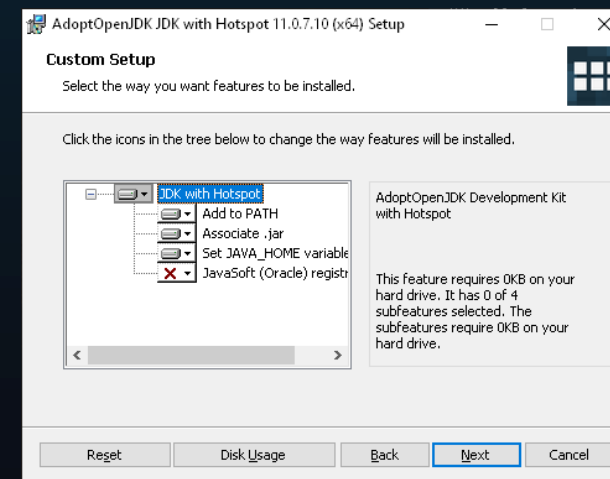
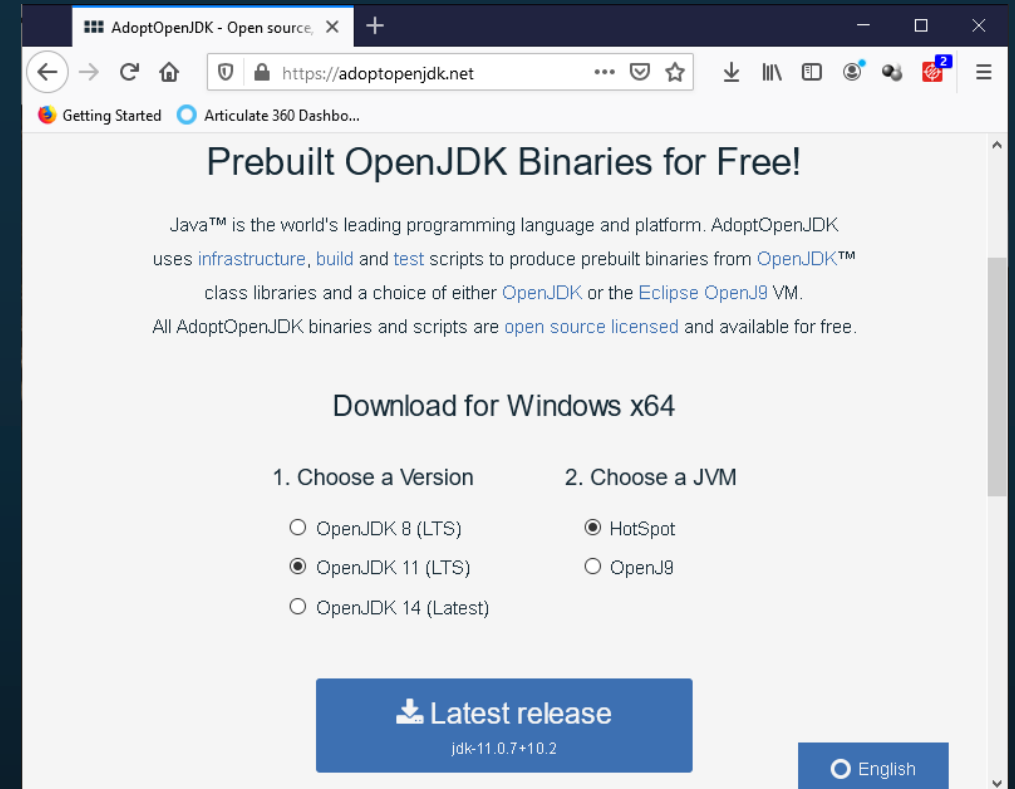
C:\Users\macga>_
```

Java Self-install (Admin rights)

Navigate to
adoptopenjdk.net

and install the latest OpenJDK 11
with HotSpot as shown.

Ensure that Add to PATH, Associate
.jar, and Set JAVA_HOME are
selected for install.



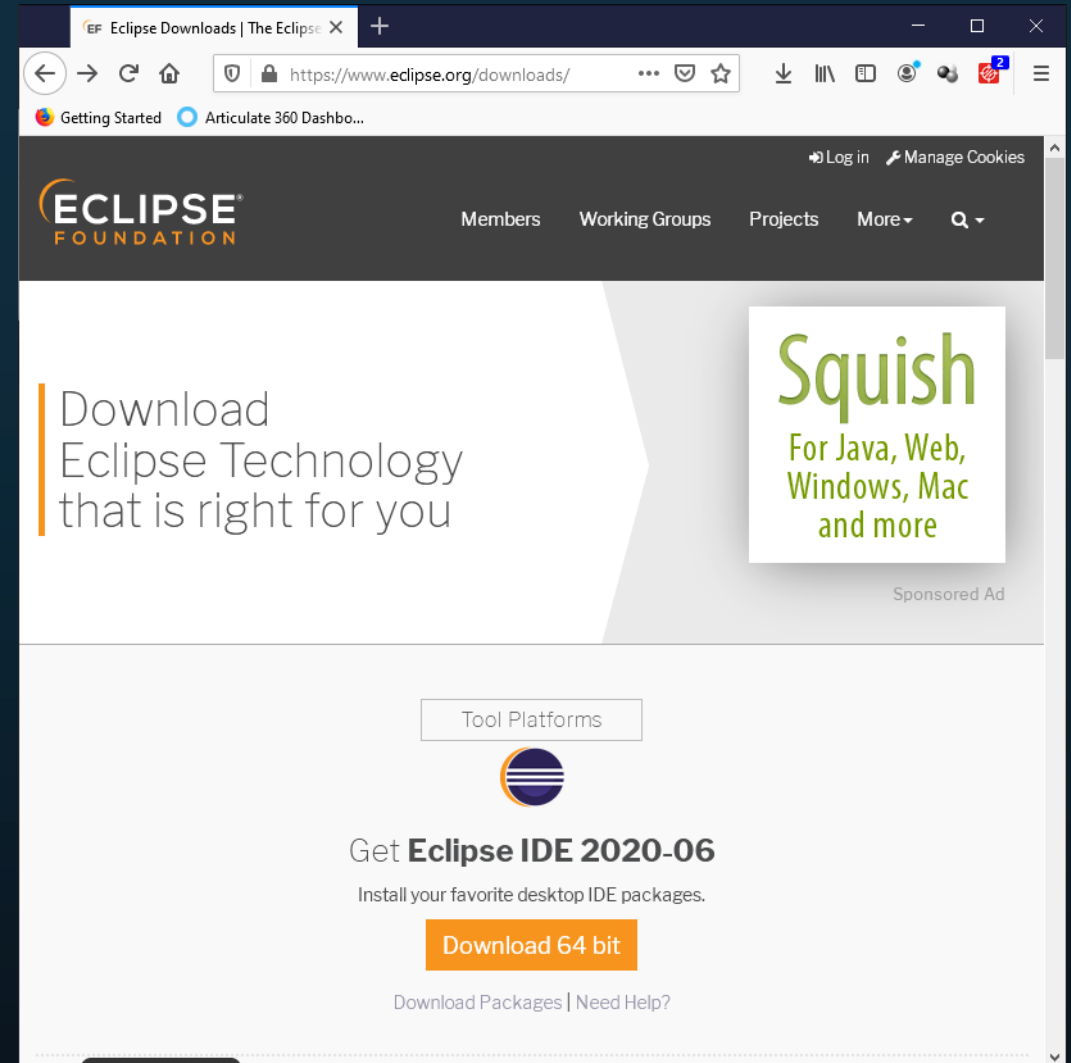
Eclipse Self-install

Navigate to

www.eclipse.org/downloads/

and download the latest version of Eclipse IDE.

Choose “Eclipse IDE for Java Developers”



OneIT install

- Search OneIT for OpenJDK
- Request the latest version 11 available.

Eclipse is nowhere close to up to date on OneIT. I'd recommend downloading the Eclipse installer and asking IT to install it once OpenJDK is installed. You *might* be able to install it yourself, but I've no way to test that on my machine.

Until Then

Load up the online Java compiler at

https://www.tutorialspoint.com/compile_java_online.php

A Simple Java Program

Hello World

```
package com.example.simpleprograms;
```

```
public class HelloWorld {
```

```
    public static void main(String [] args) {  
        System.out.println("Hello World!");
```

```
    }
```

```
}
```

Hello World

```
package com.example.simpleprograms;
```

```
/* Packages in Java are used to organize and categorize our files.
```

```
Using an organization's domain in its package structure prevents naming conflicts  
among files from different organizations. Packages exactly mirror the project's folder  
structure on disk.
```

```
This has been a multi-line Java comment.*/
```

```
public class HelloWorld {
```

```
    public static void main(String [] args) {  
        System.out.println("Hello World!");  
    }
```

```
}
```

Hello World

```
package com.example.simpleprograms;
```

```
public class HelloWorld {
```

```
// This line creates a class in Java. Everything that's part of the class goes in between
```

```
// the curly braces which, as you can see below, can nest. The indenting in the class
```

```
// is for readability only. Only the curly braces delimit blocks of code.
```

```
// These have been four single-line comments.
```

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

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

```
    }
```

```
}
```

Hello World

```
package com.example.simpleprograms;
```

```
public class HelloWorld {
```

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

```
// This is a method declaration. Most of the code you write needs to be in a method as well as
```

```
// a class. This particular method is special, in that it allows this class to be executed.
```

```
// Don't worry about "public" and "static" for now, although they have to be there.
```

```
// "void" is a keyword indicating that this method does not return a value. "main" is always void.
```

```
// "main" is the name of the method. You can use it for a different method that doesn't start a program,
```

```
// but the method that starts a program must look like this. You can also call this from a running
```

```
// program to start this one running.
```

```
// "String [] args" is the method's parameter list (one item, in this case). For main, this allows for command
```

```
// line parameters to be passed in when the program is run. It is an array of Strings named "args".
```

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

```
    }
```

```
}
```

Hello World

```
package com.example.simpleprograms;
```

```
public class HelloWorld {
```

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

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

```
// This is what all the trouble is for. Note that Java is not a language designed around writing  
// simple scripts. For instance "print("Hello World!")" is the entire Python program that does the  
// same thing.
```

```
// System is a class. out is a static object in that class (we'll look at static later). println is a method  
// in out's class. System.out.println sends whatever we send it to a console window on  
// whatever platform we're running on. "Hello World!" is the argument we're sending to println.  
// The semicolon terminates simple statements.
```

```
    }
```

```
}
```

Programming Concepts

Section topics

- Variables
- Statements and Expressions
- Data types
- Decisions and loops
- Functions and Methods
- Classes
- Collections
- Compiling and Interpreting

Variables

Variables

A variable is a name for a memory location.

Memory Address	Value
1000	
1004	
1008	
1012	
1016	
1020	
1024	

Variables

A variable is a name for a memory location.

`a = 27`

The computer has assigned address 1024 to our variable “a”. Assigning 27 to “a” stores it in 1024.

Memory Address	Value
1000	
1004	
1008	
1012	
1016	
1020	
1024	27

Variables

A variable is a name for a memory location.

`a = 27`

`b = 500`

Likewise for “b” and 1020.

Memory Address	Value
1000	
1004	
1008	
1012	
1016	
1020	500
1024	27

Variables

A variable is a name for a memory location.

`a = 27`

`b = 500`

`c = a`

“c” is address 1016. Assigning “a” to it copies the value from 1024 into 1016

Memory Address	Value
1000	
1004	
1008	
1012	
1016	27
1020	500
1024	27

Variables

A variable is a name for a memory location.

`a = 27`

`b = 500`

`c = a`

`a = 40`

Assigning 40 to “a” now just puts it in 1024. “c” keeps 27, as that value was copied into 1016.

Memory Address	Value
1000	
1004	
1008	
1012	
1016	27
1020	500
1024	40

Let's Run It

In your online compiler, remove the line

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

and replace it with

```
int a = 27;
```

```
int b = 500;
```

```
int c = a;
```

```
System.out.println(a + ", " + b + ", " + c);
```

Let's Run It

Next, put the line

```
a = 40;
```

On a new line after

```
int c = a;
```

And run again.

Statements and Expressions

Statements

- A statement is any piece of code that can change the state of the program.
 - The values of program variables
 - The module the program is executing in
 - The current screen displayed to the user

Expressions

- An expression is any piece of code that evaluates to a single value
 - A mathematical expression: $2 \times (a + 7)$
 - A comparison: $a < 10$ (evaluates to either true or false)
 - A compound expression: $a < 10 \text{ and } (2 \times (a + 7)) > c$
 - A value itself: 6
 - A variable, which evaluates to whatever is in its memory address.

Statements and Expressions

A part of your code can be a statement, expression, both, or neither. We'll look at how those apply as we go forward through the course.

Data Types

Data types

Programming languages vary widely in how they handle data. We'll restrict ourselves to concepts relevant to Java.

Primitive Data Types

Primitive data is data that's stored directly in the memory address a variable represents, such as the integers in our variable example.

Java supports four categories of primitive data

- Integers
- Floating point numbers
- Characters
- Boolean values (True and False)

Integers

- Integers in Java are represented as 2s-complement values, if you feel like looking that up.
- The key gotcha for this type of number is that if you add 1 to the highest value for a given type, you get the lowest value of that type.
- For instance, an 8-bit integer holds values from -128 to 127. $127+1$ gives you -128 and $-128 - 1$ gives you 127.

Let's Run It

Delete the text inside of main and replace it with the following:

```
int a = Integer.MAX_VALUE;
```

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

```
a = a + 1;
```

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

Floating Point data

- In Java, as in most languages, floating point numbers and arithmetic follow the IEEE 754 standard (if you want to look that up).
- It's similar to scientific notation with standards for how the bits are divided up among the parts of the number.
- Like scientific notation, floating point numbers do not have exact precision and even within their precision range have rounding errors in math.
- For instance: 3.14, 4, and 7.14 can all be exactly represented. $3.14 + 4$ doesn't not result in exactly 7.14, however.

Let's Run It

Again, replace the code inside main with the following:

```
System.out.println(3.14 + 4);
```

Characters and Booleans

- Character data in Java are values from the Unicode character set
 - These are integer values and characters can be converted to integers and vice-versa
- Boolean values are the keywords “true” and “false”. They cannot be converted to other primitive types.

Compound data types

- Data can be combined together to make more complex types.
 - In Java, this can only be done as a Class
 - A collection of characters is part of a String.
 - A pair of integers could be made into a fraction or ratio.
 - A person could be represented with any number of types of data, depending on how the type needs to be modeled.

Collections

- A particular kind of compound type is the collection, which is intended simply to hold and access a number of items of some other type.
 - Java has a relatively simple Array type that we'll cover
 - It also has an extensive library of collections for various purposes. We'll look at the List from that library.

Decisions and Loops

Decisions

- Like most languages, Java has the ability to create if-then-else style statements.
 - These allow one to take action based on whether particular conditions are true or false. For example
 - If there's a chance of hail tonight, I'll pull my car into the garage, otherwise I'll leave it out.
 - If I get at least 85% on my final exam I'll pass the class. Otherwise I'll fail.

Let's Run It

Place the following in main:

```
int score = 90;  
if (score >= 85) {  
    System.out.println("I Passed!");  
}  
else {  
    System.out.println("I failed...");  
}
```

Then change the score so you fail. 😊

Decisions

- Decisions can chain and nest for more complex situations
 - Assigning a grade based on multiple ranges of points
 - If my relocation gets approved for July or August, I'll use all my PTO for that
 - If not, then if Covid-19 drops off enough by August, we'll take a vacation, if not, we won't.
 - If we do take a vacation, then if it's especially hot in Texas, we'll go to the mountains, if not, we'll go to a beach.
 - If we don't take a vacation, then if my workload is overwhelming, I'll still take a week off. If not, I'll just take a couple long weekends.

Loops

- Loops allow us to repeat actions a number of times or until a condition is met.
 - Accept data from the user until she indicates she has no more to provide
 - Output the result of performing a mathematical function on all even values between 1 and 50.
 - Like decisions, loops can be nested.
 - Create a multiplication table for all values from 1-100

Let's Run It

Replace the code in main with

```
int num = 1;
```

```
while (num <= 10) {
```

```
    System.out.println(num + " x 2 = " + num * 2);
```

```
    num = num + 1;
```

```
}
```

Functions and Methods

Functions and methods

- Functions, methods, procedures, subroutines, and other structures whose names I'm not thinking of all allows us to modularize the flow of our program.
- They allow us to place frequently used code in its own block that can only be interacted with by calling it and acting on its return value.

Functions

- Mathematically, functions are one-way transformations from some list of values to a single resulting value. For instance
 - The power function takes in a base and a power and evaluates to the result of raising the base to the power.
 - The square root function takes in a single non-negative value and evaluates to the non-negative value which squares to it.

Functions

- In most languages, the data a function accepts are called its parameters and its result is called its return value.
- The actual values that are sent to a function are called arguments.
- Unlike mathematical functions, most languages allow functions to have no parameters and/or no return value.

Functions

- Functions can be used for any code you like, and should be used for non-trivial code that would otherwise be repeated.
 - Displaying information on a page.
 - Validating user input.
 - Converting data from one form to another.
 - Opening a connection to a database or network resource.

Methods

- Methods are very much like functions, except that they are part of classes.
- As we'll see, everything you write in Java has to be in a class, so all of our functions will be methods.

Let's Run It

Put the following code *outside of main* but inside of HelloWorld. It can be above or below.

```
public static int doubler(int in) {  
    return in * 2;  
}
```

Then, in main, replace “num * 2” with
“doubler(num)”

Classes

Classes

- Classes are the core of Object-Oriented Programming (OOP)
 - The shift in OOP is the organization of a program around its data (in classes) rather than around its behavior (in functions).
 - Objects are instances of classes, just as 27 is an instance of an integer. Objects of the same class have the same structure to their data, but different values.

Classes

- A class consist of:
 - Variables for the data the class has to represent.
 - Methods for controlling access to that data.
 - Some of those methods make up the class's *public interface*.
 - Other methods, along with the Class's data, make up its *private implementation*

Classes

- For example, a class representing a Fraction
 - Its public interface would include mathematical operations and the ability to display the fraction in a human-readable form.
 - Its private implementation would include the numerator and denominator variables – which we would *not* want someone to be changing independently of each other.

Classes

- Defining classes and the interactions among them is the heart of OOP.
- Depending on the product under development, this design ranges from about as simple to about as complex as you can imagine.

Collections

Collections

- Some classes are intended not to model some component of a particular problem, but to serve as containers for use in whatever problem.
- Examples include arrays, lists, sets, queues, dictionaries, trees, maps, and many more.
- We'll look at arrays and lists, which are relatively simple but will give you insight into how to work with other ones as you learn about them

Arrays

- Arrays are the simplest type of collection. They are:
 - A contiguous block of memory holding some number of objects of the same type
 - Of a fixed sized once they're created
 - Only accessible via the index, or location of each element.

Lists

- Lists are similar to arrays, in that they are an ordered collection of elements of the same type.
 - Lists in Java (and typically) will automatically resize if needed.
 - Lists typically have methods for accessing the end of the list directly, checking for values in the list, merging lists together, and various other helpful functions that need to be written externally for an array.

Compiling and Interpreting

Compiling and Interpreting

The program you write is called source code. In order to run, it needs to be converted into machine language code for the given hardware and operating system.

At a high level, this is done via compiling or interpreting.

Compiling

- In a compiled language the following steps (may) occur:
 - The programmer writes source code
 - There may be a pre-processor which inserts library code into the programmer's code
 - The source code is compiled into machine language to be readable by the operating system
 - The source code may be linked with other files to form a monolithic executable program.

Interpreting

- In an interpreted language, the code is compiled line by line as it is run.
 - This is much less efficient for larger programs.
 - It is more efficient for simple scripts, especially if they change frequently.
 - Not compiling removes a step that can detect and report errors, so an interpreted program will generally only fail as it's running.

Java

- Java is a compiled language.
 - It compiles to a *Java Virtual Machine* (JVM) rather than natively to the underlying hardware and operating system.
 - This allows Java to run seamlessly across multiple platforms, as it's the JVM that's natively compiled.
 - It contains neither a preprocessing nor a linking step. All linking among files and to library files is done dynamically at run time.

Next Steps

Next Steps

- Get Java and Eclipse installed.
 - The outer bound for needing this to work successfully is session 5.
 - Prior to this you can continue to follow along with everything other than making Eclipse projects using the web-based compiler at https://www.tutorialspoint.com/compile_java_online.php
 - This can still be used after this, although your practice will be less realistic at that point.