Quick Reference Guide To Programming In Java

C H A P T E R 0 – I N T R O D U C T O N

* Java is a high level, modern programming language designed in the early 1990s by Sun Microsystems, and is currently owned by Oracle. High level programming means that it is closer to human language than machine language. An example of machine language would be machine code and assembly. And of course, high level languages include Java, Python, etc.
* Java is platform dependent. This means that you only need to write the program once to be able to run it on a number of different platforms. You can write a program once, on a Windows machine, and still be able to run it on a Macintosh, Linux, etc. Java guarantees that you will be able to "Write Once, & Run Anywhere". Meaning, to distribute your application, you only need to create only version – one JAR file. And as long as the other machine has JDK, and JRE, it will be able to run it. Also, JDK = Java Development Kit, and JRE = Java Runtime Environment. JRE allows you to run Java files (JARs), and JDK allows you to code in Java. You need to download and install both to run Java applications, and code in Java.
* More than 3 billion devices run Java. Java is used to develop apps for Android OS, Windows OS, and Web Applications. These apps range from simple calculators to complex applications like media players and antivirus programs. A lot of popular games are coded in Java. A great example of this would be Runescape, developed by Jagex Ltd.
* The basic development process of any Java program follows a specific cycle. The first step is to design the application. What is its purpose? What will it look like? What will it do? Once you have that figured out, it's time to write it. After writing it, you test it and squash bugs. Make sure you rigorously test your application before publicly distributing it. Bugs can take away from the experience and cause unwanted consequences. The source code is packaged with an extension ending in ".java". It will look a little something like this: "NAME_OF_APP.java". For example: "Calculator.java". Do not lose this file and only give it to trusted people, unless you are planning on making it open source.
* When you are done testing and squashing bugs, you are ready to distribute your application to the general public. Typically, most Java apps are distributed using a Java archive file, which has the ".jar" extension. You may have come across these at some point in your lifetime. If you didn't have JRE or JDK installed, the file is an archive and upon extraction you get some weird folders like "META-INF", "Resource", and "ORG". But if you do have JDK/JRE installed then you're fine.
* The Java Development Kit (JDK) is a software development environment for Java Applications and applets. It includes: The **Java C**ompiler (javac), The **J**ava **Ar**chiving Tool (JAR), The **J**ava **D**e**b**ugging Tool (JDB), and **J**ava **R**untime **E**nvironment. Head over to www.oracle.com to download JDK and JRE for free. While you're at it, install Eclipse and get it running!
* The **J**ava **V**irtual **M**achine (JVM) is an execution environment for Java applications. It interprets compiled Java binary code (called byte code) to enable a computer's processor to carry out a Java program's instructions. Java was designed to allow applications to be executed on any platform without having to be rewritten or recompiled. The Java Virtual Machine makes this possible. The Java compiler reads Java language source (.java) files, translates the source into Java byte code and places the byte code into class files. These class files end in ".class". The class files can then be executed on the JVM. In a nutshell, JVM translates byte code into machine code. Remember, machine code is something the computer can read.
* In other programming languages, the compiler will typically produce code for a particular system. In Java, the compiler produces code for a virtual machine. Every device that has a JVM installed is able to translate and run your ".class" files. These files are created when you run your ".java" files through a compiler (remember javac). The ".class" files contain abstract instructions for JVM. The JVM is a main component of Java architecture, and is part of the JRE. The JVM is operating system dependant. Meaning, the JVM must translate the byte code into machine language, and the machine language depends on which operating system is being used. The JVM is also responsible for allocating required memory.

C H A P T E R 1 – H E L L O W O R L D

* Just about every single Java book discusses the "Hello World" program. This guide won't! That program is so boring and lame. Instead, we will take a look at a derivative of it, something more fun and interesting. It looks like this:

```
package firstprogram;
        public class Firstprogram {
                public static void main (String[] args) {
                        System.out.println("Wuzzup niggas");
                }
        }
```

* Now before you get mad at such boring code, just bare with me for at least 10 minutes and understand this code. Please! Go ahead and paste this code into Eclipse, click run and see what happens. The best way to learn is through real practice!

1. package firstprogram;
* The name of the package is called firstprogram. A package is where all of the classes go. Essentially, the package is your entire application. All of the code goes inside the package and is distributed among the class or classes. Take notice at the semicolon at the end of the line. In Java, 99% of lines end in either a semicolon (;) or a curly bracket ({) OR (}).

2. public class FirstProgram {
* This statement creates the class "FirstProgram". Inside this class is where the bulk of your code goes. Now, that's not entirely true because you can create multiple classes, which you will learn later, but as a beginner, 99% of your code goes in the main class. "Public" just means that it is visible and can be accessed by other classes, so you can call on it. There's also "Private" which is the opposite of "Public", and JVM cannot invoke it. "Class" is where the code will be, and of course the name is "FirstProgram". Notice how the class name is nearly identical to the package name with a few minor differences. The only difference is that the name is in lowercase for 'package' and the words are uppercase for 'class'. Also, the two must be the same, so don't ever try changing them. And again, notice how the line ends in a curly brace.

3. public static void main (String [] args) {
* "Public": Means that the method can be called from anywhere. You can use private so JVM cannot call upon it
* "Static": Means that you can call it without any object instantiation. You will develop a better understanding later
* "Void": This is self explanatory and means that the function will not return anything. Void = Completely Empty
* "Main": Every Java program starts off with a main method. When the program is executed, JVM looks for a main method
* "String [] args": Declares a parameter named 'args', which is an array of instances of the class String (only input allowed)

4. System.out.println("Wuzzup Niggas");
* This is kind of self-explanatory. This line outputs whatever statement is in the quotes to the console. 'System' is a class in the java.lang package. 'Out' is a static data member in the system class. And finally 'println' is a method that prints the statement out to the console. Note: Instead of 'println' you can also use 'print', and the difference is that the first one adds a line while 'print' does not. So, if you were to use 'print' in a manner like this:

```
System.out.print("Without"); // Notice how the 'ln' is gone from 'print'
System.out.print("LN");
```

* The console output would be:

```
WithoutLN
```

* And if you were to use "println", then the output would be:

```
Without
LN
```

*** You can add lines with the next line operator [\n]. This can come after the first quote, or before the last quote. For example, if you were to do it after the first quote, and use 'print' and not 'println', it would look like this:**

```
System.out.print("Without");
System.out.print("\nLN");
```

* And the output of it will be:

```
Without
LN
```

* Go ahead and try it for yourself. The best way to learn Java is to practice it daily and learn it yourself. Build some shit-kicker programs and debug them, improve them, distribute them, etc. Learning through experience is the best technique!

CHAPTER 2 – THE BASICS

* A) COMMENTS: The purpose of including comments in your code is to explain what the code is doing. Java supports both single and multi-line comments. All characters that appear within a comment are ignored by the Java compiler. A single-line comment starts with two forward slashes and continues until it reaches the end of the line. For example:

```
// This is a single-line comment before the code
System.out.println("Javaaa"); // This is a single-line comment after the code
```

* Adding comments to your code is a good practice to get into because they provide clarification of the code. When you refer back to it, you can just read the comments and immediately understand what you were trying to accomplish. Comments can be also helpful for others to understand your code. As a beginner, comment every – damn – thing!

* Another type of comment is multi-line comments. These type can span multiple lines. You start this type with a forward slash followed by an asterisk, and end it with an asterisk followed by a forward slash. For example:

```
/* Sweet Baby Jesus, Oh LAWWDD!!!
This comment spans multiple lines!!!
How will we ever surviveeeeeeeee!!!
*/
```

*Note that Java does not support nested multi-line comments, but you can put single-line comments in multi-line comments. So for example, you can do the following:

```
/* This Is A
Nasty // Nested // Nightmare // Niagara
Comment
*/
```

* But you CANNOT do the following:

```
/*
You
/*
Cannot
*/
Do This Mate
*/
```

* B) VARIABLES: Variables are used to store data for processing. A variable is given a name, such as area, age, height, etc., and then they are assigned types. Some examples of these types include, but are not limited to:
* "int": Used for integers (whole numbers) such as 420. However, ints have a range from –2147483648 to +2147483648.
* "double": Used for decimal numbers such as 3.14149. Furthermore, doubles have a greater range than ints. There range is from –9223372036854775808 to +9223372036854775808.
* "String": Used to store sentences, and words, such as: "Dog", "Bob", and "#DrainTheSwamp & #Hillary4Prison".
* "Char": Stands for character and holds a single character such as: "D", "O", "N", "A", "L", "D", and "T".
* "Boolean": Only has two possible values, and they are true and false.

* You can declare variables like so:

```
int x = 12;
double y = 3.14;
String name = "David"
```

* Note: An “int” can only occupy integers and not decimals or strings. Also, you can convert from int to double and vice versa but it is possible you will lose information when going from int to double (the decimals or the some of the value).
* You can also declare variables like this:

```
int x; // Notice how this style takes two lines
x = 12;
```

* If you wanna declare multiple variables of the same type, do this:

```
int x, y, z = 10; // Try this for yourself! The best way to learn is practice!
```

* C) PRIMITIVE OPERATORS: Also known as math operators, these include addition (+), subtraction (–), multiplication (*), division (/), and modulus (%). Java provides a rich set of operators to use in manipulating variables. A value used on either side of an operator is called operand. For example, the number 6 and 3 are operands of the plus operator: int x = 6 + 3

* ADDITION: The (+) operator adds together two values, such as two constants, a constant and a variable, or a variable and a variable. Below are a few examples:

```
int sum1 = 60 + 9;           // Two constants being added together
int sum2 = sum1 + 282;       // A variable and a constant being added
int sum3 = sum1 + sum2;      // Two variables being added together
```

* Note: Always remember to add the semicolon at the end of your variables! Also, it is important to be able to read the code in plain English. This is what I mean by this:

```
--------------------
int sum1 = 60 + 9;
^ An integer named ‘sum1’ is equal to sixty plus nine
--------------------
int sum2 = sum1 + 282
^ An integer named ‘sum2’ is equal to an integer named ‘sum1’ plus 282
--------------------
int sum3 = sum1 + sum3
^ An integer named ‘sum3’ is equal to another integer than ‘sum1’ plus ‘sum2’
--------------------
```

* SUBTRACTION: The (–) operator subtracts one value from another. It works in the same manner as the (+) operator and all continues continue to apply. Below are some examples:

```
int sum1 = 79 – 10;
int sum2 = sum1 – 9;
int sum3 = sum1 – sum2;
```

* MULTIPLICATION: The (*) operator multiples two values together. For example:

```
int sum1 = 1000 * 2;         // ‘sum1’ = 1000 x 2 = 2,000
int sum2 = sum1 * 10;        // ‘sum2’ = 2000 x 10 = 20,000
int sum3 = sum1 * sum2;      // ‘sum3’ = 20000 x 2000 = 40,000,000
```

* DIVISION: The (/) operator multiples two values together. For example:

```
int sum1 = 1000 / 5;            // ‘sum1’ = 1000/5 = 200
int sum2 = sum1 / 2;            // ‘sum2’ = 200/2 = 100
int sum3 = sum2 / sum1;         // ‘sum3’ = 100/200 = 0 (See below on why)
```

* Note: Even though 100/200 is actually 0.5, recall how ‘int’ can only store integers and not decimals, so 0.5 gets rounded down to 0 and ‘sum3’ = 0. For an int, ALL decimals get rounded down!!! Even if it is 3.8 or 3.9, Java will always round down to 3, unless instructed otherwise. You need to tell Java how to round, or else it will round down. For example:

```
System.out.println(12/7); // Outputs 1 to console
System.out.println(99/10); // Outputs 9 to console
```

* MODULUS: The modulus (aka modulo) or remainder operator performs an integer division of one value by another, and returns the remainder of that division. The operator for the modulo operation is the percentage (%). Remember doing long division in elementary school? Yeah, it is something like that. If you were given 100 % 9, think about it this way: How many times can 9 fit in 100… 11 times with 1 left over. So 100 % 9 is 1. And it works because 99 x 11 = 99, and 99 + 1 =100.

```
int value = 23;
int res = value % 6;
System.out.println(res);
```

* So 6 can fit into 23 about 3 times. This is because 6 x 3 = 18, and 6 x 4 = 24. Therefore, 3 is a better fit, and the remainder is 5 because 23 – 18 = 5. This is how the mod operand works. It might seem completely useless at first but believe me when I say it is really useful for things like finding out if an ‘int’ is odd or even. If the ‘int’ % 2 returns 0, then it is even, else it is odd. There are many other uses of the mod operand that make it extremely useful to us programmers!

* D) INCREMENT & DECREMENT: An increment (++) or decrement (– –) operator provides a more convenient and compact way to increase or decrease a variable by one. For example, below is how you would normally do it:

```
int x = 5;
x = x + 1;
```

* The value of x is 6. However, x = x + 1 is quite a lengthy and can be shortened to x++ to save time and money. So you can rewrite the above line of code into (and still achieve the same result):

```
int x = 5;
x++;
```

* The (++) or (– –) can be written before and after the ‘x’. This is known as prefix (before) and postfix (after). In prefix, the variable is incremented first, and then assigned to the variable, and in postfix, the variable is assigned first, and then incremented. For example, this is a prefix:

```
int x = 34;
int y = ++x;
System.out.println(“x = ” + x + “, y = ” + y); // Y is equal to 35, X = 35
```

* So here’s what’s happening: An integer named ‘x’ is assigned a value of 34. Then, another integer named ‘y’ is assigned a value of “1 + 34”, giving it a total value of 35. Hence, ‘y’ is assigned a value of 35. Also, since 1 is added to ‘x’ it is now 35, akin to ‘y’. If you were to output ‘x’ to the console, it would be 35. The System.out.print… does this for you. Try it!!!

* Also, in case you didn’t already know but you cannot have two integers of the same name. Same rule apples with strings, doubles, chars, Booleans, etc. They can have the same value, but not the same name. So you cannot have two int x ’s.

* The following is an example of postfix in the increment/decrement operand:

```
int x = 34;
int y = x++;
// System.out.println("x = " + x + ", y = " + y); // Y is equal to 34, X = 35
```

* In plain English, here is what is happening with this code: An integer named 'x' is assigned a value of 34. Then, another integer named 'y' is assigned a value that is equal to 'x'. Hence, 'y' equals 'x', and since 'x' is 34, so is 'y'. And finally, 'x' gets one added to it, so it now becomes 34 + 1 = 35. I have summarised the difference between postfix and prefix below:

```
int y = ++x; // Prefix: Adds one to the variable 'x' BEFORE the variable 'y' is assigned to it
int y = x++; // Postfix: Adds one to the variable 'x' AFTER the variable 'y' is assigned to it
```

* Try and solve the console output for the following block of code:

```
int a = 69;
double c = 8.3;
System.out.println(a++); //What would the console print?
```

* E) ASSIGNMENT OPERATORS: You can extend the shorthand format of the increment and decrement, and apply it to the addition, subtraction, multiplication, division, and modulus operands, making it easier to write cleaner and simpler code. Addition short-hand looks like: (+=), Subtraction short-hand looks like: (–=), Multiplication short-hand looks like: (*=), Division short-hand looks like: (/=), and Modulus looks like: (%=). In practice, you would use these like so:

```
int m = 10;        // Declare an integer named 'm' and assign it a value of 10
int n = 5;         // Declare an integer named 'n' and assign it a value of 5
m += n;            // 'm' is equal to 'm' + 'n' // So m = m + n = 10 + 5 = 15 // Therefore, the new value of 'm' is 15
```

* When you paste this into Java, replace the last line in the above block of code, with anyone of these lines below. Try it!

```
m –= n;            // 'm' is equal to 'm' – 'n' // So m = m – n = 10 – 5 = 5 // The new value of 'm' is 5
m *= n;            // 'm' is equal to 'm' x 'n' // So m = m x n = 10 x 5 = 50 // The new value of 'm' is 50
m /= n;            // 'm' is equal to 'm' / 'n' // So m = m / n = 10 / 5 = 2 // The new value of 'm is 2
m % = n;           // 'm' is equal to 'm' % 'n' // So m = m % n = 10 % 5 = 0 // The new value of 'm' is 0
System.out.println("m = " + m + ",n = " + n); // This prints the value of 'm' and 'n' to the console
```

* F) STRINGS & CHARS: A String is an object that represents a sequences of characters. For example, "hello" is a string of 5 characters, and "jog" is a string of 3 characters. Before you proceed to analyze the example below, take note of the (+) operator between the strings. The addition/plus operator adds two strings together to make a new string. This process is called concatenation. The dictionary definition of concatenation is: linking things together in series. For example:

```
String first_name = "David";
String last_name = "Packouz";
System.out.println("My name is " + first_name + " " + last_name);
```

* If you wanted to store a single letter, it would be wise to use the char data type. Pay close attention to how you have to use single quotes for the char data type, and double quotes for the string data type. You cannot mix and match and use either or, or as you please. Do not confuses these, or else you will get an error! An example of the 'char' data type:

```
char Donald = 'D';
char John = 'J'
char Trump = 'T';
System.out.println("No one can beat " + Donald + John + Trump);
```

* G) GETTING USER INPUT: Applications are all about getting input from the user. After all, somebody else is going to be using your application so it's pretty stupid if they cannot provide it with meaningful data. The easiest way, but not most intuitive way, to get user input in Java is to important the Scanner class like so:

```
import java.util.Scanner;
```

* This line goes right after the package and before public class, like so:

```
package name;
import java.util.Scanner; // When you import any class, it comes in between this space!
public class Name {
	public static void main (String [] args) {
		//Logic Code Goes Here
	}
}
```

* In order to use the Scanner class, you need to create an instance of the scanner by using the following line of code:

```
Scanner NAME_OF_VARIABLE = new Scanner (System.in);
```

* You can now read different kinds of input that the user enters. For example:

nextInt() – Read an int
nextDouble() – Read a double
nextLine() – Read a complete line/sentence
next() – Read a word; a single word

* Tip: Think of nextLine() and next() as String and char, respectively. nextLine() and String contain more letters than next() and char. nextLine() and String are for heavy duty sentences, while next() and Char are for light uses. Furthermore, you can read other types of data such as float, short, long, Boolean, etc. To do this, simply follow the same format as above, and the commands are: nextFloat(), nextShort(), nextLong(), nextBoolean(), etc. The following example demonstrates how to accept a line/sentence from the user and output that right back to them. Try to understand it before reading the "ELI5".

```
package testing;
import java.util.Scanner;
public class Testing;
	public static void main (String [] args) {
		Scanner input = new Scanner(System.in);
		System.out.println(input.nextLine());
	}
}
```

* 1) import java.util.Scanner; = Java has a lot of libraries. Imagine a public library, and you want to check out a book. You know what book you want to check out but the librarian doesn't. So you tell the librarian that I want to check out "Java For Dummies". The librarian finds the book by searching it via the ISBN number and gives it to you. The libraries in java are similar to public libraries. There are tons of books you can check out, as there are tons of classes you can import. These are written by other people; authors and developers, respectively. Heck, even you can make your own book or library if you wanted to and share it with other people. Most of the books are free to check out, just like most of the classes. In order to check out a book, you need to tell the librarian the name of the book, just like in Java, you need to tell Java the name of the class, and only then can it fetch it for you. After Java fetches it for you, you are free to use it how you want, just like the book the librarian hands you. This import calls for the Scanner class. You need to call it before you can use it.

* 2) Scanner myVar = new Scanner(System.in); = In order to use the Scanner class, you need to create an instance or an object of it. Just because you were able to check out the book, doesn't mean you can immediately start to read it. You gotta get home, get under the covers, and get comfy. In Java, you need to set it up for use. This means happens by giving it a name (i.e. myVar; but you can call it whatever you want, even SashaGreyGetsBBC). The new Scanner means that it is a new Scanner object and System.in means that it takes input.

* 3) System.out.println(myVar.nextLine()); = This prints whatever the user typed into the scanner into the console field. myVar is of course the name of the Scanner and nextLine() is anticipating that the user will provide a word. However, if it was nextInt(), and you were to type a word, it would crash the program. The only reason that you can input numbers and use nextLine() is because Java will convert the number to a string, like this:

```
String num = "33"; // This is what happens. This is allowed in Java
int nums = "red"; // This is not allowed in Java and will create errors
```

QUIZ 1 – BASIC COMPREHENSION QUESTIONS

* 1. Write a program that calculates the area of a square. The measurements are provided by the user, so code a way for the user to input them and Java to accept them. Try it on your own before looking at the answer. If you went wrong wrong, try to think where you went wrong, and why. Also, be sure to add fail safes!

```
------------------------ ANSWER ------------------------
package testing;
import java.util.Scanner;
public class Testing {
	public static void main (String [] args) {
		System.out.println("Please enter the length/width of the square");
		Scanner myVar = new Scanner(System.in);
		double length = myVar.nextDouble();
		double area = length * length;
		System.out.println("The area of the square is " + area);
	}
}
------------------------ ANSWER ------------------------
```

* Some common mistakes people make is not making the variable a double and sticking with int. Always account for all possible errors when designing and programming an application. Programs are made for humans and humans are the most flawed beings alive. It's a good thing that Java can catch exceptions, which you will learn later in your Java career! Also, try repeating this question, but instead of a square, calculate it for a rectangle, triangle, etc. Keep practicing! Here's a hint for this: You can solve this by using two scanners to accept input from the user, and with only one. Think different.

CHAPTER 3 – CONDITIONALS & LOOPS

* Conditional statements are used to perform different actions based on conditions. The "if statement" is one of the most frequently used conditional statements. If the "if statement's" condition expression evaluates to be true, then the block of code inside the "if statement" is executed. If the expression is found to be false, the first set of code after the end of the "if statement" (after the curly brace) is executed. Below is an example of the syntax, and how it is used:

```
if (condition to test goes here in brackets) {
	// If condition is found to be true, then this block of code, is executed
	// All code inside these two braces is executed
}
// If condition is found to be false, the code outside is executed
```

* Notice how the first line does not end in a semi-colon, but rather in a brace. The entire block of code you want to execute is placed inside the two braces. If the condition is found to be false, the code outside the braces runs instead.

* “If statements” are used to test if an expression is found to be true or false. For example:

```
int age = 17;
if (age > 18) {
        System.out.println(“You can enter the club!”);
}
```

* Below are a list of comparison operators to test and form a condition. You can use any one of them for testing:

(<) = Less Than
(>) = Greater Than
(!=) = Not Equal To
(==) = Equal To *[Note: To test if two things are equal, you cannot use (=), you have to use (==)]*
(<=) Less Than Or Equal To
(>=) Greater Than Or Equal To

* Here is an example of how the ‘equal to’ operand can be used:

```
int age = 21;
if (age == 21) {
        System.out.println(“Gratz, you can now legally buy alchohol);
}
```

* An “if statement” can be followed by an optional “else statement” which executes when the condition evaluates to be false. This is useful when an event is binary; it’s either a yes or no, true or false – there are only two outcomes. Example:

```
int age = 17;
if (age >= 18) {
        System.out.println(“Welcome to the club!”);
} else {
        System.out.println(“Sorry you’re to young”);
}
```

* So in this case, based on the person’s age, the program will grant them access to the club, or forbid it. This is a good example of a binary scenario. The person is either to young or the right age to enter the club. Furthermore, if needed, you can also nest “if-else statements”. This means you can use one “if-else statement” inside another “if” or “else” statement. You can nest as many as you want or need! This is good for catching exceptions. For example:

```
int age = -11; // Note how age cannot be negative so this is obviously wrong
if (age >= 0) {
        if (age >= 18) {
                System.out.println(“Welcome”);
        } else {
                System.out.println(“To Young”);
        }
} else {
        System.out.println(“Not Possible”);
```

* In a nutshell, this block of code checks if the person’s age is greater than 0. If it is, it proceeds, if not it will print out “Not possible”. If the person’s age is greater than 0, the program will check if it is greater than or equal to 18, or less than 18. If greater than or equal to 18, then it will print out “Welcome”, and if not, then it will print out “To young”.

* Instead of nested “if-else statements”, you can use the “else-if statements” to check multiple conditions. For example:

```
int age = 25;
if (age <= 0) {
        System.out.println("Error");
} else if (age <=18) {
        System.out.println("Young");
} else if (age < 60) {
        System.out.println("Welcome");
} else {
        System.out.println("Old");
}
```

* Here’s what this block of code does. First, an ‘int’ named age is declared and given a value of 25. If the value of age is less than or equal to 0, then print out “Error”. Else, if the value of age is less than or equal to 18, then print out “Young”. Else if the value of age is less than 60, then print out “Welcome”. And if none of the conditions are satisfied, then print out “Old”. This is only printed out to console if the value of age is greater than or equal to 60. Also, the order of statements is really important. Reads the code from top to bottom, in sequential order. This means that if you were the switch the “else-if statements” up, the program would not run properly. For example:

```
int age = 25;
if (age <= 0) {
        System.out.println("Error");
} else if (age < 60) {
        System.out.println("Young");
} else if (age <= 18) {
        System.out.println("Welcome");
} else {
        System.out.println("Old");
}
```

* In this block of code, the cases are switched. The program will first check to see if age is less than 60, and then based off of that answer it will either continue or exist. This is an incorrect order of “if-else statements” because now all ages from 1 to 59 are gonna print “Welcome”, and you cannot have minors in a club; that is illegal!

* A) LOGICAL OPERATORS: Logical operators are used to combine multiple conditions. For example, instead of this:

```
if (age > 18) {
        if (money > 500) {
                System.out.println("You can join da club!");
        }
}
```

You can do this:

```
if (age > 18 && money > 500) {
        System.out.println("You can join da club!");
}
```

* Both methods only execute if both conditions are satisfied, but the second one is better is terms of minimalistic coding. The first block of code takes 4 lines, while the second block takes 3 lines. However, both get the job done!

* Below are a few logical operators you can use:

(&&) – Conditional [AND]
(||) – Conditional [OR]
(?.) – Ternary (Shorthand for “if-else statement”)

* Here is an example of how you would use the OR (||) operator:

```
int age = 25;
int money = 100;

if (age > 18 || money > 500) {
	System.out.println(“Welcome!”);
}
```

* As you can see, you no longer need to be at least 18 years of age and 500 dollars to enter the club, you can be 18, and broke AF to enter, or a rich Saudi toddler with an inheritance greater than 500 dollars to enter LOL. The following is an example of how you would use the Ternary (?.) operator. So, instead of writing this mess:

```
int i = 0; int m; // You can initialize variables like this – just make sure the semicolon between the variable is present!
if (i == 0) {
	m = 10;
} else {
	m = 5
} System.out.println(m); // Prints 10 // Side note: You can start your next line of code right after the brace or semicolon!
```

Using the ternary operator, you can shorten the above block of code from 7 lines to just 1:

```
int i = 0; int m = (i == 0)? 10:5; System.out.println(m); // Prints 10
```

* Another operator you should become famililar with is the NOT (!) logical operator, which is used to reverse the logical state of the operand. If a condition is true, the NOT logical operator will make it false. For example:

```
int age = 25;
if (!(age > 18)) {
	System.out.println(“To Young”);
} else {
	System.out.println(“Come In”);
}
```

* Copy and paste this code into Eclipse and see how it works! The second line (!(age > 18)) is read as, “if age is NOT greater than 18”. Although you can get away with using nested “if statements” and “if-else statements”, it is good to know the NOT logic operator. Also, if you are planning on using the NOT logic operator, remember the flip the direction of the operand. For example, here’s how you would test and “if” condition both ways:

```
if (!(age > 18)) // Using the NOT operator, the operand is “>”.
if (age < 18) // Not using the NOT operator, the operand is “<”.
```

* So if you were to use to the NOT operand, remember to switch the signs or else Java will check the reverse conditions, and all of a sudden you’re letting minors into a club. However, you can get by just fine without knowing and using the NOT operand. A lot of developers just stick to regular “if”, and “if-else statements”. But do know the difference!

* B) SWITCH STATEMENT: A switch statement tests a variable for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case. Here is an example of the syntax:

```
switch (expression) {
        case value 1:
                // Statements
                break; // Optional
        case value 2:
                // Statements
                break; // Optional
        case value k:
                // Statements
                break; // Optional
        default: //Optional
                // Statements
        }
```

* When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line after the switch statement. Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached. The example below tests a day against a set of values:

```
int day = 3;

switch (day) {
        case 1:
                System.out.println("Monday");
                break;
        case 2:
                System.out.println("Tuesday");
                break;
        case 3:
                System.out.println("Wednesday");
                break;
        case 4:
                System.out.println("Thursday");
                break;
        case 5:
                System.out.println("Friday");
                break;
}
```

* The output of this block of code is "Wednesday". This is because int day = 3, and corresponds to case 3, so everything under case 3 will execute, thus Java will print "Wednesday" to the console. If we changed "day" to 4, and then let it run, everything under case 4 would execute, and Java would print "Thursday" to the console. Note: You can have any number of case statements within a switch. Each case is followed by the comparison value and a colon. A switch statement can have an optional default case. The default case can be used for performing a task when none of the cases match. It is not mandatory, so you do not have to include it, but it would be wise to use it to catch exceptions and errors. For instance, let's say you want the user to pick either Saturday or Sunday, and instead they pick a different day. You can program it like this to catch the exceptions:

```
int day = 3;
switch (day) {
	case 6:
		System.out.println("Saturday");
		break;
	case 7:
		System.out.println("Sunday");
		break;
	default:
		System.out.println("Weekday"); // This is the output
	}
```

* So if the user picks a different day other than Saturday or Sunday, you can notify them to change it. The switch statement allows you to look for a certain case and execute code for it. Of course you can get by with "if statements" but switch statements are also valid to use and applicable for coding. Also, notice how the default statement does not require a "break" because it is already the last case in the "switch" code. The purpose of the "break" statement is to stop the execution of code because the case has already been found and there is no need to check other statements as it is a huge waste of time and precious resources. When you start coding big applications, you will find it useful!

* C) LOOPS: A loop statement allows you to repeatedly execute a statement or group of statements. It continues to execute as long as a given condition is true. There are three types of loops, "While Loops", "For Loops", and "Do While Loops". All are explained below with great detail, so read and analyze it with a fine-tooth comb!

1. WHILE LOOP:

```
int x = 0;
while (x < 4) {
	System.out.println(x);
	x = x + 1; // Recall you can even use x++ to increment the value of 'x' by 1
}
```

* The while loop checks for the condition (x < 4). If it evaluates to be true, it executes the statements within its body. Then it checks for the statement again and repeats until it is found to be false. Also, notice the "x = x + 1". If it was not there, then the loop would execute forever. This is known as an infinite loop, and would run forever and cause the program to crash because the value of 'x' never increases, hence the statement will always be true. Another important concept to learn is the ability to code cleanly and concisely. The less lines, the better, and sometimes it is faster, and the file size is smaller. For instance, the above block of code can be simplified to:

```
int x = 0;
while (x < 4) {
	System.out.println(x++);
}
```

* Notice how we shortened the "x = x + 1" into "x++" and merged it into the "println" statement. By doing this, we went from 5 lines of code, down to 4. Clean, concise, and efficient code is really important especially for large scale programs. In a "while loop", when the expression tested is found to be false, the loop body is skipped and the first statement after the while loop is executed. This applies to all "if statements", and loops. For example:

```
int x = 6;
while (x > 10) {
	System.out.println(x++);
}
System.out.println("X is less than 10"); // Output is only this statement – nothing else is printed to console
```

2. FOR LOOP:

* A "for loop" allows you to efficiently write a loop that needs to execute a specific number of times. In essence, a "for loop" is just like a "while loop" but it is more efficient and cleaner/concise. Here is the syntax:

```
for (initialization; condition; increment/decrement) {
	// Code To Be Executed, If Condition Found True, Goes Here
}
```

* So here's what is happening when you use a "for loop". Understand this before moving on to the next example:
i. Initialization: The expression executes only once during the beginning of the loop
ii. Condition: This is evaluated each time the loop iterates. The loop executes the statement repeatedly, until this condition is returned false
iii. Increment/Decrement: This executes after each iteration of the loop

```
for (int x = 1; x <= 5; x++) {
	System.out.println(x);
}
```

* This block of code outputs numbers from 1 all the way to 5. Notice the semicolon after the initialization and condition in the syntax. The first thing that happens is the variable 'x' is initialized. Then, Java checks to see if 'x' is greater than or equal to 5. If the condition is found to be false, 'x' is printed out to the console, and then 'x' is incremented by 1. You can have any type of condition and any increment amount in the loop. The example below only outputs even values to 10:

```
for (int x = 0; x <= 10; x += 2) {
	System.out.println(x);
}
```

* A "for loop" is best when starting and ending numbers are known. So if you know the final product, then "for loops" are easy to implement and an excellent choice. "While loops" are just easier to code and conceptualize.

3. DO WHILE LOOPS

* A "do while loop" is similar to a "while loop", except that a "do while" loop is guaranteed to execute at least one time. This is because the conditions appears at the end of the loop, so the statements in the loop execute at least once before tested. Here is an example of a "do while" loop, outputting numbers from 1 to 5:

```
int x = 0;
do {
	System.out.println(x++);
} while (x < 5);
```

* Even with a false condition, the code WILL run once! Take a look at the example below:

```
int x = 1;
do {
	System.out.println(x++);
} while (x < 0);
```

* This block of code outputs 1. Even though 1 is not greater than 0, the code runs at least once. Go ahead and paste it into Eclipse and try it for yourself!

4. LOOP CONTROL STATEMENTS

A) BREAK: The break and continue statement change the loop's execution flow. The break statement terminates the loop and transfers execution to the statement immediately following the loop. For example:

```
int x = 1;
while (x > 0) {
	System.out.print(x + ", ");
	if (x == 4) {
		break;
	}
	x++;
}
```

* The output of this block of code is:

```
1, 2, 3, 4
```

* The above block of code is identical to this while loop, the major difference is that it is missing the "if statement":

```
int x = 1;
while (x < 5) {
	System.out.println(x++);
}
```

* Without the "if statement" it would run forever. The "if statement" allows the code to continue after the "while loop".

B) CONTINUE: The continue statement causes the loop the skip the remainder of its body and then immediately retest its condition prior to reiterating. In other words, it makes the loop skip to its next iteration. For example:

```
for (int x = 10; x <= 40; x += 10) {
	if (x == 30) {
		continue;
	}
	System.out.println(x + ", ");
}
```

* This block of code outputs:

```
10, 20, 40
```

* As you can see, it skips printing the number 30, as directed by the continue statement. If 'x' is equal to 30, then continue the code as normal, but do not print it. This is essentially what is happening in the above block of code.

Q U I Z 2 – C O N D I T I O N A L C O M P R E H E N S I O N

* 1. Write a program using a "for loop" print only even numbers from 1 to 100. Then, modify it to print only odd numbers from 1 to 100. Try this on our own, before looking down at the answer.

```
for (int xf = 0; xf <= 100; xf += 2) {
	System.out.println(xf);
}

// To print only odd numbers, you would change "xf" 's starting value to 1
```

* 2. What are (&&) and (||), and what do they do? Provide an example for each.

These are operands to check conditions on statements such as “if”, “while”, etc. The (&&) operators checks to see if both conditions are true. For instance, a && b checks if both ‘a’ and ‘b’ are true. On the other hand, (||), checks to see if only one statement is true. For instance (a || b) checks to see if either ‘a’ or ‘b’ is true.

* 3. Write a program that will add all the numbers from 1 to 100. i.e. 1 + 2 + 3 + 4 + 5 + … + 100.

```
int one = 1; int tots = 0;
while (one < 101) {
        tots += one;
        one += 1;
}
```

* 4. Write a program that will calculate the following in the same manner as question three: 3 x 7 x 11 x 15 x … x 168. Hint: You gotta think big. Bigger than int; it’s too small for you!

```
double xyz = 1;
double xxy = 3;
double county = 4;

while (xxy < 168) {
        xyz *= xxy; xxy += county;
        if (xxy == 167) {
                System.out.print(“The answer is: ”);
        }
} System.out.println(xyz);
```

* 5. Write a program that asks the user for their age, starting from 1 years old, all the way to their age. i.e. The program will ask you, “Are you 1”, “Are you 2”, “Are you 3”, etc., all the way till you say yes to your age.

```
int age = 1;
Boolean user = false;
Scanner newAge;

while (user == false) {
        System.out.println(“Are you ” + age1 + “?”);
        if (age == 1) {
                System.out.println(“Yes = 1 & No = 2”);
        }
        newAge = new Scanner (System.in);
        if (newAge.nextInt() == 1) {
                System.out.println(“Gratz you are ” + age);
                User = true;
        } else {
                age++;
        }
}
```

* If you struggled with this question, don’t worry because this question is hard and was designed to be tricky. The trick is that you have to to use numbers to find out if the person’s age has been reached or not. What this means is, the user cannot enter in “yes” or “no”, they must enter a number to represent their decision. Initially, you probably tried to use strings, but it didn’t work. That’s because comparing strings in Java is done using the (.equals()) function. For example:

```
String ans = “yes”
if (ans.equals(“yes”) || (ans.equals(“Yes))) {
	System.out.print(“Cool!”);
} else {
	System.out.println(“No”);
}
```

* Another thing I wanna talk about is code “Robustibility”. This is not a legit word but it is a measure of how robust your code is. Meaning, can it catch exceptions and errors? Can it properly handle different input? Will it crash? How easily can some one crash it? For question 3, the sample code provided will easily crash if the user were to enter a word like ‘yes’ or ‘no’, and it would crash if the user entered a very lrage number like 987654321987654321987654321. Take a look at the following block of code. Paste it into Eclipse and play around with it as much as you can! It’s un-crash-able ☺

```
int agem = 1;
Scanner newAge;
boolean user = false;

while (user == false) {
	System.out.println(“ Are you “ + agem + “?”);
	if (agem == 1) {
		System.out.println(“Enter Yes/No”);
	}
	newAge = new Scanner (System.in);
	String age = newAge.next().toLowerCase();
	If (age.equals(“yes”)) {
		System.out.println(“\nGRATZ!!! You are” + agem);
		user = true;
	} else if (age.equals(“no)) {
		agem++;
		System.out.println(“”);
	} else {
		System.out.println(“\nWRONG INPUT!!! TRY AGAIN!!! \n”);
	}
}
```

* See the difference between this block of code and the other ones? This one will not crash no matter what input and how large the number is. You can even enter things like “yEs”, “nO” or “yeS”, and it will still accept it and work properly. Another key thing is to have properly formatted code that looks nice and has a good-looking output window. Having robust code that is clean and properly formatted is super duper important. Robust code is even acknowledged by gods!

C H A P T E R 4 – A R R A Y S

* An array is a collection of variables of the same type. When you need to store a list of values, such as numbers, you can store them in an array, instead of declaring separate variables for each number. To declare an array, you need to define the type of elements with square brackets. For example, to declare an array of integers:

```
int [] arr;
```

* The name of the array is ‘arr’. The type of elements it will hold are ‘ints’. Java knows you want to use the array function because of the square brackets ([]). If you did not have square brackets, you would be declaring an ‘int’. Now you need to define the array’s capacity, or number of elements it will hold. To do this, use the word ‘new’:

```
int [] arr = new int [5];
```

* The code above declares an array of 5 integers. In an array, the elements are ordered and each has a specific and constant position, which is called an index. In an array, the first position starts at 0. So, for our array, the positions are:

```
___    ___    ___    ___    ___
0      1      2      3      4
```

* To reference elements in an array, type the name of the array followed by the index position within a pair of square brackets. For example, to occupy the second position of our array called ‘arr’, you would type:

```
arr[2] = 42; // The SECOND position in our array holds a value of 42
```

* Java provides a shortcut for instantiating arrays of primitive types and strings. If you already know what values to insert into the array, you can use an array literal. For example:

```
String [] myNames = {“A”, “B”, “C”, “D”};
System.out.println(myNames[2]);
```

* Note: Sometimes you might see the square brackets ([]) placed after the array name. However, the preferred way is to place the brackets after the array’s data type. For example:

```
String [] names; // This is preferred
String names []; // But this works too
```

* You can access the length of an array (the number of elements it stores) via its length property. For example:

```
int [] Art = new int[5];
System.out.println(Art.length); // Output is 5
```

* Now that we know how to set up and get array elements, we can calculate the sum of all elements in an array by using loops. The “for loop” is the most used loop when working with arrays as we can use the length of the array to determine how many times to run the loop. Below is a block of code that will calculate the sum of all elements in an array:

```
int [] arr = {6, 42, 3, 7}
int sum = 0;
for (int x = 0; x < arr.length; x++) {
        sum += arr[x];
}

System.out.println(sum); // Outputs 58 (Because 6 + 42 + 3 + 7 = 58)
```

* In the code above, we declared a variable sum to store the result and assigned it 0. Then we used a “for loop” to iterate through the array, and added each element’s value to the variable. In simpler terms, we made an integer array called ‘arr’ and stored the values 6, 42, 3, 7 in it. Then we made an interger named ‘sum’ and gave it a value of 0. Using a “for loop”, we added up all the values in ‘arr’ and gave it to ‘sum’. We were able to do this by creating another integer named ‘x’ and gave it a value of 0. As long as ‘x’ is less than the length of ‘arr’, ‘sum’ gets whatever value ‘arr’ is storing at ‘x’ position.

* A) ENHANCED FOR LOOPS: The “enhanced for loop” (sometimes called a “for each loop”) is used to traverse elements in arrays. The advantages are that it eliminates the possibility of bugs and makes the code easier to read. For example:

```
int [] primes = {2, 3, 5, 7};
for (int ssh: primes) {
	System.out.print(ssh + “, “);
}
```

* The output of this is:

```
2, 3, 5, 7
```

* The “enhanced for loop” declares a variable of a type compatible with the elements of the array being accessed. The variable will be available within the for block, and its value will be the same as the current array element. So each iteration of the loop, the variable ‘t’, will be equal to the corresponding element in the array. This sounds confusing as shit so let me break it down for you. First of all, we declare an array of type ‘int’ and store the values, 2, 3, 5, 7, inside of it. Then, using a “for loop”, we iterate the array. We create a new integer named ‘ssh’, and print it out, and then move onto the next number. The first element is 2, is 2 is assigned to ‘int ssh’, then printed out. The next number is 3, and the process is repeated until all elements in the array are printed out into the console. You can do the same for strings. i.e.:

```
String [] primes = {“Grapes”, “Apple”, “Banana”};

For (String prim: primes) {
	System.out.println(prim + “, “);
}
```

* The output of this is:

```
Grapes, Apple, Banana
```

* The same thing for ‘int” is happening for this block of code. First, an array of type strings is created. It is called ‘primes’ and “Grapes”, “Apple”, and “Banana” are assigned to it. Then, using a “for loop”, a string named ‘prim’ is created and assigned the first item in the array named ‘primes’. Then ‘prim’ is printed out to the console. Then, the next item in ‘primes’ is assigned to ‘prim’ and the process is repeated until all items are printed out. Also, notice the colon after the variable. After “String prim”, there is a colon. The same rule applies for “int ssh”.

* B) Multi – Dimensional Arrays: These are arrays that contain other arrays. The two-dimensional array is the most basic multi-dimensional array. To create multi-dimensional arrays, place each array within its own set of square brackets. i.e.:

```
int [][] samus = {{1, 2, 3}, {4, 5, 6}};
```

* This declares an array with two arrays as its elements. To access an element in the two-dimensional array, provide two indexes, one for array, and another for the element inside that array. The first number is the array you want to access and the second number is the position in that array you want to output. The following example prints the first element in the secondary array of ‘samus’:

```
int [][] samus = {{1, 2, 3}, {4, 5, 6}};
System.out.println(samus[1][0]); // Output of this is 4
```

* The array’s two indexes are called row index and column index. You can get and set a multidimensional array’s elements using the same pair of square brackets. For example:

```
int [][[] arr = {{1,2, 10}, {4, 5, 6}, {7, 8, 9}}; // This is an example of a three dimensional array
System.out.println(arr[1][1]); // This prints the second item in the second array
System.out.println(arr[0][2]); // This prints the third item in the first array
arr[0][2] = 3; // This assigned the third item in the first array a value of 3
System.out.println(arr[0][2]); // This prints the third item in the first array
```

* The above two dimensional array contains three arrays. In Java, you are not limited to two-dimensional arrays. Arrays can be nested within arrays to as many levels as your program needs. To have multi-dimensional arrays, just have the appropriate number of curly brackets. However, keep in mind that the more arrays, the harder it is to maintain. Furthermore, don't forget that all array members must be of the same type. So you cannot have 'strings' and 'ints' in the same array. Arrays must be 'int' only or 'String' only, etc.

Q U I Z 3 – A R R A Y S

* 1. What is the output of this code?

```
int arr[] = new int[3];
for (int i = 0; i < 3; i++) {
	arr[i] = i;
}

int res = arr[0] + arr[2];
System.out.println(res);
```

* Alright, so let's break down this code, line by line, and analyze it. First of all, an array of type 'int' is declared and assigned four place holders. We know this because of the 'int[3]'. So, 'int arr' looks something like this:

```
___     ___     ___     ___
0       1       2       3
```

* Next, using a for loop we iterate 'arr' with values. So, we create a new 'int' named 'i'. As long as 'i' is less than 3, we add one to 'i', and assign that value to 'arr'. So, the first position is 'arr' is 0 because 'i' starts off as 0. Now, 'int arr' looks something like this:

```
[0]     ___     ___     ___
0       1       2       3
```

* The key piece of code is:
arr[i] = i;

When i = 0, then arr[0] = 0
When i = 1, then arr[1] = 1
When i = 2, then arr[2] = 2

And 'i' can only go up to a max value of 3 because of the condition (i < 3), so arr[3] does not have any value assigned to it. Go ahead and paste this code into Eclipse and try to print out arr[3], and see what happens!

Now, 'arr', looks like this:

```
[0]     [1]     [2]     ___
0       1       2       3
```

So, 'int res' is equal to arr[0] which is 0, plus arr[2] which is 2. The sum of these two numbers is 2. 'int res' = 2

* 2. What is the output of this code?

```
int result = 0;
for (int ill = 0; ill < 5; ill++) {
        if (ill == 3) {
                result += 10;
        } else {
                result += ill;
        }
}

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

* First of all, an integer named result is declared and assigned a starting value of 0. Then, using a for loop, 'result' is iterated. An integer named 'ill' is created and assigned a value of 0. As long as 'ill' is less than 5, it's value is incremented by one and the code below it is executed. So if 'ill' is equal to 3, then result gets 10 added to it or else result gets whatever value 'ill' is currently assigned. Let's break this down further by analyzing what happens when ill = 1, 2, 3, etc. When 'ill' = 0, 'ill' does not equal to 3 so the else statement is executed. The integer 'result' gets whatever value 'ill' is added to it. So 'result' is 0 and 'ill' is 0, hence the final value is 0. When 'ill' = 1, 'ill' does not equal to 3 so the else statement is executed. The integer 'result' gets whatever value 'ill' is added to it. So 'result' is 0 and 'ill' is 1, hence the final value is 1. When 'ill' = 2, 'ill' does not equal to 3 so the else statement is executed. The integer 'result' gets whatever value 'ill' is added to it. So 'result' is 1 and 'ill' is 2, hence the final value is 3. When 'ill' = 3, 'ill' DOES equal 3 so the first part is executed. 'Result' gets 10 added to it, so 3 + 10 is equal to 13. When 'ill' = 4, 'ill' does not equal to 3 so the else statement is executed. The integer 'result' gets whatever value 'ill' is added to it. So 'result' is 13 and 'ill' is 4, hence the final value is 17. When 'ill' is 5, the condition 'ill' is less than 5 is no longer true and the for loop stops, and the next line after the for loop is executed which would be the print statement. The final value of 'result' is 17 and gets printed out to the console. Therefore the output of this console is 17!

* 3. Write a program that accept 5 numbers from the user, put them in an array, and then using an "enhanced for loop", it will calculate the sum of all elements in that array and output it to the console. Your program does not need to have safe-guards, it just needs to work! Even though this is easy, feel free to consult the internet.

```
byte ic = 0;
double ace [] = new double [5];
Scanner scc;

System.out.println("Enter five numbers please!");

While (ic < 5) {
        scc = new Scanner(System.in);
        ace[ic++] = scc.nextDouble();
}

double resu = 0;
for (double del: ace) {
        resu += el;
}

System.out.println("The sum is " + resu);
```