LESSON 1: HELLO WORLD

Press CTRL + Spacebar to bring up the code hinting in Eclipse and Netbeans

i.e. Type main then press CTRL + Spacebar

i.e. Type syso then press CTRL + Spacebar

The main method is where the program begins – whatever is inside it, is what runs

You can print stuff to the console using: System.out.println(“Print To Console”);

Comments are notes that explain what the code is doing. It is completely ignored by the compiler, thus it is only relevant to you, the developer. To insert comments, type “//” followed by your comment. i.e.:

// This is a comment

LESSON 2: TYPES & VARIABLES

This is how you create an integer: int number = 5;

The variable above is of type int, called ‘number’ and holds a value of 5

Every statement must end in a semicolon ( ; )

You can output the value of ‘number’ to console like so: System.out.println(number);

You can manipulate variables by adding or subtracting values like so: number = number + 5

In the statement above, the right side is being evaluated and that is being assigned to the left side. So, ‘number’ plus 5 is 5 + 5 and that is assigned to ‘number’. Hence ‘number’ holds a value of 10

You can add variables together as well. Imagine this:

int variable1 = 5;

int variable2 = 7;

System.out.println(variable1 + variable2); // Prints out 12 to console because 5 + 7 = 12

An ‘int’ can only hold integers, and not a fractional value. This is because computers store integers and fractions in memory, very differently

To make a fractional value [aka floating point number], use double like so:

double x = 2.5;

You cannot assign fractional values to ints without getting an error. This is because you lose precision [i.e. the decimal]. However, you can assign integers to doubles.

To hold text, use a string like so: String hello = “Hello World!”;

And you can print it to console like this: System.out.println(hello);

String concatenation is when you combine strings using the plus operator. For example:

String first\_name = “John ”;

String last\_name = “Smith”;

System.out.println(first\_name + last\_name);

LESSON 3: METHODS

You can create more methods by placing them after the curly brace of the main method. For example:

static void sayHelloWorld() {

System.out.println(“Hello World”);

}

And you can call the method above as many times as you want. When a method runs, it jumps down to where method is, the code inside of it is executed it, and then Java picks up where it left off

You can even pass parameters in methods like so:

static void sayHelloTo (String name) {

System.out.println(“Hello, ” + name);

}

Methods can even return values. For example:

static int returnFive() {

return 5;

}

The keyword void means that nothing is returned. If something is returned, void is replaced by the type of return value [i.e. int, double, string, etc.]

Methods can take a parameter, manipulate it and return it. For example, the following method takes a parameter of type int and returns it’s square:

static int square(int x) {

return (x \* x);

}

LESSON 4: VARIABLE SCOPE

Every method including the main method have local variables. This means that variables declared in the main method do not exist and cannot be accessed outside the main method. Variables declared in method doSomething(), only exist in that method and cannot be accessed in the method doSomethingElse().

You can declare global variables by declaring them outside of any method. Global variables can be declared in any method and manipulated in any method.

Variables declared outside of any method are referred to as class variables, and variables declared in a method are local variables

The ‘static’ keyword means that you don’t have to instantiate it in order to use it

The ‘final’ keyword is used to make the variable immutable - meaning it can’t be manipulated/changed

The ‘private’ keyword makes the variable only accessible within the class, and ‘public’ makes it accessible within the package

LESSON 5: CONDITIONALS

Use ‘If’ statements and ‘while/for’ loops to execute code conditionally. This means that if the conditions are not met [i.e. a certain variable is not a certain value], the code will not execute

A Boolean holds two values, true or false.

Here is an example of an ‘if’ statement:

if (true) { // You can change ‘true’ to ‘false’ and Java skips the code inside the ‘if’ block

System.out.println(“Hello”);

}

In conditionals, everything comes down to Booleans – something is either true or false. When Java evaluates the expression of a conditional, it simplifies it down to either true or false. For example

if (5 > 10) { } // Evaluates to false

if (1 < 10) { } // Evaluates to true

‘If-Else’ statements are used to execute certain blocks of code. For example:

if (true) {

System.out.println(“Hello”); // Only this line of code gets executed

} else {

System.out.println(“Goodbye”); // This line is skipped

}

Operands in Java include: >, <, >=, <=, !=, ==

In order: Greater than, less than, greater than or equal to, less than or equal to, not equal to, equals

!= translates to logical not and is used to test if two things are not equal to each other

Pro tip: x = 5 means that 5 is being assigned to 5, AND, x == 5 tests the equality if x is equal to 5

You even test multiple conditions using the AND operator [ && ] and the OR operator [ || ]

For example:

if (x > 5 && x ==6) { // This will only execute if ‘x’ is greater than 5 AND if ‘x’ is equal to 6

System.out.println(“TRUE”);

}

if (x > y || x < 5) { // This will only execute if ‘x’ is greater than ‘y’ OR if ‘x’ is less than 5

System.out.println(“TRUE”);

}

You can even nest ‘if’ statements like this:

if (true) {

if (false) {

System.out.println(“Hello”);

}

}

You can use an if-else-if ladder to test multiple conditions. For example

If (x == 5) {

System.out.println(“If ‘x’ is equal to 5 then this is printed to console”);

} else if (x == 7) {

System.out.println(“If ‘x’ is equal to 7 then this is printed to console”);

} else if (x == 8) {

System.out.println(“If ‘x’ is equal to 8 then this is printed to console”);

} else if (x == 10) {

System.out.println(“If ‘x’ is equal to 10 then this is printed to console”);

} else {

System.out.println(“If ‘x’ is does not 5, 7, 8, or 10, then this is printed to console”);

}

Your if-else-if ladder should be organized from most general to most specific. This is because Java only executes one expression from the entire ladder

LESSON 6: LOOPS & DEBUGGER

Use loops to run code over and over again. You can use while loops, for loops, and do while loops.

For example, this is a while loop:

int counter = 0;

while (counter < 10) {

System.out.println(counter);

counter = counter + 1;

}

The code above prints numbers from 0 to 10 to console.

A do while loop executes the code at least once regardless of the expression. This is because in a do while loop, the condition is tested AT THE END. For example:

int x = 10;

do {

x++;

System.out.println(x);

} while (x < 10);

11 gets printed to console because ‘x’ gets incremented by 1 and then gets printed to console and then, in the end, gets tested in the condition. The following example is a for loop:

for (int y = 0; y < 10; y++) {

System.out.println(y);

}

A for loop consists of 3 parts: The initializer, the condition, and the increment/decrement.

Initializer: int y = 0 [The variable y is first initialized]

Condition: y < 10 [A condition is made to be tested]

Increment/Decrement: y++ [A variable in the condition is manipulated]

LESSON 7: USING OBJECTS

Create objects using the ‘new’ keyword. For example: JFrame window = new JFrame();

Object Oriented Programming makes thinking about programming closer to real life. In OOP, everything is an object and objects have properties and characteristics.

Use the code assist/hinting feature to reveal all the methods of an object

A string is also an object – so when you create a string, you are creating an object

LESSON 8: CLASSES & OBJECTS

A class is kind of like a blueprint for an object. You only need one blueprint to build a house, but using that one blueprint you can build as many houses as you want.

Once you have created your class, you can instantiate as many objects as you want from the class. And each object will be independent from the others

Each object can have its own variables and methods. So changing one object, doesn’t affect the others

We can create an instance of a class or an object using the ‘new’ keyword, like so:

JLabel label = new JLabel();

Person person = new Person();

When a method or variable is static, it becomes a member of the class itself and not an instance of the class. In other words there is only one variable for that class, as there is only one variable in memory

When we create an object we do so by calling the constructor with the keyword ‘new’ and the name of the class followed by parenthesis.

When an object is created, the code inside the constructor is executed every time the object is created.

We can pass data to the constructor as we do with methods

Use the ‘this’ keyword to refer to the object created inside the class

Good coding practicing involves generating getters and setters to get and set the values of variables. Using getters and setters helps with encapsulating data and preventing unauthorized manipulation. Also, variables should be declared private

LESSON 9: REFERENCE & VALUE TYPES

Value types in Java include: byte, short, int, long, float, double, boolean, and char

Anytime you are using a variable [see above], you are using a value type. And anytime you are using something else [i.e. String, JLabel, etc.] you are using a reference type.

Java passes by value, not reference. So when an int is passed, Java passes the value, not the int itself

When we pass a reference type, we pass a reference that takes us to the object that allows us to manipulate the object itself

LESSON 10: ACCESS MODIFIERS

Access modifiers put restrictions on how parts of your code can be accessed

The access modifiers in Java are: public, private, protected, default

Public: Can be accessed anywhere in the program as long as you have an instance of the class to call it from

public void doSomethingPublic() { }

Private: The method cannot be accessed anywhere other than inside of this class

private void doSomethingPrivate() { }

Protected: The method can only be accessed inside the package and from subclasses of the class

protected void doSomethingProtected() { }

Default: The method can only be accessed inside the package

void doSomething() { }

In order of most restrictive to least: private, default, protected, public

The above access modifers can also be applied to variables. For example:

private int x;

public int y;

As a general rule of thumb, methods and variables should be private as possible. This helps make the program more secure and simplifies the interfaces. You can use getters and setters to get and set the values of variables

LESSON 11: PACKAGES

Packages are always in lowercase [i.e. testing] while classes have the first word capitalized [i.e. TypesAndVars]

Classes that are in the same package, don’t need to be imported

You can think of packages like folders, and you can create more folders inside folders to resemble a directory structure. Hence, you can create packages inside packages to help organize your code

Use the fully qualified name to avoid using the import statement and distinguish similar objects

The package declaration is the very first line of code in a Java file [with the exception of a comment]

LESSON 12: DATA STRUCTURES & ARRAYS

An array is a data structure – contains multiple elements of a variable. i.e. Contains many ints or doubles

Here is how you declare and initialize an array to hold 4 elements:

int[] name = new int[4];

name[0] = 5;

name[1] = 4;

name[2] = 3;

name[3] = 2;

Note: Array indices start at 0, not 1. So an array holding 4 elements looks like this:

5 4 3 2

--- --- --- ---

0 1 2 3

Arrays work hand-in-hand with for loops. When iterating arrays, you can use an enhanced for loop, which is tailored specifically for arrays.

When you try to assign or get a value outside the range of the array leads to an error. For example:

name[4] = 6; // This throws an exception because 4 is outside the element of the array ‘name’

You can even initialize arrays like this:

int[] numbers = new int[] {5, 4, 3, 2, 1, 6, 7, 8, 9, 0};

You can create arrays to store anytime of data, including objects and variables

Also, an array is actually an object and you can call methods on it [i.e. Get length of the array]

LESSON 13: CONSTANTS

A constant is a value that never changes, and they are static [i.e. Math.PI]

By convention, constant values are CAPITALIZED

Variables that will never change [i.e. Screen resolution] are excellent candiates to be constant

You can make a variable constant by using the ‘final’ modifier. Also, you should add the ‘static’ keyword as well. For example:

static final int WIDTH = 800;

static final int LENGTH = 600;

Constants can be public because there is no harm in accessing variables directly if you can’t alter it

Since constants are static, you should access them through the class name, and not an instance of the class

For example: Constants.WIDTH, Math.PI, Constants.LENGTH

LESSON 14: INHERITANCE

Inheritance in OOP is somewhat similar to thinking about inheritance in a parent and a child. For instance, a child might inherit an eye color from a parent. In general, the child can inherit traits from the parent(s)

Use the ‘extends’ keyword to inherit classes. i.e.: public class Child extends Parent { }

Every class in Java automatically inherits a class called Object

When dealing with inheritance, there is a one way relationship between the parent class and child class

Any of the methods and variables inside parent will be inherited by the child. And you can add more methods and variables to the child class.

The child class and the parent class have a ‘”is-a” relationship. The child is a parent, but the parent isn’t a child.

Think about it this way. You have a class called Mammal which is the base class, and a class called Dog which is the sub class. A dog is a mammal, but a mammal is not necessarily a dog, it can also be a cat. Hence, this relationship is only one way.

Parent class = Base class = Super class

The class below inherits from the class above

Child class = Derived class = Sub class

You can create as many classes as you want and they can all extend one parent class

LESSON 15: OVERRIDING METHODS

Use ‘@Override’ to ignore the methods in the parent class and use the methods in the child class. Overriding a method allows you to add to or completely change the functionality of the inherited method from the parent class

The keyword ‘super’ refers to the instance of the parent class

LESSON 16: ABSTRACT CLASSES & METHODS

Adding an ‘abstract’ modifier prevents that class to be instantiated

Abstract classes exist to be extended

The abstract keyword represents all objects. i.e. An abstract game-object can be a player, enemy, tree, weapon, etc.

Polymorphism: One form, many implementations

You can have private variables, regular old methods, etc. inside an abstract class. Adding the abstract modifier allows you to use abstract methods, and prevents you from creating instances of the abstract type