JAVA PROGRAMMING

Programming Fundamentals:

* Variables
* Data Types
* Operators
* Conditions
* Loops
* Methods

Object Oriented Programming:

* Concrete Classes
* Objects
* Abstractions
* Encapsulation
* Inheritance
* Polymorphism
* Exceptions
* Abstract Classes
* Interfaces
* File Input/ Output

Concept of Programming:

* Programming – The act of writing a computer program (a set of instructions of a computer).
* Program is a set of instructions that tell a computer what to do.
* Object Code – Refers to low level code which is understandable by machine. It is generated from source code after going through compiler.
* We use programs to interact / talk with computers.
* To write the programs we use programming languages.
* Program are written in a language a computer can understand.
* Programming Language is a set of instructions that a particular computer is designed to execute-run.

Machine Language

* A compute’s native language. (has to do with where the computer is coming from).
* Uses zero and one (0/1) – Binary Language.
* Machine language is dependent (works on certain machines and might not work on certain machines) and differs among different types of machines.

Assembly Language:

* Was developed to make programming easier.
* Machine dependent.
* Introduced keywords (add, sub, ...)
* A program called “assembler” translates assembly code to machine code.

High-Level Language:

* A new generation of programming languages.
* Uses English words, which is easy to learn and use.
* Machine independent (run on different machines).
* Instructions are called “statements”.
* A program written in a high-level language is called a “source program” or “source code”

NB. All instructions written in other programming language must be translated to machine code instructions.

A “compiler” or an “interpreter” is used to translate source code to machine code.

Compiler Translate all source code into machine code.

Translator Translate each statement into machine code and executes it right away.



JAVA

* Developed at Sun Microsystems purchased by Oracle in 2010.
* Developing software that run on mobile, desktop and servers.
* Machine independent.
* API (Application Programming Interface)
* Also known as library.
* Contains predefined Java code that can use to develop Java programs.
* 3 Java Editions:
  + Java Standard (SE) – foundations of all other editions.
  + Java Enterprise (EE)
  + Java Micro (ME)
* Java Development Kit (JDK)
  + Set of programs that enable us to develop our programs
  + Contains Java Runtime Environment (JRE)
  + JRE and JDK contains Java Virtual Machine (JVM)
  + JVM executes our Java programs on different machines.
* Integrated Development Environment (IDE)
  + A program that allows us to write (source code).
  + Able to compile our program to machine code.
  + Tools helps us to debug errors (find errors).
  + Build our program to be executed by JVM.
  + Run our program.
  + NB. Development is faster and easier.
  + e.g. NetBeans, Eclipse, IntelliJ IDEA

Class:

* A Blueprint/representation to create Objects.
* Object is an instance/exemplar of a Class.
* Creating an object using Class keyword.
  + class class\_name {
  + Code block
  + }

Method:

* Group of instructions to do a specific task.
* We have a special method called “main()”
* 4 main parts of each method.
* Every method is written inside a class.
* The main method is automatically called when we run our JAVA program.
  + It is the first method that is called.
  + It I the starting point of execution of our program.
  + We cannot have Java program without main method.
  + public static void main(String[] args) {
  + Code block
  + }

Access Modifies:

* Specify how to access classes and methods.
* Public Access Modifies
* Private Access Modifies
* Protected Access Modifies
* Default Access Modifies
* (Public, Private, Protected and Default)- Keywords of Access modifies.

Naming Conventions:

* How to write names in programming
* Pascal case conventions (used in naming Classes).
  + ThisIsPascalConvention
* Camel case conventions (used in naming Variables and methods).
  + thisIsCamelConvention
* Snake case conventions (used in naming Constants)
  + this\_is\_snake\_convention
* public class main {
  + public static void main(String[] args) {
  + code block
  + }
* }

Package – a container for classes.

Code block – is a code that is inside the braces {}.

Package -> Classes -> Method

Field:

* a single piece of data or attributes.
* represents a specific characteristic or property.

Object:

* a collection of related fields.
* Encapsulate data and behaviour (methods).
* Instance of a class.

Class:

* a blueprint or template for creating objects.
* Defines the structure and behaviour of an object
  + Class -> Vehicle (defines the properties and behaviour of a vehicle.)
  + Object -> MyCar (an instance of the vehicle class, with its own attributes and methods.)
  + Field -> Colour (a single attribute of the MyCar object)

Strings

* ­Groups of characters (text) put in double quotes “ “.
* Strings in Java should be put in double quotes.

Calling Println():

* “println” means print in a new line (ln).
* “ln” is to break into a new line.
* “print” is not going to break into a new line.

What is System.out?

* “out” is an object of the “PrintStream” Class.
* “out” has the “print()” and “println()” methods.
* “system” is a class (pascal case).
* “out” is inside system.
* “out” is a field of the “system” class.
  + System.out.println(“First Java Application”);
* “System” is a class
* “out” is an object that belongs to the “system” class.
* “println()” is a method. A method always followed by parenthesis ().
* “dot (.)” separate classes and methods and objects.
* “First Java Application” is the literal string that is the argument to the “println” method.

Comments:

* Text ignored by the compiler.
* To write notes.
* To find errors.

Public keyword:

* Is an access modifier, accessible everywhere
* Means it can specify how to access classes, methods and fields.
* Can be inside a class.
* Outside of a class.
* Inside of a package.
* Outside of a package.

Private keyword:

* Access level only inside a class.
* Only allowed us to access them in the same class.

Static keyword:

* Non-access modifier.
* You can access fields/ methods using the class name.
  + e.g. System.out
* “out” is a static field of “system”.
* “static” keyword allows us to call the method using a class name
* “static” allow to access methods and field using the class name.

The Void Return Type:

* Return Types – the type of data that a method returns or gives.
* “main()” has a void return type.
* void means nothing.
  + e.g. getUserName() , this method must return a surname.
  + e.g. printUserName(), this method has a void return.

Command Line Arguments:

* Data given to the main method.
  + e.g. public static void main (String[] args)
* “string [ ]”
* “args” each argument would be a string
  + e.g. System.out.println(arg[0])

Types of Errors:

* Syntax Error
* Runtime Error
* Logical Error
* Common Error

Overview of the Basics:

* Variables – a container that stores some data.
* It is called a variable because the value inside it can change.
* Data Types – Text, Numbers, Boolean, user defined types.
* Operators
  + Arithmetic relational operators (+, -, \*, /)
  + Logical operators (and, or)
* Conditions – used to control what to execute in our program.
  + e.g. if a condition is true -> do something
  + if it is not true -> do another thing.
* Loops – use to run some cod more than once.
* Types of Loops:
  + While Loop
  + Do while Loop
  + For Loop
  + For each Loop
* Methods – divide our code into smaller pieces.
  + Less code in “main()”.
  + Our program will be easier to maintain and debug.
  + Our program will be easier to understand.

Intro to Variables:

* Declaring a variable means allocating space inside our memory.
* We put a type of a variable.
* We put a name of the variable.
  + Type – String
  + Name – myName
  + String myName;
* We declared a variable called “myName” and can store a “String”.
* To declare multiple variables
  + Type – String
  + Name – myName1, myName2, … ;
  + String myName1, myName2;
* A variable must be declared before it can be used.

Assignment:

* Used to store or put a value inside a variable.
* We use assignment operator (=).
  + e.g. variableName = expression
* An expression is anything that produces or give a value.

Initialization:

* Assigning a value to a variable when declaring it.
  + e.g. String myJob = “Programmer”;
* Each variable has a unique name.
* When assigning a value to a variable to not define it again.
* Declare a variable:
  + String myJob;
* Assign and initialize the declare variable:
  + myJob = “Programmer”;
* Declare, assign and initialize the variable same time:
  + String myJob = “Programmer”;

Constants:

* A variable whose value cannot be changed.
* To define a constant, we use the “final” keyword.
* Constants names are written in upper case & use snake case convention.
* You get a syntax error trying to change the value of a constant.
* Initializing Constants:
* ­Final type = Value;
  + e.g. Final String company\_name = “Neso Academy”;
  + System.out.println(company\_name);
* Benefits of using constants:
  + Value will not be changed.
  + Cannot type same value multiple times.
  + Descriptive name makes program easy to read and understand.

Identifiers:

* Are names that identify the elements in a program.
  + e.g. names of classes, methods and variables.
* Naming Rules:
  + Every identifier must obey some rules.
  + Identifiers contains letters, digits, underscore and dollar sign.
  + Cannot create identifiers using keywords.
* Case Sensitivity:
  + numberOfStudents and numStuds are not the same.
  + userName and uName still are not the same.
* Do not use “&” character when naming identifiers.

Intro Data Types:

* Integers:
  + Numbers without a decimal part.
* Characters:
  + All characters on the keyboard and more.
  + Characters are surrounded with single quotes.
* Strings:
  + Group of characters (text).
* Booleans:
  + Represents True and False.
  + Used to create conditions.
* Byte, Short and Long Data types:
* Byte:
  + A type used with integers.
  + e.g. byte b1 = -128;
  + byte interval (-128 ; 127)
  + A number used that is lesser than the minimum and bigger than the maximum is an error.
* Short:
  + Short s1 = 5000;
  + Same as short.
* Long:
  + A letter “1” or “L” should be added to tell the compiler that a number is a long not an int.
  + e.g. long L1 = -1839;

Types of Conversion:

* byte -> short -> int -> long
* A “long” can store an int, short and byte (L1 = i1 + s1 + b1).
* An “int” can store a short and byte (I1 = s1 + b1).
* A “short” can store a byte (s1 = b1).
* A “byte” can only store a byte.

Bytes and Values

* bit is the smallest unit to measure memory.
* A byte is equal to 8bits.
* Our memory is divided into bytes.
* An int variable allocates 4bytes.
  + bytes: 1Byte = 8bits
  + short: 2Bytes = 16bits
  + int: 4Bytes = 32bits
  + long: 8Bytes = 64bits
* Which Data Type to Use
  + Do not waste memory.
  + Working with small numbers.
  + Do not use a long, use (byte, short, int).

Double and Float:

* Floating-point numbers:
  + Numbers that have a decimal part.
  + double and float.
* Double data type
  + A type used with floating-point numbers. 15 decimals.
  + double number, 4.9 \* 10 -324
  + Double takes 8Bytes in memory.
* Float data type
  + A type used with floating-point numbers. 6-7 decimals.
  + float number, use letter “F”, 1.4 \* 10 -45
  + Float takes 4Bytes in memory.

Boolean Data Type:

* A type used with Boolean/ logic values.
* boolean c;
  + true or false

String Data Type:

* String is a class
  + This class contains some static methods.
  + It starts with a capital letter and contain dot operator.
* String objects
  + Variable of type String, also have some methods.

Primitive Types and Reference Types:

* Primitive Types
  + Types that hold simple value.
  + e.g. byte, short, int, long
  + float, double, char, boolean.
* Reference Types
  + Types that hold complex values (Objects)
  + e.g. String
* The value references the variables (visa – versa).
* When dealing with primitive types of each variable is a box and inside this box we have value.
* Both variable and values are different.
* But with Reference type they both reference the same value.

Instantiating a String Object:

* Instantiating an Object
  + Creating an object of a class
  + e.g. ClassName objectName = new ClassName(Parameters);
  + “ClassName” is a class name.
  + “objectName” name of the object or variable.
  + “new” used to create objects.
  + “ClassName (Parameters)” a special method called Constructor.
* Instantiating a String Object
  + Creating a variable/object of type String, using new keyword.
  + If we do not use a new keyword our object will reference the same value.

String are Immutable (not changed):

* Immutable Objects
* Objects whose contents cannot be changed.
* Immutable objects are created from immutable classes.
  + String str = “Old Value”; the old value cannot be changed
  + str = “New Value”; with new value.
  + Object “str” will have to reference the “New Value”.
  + “Old Value” will be removed under Java Garbage Collector (JGC).
* Strings are immutable, no need to use “New” keyword.

The Scanner Class:

* We create an object “input” that reads data from the “System.in” keyboard.
  + Scanner input = new Scanner (System.in);
  + “Scanner” – Class
  + “input” – variable/object
  + “Scanner (System.in)” – Constructor
  + “(System.in)" – Parameters
* Input methods
  + Methods used to read specific types of data from the keyboard
  + e.g. Scanner sc = new Scanner (System.in);
  + System.out.print(“Enter your name and age: “);
  + System.out.println(sc.nextLine() + “ ! You are “ + sc.nextDouble() + “ years old!”);
  + Run = Output

Increment and Decrement Operators:

* Increment Operator
* Used to increase the value of a variable by 1.
* Increment operator (++) “++i”;
* Can be placed before the variable (pre-increment) ++i
* Can be placed after the variable (post-increment) i++
* Post-Increment:
  + e.g. int i = 4;
  + i++;
  + System.out.println(i); = 5
* Pre-Increment:
  + e.g. int i = 4;
  + ++i;
  + System.out.println(i); = 5
* “++i” increments the value of “i” by 1 and uses the new value in the statement.
* “i++” increments the value of “i” and uses the original value in the statement.
* Decrement Operator
* Used to decrease the value of variable by 1.
* Decrement Operator (--) “--i";
* Can be placed before the variable (pre-decrement) --i
* Can be placed before the variable (post-decrement) i--
* post-decrement:
  + e.g. int i = 4;
  + i--;
  + System.out.println(i); = 3
* pre-decrement:
  + e.g. int i = 4;
  + --i;
  + System.out.println(i); = 3
* “--i” increments the value of “i” by 1 and uses the new value in the statement.
* “i--” increments the value of “i” and uses the original value in the statement.

Casting

* Converting a data type to another type.
* Implicit casting: Happens automatically when converting from a narrower range data type to wider range data type.
  + e.g. converting an int to a double, float and long.
  + Converting a float to a double.
* Explicit casting: Does not happen automatically
* Should be done by the programmer when converting from a wider to a narrower data type.
  + e.g. int i1 = (int) 4.5;
  + int i2 = (int) 8L;
  + float = (float) 4.5;
* The programmer tells Java to do the casting:
  + (int) 4.5 -> 4 (data loss).

Relational Operator:

* Operator used to make expressions:
  + Equality Operator (==) Test if two expressions are equal.
  + Inequality Operator (! =) Test if two expressions are not equal.
  + Greater than Operator (>) Test if an expression is greater than the other.
  + Greater or equal to Operator (>=) Test if an expression is greater or equal to the other.
  + Less than Operator (<) Test if an expression is less than the other.
  + Less or equal to Operator (<=) Test if an expression is less or equal to the other.
* NB. All these operator produces a Boolean value when used.

Logical Operator:

* Operators used to construct complex conditions:
  + Logical AND Operator (&&) Returns True if both conditions are True.
  + Logical OR Operator (||) Return True if one statement is True.
  + Logical NOT Operator (!) Reverse result, return False if result is True.

Conditional/Ternary Operator:

* An operator that evaluates an expression based on a condition:
  + e.g. Boolean expression? expression1: expression2
  + The expression will evaluate expression1 if boolean-expression is true
  + The expression will evaluate expression2 if boolean-expression is false
  + e.g. int a = 10;
  + int b = 15;
  + int max = a > b ? a : b;
  + if “a” > “b” then ? max is “a”.
  + if “a”! = then: max is “b”

If Statement:

* Used to execute a piece of code based on a condition.
  + if (boolean-expression)
    - statement
  + More than one
  + if (boolean-expression){
    - statements
  + }
* The statement(s) are executed if the boolean-expression value true.
* If the boolean-expression evaluates to false, the statement(s) will be skipped.

If-Else Statement:

* An optional block of an if statement
  + if (boolean-expression)
    - statement
  + else
    - statement
  + More than one
  + if (boolean-expression) {
    - statement
  + } else {
    - statement
  + }
* The statement(s) in the “else” statement are executed only if the boolean-expression evaluates to false.

Nested if-else Statement:

* Using if-else statements inside if-else statements
  + if (boolean-expression) {
    - statement
  + } else {
    - Statement
    - if (boolean-expression) {
      * statement
    - } else {
      * Statement
    - }

Switch Statement:

* Used to execute different cases based on equality.
  + switch (expression) {
    - case expression\_1:
      * + statements;
  + }
* Default used if the condition does not apply to all the cases, then default will be executed.
* Break causes program flow to exit from the entire switch statement and resume at the next statement outside the switch.

Switch vs. if-else statement:

* Using if-else statement only one block will be executed.
* Using switch statement more than one case may be executed.
* Break keyword:
  + int n = sc.nextInt();
  + switch (n % 2) {
    - case 0:
      * System.out.println(“even”);
      * break;
    - default:
      * System.out.println(“odd”);
  + }

The While Loop:

* Loops are used to execute a block of code more than once.
  + int i = 1; (start at 1)
  + while (i <= 5) (stop at 5)
  + System.out.println(“Hello”);
  + i++; (we are changing the value of i)
  + NB. This code will print “Hello” 5 times as long as (i <= 5); print “Hello”
* If you want the number + “Hello” to keep adding like “Hello 1”, you should concatenate (“Hello ” + i ) ;
  + while (boolean-expression)
    - statement
  + More than once.
  + while (boolean-expression) {
    - statement(s)
  + }
* The statement(s) will keep on executing as long as the boolean-expression evaluate to true.
* Each execution is called an iteration.

Infinite Loops:

* The loop that does not end.
* Do this code while (boolean-expression) is true.
* The do-while loop.
  + do
    - statement
    - while (boolean-expression);
  + More than once
  + do {
    - statement(s)
    - while (boolean-expression);
  + }

While vs. Do-While:

* While: check condition then execute.
* Do while: execute then check condition.

The for-loop:

* The statement(s) will keep on executing as long as the condition evaluates to true.
  + for (variable; condition; change)
    - Statement
  + More than once
  + for (variable: condition; change) {
    - statement(s)
  + }

For-loop vs. While-loop:

* While loop
  + int i = 1;
  + while (i <= 5) {
    - statement(s)
    - i++;
* For loop
  + for (int i = 1; i <= 5; i++) {
    - statement(s);
  + }
* Both loops are doing the same thing.
* The loop starts from the declared variable “int i = 1”.
* Then check the condition “i <= 5”.
* If the condition is true, we execute the statement.
* When the body of the loop “statement(s)” finish executing.
* Then make change by incrementing “i++”.
* Key:
  + When we declare the variable inside the loop we cannot use outside it.
  + The variable can be declared outside and set inside the loop. Means that it can be accessed outside the loop.
  + The variable can be initialized outside.
  + NB. “i++” and “++i” are the same in the for-loop.

Infinite for-loop:

* It is an infinite for-loop because the condition is true or empty.
  + for ( ; ; )
    - statement(s)
  + for ( ; true; )
    - statement(s)

For-loop just like a while loop:

* Use a for-loop just like a while loop
  + int i = 1;
  + for ( ; i <= 5; ) {
    - statement(s);
    - i++;
  + }
  + int i = 1;
  + while (i <= 5) {
    - statement(s);
    - i++;
  + }

Nested loop:

* A loop can be nested inside another loop
* Each time the outer loop is executed, the inner loops are executed again.
  + for (int i = 0; i < 3; i++) { Rows
    - for (int j = 0; j < 2; j++) { Columns
      * statement(s)
    - }
    - statement(s)
  + }
* Each time we execute the outer loop we are re-executing the inner loop from the beginning.
* And after we finish executing the inner loop, we continue executing the outer loop.
* The outer loop is responsible for the rows and inner loop is responsible for the columns/values.

Break and Continue:

* Special keywords that can be used within loops
* Break: exists the loop regardless of the condition.
* Continue: skips the rest of the loop’s body. Continue executing the loop as if we finished executing the body.
  + int num;
  + while (true) {
    - System.out.println("Enter number between 1 and 10: ");
    - num = sc.nextInt();
    - if (num < 1 || num > 10)
      * continue;
    - break;
  + }
  + System.out.println(num + " is between 1 and 10.");

Scope and Local Variables:

* Scope of a variable is the part of the program where the variable can be referenced or used.
* A variable defined inside a method is called a local variable.
* A scope of local variable starts from its declaration and continues to the end of the block that contains the variable.
* A local variable must be declared and assigned a value before it can be used.
* Parameters are also local variables; their scope is the whole method.
  + public static void main(String[] args)
  + “String[] args” is a local variable that is only used in the main method.
  + “args” is a variable that is a local variable to the main method.

Methods:

* Void – is a method that does not return a value.
  + The return type is void.
* Value Returning Method – a method that returns a value.
  + The return type is the type of a data that will be returned.
    - public static int sum(int x, int y)
      * return int;
* The return keyword – is a keyword that can be used to return a value from a method.
  + When the return statement is executed, we will exit the method.

Passing Arguments by Value:

* Parameters and arguments
* Arguments are actual value(s) that are given to a method.
  + public static void main (String[] arg) {
    - sum (1; 2)