First Java Program, i/o , datatypes and debugging

Main file (Main.java)

*class Main {*

*}*

Note – classes are written in capital letters according to the programming conventions

Public keyword

Generally classes are closed in nature ie it is not accessible to other classes, packages etc

And as the name suggests public keyword makes the class public to other files, classes etc

*public class Main {  
}*

Function – collection of code which can be used again and again

**Writing the first Code**

*public class Main {  
public static void main (String[], args){  
system.out.println(“Hello” World!”);  
}  
  
} //syntax of this program to be explained later*

***.java to .class****javac Main.java -🡪 Main.class( Byte Code)*

***Running the .class file****java Main.class -🡪 Hello World! (Output)*

***Explanation of syntax used in our first code***

***public*** *(in Main class )*- to make the class accessible to other classes, files etc

***main****(in function /method inside main class)*- like c++, Java require the main keyword to execute its “main” or the base function

***public****( in the main function/method)*

-to make the function accessible to other functions as well

***Static***

* Used to use variables , functions etc inside a class without actually creating an instance of an object

***Void***

* *Not expecting the function to return anything*

***String[], args***

* *Array in the command line interface*

***Changing location of the .class file***

*In git bash*

*(cd – your folder location )*

***Command - “*** *javac -d location Main.java*

*Note : - two dotes(period) means previous directory*

***Environment Variables / Path Variables***

*To access executable files in java which is only possible by specifying a path for them*

***Package (eg com.company)***

*Packages generally restricts the file content from alien files or programs*

*The period (“.”) means the subfolder of the specified file /package*

***Sout explanation***

***System class***

*Contains several useful class fields and methods*

*(Watch at 35:23)*

***Out***

*Part of the system class*

***println***

a method inside PrintStream

prints the input with a new line

**Taking inputs**

**Scanner class**

A simple text scanner which can parse primitive datatypes and strings using regular expressions

**import** java.util.Scanner

**code**

Scanner input = new Scanner(System.in)

//here first scanner is the return type of the variable

// input is the variable name

// new keyword creates instance of scanner for input variable

//System.in is the parameter for the constructor function, System.in mean the keyboard

input.next()

//take the first string before occurrence of space

input.nextline()

//takes the whole typed string and returns the same

***Primitives (47:07)***

*Primitives are data types you can’t break further*

***In java strings is not a primitive data type because It can be break into charracters***

*main types are*

*1)Int*

*2)char*

*3)float*

*4)long*

*5)double*

*6)Boolean*

*//Decimal values are of type double by default that’s why you have to specify “f” in float*

*//int values are by default added thus while declaring long you have to specify “L”*

***Wrapper classes***

*Classes of primitive data types for inheriting various methods in primitive classes as well*

*//more in OOP*

***Int input programs***

***System()*** *corresponds to keyboard inputs*

***Literals and Identifiers***

***`int a = 10***

***//****here a is the identifier*

*// 10 is the literal*

*// int*

***Writing as a currency***

*Int a = 25\_000\_000; ( underscores will be ignored)*

***String input***

*String name = input.nextLine() // Scanner* input = Scanner(System.in)

*System.out.println(name)*

***Sum of two numbers***

*you can typecast int to float if return type of the variable is float but you can’t convert float to int if the return type is int*

***IMP:- Type conversion / Type Casting***

*There could be cases when a type of a literal of variable may not be compatible with the return type of same variable , if in some cases the data types are compatible with each other than an automatic type conversion will happen other wise we have to convert the type of the variable explicitly*

*Simply explained the return type should be greater than the source type /actual data type*

***// eg:- Float num = input.nextInt()***

***//int :- 4 bytes, float:-6 bytes***

***Type casting***

*The process of explicitly converting /compressing a bigger data type into smaller data types*

*//****eg:- int num = (int)(5.6)***

***Automatic Type Promotion in expression***

*Process of automatically promoting smaller data types into a larger data type to effectively perform an expression*

***//eg:- byte malformedByteType = (byte) ( 257)  
// sout(malformedByteType) // result would be 1 257%256***

**//eg** byte numOne = 100;  
byte numTwo = 100;  
byte numThree = 4;  
int result = (numOne \* numTwo)/numThree;  
System.*out*.println(result);

//output would be 2500 i.e. a result of a valid logic/expression

It will transform a char into ASCII/Unicode as well

//**int a = “a” //result :-65**

**Rules of type promotion**

1. Char, byte, short would be promoted to bigger data type (int)
2. If in the expression there is a presence of long, double etc data types then it would be converted to that datatype respectively

**//eg:- int a = 10  
// char b = “a”**

**// int result = a\*b**

**//System.out.println(result)**

**// output would be an int**

**Loops**

**If , while**

**//** left at 1 hour 29 minutes