JAVA

→ First program gets compiled and then Interpreted

i) DATATYPES

→ There are two kinds of datatypes =
i) Primitive = which are already present in the library of program. Can say a reserved
→ It is further divided into three parts =
Integrals = byte, short, int, float, double, long.
Character = char =
Boolean = bool
Note =
→ Here l/L should be specified after long's constant value else compiler will save that value as Integer type.
→ Similarly with float i.e. F/f because default int double is default data-type for decimal value and if not specified the same thing will get stored as double in backend.

ii) LITERALS =
→ The phenomina of giving constant values to the variables.

→ Number function

iii) KEYWORDS = These are already reserved w
in the programming languag
and the compiler.
→ Cannot be used as an Identifier.

iv) OPERATORS = They are various kind of
operators =
i) Arithmetic Operator = Performs various
complex and simple mathematical calcula
such as = +, -, *, /, %

ii) Logical Operator = Performs various logi
~~comparisions~~ termini
logics such as =
i) && (AND - OPERATOR) = If both values
correct then only
will give boolean value true else if
one condition among multiple is false
then that particular condition will dom
nate

ii) || (OR - Operator) = Atleast one of th
multiple conditions
must be ~~true~~ correct to get the boolean
value true

iii) Comparison Operator = Compares betw
two or mo
credintials such as = (==), (<=),
(>=), (<), (>)

iv) Bitwise Operator = Smallest display
by 2, 1.

Format of Java Program =

```
package com.company;
```

package name according to system

```
public class class_name
{
    public static void main (String[] args)
    {
        // body
    }
}
```

File-Name saved

Note: For printing something
System.out.print(" ");
is used.
→ But to display it in another line
System.out.println(" "); is used.
→ The shortcut is
sout + ↵

PRECEDENCIES AND ASSOCIATIVITY =

Precedencies = The ranking of the Operators to be used in the order in the program.

Associativity = The Execution of the operators

→ Member function

according to its precidence.

- Note = The higher the precidence, the earlier will its execution
→ The precidence of * is greater than that of + and -.
→ The precidence of * and / are same then the execution will start from left to right

Datatypes =
→ According to the values, the datatypes assumed in the compiler will be done.

→ Integer datatype will give Integer Value
→ Short + Integer = Integer Value
→ byte + short = Integer
→ long + float = float
→ Integer + float = float
→ Character + Integer = Integer
→ float + double = double
→ long + double = double
∴ long + Integer = long

→ Java is not as flexible as C++ or Other Object - Oriented Programming.

INCREMENT AND DECREMENT OPERATORS =
→ I++; ← First I will get printed and then gets Incremented
→ ++I; ← First I will get Incremented and then gets Executed

Page No.:
Date: / /

Ex =

B = I ++

→ Here, first Value of I is Assigned to B and then Incremented.

~~int B~~

B = ++I;

→ Here, first Value of I is Incremented then assigned to B.

Note = For Character type values calso the Rules are Same.

STRINGS =

→ The combination of characters.
→ It has special support by java.

Can be declared in two types =

i) String String_name = " ";
ii) ~~String~~ ~~Scanner~~ String String_name = new String ("

To display strings there are two types

i) Scanner Object_Name = new Scanner (Syste
String String_name = Object_name.next()
System.out.println (String_Name);

Here If Input is : MY NAME IS HARSHIDA
Output Outcome = MY

→ Member function

Hence to display full line =
The another method is
Scanner Object_Name = new Scanner (System.
String String_Name = Object_Name.nextLine();
System.out.println(String_Name);

If the Input Lines = MY NAME HARSHIDA.
Output Lines = MY NAME HARSHIDA

STRING METHODS :
There are various Methods in Strings to display and Identify various factors.

~~St~~ Length = Displays the length of the String including spaces.
Syntax = String String_Name = new String (" ");
~~String n~~ ~~int~~
String Value = String_Name.length();
System.out.println (Value);
Or
String String_Name = new String_Name (" ");
System.out.println (String_Name.length());

ii) toLowerCase = Changes all character and Elements of string to lower-case.
Syntax = String String_Name = new String (" ");
System.out.println (String_Name.toLowerCase());

) toUpperCase = Changes all Elements of the string to the Uppercase.

Syntax = String string-Name = new String (" ");
System.out.println (String-Name.toUppercase());

) Trim = Removes the Unwanted space of the Elements of the string.

Syntax = Scanner Object-Name = new Scanner (System.in);
String String-Name = Harshida.nextline (" ");
String Value1 = Object-Name.Trim();
System.out.println (Value);

Or

String String-Name = new String (" ");
System.out.println (String-Name.Trim());

) substring (int A) = Displays values from the Index from the starting, Includes it the values from starting

Syntax = String String-Name = new String (" ");
System.out.println (String-Name.substring (int A));

Ex = String Name = new String ("HARSHITA");
System.out.println (Name.substring (3));

↳ Number function

Output = RSHIDA.

vii) substring (int A, int B) = Includes the Utmost of the starting Index and Excludes the End

Syntax = String String_Name = new String (" ");
System.out.println (String_Name.substring(2, 7));

Ex = String Name = new String ("HARSHIDA");
System.out.println (Name.substring(2,

Output = RSHI.

viii) charAt = It represents character at the particular Index

Syntax = String String_Name = " ";
System.out.println (String_Name.charAt(int));

Ex = String String_Name = "Harshida";
System.out.println (String_Name.charAt(3));

Output = S

ix) indexOf = Represents the Index of particular Element of the string

Syntax = string String_Name = " ";
System.out.println (String_Name.indexOf("

Ex = String Name = "HARSHIDA";
System.out.println (Name.indexOF("A"));

Output = 1
Note = Represents the Index of first positioned Element

x) replaces = Replaces Element or particular bunch of Elements in the string with another specified terms

Syntax = String String_Name = " ";
System.out.println (String_Name.replace(" ", " "));

Ex = String Name = "Harshida";
System.out.println (Name.replace ("a", "I"));

Output = HIrshidI

x) startsWith = It gives the boolean value that either that particular string is starting with that character or not

Syntax = String String_Name = new String (" ");
System.out.println (String_Name.startsWith(" "));

Ex = String Name = "Harshida".
System.out.println (Name.startsWith ("H"));
↳ Number function

Output = False

Ex = String Name = "Harshida";
System.out.println(Name1.startsWith("H"));

Output = True

xii) endsWith = Displays the boolean value that either the string is ending with that particular value or not.

Syntax = String string_Name = new String(" ");
System.out.println(string_Name.endsWith(" "));

Ex
Output = String Name = "Harshido";
System.out.println(Name.endsWith("o"));

Output = true

xiii) equals = Displays the boolean value that either the given condition is equal to string or not. Case Sensitive.

Syntax = String String_Name = " ";
System.out.println(String_Name.equals(Condition

Ex = String Name = "harshida";
System.out.println(Name.equals("HARSHIDA"));

Output = FALSE (Case Sensitive).

(11)

xiii) equalsIgnoreCase = displays boolean value for whether the given condition is equal to the string or not. (Not Case Sensitive)

Syntax = String String-Name = " ";
System.out.println(String-Name.equalsIgnoreCase());

Ex = String Name = "HARSHIDA";
System.out.println(Name.equalsIgnoreCase("HARSHI

Output = True (Not Case-Sensitive)

~~Loops~~ = LOOPS =

→ Thus are used to represent the repetitive output till the boolean value of the program turns to be false.

There are various kinds of loop =

i) while = This first checks the condition and then gets executed.
→ Gets executed till the condition becomes false.

Syntax = while (condition)
{
// body to be executed.
}

→ Member function

(2)

Ex = Scanner Harshida = new Scanner (System.in);
int N;
System.out.print("Enter the value of N=");
int N = Harshida.nextInt();
while (N>0)
{
System.out.println(N);
N--;

Input = Output = Enter the value of N = 10

10
9
8
7
6
5
4
3
2
1

ii) do-while loop = It first executes the statement and then checks the condition.
→ The loop gets executed atleast Once.

Syntax = do
{
// body;
}
while (condition);

Ex = Scanner Harshida = new Scanner (System.in);
System.out.println("Enter the Number =");

```
int N = Harshida.nextInt();
do
{
System.out.println("Hello World!");
N--;
}
while (N>0);
```

Output = Enter the Value of N = 2
Hello World! ← 2
Hello World! ← 1
Hello World! ← 0
} Indices

iii) FOR LOOP = This is similar to while and do-while loops.

Syntax = For (int I = condition1;
I (RELATIONAL_OPERATOR) COMPARISION_OPERATOR) CONDITION2;
I (ARITHMETIC_OPERATOR))
{
// body of the PROGRAM.
}

Ex =
```
int [] ARR = {1, 2, 3};
For (int I=0 ; I < ARR.length;
I++)
{
System.out.println(ARR[I]);
}
```

Output =
1
2
3

→ Member function

iv) FOR-EACH LOOP = Similar to for loop but represent all items of all the Indices.

Syntax = FOR (int ELEMENT : CONDITION)
{
System.out // body of the PROGRAM
}

Ex = int [ ] ARRAY = { 10, 20, 30, 40 } ;
FOR (int ELEMENT : ARRAY)
{
System.out.println(ELEMENT);
}

Output = 10
20
30
40

* ARRAYS :
⇒ Arrays are used to store the Elements of similar datatypes.

There are two kinds of Arrays :
i) One-dimensional Arrays.
ii) Multi-dimensional Arrays.

There are various methods to declare Arrays

i) Initializing and Memory Allocation Separately
Ex : int [

ii) Initializing + Memory Allocation.

Syntax = datatype [ ] Array_name;
Array_name = { };

iii) Initializing + Declaration

Syntax = ~~int [ ]~~
datatype [ ] ARRAY_NAME = { };

→ One - dimensional Array =
Single Arrays

Ex = int [ ] ARR = {1, 2, 3, 4, 5};

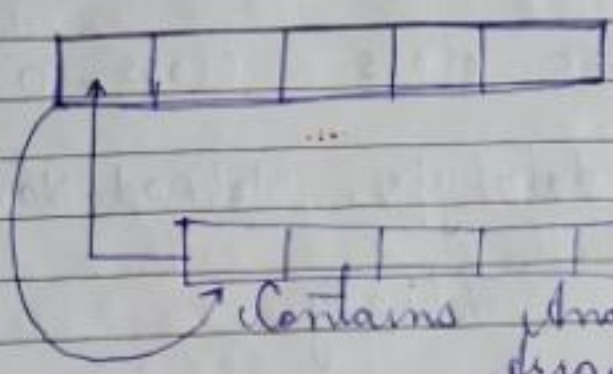
☞ Note = In the third case, memory Allocation is by Itself.

→ Two - dimensional Array =
Arrays Under Array.

→ Member function

Syntax = datatype [][] ARRAY-NAME = {{ }, { }};

Ex =



← Contiguous Array's Memory blocks

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

* JAVA METHODS =
→ Java methods is similar as functions
→ Used to perform specific task on restriction to the repitition of logics
→ Declared inside the class of the program
Can be declared by two types =

i) Statically =

```
public class class_name
  {
   static datatype Method-name (Parameters)
     {
       // body;
     }
  public static void main (String[] arg
  {
     int c;
     c = Method-Name (Parameters);
  } System.out.println (c);
```

ii) If not statically, then by the declaration of objects.

```
public class  class_name
  {
    datatype  Method-Name (Parameter)
      {
        //body;
      }
public static void main (String [ ] args)
  {
     class_name  object_name =
                          new class_name
     object_name. Method-Name [ ] { };
  }
```

Ex =

```
public class JAVA_METHODS
  {
    int METHODS (int A, int B)
    {
      int z;
      IF (A > B)
        {
          z = (A+B) * 5;
        }
      else
        {
          z = (A+B) / 5;
        }
      return z;
    }
```

→ Member function

public static void main(String[] args)
{ int x = 10, y = 20; obj
JAVA-METHODS METHODS = new
JAVA-METHODS();
obj.METHODS(x1, x2);
System.out.println(obj.METHODS(X,Y));
}

METHOD OVERLOADING =
→ The methods of same name but performing different tasks

Note = In method Overloading the datatypes of methods of same name performing different tasks cannot be varied

Syntax = datatype Method-Name (Parameters)
{
// body
}

datatype Method-Name (Parameter1, Parameter2)
{
//body
}

:
:

public static void main(String[] args)
{
METHOD_NAME ('var').
METHOD-NAME (Var1, Var2).
}

Note = The variables, method name and functionality allocated to each is identified by the compiler itself.

* VARARGS =
→ To reduce the coding time in declaring, initialising and calling the elements and its values, Varargs are used.

Syntax = datatype Method-Name (int ... ARR)
{
//body;
}
public static void main(String[] args)
{
Class_Name object_name = new Class_Name();
object_name.Method-Name (Values);
}

For Giving atleast one value to the Varargs following syntax is used:-

static
datatype ~~metho~~ METHOD-NAME (int A, int ... ARR)
{
//body;
}
public static void main (String[] args)
{
METHOD_NAME (val1, val2, val3, ---); Always until in there

→ Member function

* JAVA RECURSION :
→ In Java the function calling itself, is termed as Java Recursion

Syntax = static datatype method-name (Parameters)
{
// body with Functionallity;
return-type if required;
}

public static void main(String[] a
{
method-name (Values);
}

Ex = static int ADD(int A, int B)
{
int C;
C = A+B;
System.out.println("The Sum of Two Numbers Is =" + c);
return c;
}
public static void main(String[] arg
{
ADD(Val1, Val2);
}

IMPORTANT QUESTIONS :-

1. MATRIX

```
public static void main (String[ ] args)
{
int [ ][ ] MATRIX1 = {{1,2,3},{4,5,6}};
int [ ][ ] MATRIX2 = {{7,8,9},{10,11,12}};
int [ ][ ] SUM = {{0,0,0},{0,0,0}};
for (int I = 0; I <= MATRIX1.length; I++)
{for (int J = 0; J <= MATRIX1[I].length; J++)
{
System.out.printF ("The location
OF I And J IS %d
And %d", I, J);
SUM[I][J] = MATRIX1[I][J] + MATRIX2
[I][J];
}
}
FOR (int I=0; I <= MATRIX1.length; I++)
{
FOR (int J=0, J <= MATRIX1[I].length;
J++)
{
System.out.println (SUM[I][J] + " ");
SUM[I][J] = MATRIX1[I][J] +
MATRIX2[I][J];
}
}
}
```

→ Member function

2. MAXIMUM / MINIMUM OF NUMBERS IN ARRAY.

```
public static void main (String[ ] args)
 {
    int [ ] ARR = {21,22,23};
    int max = Integer.MIN_NUMBER;
    for (int I=0; I<=ARR.length; I++)
      {
        if (ARR[I] > MAX)
          {
            ~~cout<<~~ max = ARR [I];
          }
      }
    System.out.println("The Maximum
                        Number Is :" +
                                max);
 }
```

3.
```
*
* *
* * *
```

```
public static void main
                (String [ ] args) {
    {
    Scanner HARSHIDA = new Scanner
                        (System.in)
    System.out.println ("Enter Num
                        of Conditions :");
    int CONDITIONS = HARSHIDA.nextIn
                                ();
For (int I=0; I<=CONDITIONS; I++)
    {
    System.out.println(" ");
      for (int J=0; J<=I; J++);
    ) { System.out.print ("* ");
```

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

SORTING OF ARRAY :-

```
public static void main (String [ ] args)
{
Scanner HARSHIDA = new Scanner (System.
boolean isSorted = false;
int [ ] ARR = { 21, 22, 23 };
For (int I = 0; I < ARR.length; I++)
{
  IF ((ARR[I]+1) > ARR[I])
    {
      isSorted = true;
      break;
    }
}
IF (isSorted)
{
  System.out.println ("The Array
          is Sorted In Descending
    Order.");
}
else
{
  System.out.println ("The Array
          Not Sorted.")
}
}
```

→ Number function

5.

```
* * * *
* * *
* *
*
```

```
public ~~etc~~ static void main
                (String [ ] args)
{
Scanner HARSHIDA = new Scanner
                (System.in));
System.out.println("Enter the number
                OF conditions =");
int CONDITIONS = HARSHIDA.nextInt();
for ~~(int I=0; I<= CONDITIONS~~
        (int I : CONDITIONS ; I >=0 ; I--)
{ for (int j=0 ; j<= I ; j++)
    {
    System.out.print ("*");
    }
}
System.out.println ();
}
```

OOP's

→ Object - Oriented Programming is the connection of Computer programming to the real world.

* Properties of Object - Oriented Programming :

i) Class = Class is User - defined and contains objects of similar properties inside it
→ Blueprint of Object

ii) Encapsulation = Wrinding up of Information and functions into single unit.

iii) Abstraction = Hiding data and Information from public
→ Showing only Important data and Information

iii) Inheritance = One class assuming it's properties and characteristics from another class.

→ The class from which properties and characteristics are assumed is termed as super class or base class or parent class.

→ Member function

→ The class which drives it properties and characteristics from another class is turned as sub-class or child class or driven class.

* CREATING CLASS IN JAVA =

```
class   class_name
{
    //body;
}
public  class  Main
{
    public static void main (String[ ] args)
    {
        class_name  object_name = new class_name();
    }
}
```

* Access - Specifiers In Java =

i) Private = The Information are not accessible outside the class

ii) Default = By default, if the access-specifier is not mentioned then it is of default type

iii) Public = These Information are accessible and called outside the class as well.

iv) Protected = This gives security to the Information and data within the class as well.

Ex 1

```
class SECTION
  {
    int COST; // DEFAULT ACCESS-SPEC
    public void setCOST (int A)
      {
        COST = A;
      }
    public int getCOST ( )
      {
        return COST;
      }
  }
public class Main
  {
    public static void main (String[] args)
    {
      SECTION ABC = new SECTION ( );
      ABC.setCOST (100);
      System.out.println ("THE COST
          OF ENTERING IS =" +
          ABC.getCOST ( ) + "Rs.");
    }
  }
```

Output = THE COST OF ENTERING IS = 100Rs.

Ex 2 =

```
class SECTION
{
  private int COST;
  public void setCOST (int cost)
    {
      this.COST = COST;
    }
```

→ Member function

```
}
public int getCOST()
  {
    System.out.println("COST IS=");
      return this.COST;
  }
}
public class Main
  {
    public static void main(String[]
                                args)
    {
      SECTION ABC = new SECTION();
      ABC.setCOST(100);
      System.out.println("COST IS =" +
                          ABC.getCOST());
    }
}
```

Output = COST IS = 100

Note : i) There is only one public class in the whole program that is the name of java class file

* METHODS EXTENDED =

i)
```
public class Main
{
    int Foo(int A)
      {
        return A;
      }
    public static void main(String[]args)
```

```
    {
        Main ABC = new ABC(); // Object calling
        ABC.Foo(100)
        System.out.println("ABC.Foo(100));
    }
}
```

Output = 100

ii)
```
public class Main
    {
        static int Foo(int A)
        {
            return A;
        }
        public static void main (String[] args)
        {
            System.out.println(Foo(100));
        }
    }
```

Output = 100

→ In the given above two examples there are two differences =
i) Presence of static keyword as Initialization of methods.
ii) Creation and Initialization of the object.

→ Member function

Explanation = This is Lucara =

a) Static keyword is applicable for all the objects Hence, Initialization of object is not necessary

b) If the method is not Initialized by the static keyword then pointing i.e., Initialisation of objects is necessary for which the method has been passed.

Syntax = public class Main
{
Static or not ⟶ datatype identifier (Parameter)
{
Without static keyword = // body;
}
public static void main
(String[] args)
{
Main Object_Initialization =
new Main();
object_Initialization. identifier
(value)
}
}

Syntax = public class Main
{
With static keyword = static datatype Identifier (Parameter)
{ //body;
}
public static void main(String[] args)
{ Identifier (Value);

INHERITANCE :-

→ Inheritance is a mechanism in which one or more class derives its properties and characteristics from another class. This phenomena is termed as Inheritance.

1 → Sub-class / Child-class / Derived-class :-

→ The class which derives its properties and characteristics from another class/ classes.

2 → Super-class / Parent-class / Base-class :-

→ The class/ classes from which properties and characteristics of another class/ classes are derived.

Note :- Supports phenomena of Reusability of code.

Syntax :-

```
class class-name //base class
{
  // body;
}
class class-name1 extends class-name
// Here class name1 is derived class Inheriting all credintials and properties from base class (class-name)
{ // body;
}
```

→ Member function

```
public class Main()
  {
    public static void main (String[] args)
      {
        class-name1 obj = new class-name1;
                                  // derived class object Initialisation
to access credintials of base class as well
      }
  }
```

```
Ex = class SECTION // Base Class
       {
         int x;
         public void setSECTION(int x1)
         {
           x = x1;
         }
         public int getSECTION()
           {
             return x;
           }
       }
class NAME extends SECTION // derived name is derived class and section is base class.
{
  int y;
public void
    setY(int y1)
  { y = y1;
  }
```

```
    public void getY()
      {
        return Y;
      }
  }
public class Main
  {
    public static void main (String[]
                                args)
    {
      NAME ABC = new NAME();
      ABC.setSECTION(1);
      System.out.println(ABC.getSECTI-
                                ON());
      ABC.setY(2)
      System.out.println(ABC.getY());
    }
}
```

ii)
```
public class Main
  { public static void main (String[]
                                args)
    {
      SECTION ABC = new SECTION();
      ABC.setSECTION(3);
      System.out.println(ABC.getSECTION());
      ABC.setY(4);
      System.out.println(ABC.getY());
    }
}
```

→ Member function

→ Here are two parts
a) Object Initialization by derived / child / sub class :

Output = 1
2

→ because base / derived / sub class can Inherit characteristics of parent / base / sub class.
b) Object Initialization by base / parent / super class :

Output : Error
→ because, parent / base / super class cannot Inherit characteristics of child / derived / sub class.

* ~~METHOD~~ INHERITANCE WITH CONSTRUCTOR AND ~~OVERRIDE~~ OVERLOADING

Ex :

```
class SECTION
{
    public void ALLOTED (String NAME)
    {
        System.out.println ("The Alloted section =" + NAME);
    }
    SECTION ( ) // CONSTRUCTOR WITHOUT PARAMETER AND ARGUMENT
    {
        System.out.println ("I AM CONSTRUCTOR OF BASE CLASS.");
    }
```

```
SECTION (int A) // CONSTRUCTOR WITH PARAMETERS AND ARGUMENTS
{
  System.out.println
        ("This is over-loaded
        constructor of base class
with value =" + A);
  }
}

class UNIVERSITY EXTENDS SECTION
                // Here, University is
                derived class deriving
                its properties from base
                class "section".
{  @Override
   UNIVERSITY (int A) // Constructor without
   {                     Arguments
     SUPER (A);
     System.out.println ("This is
                constructor of
                derieved class");

   }
}
public class Main
   {
     public static void main (String[]
                              args)
      {
        UNIVERSITY ABC = new UNIVERSITY();
        ABC.UNIVERSITY (10);
      }
   }
```

→ Member function

In the above program :-

i) The derived class constructor is argument free but then also an argument of Integer type has passed, that is the Inheritance of parameter from base class.

ii) Super keyword = This keyword is used when there is Inheritance of Arguments from base class constructor to derived class constructor for giving values.

Syntax = Derieved-class-name (Argument OF Base Class)
{
Super (Identifier-OF-Argument OF-Bose-class);
}

iii) Override keyword = This is placed just above the Constructor of derived class to check wether the properties are overloaded or not.

→ Used without semi-colon (;), and with @ (at-the-rate of sign).

→ Ex = @ Override.