Unit 1

Q] Features of Object-oriented Programming :-

Features of OOP :-

1) Object
2) Class
3) Data Hiding and Encapsulation
4) Dynamic Binding
5) Message Passing
6) Inheritance
7) Polymorphism

1) Object :- Object is a collection of number of entities. Objects take up space in the memory. Objects are instances of classes. When a program is executed, the objects interact by sending messages to one another. Each object contain data and code to manipulate the data. Objects can interact without having to know details of each others' data or code.

2) Class :- Class is a collection of objects of similar type. Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to

that class. Eg:- grapes, bananas and oranges are the members of class fruit.

Example:-

fruit orange;

In the above statement object mango is created which belongs to the class fruit.

Note :- Classes are user define data types.

3) Data Abstraction and encapsulation:- Combining data and functions into a single unit class and the process is known as encapsulation. Data encapsulation is important feature of a class. Class contains both data and functions. Data is not accessible from the outside world and only those functions which are present in the class can access the data. The insulation of the data from direct access by the program is called data hiding or information hiding. Hiding the complexity of program is called Abstraction and only essential features are represented. In short we

can say that internal working is hidden.

4) Dynamic Binding :- Refers to linking of function call with function definition is called binding and when it takes place at run time is called dynamic binding.

5) Message Passing :- The process by which one object can interact with other object is called message passing.

6) Inheritance :- It is the process by which object of one class acquire the properties or features of objects of another class. The concept of inheritance provide the idea of reusability means we can add additional features to an existing class without modifying it. This is possible by driving a new class from the existing one. The new class will have the combined features of both the classes.

Example :- Robin is a part of the

class flying bird which is again a part of the class bird.

7) Polymorphism:- It means ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

Example:- Operator Overloading
Function Overloading

Q) History of Java :-

Java Programming language was written by James Gosling along with two other person 'Mike Sheridan' and 'Patrick Naughton' while they were working at Sun MicroSystems. Initially it was named oak Programming language.

Java Releases

1) Initial Java Versions 1.0 and 1.1 was released in the year

1996 for Linux, Solaris, Mac and Windows.

2) Java version 1.2 (Commonly called as Java 2) was released in the year 1998.

3) Java version 1.3 codename Kestrel was released in the year 2000.

4) Java version 1.4 codename Merlin was released in the year 2002.

5) Java version 1.5/Java SE 5 codename 'Tiger' was released in the year 2004.

6) Java version 1.6/Java SE 6 codename 'Mustang' was released in the year 2006.

7) Java version 1.7/Java SE 7 codename 'Dolphin' was released in the year 2011.

8) Java Version 1.8 is the current stable release which was released in the years 2015.

Q) Features of Java

The prime reason behind creation of Java was to bring portability

and security features into a computer language. Beside these two major features, there were many other features that played an important role in moulding out the final form of this outstanding language. Those features are:-

1) Simple

Java is easy to learn and its syntax is quite simple, clean and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

Eg:- Pointers and Operator Overloading are not there in java but were an important part of C++

2) Object Oriented

In java everything is Object which has some data and behaviour. Java can be easily extended as it is based on Object model.

3) Robust

Java makes an effort to eliminate error prone code by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

4) Platform Independent

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.

5) Secure

When it comes to security,

PAGE NO.....
DATE...../...../.....

Java is always the first choice with java secure features it enables us to develop virus free, temper free system. Java program always run in Java runtime environment with almost null interaction with System OS, hence it is more secured.

6) Multi Threading

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like while typing, grammatical errors are checked along.

7) Architectural Neutral

Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

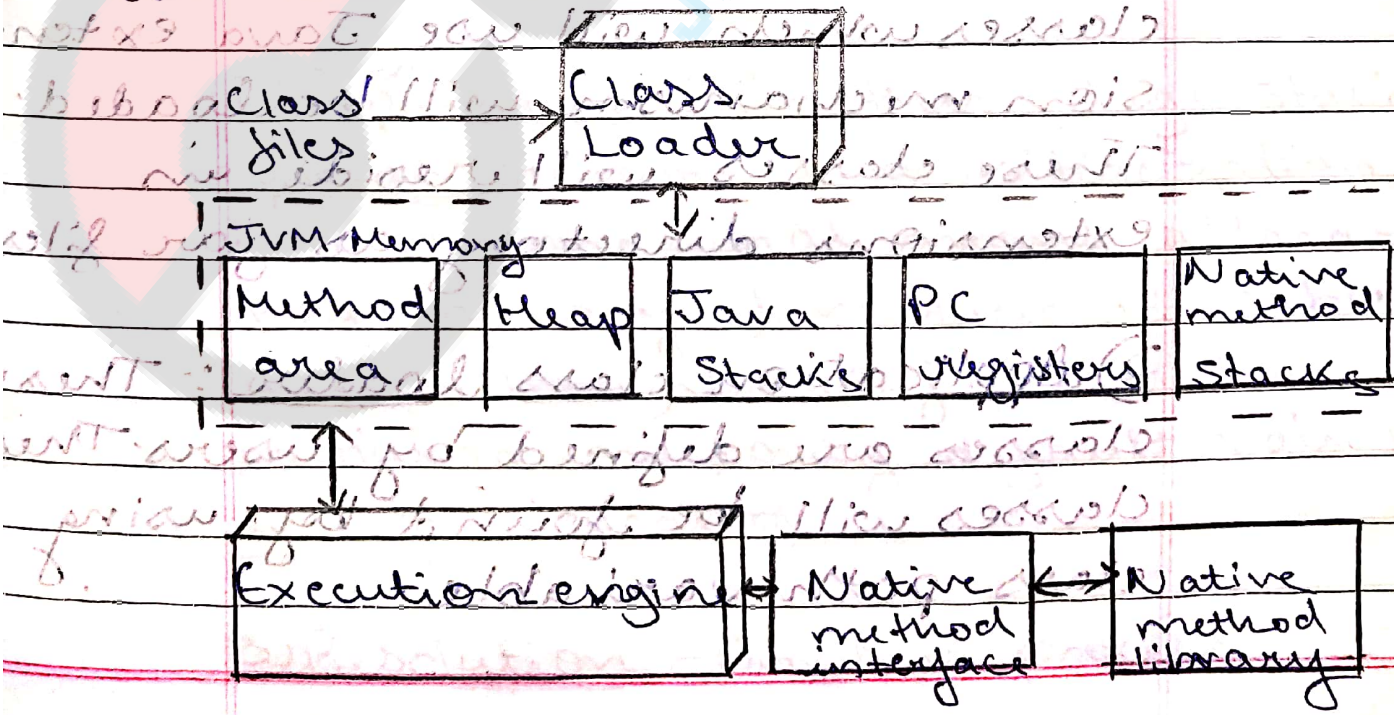
8) Portable

Java Byte code can be carried to any platform. No implementation dependant features. Everything related to storage is predefined, example: size of primitive data types

9) High Performance

Java is an interpreted language so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

Q) JVM Architecture



PAGE NO.....
DATE. ...../...../.....

Class loader

Class loader sub system is responsible for class loading, linking and initialization.

Loading

Here loader will search for the classes and load in order.

It will contains 3 parts:

1) Bootstrap class loader :- It loads classes that are related to java platform and the classes which are in bootstrap path which is present in rt.jar. Actually rt.jar contains all compiled classes.

2) Extension class loader :- Here the classes which will use Java extension mechanism will be loaded. These classes will reside in extensions directory as .jar files.

3) Application class loader :- These classes are defined by users. These classes will be found by using class path variable.

After loading object is created for the class file. It is used to represent memory in heap and this object is used by the programmer to retrieve information.

Linking

Linking involves verification, preparation, Resolution.

1) Verification :- It will verify whether the byte code is properly formatted that means the binary representation of the class has followed the structural constraints or not. If not followed then verify Error must be thrown. This Error is sub class of linkage Error.

2) Preparation :- In this stage static fields creation and initialization will be done. That means allocating memory for static variables and initialization of those. In initialization, only default values will be allocated.

3) Resolution :- In this process

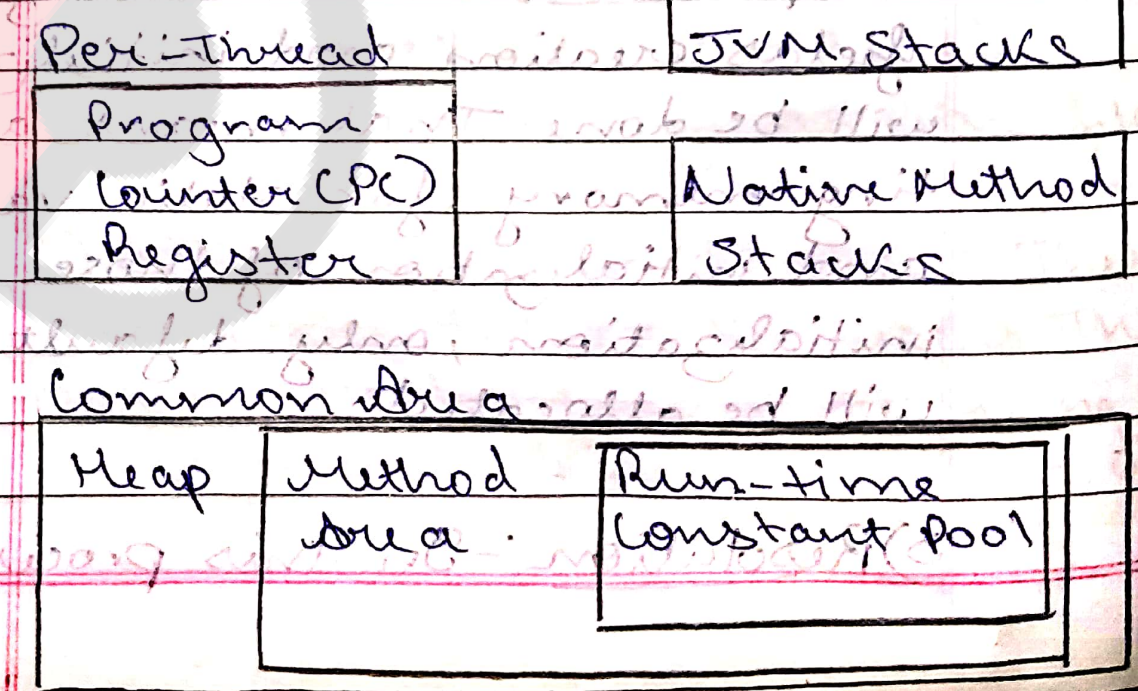
symbolic references will be dynamically replaced with their direct references (concrete values)

Initialization

In this phase classes and interfaces will be initialized. This will be done by executing the initialization method of the class or interface.

Runtime data areas (memory)

JVM organizes memory in several runtime data areas. For the execution of the program, these data areas are needed. Some memory will be shared among all the threads.



1) Method area :- This is logically a part heap space which will contain the class structure. It stores per class structures means run time constants and static variables, methods, constructors, class names and also class type information. It is a shared resource. Only one method area will reside in JVM. In Runtime constant pool string literals will be stored. Here literals don't relate to any object instances. Run time constant pool doesn't relate to object instances.

2) Heap :- Here information about objects will be stored. If we create object, heap space will be allocated. If object dies then memory garbage is collected. It is common space shared among all threads.

3) Stack area :- It is not shared memory. Per every thread one run time stack will be created.

It holds local variables, parameters, intermediate results and other data. It plays role in method invocation and return. JVM stores thread information in discrete frames. These frames will be stored in JVM stacks with push and pop operations. Here stack memory need not to be contiguous and it is dynamically expanded. If memory is not sufficient for expansion it will throw out of memory error.

4) PC register:- Whenever new thread is created then it will get pc register. PC register will store the address of current instruction to be executed.

5) Native method stacks:- Native method stack will store native methods. This is also not a shared resource. Native method is java method but implementation will be another language mostly in

C. These methods usually used to interface the systems calls and libraries.

Execution engine

Execution engine will execute the java byte code that presents in run time data areas. Each byte code instruction contains opcode and operand. With the help of both execution engine will execute Java Byte code should be changed to machine understandable. It should be done by compiler or interpreter.

Interpreter

It can interpret the byte code instructions line by line. It can interpret faster than compiler. But the repeated code should also be interpreted again. This is only the disadvantage about the interpreter.

JIT compiler (just in time)

It is mainly used for repeated codes. As this is the disadva-

PAGE NO.....
DATE. ...../...../.....

stage of the interpreter to compensate that disadvantage this compiler came into the picture. The compiler takes more time than interpreter. But here JIT compiler will improve the performance. Actually every method will be interpreted first. If call count of this method increases more than JIT threshold than that method will be compiled by JIT compiler. JIT compiler will translate byte code into native code. After that only native code will be executed. That means JIT compiler will be invoked at the appropriate time and operated based on method frequency.

Java Native Interface:

Java supports native codes through JNI interface. Native methods are useful for system calls and other issues related to memory management and performance issues. JNI is a framework that will be used

to interface the native applications. It will enable java application to call and to be called by native applications.

Java Native Interface

Java supports native codes through JNI interface. Native methods are useful for system calls and other issues related to memory management and performance issues. JNI is a framework that will be used to interface the native applications. It will enable java application to call and to be called by native applications.

Native Method libraries

These native method libraries contain native methods. Our system hardware may have some special capabilities. To use those special qualities for our application we need native methods. Some native methods provide extra speed.

for memory management also some native methods will be designed. Usually, native methods will be written in C or C++.

Q] Differences between C++ and Java

Comparison Index	C++	Java
Platform Independent	C++ is platform dependent	Java is platform independent
Mainly used for	C++ is mainly used for system programming	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems	Java was designed

	and applications programming. It was an extension of C programming language.	and created as an interpreter for printing systems but later extended as a support network computing It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement	Java doesn't support the goto statement
Multiple inheritance	C++ supports multiple inheritance	Java doesn't support multiple inheritance through class It can be achieved by interfaces in java.

Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes

		this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent
Call by value and Call by reference	C++ supports both call by value and call by reference	Java supports call by value only. There is no call by reference in java
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.

Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words non-static methods are virtual by default.
Document-ation comment	C++ doesn't support documentation comment.	Java supports documentation comment /** */ to create documentation for java source code.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills

		zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Q) Data types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1) Primitive data types: The primitive data types include boolean, char, byte, short, int,

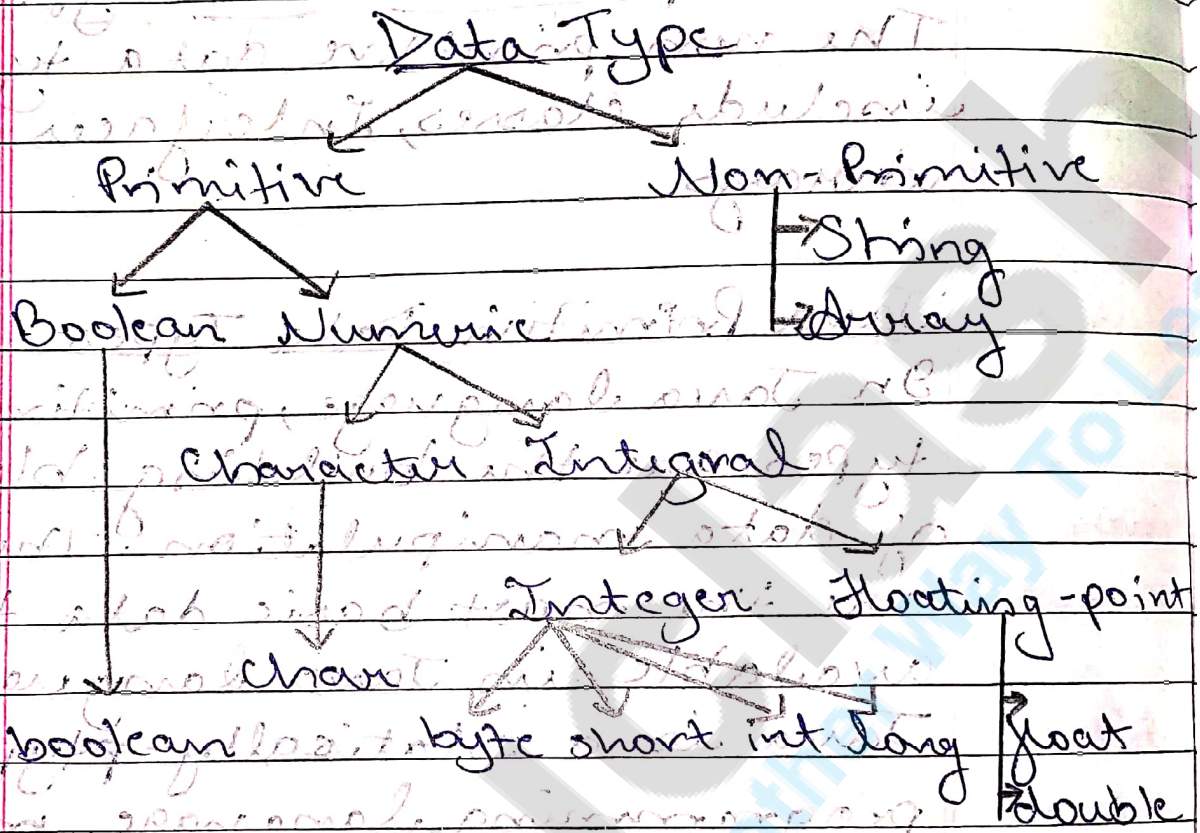
long, float and double.

2) Non-primitive data types:-
The non-primitive data types include classes, Interfaces and Arrays.

Java Primitive Data Types
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language. Java is a statically-typed programming language means, all variables must be declared before its use. That is why we need to declare variables type and name.

There are 8 types of primitive data types
1) boolean data type
2) byte data type
3) char data type
4) short data type
5) int data type
6) long data type

7) float data type
8) double data type



Boolean Data type:
The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.
The boolean data type specifies one bit of information, but its "size" can't be defined precisely.
Eg:- Boolean one = false

Byte Data type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0. The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example : byte a=10, byte b=-20

Short Data type

The short data type is a 16 bit signed two's complement integer. Its value range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example :- short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2^{31} to $2^{31}-1$ (inclusive). Its default value is 0. The int data type is generally used as a default data for integral values unless if there is no problem about memory.

Example : int a = 100000, int b = -200000

long Data type

The long data type is a 64-bit two's complement integer. Its value-range lies between (-2^{63}) to $(2^{63}-1)$ (inclusive). Its default value is 0. The long data type is used when

you need a range of values more than those provided by int.

Example: long a = 1000002L,
long b = -2000002L

Float Data type

The float data type is a single precision 32 bit floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F

Example: float f1 = 234.5f

Double Data type

The double data type is a double-precision 64-bit floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type

also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data type

The char data type is a single 16-bit Unicode character. Its value range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.

Example: char letterA = 'A'

Q) Variable

A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype. Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

Types of variables

There are three types of variables in java.

1) local variable
2) instance variable
3) static variable

1) local variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

PAGE NO.....
DATE. ...../...../.....

2) Instance Variable
A variable declared inside the class, but outside the body of the method, is called instance variable. It is not declared as static.
It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable
A variable which is declared as static is called static variable. It cannot be local.
You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Eg to understand the types of variables in java

```
class A{
  int data=50;//instance
                variable
```

```
static int m=100; // static variable
void method(){
int n=90; // local variable
}
}//end of class
```

Q) Expressions

Expressions consists of variables, operators, literals and method calls that evaluates to a single value

Eg:-

```
int score;
score=90;
```

Here, score=90 is an expression that returns int

```
Double a=2.2, b=3.4, result;
result=a+b-3.4;
```

Here, a+b-3.4 is an expression

```
if (number1 == number2)
System.out.println("Number 1 is larger than number 2");
```

Here, number 1 == number 2 is

an expression that returns Boolean. Similarly, "Number 1 is larger than number 2" is a string expression.

Q) Operators

Java provides a rich set of operators environment. Java operators can be divided into following catagories:-

1) Arithmetic operators
2) Relation operators
3) Logical operators
4) Bitwise operators
5) Assignment operators
6) Conditional operators
7) Misc operators

Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Description
+	adds two operands
-	Subtract second operands from first

*	multiply two operand
/	divide numerator by denumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Relation operators

The following table shows all relation operators supported by Java

Operator	Description
==	check if two operand are equal
!=	Check if two operand are not equal
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand

<=	Check if operand on left is smaller than or equal to right operand

logical operators

Java supports following 3 logical operators. Suppose a=1 and b=0;

Operator	Description	Example
&&	logical AND	(a&&b) is false
||	logical OR	(a||b) is true
!	logical NOT	(!a) is false

Bitwise operators

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now lets see truth table for bitwise &, | and ^

a	b	a&b	a|b	a^b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise shift operators shift the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence.

Example:

a = 00001000

b = 2

a << b = 00100000

a >> b = 00000010

Assignment Operators

Assignment operator supported by Java are as follows:

PAGE NO.....

DATE. ...../...../.....

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	multiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the	

	result to left operand.	
%=	calculate modulus using two operands and assign the result to left operand.	a %= b is same as a = a % b

Misc operator

There are few other operator supported by java language.

Conditional operator

It is also known as ternary operator and used to evaluate Boolean expression.

expr1 ? expr2 : expr3

If expr1 condition is true? Then value expr2 : Otherwise value expr3

instance of operator

This operator is used for object reference variables. The operator checks whether the object is of particular type

(class type (or interface type))

Q) Arrays
An array is a collection of similar data types. Array is a container object that hold values of homogenous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.
An array can be either primitive or reference type. It gets memory in heap area. Index of array starts from zero to size-1

Features of Array
- It is always indexed. Index begins from 0.
- It is a collection of similar data types.
- It occupies a contiguous memory location.

Array Declaration
Syntax:-

datatype [] identifier;
or
datatype identifier [];

Both are valid syntax for array declaration. But the former is more readable.

Example
int [] arr;
char [] arr;
short [] arr;
long [] arr;
int [][] arr;

Initialization of Array
new operator is used to initialize an array.
Example:
int [] arr = new int [10];
or
int [] arr = {10, 20, 30, 40, 50};

Accessing array element
To access nth element of an array.
Syntax
arrayname [n-1];

PAGE NO.....
DATE...../...../.....

Example: To access 4th element of a given array

```
int [] arr = {10, 20, 30, 40};
System.out.println("Element at 4th place" + arr[3]);
```

The above code will print the 4th element of array arr on console.

Multi-Dimensional Array

A multi-dimensional array is very much similar to a single dimensional array. It can have multiple rows and multiple columns while single dimensional array, which can have only one full row or one full column.

Array Declaration:

Syntax

```
datatype [] [] identifier;
```

or

```
datatype identifier [] [];
```

Initialization of Array

new operator is used to initialize an array.

Example:-

int [][] arr=new int [10][10];

or

int [][] arr={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};

Accessing array element:-

For both row and column, the index begins from 0.

Syntax:

array_name [m-1][n-1]

Example:

int arr[][]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};

System.out.println("Element at (2,3) place" + arr[1][2]);

9] Control Statements / Control Structures

A control statement works as a determiner for deciding the next task of the other statements whether to execute or not. An if statement

decides whether to execute a statement or which statement has to execute first between the two. In Java, the control statements are divided into three catagories which are selection statements, iteration statements and jump statements. A program can execute from top to bottom but if we use a control statement we can set order for executing a program based on values and logic.

Control flow statements control the flow of a programs execution. Here flow of execution will be based on state of a program. We have 4 decision making statements available in Java.

Simple if Statement

Simple if statement is the basic of decision making statements in Java. It decides if certain amount of code should be executed based

on the condition.

Syntax:

if (condition){
Statement 1; // if condition becomes true then this will be executed
}
Statement 2; // this will be executed irrespective of condition becomes true or false

Example

```
class ifTest
{
    public static void main(String args[])
    {
    int x=5;
        if (x>10)
    System.out.println("Inside If");
System.out.println("After if statement");
    }
}
```

Output

After if statement

if..else statement

In if...else statement, if condition is true then statements in if block will be executed but if it comes out as false then else block will be executed.

Syntax:

```
if (condition)
{
Statement 1; // if condition becomes true then this will be executed
}
```

Example:

```
class ifElseTest
{
  public static void main (String args [])
  {
    int x = 9;
    if (x > 10)
System.out.println("x is greater than 10");
    else
System.out.println("x is less than 10");
System.out.println("After if
```

```
else statement");
}
}
```

Output:

```
i is less than 10
After if else statement
```

Nested if statement

Nested if statement is if inside an if block. It is same as normal if...else statement but they are written inside another if...else statement.

Syntax:

```
if(condition1){
Statement 1; //executed when
        condition 1 is true
if (condition 2){
Statement 2; //executed when
        condition 2 is true
}
else{
Statement 3; //executed when
        condition 2 is false
}
}
```

Example:

```
class nestedifTest
{
public static void main(String
        args[]):
{
int x=25;
if(x>10)
{
if(x%2==0)
System.out.println("i is
greater than 10 and even
number");
else
System.out.println("i is gre-
ter than 10 and odd number");
}
else
{
System.out.println("i is less
than 10");
}
System.out.println("After
nested if statement");
}
}
```

Output:

i is greater than 10 and odd number.
After nested if statement

if...else if...else statement

if...else if statements will be used when we need to compare the value with more than 2 conditions. They are executed from top to bottom approach. As soon as the code finds the matching condition, that block will be executed. But if no condition is matching then the last else statement will be executed.

Syntax:

```
if (condition2) {
Statement 1; // if condition 1 becomes true then this will be executed
}
else if (condition2) {
Statement 2; // if condition 2 becomes true then this will be executed
}
:
```

else
Statement 3; // executed when no matching condition found
}

Example

```
class ifelseifTest
{
    public static void main(String args[])
    {
        int x=2;
        if (x>10)
        {
            System.out.println("i is greater than 10");
        }
        else if (x<10)
        {
            System.out.println("i is less than 10");
        }
        else
        {
            System.out.println("i is is 10");
        }
        System.out.println("After if else if ladder statement");
```

}

Output:-

i is less than 10

After if else if ladder statement

Switch Statement

Java "switch" statement compares the value and executes one of the case blocks based on the condition. It is same as if..else if ladder. Below are some points to consider while working with switch statements:

- case value must be of the same type as expression used in switch statement.
- case value must be a constant or literal. It doesn't allow variables.
- case values should be unique. If it is duplicate, then program will give compile time error.

Example :-

```
class switchDemo2
{
public static void main (String
  args[])
{
 int i=2;
switch(i)
{
  case 0:
System.out.println("i is 0");
 break;
 case 1:
System.out.println("i is 1");
 break;
case 2:
System.out.println("i is 2");
 break;
 case 3:
 System.out.println("i is 3");
 break;
 case 4:
 System.out.println("i is 4");
 break;
 default;
 System.out.println("i is not in
   the list");
}
}
}
```

Looping Statements in Java

looping statements are the statements which executes a block of code repeatedly until some condition met to the criteria. loops can be considered as repeating if statements. There are 3 types of loops available in Java.

While

while loops are simplest kind of loop. It checks and evaluates the condition and if it is true then executes the body of loop. This is repeated until the condition becomes false. Condition in while loop must be given as a Boolean expression. If int or string is used instead, compile will give the error.

Syntax:

```
while(condition)
    statement;
```

Example:

```
class whileLoopTest
{
    public static void main(String args[])
    {
        int j=1;
        while(j<=10)
        {
            System.out.println(j);
            j=j+2;
        }
    }
```

Output:

```
1
3
5
7
9
```

Do... while

Do... while works same as while loop. It has only one difference that is do... while, condition is checked after the execution of the loop body. That is why this loop is considered as exit control loop. In do...

while loop, body of loop will be executed at least once before checking the condition.

Syntax:

```
do {
statement 1;
} while(condition);
```

Example:

```
class dowhileLoopTest
{
public static void main(String args[])
{
int j=10;
do
{
System.out.println(j);
j=j+1;
} while(j<=10)
}
}
```

Output:

10

For

It is the most common and widely used loop in Java. It is the easiest way to construct a loop structure in code as initialization of a variable, a condition and increment / decrement are declared only in a single line of code. It is easy to debug structure in Java.

Syntax:

```
for(initialization; condition;
increment/decrement)
{
  statement;
}
```

Example

```
class forLoopTest
{
public static void main(String
        args[])
{
  for(int j=1;j<=5;j++)
  System.out.println(j);
}
}
```

PAGE NO.....
DATE ...../...../.....

Output:

1
2
3
4
5

For-Each loop

For-Each loop is used to traverse through elements in an array. It is easier to use because we don't have to increment the value. It returns the elements from the array or collection one by one.

Example:

```
class ForeachDemo{
public static void main(String
args[]){
int a[]={10,15,20,25,30};
for(int i:a){
System.out.println(i);
}
}
```

Output:

10
15

20
25
30

Branching statements in Java
Branching statements jump from one statement to another and transfer the execution flow. There are 3 branching statements in Java.

Break
Break statement is used to terminate the execution and bypass the remaining code in loop. It is mostly used in loop to stop the execution and comes out of loop. When there are nested loops then break will terminate the innermost loop.

Example:

```
class breakTest{
public static void main(String
args[])
{
for(int j=0; j<5; j++)
{
```

```
if (j==4)
  break;
System.out.println(j);
}
System.out.println("After loop");
}
```

Output:

```
0
1
2
3
4
After loop
```

Continue

Continue statement works same as break but the difference is it only comes out of loop for that iteration and continue to execute the code for next iterations. So it only bypasses the current iteration.

Example:

```
class continueTest
{
public static void main (String args[])
{
for(int j=0; j<10; j++)
{
if (j%2!=0)
continue;
System.out.println(j+" ");
}
}
}
```

Output:

0 2 4 6 8

Return

Return statement is used to transfer the control back to calling method. Compiler will always bypass any sentences after return statement. So, it must be at the end of any method. They can also return a value to the calling method.

Example: Here method

getwebURL() returns the current URL to the caller method.

```
public String getwebURL()
{
    String vURL = null;
    try {
        vURL = driver.getCurrentURL();
    }
    catch(Exception e){
        System.out.println("Exception occured while getting the current url:"+e.getStackTrace());
    }
    return vURL;
}
```