MODULE II

JAVA PROGRAMMING

What is Java?

Java is a high-level, general-purpose, object-oriented, and secure programming language developed by James Gosling at Sun Microsystems, Inc. in 1991. It is formally known as OAK. In 1995, Sun Microsystem changed the name to Java. In 2009, Sun Microsystem takeover by Oracle Corporation.

Java Platform

Java Platform is a collection of programs. It helps to develop and run a program written in the Java programming language. Java Platform includes an execution engine, a compiler and set of libraries. Java is a platform-independent language.

Features of Java

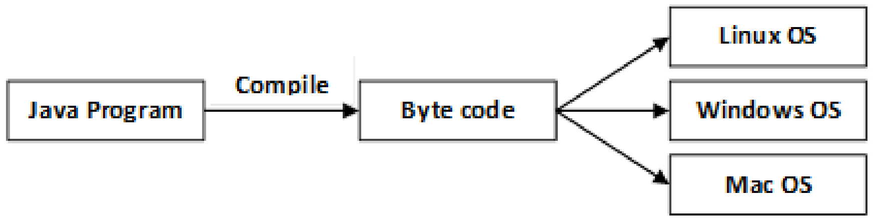
Simple: Java is a simple language because its syntax is simple, clean, and easy to understand. Complex and ambiguous concepts of C++ are either eliminated or re-implemented in Java. For example, pointer and operator overloading are not used in Java.

Object-Oriented: In Java, everything is in the form of the object. It means it has some data and behavior. A program must have at least one class and object.

Robust: Java makes an effort to check error at run time and compile time. It uses a strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.

Secure: Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.

Platform-Independent: Java provides a guarantee that code writes once and run anywhere. This byte code is platform-independent and can be run on any machine.



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

High Performance: Java is an interpreted language. Java enables high performance with the use of the Just-In-Time compiler.

Distributed: Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. It can run over the internet. EJB and RMI are used to create a distributed system.

Multi-threaded: Java also supports multi-threading. It means to handle more than one job a time.A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection)

JAVA API

In simple terms, API, or Application Programming Interface, is a connection that allows

you to make software.APIs are important software components bundled with the JDK.

APIs in Java include classes, interfaces, and user Interfaces. They enable developers to

integrate various applications and websites and offer real-time information.

Byte code

Byte codeis program code that has been compiled from source code into low-level code

designed for a software interpreter. Byte code is the compiled format for Java programs.

Once a Java program has been converted to byte code, it can be transferred across a

network and executed by Java Virtual Machine (JVM). Byte code files generally have a

.class extension.

JAVA VIRTUAL MACHINE(JVM)

A specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.

An implementation Its implementation is known as JRE (Java Runtime Environment).

Runtime Instance Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

What it does

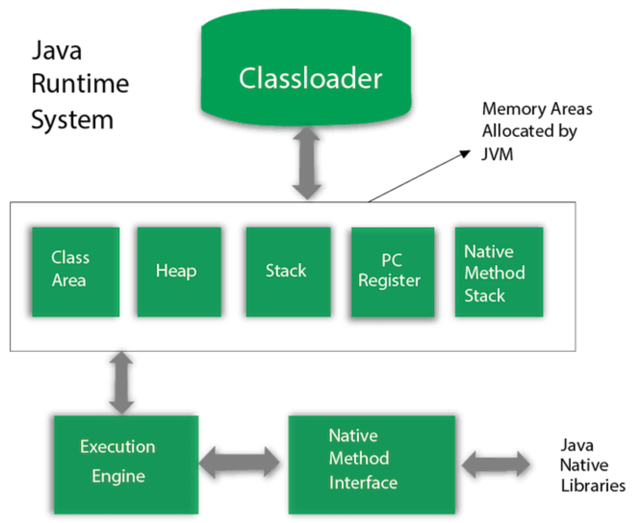
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code

- Provides runtime environment
- JVM provides definitions for the:
- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside $JAVA_HOME/jre/lib/ext directory.

System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

Interpreter: Read bytecode stream then execute the instructions.

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

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.

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

int data=50;//Here data is variable

Types of Variables

There are three types of variables in Java:

1. local variable
2. instance variable
3. static variable

Types of variables in java

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.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

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

Example to understand the types of variables in java

```
public class A

{

    static int m=100;//static variable

    void method()

    {
        int n=90;//local variable
    }
    public static void main(String args[])
    {
        int data=50;//instance variable
    }
}//end of class
```

Java Variable Example: Add Two Numbers

```
public class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}
}
```

Data Types in Java

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

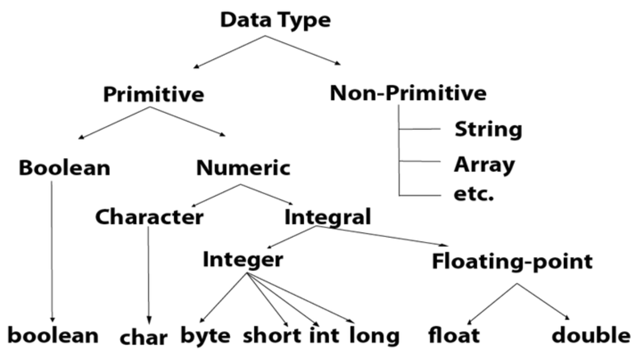
Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

Thereforeare 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



Data Type   Default Value   Default size

boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

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

Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 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 - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type 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 -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 - 1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. 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 = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 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 IEEE 754 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'

Operators in Java

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

1. Unary Operator,
2. Arithmetic Operator,
3. Shift Operator,
4. Relational Operator,
5. Bitwise Operator,
6. Logical Operator,
7. Ternary Operator and
8. Assignment Operator.
9. Java Operator Precedence
10. Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

incrementing/decrementing a value by one

negating an expression

inverting the value of a boolean

Java Unary Operator Example: ++ and --

```
public class OperatorExample{

public static void main(String args[]){

int x=10;

System.out.println(x++);//10 (11)

System.out.println(++x);//12

System.out.println(x--);//12 (11)

System.out.println(--x);//10

}}
```

Output:

```
10

12

12
```

```
10
```

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
```

```
2

0
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
public class OperatorExample{

public static void main(String args[]){

System.out.println(10<<2);//10*2^2=10*4=40

System.out.println(10<<3);//10*2^3=10*8=80

System.out.println(20<<2);//20*2^2=20*4=80

System.out.println(15<<4);//15*2^4=15*16=240

}}
```

Output:

```
40

80

80

240
```

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
public OperatorExample{

public static void main(String args[]){

System.out.println(10>>2);//10/2^2=10/4=2

System.out.println(20>>2);//20/2^2=20/4=5

System.out.println(20>>3);//20/2^3=20/8=2

}}
```

Output:

```
2

5

2
```

Java AND Operator

Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a<b&&a<c);//false && true = false

System.out.println(a<b&a<c);//false & true = false

}}
```

Output:

```
false

false
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=5;

int c=20;

System.out.println(a>b||a<c);//true || true = true

System.out.println(a>b|a<c);//true | true = true

//|| vs |

System.out.println(a>b||a++<c);//true || true = true

System.out.println(a);//10 because second condition is not checked

System.out.println(a>b|a++<c);//true | true = true

System.out.println(a);//11 because second condition is checked

}}
```

Output:

```
true

true

true

10
```

```
true

11
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
public class OperatorExample{

public static void main(String args[]){

int a=2;

int b=5;

int min=(a<b)?a:b;

System.out.println(min);

}}
```

Output:

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```
public class OperatorExample{

public static void main(String args[]){

int a=10;

int b=20;

a+=4;//a=a+4 (a=10+4)

b-=4;//b=b-4 (b=20-4)

System.out.println(a);

System.out.println(b);

}}
```

Output:

```
14

16
```

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

List of Java Keywords

A list of Java keywords or reserved words are given below:

abstract: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.

boolean: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.

break: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.

byte: Java byte keyword is used to declare a variable that can hold 8-bit data values.

case: Java case keyword is used with the switch statements to mark blocks of text.

catch: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

char: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters

class: Java class keyword is used to declare a class.

continue: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

default: Java default keyword is used to specify the default block of code in a switch statement.

do: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.

double: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.

else: Java else keyword is used to indicate the alternative branches in an if statement.

enum: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

extends: Java extends keyword is used to indicate that a class is derived from another class or interface.

final: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.

finally: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.

float: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

this: Java this keyword can be used to refer the current object in a method or constructor.

throw: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.

throws: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.

transient: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

try: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

void: Java void keyword is used to specify that a method does not have a return value.

volatile: Java volatile keyword is used to indicate that a variable may change asynchronously.

while: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {

statement 1; //executes when condition is true

}
```

Consider the following example in which we have used the if statement in the java code.

Student.java

```
public class Student {

public static void main(String[] args) {

int x = 10;

int y = 12;

if(x+y > 20) {

System.out.println("x + y is greater than 20");

}

}

}
```

Output:

```
x + y is greater than 20
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {

statement 1; //executes when condition is true

}

else{

statement 2; //executes when condition is false

}
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
if(condition 1) {

statement 1; //executes when condition 1 is true

}

else if(condition 2) {

statement 2; //executes when condition 2 is true

}

else {
```

```
statement 2; //executes when all the conditions are false

}

public class Student {

public static void main(String[] args) {

String city = "Delhi";

if(city == "Meerut") {

System.out.println("city is meerut");

}else if (city == "Noida") {

System.out.println("city is noida");

}else if(city == "Agra") {

System.out.println("city is agra");

}else {

System.out.println(city);

}

}

}
```

Output:

Delhi

4. Nested if-statement

In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1) {

statement 1; //executes when condition 1 is true

if(condition 2) {

statement 2; //executes when condition 2 is true

}

else{

statement 2; //executes when condition 2 is false

}

}
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java

Cases cannot be duplicate

Default statement is executed when any of the case doesn't match the value of expression. It is optional.

Break statement terminates the switch block when the condition is satisfied.

It is optional, if not used, next case is executed.

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The **syntax** to use the switch statement is given below.

```
switch (expression){
    case value1:
     statement1;
     break;
    .
    .
    .
    case valueN:
     statementN;
```

```
  break;
 default:
  default statement;
}
```

Consider the following example to understand the flow of the switch statement.

Student.java

```
public class Student implements Cloneable {
public static void main(String[] args) {
int num = 2;
switch (num){
case 0:
System.out.println("number is 0");
break;
case 1:
System.out.println("number is 1");
break;
default:
System.out.println(num);
}

}

}
```

Output:

```
2
```

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop
4. for-each loop

Let's understand the loop statements one by one.

Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
for(initialization, condition, increment/decrement) {

//block of statements

}
```

The flow chart for the for-loop is given below.

Control Flow in Java

Consider the following example to understand the proper functioning of the for loop in java.

Calculation.java

```
public class Calculattion {

public static void main(String[] args) {

// TODO Auto-generated method stub

int sum = 0;
for(int j = 1; j<=10; j++) {

sum = sum + j;

}

System.out.println("The sum of first 10 natural numbers is " + sum);

}

}
```

Output:

```
The sum of first 10 natural numbers is 55
```

Java for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
for(data_type var : array_name/collection_name){

//statements

}
```

Consider the following example to understand the functioning of the for-each loop in Java.

Calculation.java

```
public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

String[] names = {"Java","C","C++","Python","JavaScript"};

System.out.println("Printing the content of the array names:\n");

for(String name:names) {

System.out.println(name);

}

}

}
```

Output:

```
Printing the content of the array names:


Java

C

C++
```

Python

JavaScript

Java while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
while(condition){
//looping statements
}
```

The flow chart for the while loop is given in the following image.

Control Flow in Java

Consider the following example.

Calculation .java

```
public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

while(i<=10) {

System.out.println(i);

i = i + 2;

}

}

}
```

Output:

```
Printing the list of first 10 even numbers
```

```
0

2

4

6

8

10
```

Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do

{

//statements

} while (condition);
```

The flow chart of the do-while loop is given in the following image.

Control Flow in Java

Consider the following example to understand the functioning of the do-while loop in Java.

Calculation.java

```
public class Calculation {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i = 0;

System.out.println("Printing the list of first 10 even numbers \n");

do {

System.out.println(i);

i = i + 2;

}while(i<=10);

}

}
```

Output:

```
Printing the list of first 10 even numbers
```

```
0

2

4

6

8

10
```

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

The break statement example with for loop

Consider the following example in which we have used the break statement with the for lop.

BreakExample.java

```
public class BreakExample {

public static void main(String[]

 args) {

// TODO Auto-generated method stub

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

System.out.println(i);


if(i==6) {

break;

}

}

}

}
```

Output:

```
0
```

```
1

2

3

4

5

6
```

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

```
public class ContinueExample {

public static void main(String[] args) {

// TODO Auto-generated method stub

for(int i = 0; i<= 2; i++) {

for (int j = i; j<=5; j++) {

if(j == 4) {

continue;

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

Output:

```
0
1
2
3
5
1
2
3
5
2
3
5
```

How will you give comments in java?

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.Single-line comments start with two forward slashes (//).Any text between // and the end of the line is ignored by Java (will not be executed).This example uses a single-line comment before a line of code: System.out.println("Hello World"); // This is a commentThis example uses a single-line comment before a line of code:

```
// This is a comment
System.out.println("Hello World");
```

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

```
/* The code below will print the words Hello World

to the screen, and it is amazing */

System.out.println("Hello World");
```

List the different types of literals in java.

Literal:Any constant value which can be assigned to the variable is called as

literal/constant.

int x = 100; // Here 100 is a constant/literal.

There are five types of literals in Java.

- Integer Literals
- Boolean Literals
- Character Literals
- String Literals
- Floating Point Literals

Integral literals: The integer literal can be used to create int value. They can be used to initialize the group data types like byte, short, int and long.The java compiler treats all the integer literals as int by default. We can assign different values to integer literal.
They are:

Decimal literals (Base 10) : In this form the allowed digits are 0-9.

```
int x = 101;
```

Octal literals (Base 8) : In this form the allowed digits are 0-7.

The octal number should be prefix with 0.

```
int x = 0146;
```

Hexa-decimal literals (Base 16) : In this form the allowed digits are 0-9 and characters

are a-f. We can use both uppercase and lowercase characters. As we know that java is a case-sensitive programming language but here java is not case-sensitive.The hexa-decimal number should be prefix with 0X or 0x.

int x = 0X123Face;

Binary literals : From 1.7 onward we can specify literals value even in binary formalso, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B. int x = 0b1111;

Floating-Point literal: The floating point literal can be used to represent the decimal values with a fractional component. These literals are double data type. These literals contain the fractional parts and if the floating point literal is suffixed with letter f or F then it is float type.

These literals include the float and double data types. In floating point literals double is the default data type. To represent the float literal f is suffixed and to represent the double literal d is suffixed.

For example:

float x = 2.7f;

double x = 54.888d;

Char literal

For char data types we can specify literals in 4 ways:

Single quote: We can specify literal to char data type as single character within single quote.

char ch = 'a';

Char literal as Integral literal: we can specify char literal as integral literal which represents Unicode value of the character and that integral literals can be specified either in Decimal, Octal and Hexadecimal forms. But the allowed range is 0 to 65535.

char ch = 062;

Unicode Representation: We can specify char literals in Unicode representation '\uxxxx. Here xxxx represents 4 hexadecimal numbers.

char ch = '\u0061';// Here /u0061 represent a.

Escape Sequence: Every escape character can be specify as char literals.

char ch = '\n';

String literal

Any sequence of characters within double quotes is treated as String literals.

String s = "Hello";

String literals may not contain unescaped newline or linefeed characters.

boolean literals

Only two values are allowed for Boolean literals i.e. true and false.

boolean b = true;

What are constructors? How they are invoked in java? Also explain the

different types of constructors.

Constructor is a special method in Java which is used to initialize the object. It looks like a normal method however it is not. A normal java method will have return type whereas the constructor will not have an explicit return type. A constructor will be called during the time of object creation (i.e) when we use new keyword follow by class name.Rules for writing Constructor:

• Constructor(s) of a class must have same name as the class name in which it

resides.

• A constructor in Java cannot be abstract, final, static and Synchronized.

• Access modifiers can be used in constructor declaration to control its access i.e

which other class can call the constructor.

• A Constructor cannot have a explicit return type.

How they are invoked in java

For Example: Let's say we have class by the name "Test", we will create object for Test class like below

Test t = new Test();

This will invoke the default constructor of the Test class.

There are two type of Constructors in Java, they are

• Default Constructor (or) no-arg Constructor

• Parameterized Constructor

Default Constructor (or) No-argument constructor

A constructor that has no parameter is known as default constructor. If we don’t define a constructor in a class, then compiler creates default constructor (with no arguments) for the class.In the below code we have created the no-arg constructor which gets called during the

time of object creation (Car c = new Car())

```
public class Car
{
Car()
 {
 System.out.println("Default Constructor of Car class called");
 }
 public static void main(String args[])
 {
 //Calling the default constructor
 Car c = new Car();
 }

}
```

Parameterized Constructor

A Constructor which has parameters in it called as

Parameterized Constructors, the Parameterized constructor is used to assign different values for the different objects.

If we want to initialize fields of the class with own values, then use a parameterized constructor.

In the below example we have a parameterized constructor for the car class which set the

value for the parameter "carColor"

```
public class Car

{

 String carColor;
Car(String carColor)

 { this.carColor = carColor;

 }
 public void disp()

 { System.out.println("Color of the Car is : "+carColor);

 }

 public static void main(String args[])

 {

 //Calling the parameterized constructor

 Car c = new Car("Blue");

c.disp();

 }

}
```

Explain the Basic structure of java program?

1. A Java program involves the following sections:
2. Documentation Section
3. Package Statement
4. Import Statements
5. Interface Statement
6. Class Definition
7. Main Method Class
8. Main Method Definition

Documentation Section: You can write a comment in this section. Comments are

beneficial for the programmer because they help them understand the code. These are optional.

Package statement: You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program. Here, the package is a keyword that tells the compiler that package has been created.It is declared as:package package_name;

Import statements: This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program.
mport calc.add;

Interface statement: Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to mplement multiple inheritances within a program.

Class Definition: A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.

Main Method Class: Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements

Explain the structure of simple java program.

Example

```
//Name of this file will be "Hello.java"

public class Hello


{
 /* Description:

 Writes the words "Hello Java" on the screen */

 public static void main(String[] args)

 {

 System.out.println("Hello Java");

 }

}
```

public class Hello

This creates a class called Hello.All class names must start with a capital letter.

The public word means that it is accessible from any other classes.

/* Comments */ The compiler ignores comment block. Comment can be used anywhere in the program to add info about the program or code block, which will be helpful for

developers to understand the existing code in the future easily.

public static void main

• When the main method is declared public, it means that it can also be used by code outside of its class, due to which the main method is declared public.

• The word static used when we want to access a method without creating its

object, as we call the main method, before creating any class objects.

• The word void indicates that a method does not return a value. main() is declared

as void because it does not return a value.

• main is a method; this is a starting point of a Java program.

String[] args It is an array where each element of it is a string, which has been named as"args". If your Java program is run through the console, you can pass the input parameter, and main() method takes it as input.System.out.println(); This statement is used to print text on the screen as output, where the system is a predefined class, and out is an object of the PrintWriter class defined in the system. The method println prints the text on the screen with a new line. You can also use print() method instead of println() method. All Java statement ends with a semicolon.