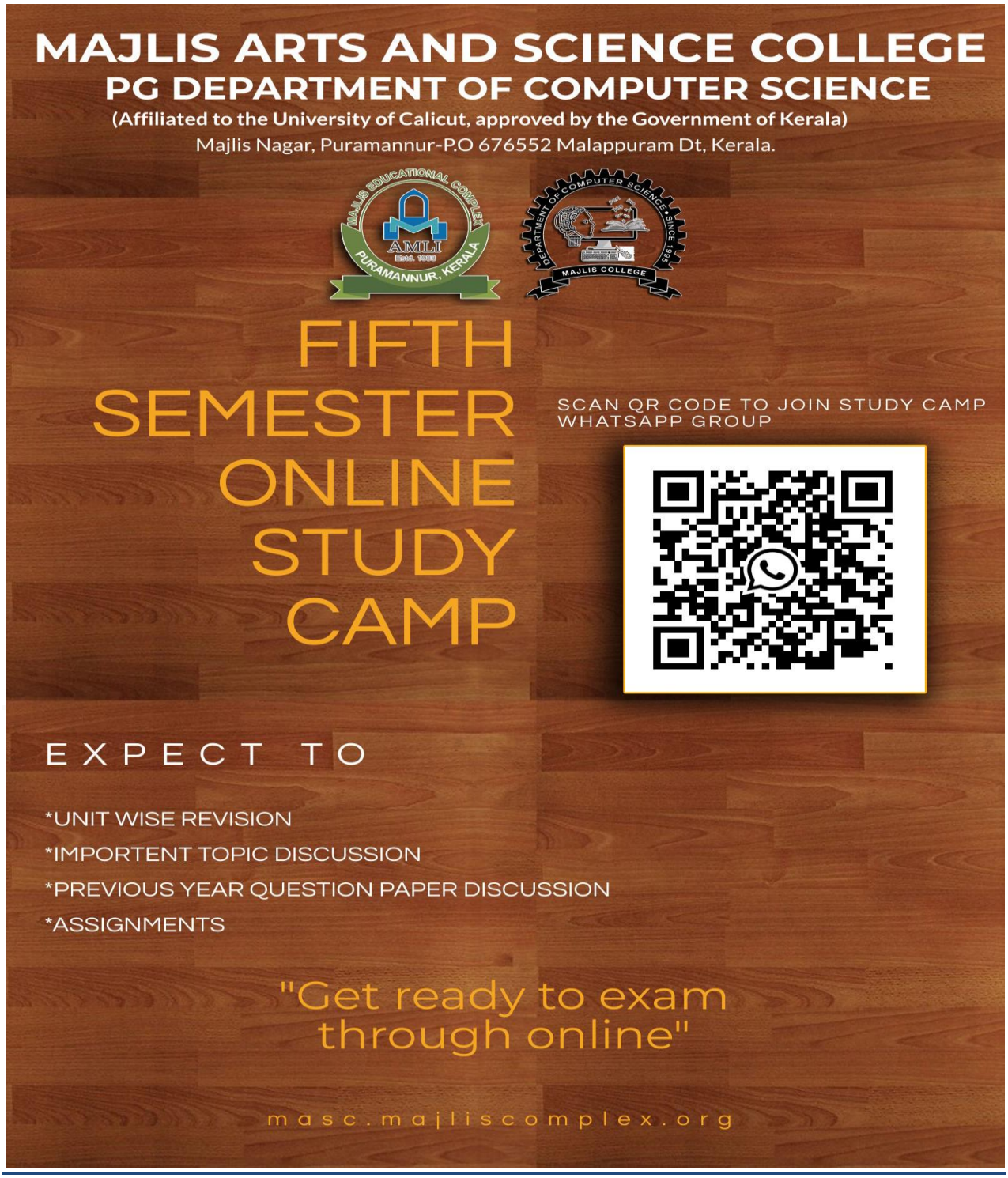


JAVA PROGRAMMING

MODULE 2

Java module 2

1. What is the importance of 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.

2. What is a 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.

3. How JVM works? Explain JVM Architecture?

   OR

4. What is JVM?

   Java Virtual Machine (JVM) is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of Java Run Environment (JRE). In other programming languages, the compiler produces machine code for a particular system. However, Java compiler produces code for a Virtual Machine known as Java Virtual Machine.

   How JVM works: First, Java code is compiled into byte code. This byte code gets interpreted on different machines

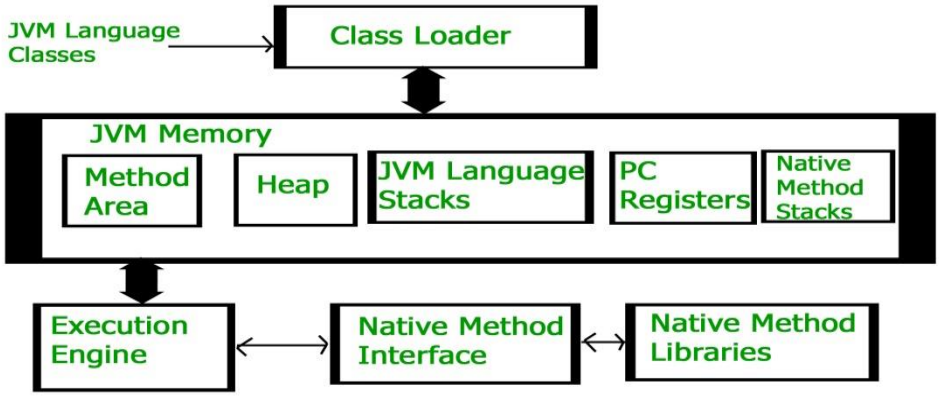
   Between host system and Java source, Byte code is an intermediary language.

   JVM in Java is responsible for allocating memory space.

   Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

   When we compile a .java file, .class files (contains byte-code) with the same class names present in .java file are generated by the Java compiler. This .class file goes into various steps when we run it. These steps together describe the whole JVM.

JVM Architecture



Class Loader Subsystem

It is mainly responsible for three activities.

Loading

Linking

Initialization

Loading: The Class loader reads the ".class" file, generate the corresponding binary data and save it in the method area. For each ".class" file, JVM stores the following information in the method area.

- The fully qualified name of the loaded class and its immediate parent class.
- Whether the ".class" file is related to Class or Interface or Enum.
- Modifier, Variables and Method information etc.

After loading the ".class" file, JVM creates an object of type Class to represent this file in the heap memory.

Linking: Performs verification, preparation, and (optionally) resolution.

Initialization: In this phase, all static variables are assigned with their values defined in the code and static block(if any). This is executed from top to bottom in a class and from parent to child in the class hierarchy.

JVM Memory

Method area: In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.

Heap area: Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.

Stack area: For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.

PC Registers: Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.

Native method stacks: For every thread, a separate native stack is created. It stores native method information.

Execution Engine

Execution engine executes the ".class" (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:

Interpreter: It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.

Just-In-Time Compiler(JIT) : It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.

Garbage Collector: It destroys un-referenced objects.

Java Native Interface (JNI) :

It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

Native Method Libraries :

It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.

5. Explain important features of java.

Following are the notable features of Java:

Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

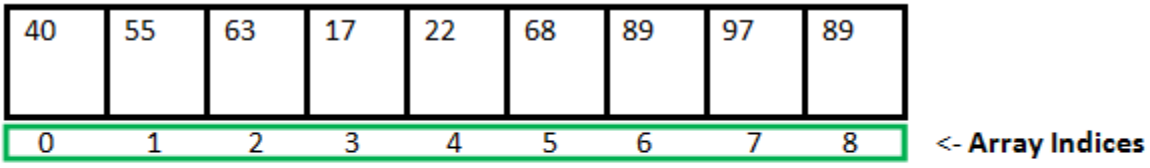
Architecture-neutral: Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on any processors, with the presence of Java runtime system.

Portable: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

Robust: Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.
Multithreaded: With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
High Performance: With the use of Just-In-Time compilers, Java enables high performance.
Distributed: Java is designed for the distributed environment of the internet.
Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

6. How can we use arrays in java?

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using the object property length. This is different from C/C++ where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The size of an array must be specified by an int or short value and not long.
- The direct superclass of an array type is Object.
- Every array type implements the interfaces Clone able and java.io.Serializable.
- Array can contain primitives (int, char, etc.) as well as object (or non-primitive) references of a class depending on the definition of the array. In case of primitive data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in heap segment.



Array Length = 9
First Index = 0
Last Index = 8

7. What is array? How one dimensional array are Created, Initialized, and Accessed?

Arrays are objects which store multiple variables of the same type. It can hold primitive types as well as object references. In fact most of the collection types in Java which are the part of java.util package use arrays internally in their functioning. Since Arrays are objects, they are created during runtime .The array length is fixed.

One-Dimensional Arrays:

The general form of a one-dimensional array declaration is

type var-name[];

OR

type[] var-name;

An array declaration has two components: the type and the name. type declares the element type of the array. The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc. or user-defined data types (objects of a class). Thus, the element type for the array determines what type of data the array will hold.

Example:

```
// both are valid declarations
int intArray[];
or int[] intArray;
```

Instantiating an Array in Java

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:The general form of new as it applies to one-dimensional arrays appears as follows:

```
var-name = new type [size];
```

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and var-name is the name of array variable that is linked to the array. That is, to use new to allocate an array, you must specify the type and number of elements to allocate.

Example:

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Array Literal

In a situation, where the size of the array and variables of array are already known, array literals can be used.

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
// Declaring array literal
```

The length of this array determines the length of the created array.

There is no need to write the new int[] part in the latest versions of Java

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

```
for (int i = 0; i<arr.length; i++)
  System.out.println("Element at index " + i + " : "+ arr[i]);
```

8. Multidimensional Arrays

Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension. Examples:

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

9. What is type casting? Explain different types.

Type casting is nothing but assigning a value of one primitive data type to another. When assign the value of one data type to another, be aware of the compatibility of the data type. If they are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not, then they need to be casted or converted explicitly.

There are two types of casting in Java as follows:

Widening Casting (automatically) –This type of casting takes place when two data types are automatically converted. It is also known as Implicit Conversion. This happens when the two data types are compatible and also when we assign the value of a smaller data type to a larger data type.

This involves the conversion of a smaller data type to the larger type size.

byte -> short -> char -> int -> long -> float -> double

Example

```
int i = 200;
 //automatic type conversion
long l = i;
 //automatic type conversion
float f = l;
```

Narrowing Casting (manually) –In this case, to assign a value of larger data type to a smaller data type, perform Explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.

This involves converting a larger data type to a smaller size type.

double -> float -> long -> int -> char -> short -> byte

Example

```
double d = 200.06;
//explicit type casting
long l = (long)d;
//explicit type casting
int i = (int)l;
```

10. What is variable?

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.

```
int data=50;//Here data is variable
```

There are three types of variables in java: local, instance and static.

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.

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.

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.

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

11. 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 comment

This 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");

12. 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 form also, 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;

13. What is the purpose of import statement?

The import statement can be used to import an entire package or sometimes import certain classes and interfaces inside the package. The import statement is written before the class definition and after the package statement (if there is any). Also, the import statement is optional.

the general form of the import statement:

import pkg1[.pkg2].(classname|*);

Here, pkg1 is the name of top-level package, and pkg2 is the name of subordinate package inside outer package separated by dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Now, you specify either

an explicit classname or a star (*), indicates that the Java compiler should import the full package. Below code fragment shows both forms in use :

import java.util.Date;

import java.io.*;

All the basic Java classes included with Java are stored in a package called java

14. Explain in detail the different types of operator in java.

Operators are the constructs which can manipulate the values of the operands. Consider the expression 2 + 3 = 5, here 2 and 3 are operands and + is called operator.

Java supports the following types of operators:

- Arithmetic Operators
- Assignment Operators
- Logical Operators
- Relational Operators
- Unary Operators
- Bitwise Operators
- Ternary Operators
- Shift Operators

Arithmetic Operators: Arithmetic Operators are used to perform mathematical operations like addition, subtraction, etc. Assume that A = 10 and B = 20 for the below table.

Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
– Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

Assignment Operators: An Assignment Operator is an operator used to assign a new value to a variable. Assume A = 10 and B = 20 for the below table.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

Relational Operators: These operators compare the values on either side of them and decide the relation among them. Assume A = 10 and B = 20.

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(A == B) is not true
!=	If the values of two operands are not equal, then condition becomes true.	(A != B) is true
>	If the value of the left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true
<	If the value of the left operand is less than the value of right operand, then condition becomes true.	(a < b) is true
>=	If the value of the left operand is greater than or equal to the value of the right operand, then condition becomes true.	(a >= b) is not true
<=	If the value of the left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true

Explain various logical and bitwise operators in java.

Logical Operators: The following are the Logical operators present in Java:

Operator	Description	Example
&& (and)	True if both the operands is true	a<10 && a<20

|| (or)	True if either of the operands is true	a<10 || a<20
! (not)	True if an operand is false (complements the operand)	!(x<10 && a<20)

Bitwise Operator: Bitwise operations directly manipulate bits. In all computers, numbers are represented with bits, a series of zeros and ones.

Operator	Description	Example
& (AND)	returns bit by bit AND of input	a&b
| (OR)	returns OR of input values	a|b
^ (XOR)	returns XOR of input values	a^b
~ (Complement)	returns the one's complement. (all bits reversed)	~a

```
int a = 58; //111010
    int b=13; //1101
System.out.println(a&b);  //returns 8 = 1000
System.out.println(a|b);  //63=111111
System.out.println(a^b);  //55=11011
System.out.println(~a);  //-59
```

Unary Operator :Unary operators are the one that needs a single operand and are used to increment a value, decrement or negate a value.

Operator	Description	Example
++	increments the value by 1. There is post-increment and pre-increment operators	a++ and ++a
—	decrements the value by 1. There is post decrement and pre decrement operators	a– or –a
!	invert a boolean value	!a

Ternary Operators in Java: The ternary operator is a conditional operator that decreases the length of code while performing comparisons and conditionals. This method is an alternative

for using if-else and nested if-else statements. The order of execution for this operator is from left to right.

Syntax:

(Condition) ? (Statement1) : (Statement2);

Condition: It is the expression to be evaluated which returns a Boolean value.

Statement 1: It is the statement to be executed if the condition results in a true state.

Statement 2: It is the statement to be executed if the condition results in a false state.

Shift Operators in Java: Shift operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number. There are three different types of shift operators, namely left shift operator()<<, signed right operator(>>) and unsigned right shift operator(>>>).

Syntax: number shift_opnumber_of_places_to_shift;

```
int a=58;

 System.out.println(a<<2); //232=11101000

System.out.println(a>>2);  //returns 14=1110
```

15. What is the difference between m++ and ++m statements in java?

m++ and ++mare very similar but not exactly the same. Both increment the number, but ++m increments the number before the current expression is evaluated, whereas m++ increments the number after the expression is evaluated.

```
int i = 3;
int a = i++; // a = 3, i = 4
int b = ++a; // b = 4, a = 4
```

16. Explain various data types used in java.

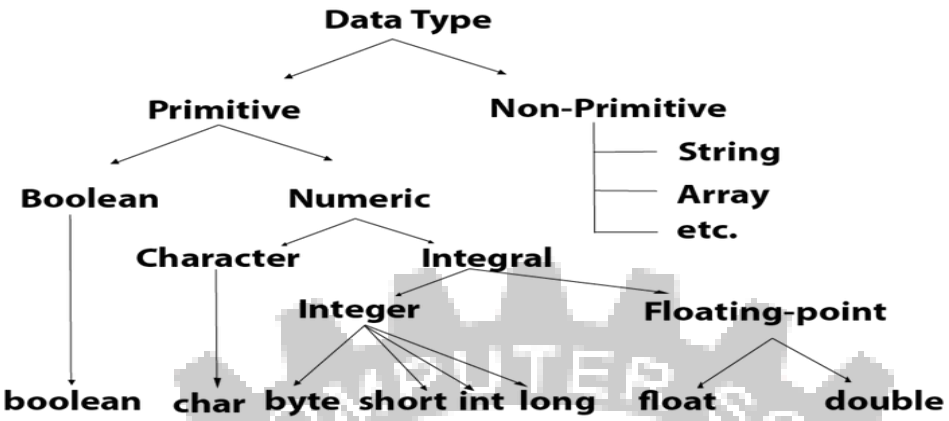
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

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.

17. Explain the primitive data types in java.

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.



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.
Example: Boolean one = false
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.
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 = -5000Example: byte a = 10, byte b = -20
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
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

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
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
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'

Non-primitive data types

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

```
class <class_name>{
  field;
  method;
}
```

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

```
interface <interface_name>{

  // declare constant fields
  // declare methods that abstract
  // by default.
}
```

array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

18. Explain the syntax of switch statement with example.

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

The syntax of the switch statement in Java is:

```
switch(expression) {
  case value :
    // Statements
    break; // optional

  case value :
    // Statements
    break; // optional

  // You can have any number of case statements.
default : // Optional
    // Statements
}
```

Example:

```
public class Test {
 public static void main(String args[]) {
    char grade = 'C';
 switch(grade) {
       case 'A' :
          System.out.println("Excellent!");
          break;
       case 'B' :
       case 'C' :
          System.out.println("Well done");
          break;
       case 'D' :
          System.out.println("You passed");
       case 'F' :
          System.out.println("Better try again");
          break;
default :
          System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

The switch statement work:

The expression is evaluated once and compared with the values of each case label.

If there is a match, the corresponding code after the matching case label is executed.

For example, if the value of the expression is equal to'D', the code after case 'D': is executed.

If there is no match, the code after default: is executed.

The following rules apply to a switch statement –

- The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

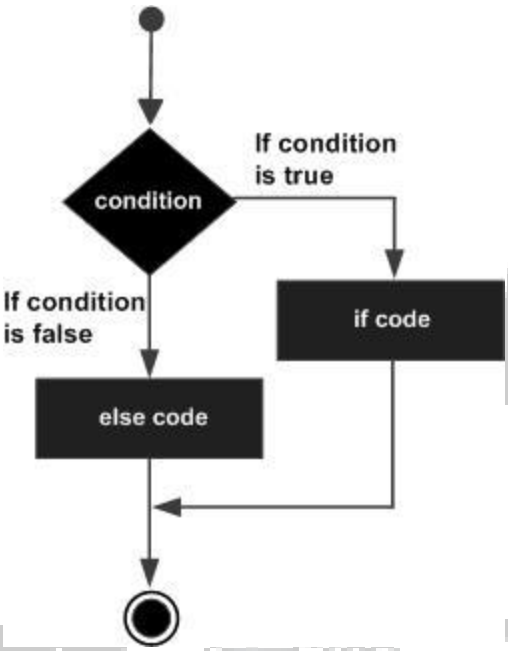
19. Explain the structure and function of if else statement with example.

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Following is the syntax of an if...else statement −

```
if(Boolean_expression) {
   // Executes when the Boolean expression is true
}
else {
   // Executes when the Boolean expression is false
}
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.




```
public class Test {
public static void main(String args[]) {
    int x = 30;
if( x< 20 ) {
System.out.print("This is if statement");
    }
else {
System.out.print("This is else statement");
    }
  }
}
```

20. Explain various looping structures in java.

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

There are three types of looping statements in Java. They are as follows:

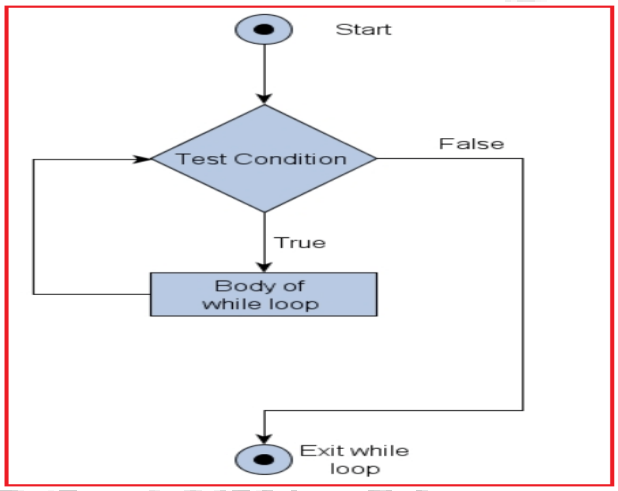
- While loop
- Do while loop
- For loop

21. Explain the structure and function of while loop construct with example.

while loop: A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
  loop statements...
}
```



While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop

Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.

When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example

```
class whileLoopDemo
{
   public static void main(String args[])
   {
      int x = 1;
  while (x <= 4)
      {
          System.out.println("Value of x:" + x);

          // Increment the value of x for
          // next iteration
          x++;
      }
   } }
```

22. Explain the working of while and do-While with syntax and example.

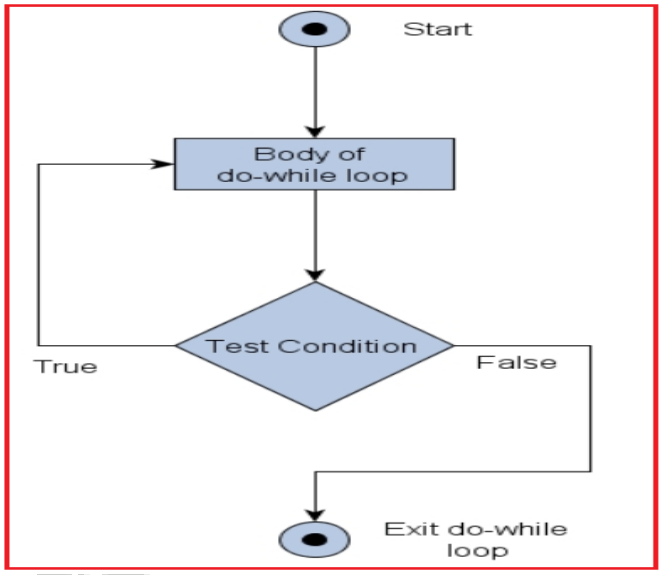
While loop working explained in above question

do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do
{
statements..
}
while (condition);
```



do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.

After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts. When the condition becomes false, the loop terminates which marks the end of its life cycle.

It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control.

Example

```
class dowhileloopDemo
{
   public static void main(String args[])
   {
```

```
        int x = 21;
        do
        {
           System.out.println("Value of x:" + x);
           x++;
        }
        while (x < 20);
    }
}
```

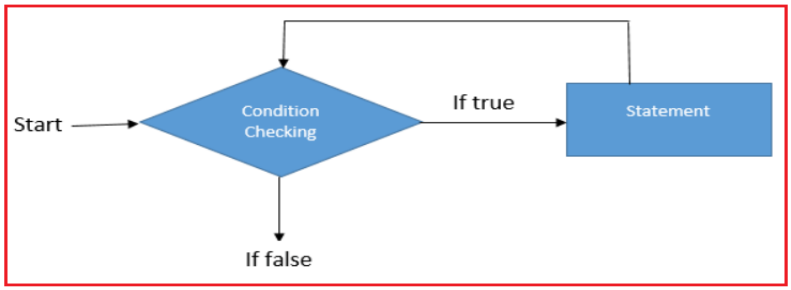
23. Explain structure and function of for statement in java with examples.

for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

The syntax to use for the loop is given below.

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

The initialization expression initializes the loop and is executed only once at the beginning when the loop begins. The termination condition is evaluated every loop and if it evaluates to false, the loop is terminated. And lastly, increment/decrement expression is executed after each iteration through the loop.



Example

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

24. Explain how data and methods are organized in an object oriented program with suitable illustration.

OOP languages involve using a series of classes to keep data and functions organized. Every class will contain variables and functions specific to that class that are called from elsewhere in the program where that class is used.

First an object has some properties and that properties represent the data

Example:- Student object has some properties like name,age,roll_noetc so that is data about the object.Let as see program

```
class Student
  {
      String name;
      int age;
      int roll_no;
  }
```

Now second come methods(functions),the methods are the operations which are performed on the data of the object.

Example:- student class has some function like getdetail,display etc.

```
class Student
 {
      String name;
      int age;
        void getdetail(){
        name="java";
        age=20;
        }
        void display(){ System.out.println("Name="+name+"Age="+age);}
        }
```

25. What are the two sections in class definition?

A class is a user defined data type which contains the set of similar data and the functions that the objects posses.A class serves as a blueprint or template for its objects.

Once a class has been defined, any number of objects belonging to that class can be created.The objects of a class are also known as instances or the variables of that class.

Syntax for defining a class as follows:

```
class class_name
{
        //variables declaration
        //methods declaration
}
```

Variables declared in the class are known as instance variables.
Variables and methods declared within the class are collectively known as members of the class.

Example

```
class Rectangle
{
        int length;
        int breadth;
        int area()
                {return(length*breadth);}
}
```

26. Explain about the classes inside classes.

Writing a class within another is allowed in Java. The class written within is called the nested class, and the class that holds the inner class is called the outer class.

Syntax

Following is the syntax to write a nested class. Here, the class Outer_Demo is the outer class and the class Inner_Demo is the nested class.

```
class Outer_Demo {
  class Inner_Demo {
  }
}
```

Nested classes are divided into two types

- Inner class\Non-static nested classes − These are the non-static members of a class.
- Static nested classes − These are the static members of a class.

Inner class means one class which is a member of another class. There are basically three types of inner classes in java.

1) Nested Inner class

Nested Inner class can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier.

```
class Outer {
  class Inner {
    public void show() { System.out.println("In a nested class method");  }
  }
}
class Main {
```

```
  public static void main(String[] args) {
Outer.Inner in = new Outer().new Inner();
in.show();
  }
}
```

2) Method Local inner classes

Inner class can be declared within a method of an outer class. In the following example, Inner is an inner class in outerMethod().

```
class Outer {
   void outerMethod()
    {  System.out.println("inside outerMethod");
      class Inner {
         void innerMethod() {  System.out.println("inside innerMethod"); }
      }
      Inner y = new Inner();
y.innerMethod();
    }
}
class MethodDemo {
   public static void main(String[] args) {
      Outer x = new Outer();
x.outerMethod();
   }
}
```

3) **Anonymous inner classes**

An inner class declared without a class name is known as an anonymous inner class. In case of anonymous inner classes, we declare and instantiate them at the same time. Generally, they are used whenever you need to override the method of a class or an interface. The syntax of an anonymous inner class is as follows –

Syntax

```
AnonymousInneran_inner = new AnonymousInner() {
  public void my_method() {
    ........
    ........
  }
};
```

Static nested classes

Static nested classes are not technically an inner class. They are like a static member of outer class.

```
class Outer {
  private static void outerMethod() {  System.out.println("inside outerMethod");   }

  // A static inner class
  static class Inner {
    public static void main(String[] args) {
      System.out.println("inside inner class Method");
outerMethod();
    }
  }

}
```

27. Explain constructor in java.

OR

28. 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();
   }
}
```

29. ## Explain the Basic structure of java program?

A Java program involves the following sections:
Documentation Section
Package Statement
Import Statements
Interface Statement
Class Definition
Main Method Class
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.

import 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 implement 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.

30. 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.

31. Differentiate between break and continue in java.

S.NO.	Break	Continue
01.	The break statement is used to terminate from the loop immediately.	The continue statement is used to skip the current iteration of the loop.
02.	break keyword is used to indicate break statements in java programming.	continue keyword is used to indicate continue statement in java programming.
03.	We can use a break with the switch statement.	We can not use a break with the switch statement.
04.	The break statement terminates the whole loop early.	The continue statement brings the next iteration early.
05.	It stops the execution of the loop.	It does not stop the execution of the loop.

32. Explain the syntax and function of break and continue statements?

The break and continue statements are the jump statements that are used to skip some statements inside the loop or terminate the loop immediately without checking the test expression. These statements can be used inside any loops such as for, while, do-while loop.

Break: The break statement in java is used to terminate from the loop immediately. When a break statement is encountered inside a loop, the loop iteration stops there, and control returns from the loop immediately to the first statement after the loop. Basically, break statements are used in the situations when we are not sure about the actual number of iterations for the loop, or we want to terminate the loop based on some condition.

Syntax:

break;

In Java, a break statement is majorly used for:

To exit a loop.

Used as a “civilized” form of goto.

Terminate a sequence in a switch statement.

example

```
for (int i = 0; i< 10; i++) {
        // Terminate the loop when i is 5
        if (i == 5)
           break;
        System.out.println("i: " + i);
     }
```

Continue: The continue statement in Java is used to skip the current iteration of a loop. We can use continue statement inside any types of loops such as for, while, and do-while loop. Basically, continue statements are used in the situations when we want to continue the loop but do not want the remaining statement after the continue statement.

Syntax:

```
continue;
```

Example

```
for (int i = 0; i< 10; i++) {
        // If the number is 2
        // skip and continue
        if (i == 2)
           continue;

System.out.print(i + " ");
     }
```

33. Explain the methods of StringBuffer class with example.

OR

34. Write a short note on StringBuffer class.

Java StringBuffer class is used to create mutable (modifiable) string.StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.

Important Constructors of StringBuffer class

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

append() method: is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

```
StringBuffer sb=new StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java
```

insert(int offset, String s)method: is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

```
StringBuffer sb=new StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello
```

replace(int startIndex, int endIndex, String str): is used to replace the string from specified startIndex and endIndex.

```
StringBuffer sb=new StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo
```

delete(int startIndex, int endIndex):is used to delete the string from specified startIndex and endIndex.

```
StringBuffer sb=new StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo
```

reverse():is used to reverse the string.

```
StringBuffer sb=new StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH
```

public intcapacity():is used to return the current capacity.The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the

capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//default 16

public intlength():is used to return the length of the string i.e. total number of characters.

StringBuffer s = new StringBuffer("Hello");

int p = s.length();

System.out.println("Length "+ p); //length 5

public Stringsubstring(int beginIndex):is used to return the substring from the specified beginIndex.

System.out.println("SubSequence = "+ str.substring(3)); // lo

35. What is this operator?

In java, this keyword is a reference variable that refers to the current object of a method or a constructor. The main purpose of using this keyword in Java is to remove the confusion between class attributes and parameters that have same names.
Following are various uses of 'this' keyword in Java:

- It can be used to refer instance variable of current class
- It can be used to invoke or initiate current class constructor
- It can be passed as an argument in the method call
- It can be passed as argument in the constructor call
- It can be used to return the current class instance

class Student{

int rollno;

String name;

Student(int rollno,String name){

this.rollno=rollno;

this.name=name;

}

36. What is finalize method in java?

It is a method that the Garbage Collector always calls just before the deletion/destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity. Clean-up activity means closing the resources associated with that object like Database Connection, Network Connection or we can say resource de-allocation. Remember it is not a reserved keyword.

Once the finalize method completes immediately Garbage Collector destroy that object.

finalize method is present in Object class and its **syntax** is:

protected void finalize throws Throwable{}

37. What is a final variable?

final variables are nothing but constants. We cannot change the value of a final variable once it is initialized.

A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to a different object.

```
public class Test {
    public static void main(String args[]) {
        final int i = 10;
i = 30; // Error because i is final.
    }
}
```

38. What are the uses of super keyword?

The super keyword in Java is a reference to the object of the parent/superclass. Using it, you can refer/call a field, a method or, a constructor of the immediate superclass.

super can be used to refer immediate parent class instance variable.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
```

In the above example, Animal and Dog both classes have a common property color. If we print color property, it will print the color of current class by default. To access the parent property, we need to use super keyword.

super can be used to invoke immediate parent class method.

```
class Animal{
```

```
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();  //invokes Animal class eat() method
bark();
}
}
```

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

super() can be used to invoke immediate parent class constructor.

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();  //invokes Animal class constructor
System.out.println("dog is created");
}
}
```

39. Explain static variables and methods with an example.

In java, static belongs to class. You can create static variables and static methods. You can call these directly by using class name, without creating instance.

Java static variables:

Static variables are belongs to the class and not to the object. These are only once, at the starting of the execution. Static variables are not part of object state, means there is only one copy of the values will be served to all instances. You can call static variable with reference to class name without creating an object. Below example shows how to create and call static variables.

Java static methods:

Static methods are also similar to static variables, you can access them with reference to class name, without creating object. Inside static methods, you cannot access instance variables or instance methods. You can only access static variables or static methods.

```
/Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
```

```
    String name;  //instance variable
    static String college = "ITS";  //static variable
 static void change(){       //static method to change the value of static variable
    college = "BBDIT";
    }
    //constructor to initialize the variable
Student(int r, String n){
rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to create and display the values of object

public class TestStaticMethod{
   public static void main(String args[]){
Student.change();//calling change method
    //creating objects
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    Student s3 = new Student(333,"Sonoo");
    //calling display method
    s1.display();
    s2.display();
    s3.display();
    }
}
```

40. Explain abstract classes and its characteristics with example.

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods.

abstract classes is a type of class in Java, that declare one or more abstract methods. A normal class cannot have abstract methods.

It needs to be extended and its method implemented. It cannot be instantiated.

Example of abstract class

```
abstract class A{}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.
Example of abstract method
abstract void printStatus();//no method body and abstract

Example

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle();
s.draw();
}
}
```

41. What the purpose is of extends keyword?

The extends keyword extends a class (indicates that a class is inherited from another class).
In Java, it is possible to inherit attributes and methods from one class to another.

- The extends is a Java keyword, which is used in inheritance process of Java. It specifies the super class in a class declaration using extends keyword. It is a keyword that indicates the parent class that a subclass is inheriting from. In Java, every class is a subclass ofjava.lang.Object.
- Extends keyword is also used in an interface declaration to specify one or more super interfaces.

```
public class A{

public int number;}

class B extends A {

public void increment() {

number++;      }
```

}

In this example, we inherit from class A, which means that B will also contain a field called number. Two or more classes can also be inherited from the same parent class.

42. Explain inheritance in java.

OR

43. Explain the different forms of inheritance with examples.

OR

44. Describe multiple and multilevel inheritance.

OR

45. Write a java program to implement multilevel inheritance

OR

46. Explain single inheritance with an example.

inheritance

It is the process of deriving a new class from the existing one in such a way that new class inherits all the members of the existing class.

The class that is inherited by other classes is called a base class or superclass or parent class.

The class which inherits the properties of base class is called derived class or subclass or child class.

The subclass inherits all the instance variables and methods of the superclass and also has its own members.

Defining a Subclass

The name of the superclass is specified in the subclass. A subclass is defined by using the extends keyword.

Syntax to define a subclass is as follows:

```
class subclass_name extends superclass_name
{
        //variables declaration
        //methods declaration
}
```

subclass_name=> name of the subclass

superclass_name=> name of the superclass

extends=> it is a keyword that indicates the superclass properties have been extended to the subclass.

Types of inheritance

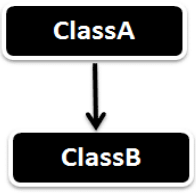
Inheritance allows the creation of a logical relationship between two or more classes. Depending on the number of classes involved and the way the classes are inherited, inheritance may take different forms, namely

Single inheritance
Multilevel inheritance
Hierarchical inheritance
Multiple inheritance

In java it is not possible to implement multiple inheritance directly, there is a concept known as interface through which multiple inheritance implemented.

Single Inheritance:

Single Inheritance is the simple inheritance of all, When a class extends another class(Only one class) then we call it as Single inheritance. The below diagram represents the single inheritance in java where Class B extends only one class Class A. Here Class B will be the Sub class and Class A will be one and only Super class.



Syntax:

```
class classA
{
 // members
}
class classB extends classA
{
 //members
}
```

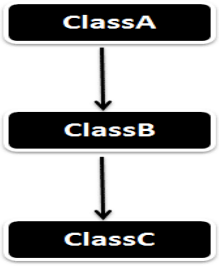
Example

```
class Shape{
   void draw()
   {
      System.out.println("Draw shape");
   }
}
class Circle extends Shape
{
   void drawCircle(){
      System.out.println("Draw Circle");
   }
 public static void main(String args[])
{
```

```
    Circle c  = new Circle();
c.draw();
c.drawCircle();
}
}
```

Multilevel Inheritance:

In Multilevel Inheritance a derived class will be inheriting a parent class and as well as the derived class act as the parent class to other class. As seen in the below diagram. ClassB inherits the property of ClassA and again ClassB act as a parent for ClassC. In Short ClassA parent for ClassB and ClassB parent for ClassC.



Syntax:

```
class classA
{
 // members
}
class classB extends classA
{
 //members
}
class classC extends classB
{
 //members
}
```

Example

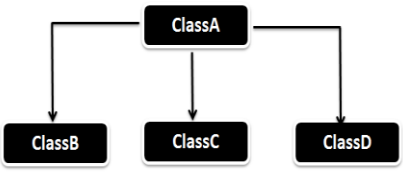
```
public class ClassA
{
   public void dispA()
   {
       System.out.println("disp() method of ClassA");
   }
}
public class ClassB extends ClassA
{
```

```
    public void dispB()
    {
        System.out.println("disp() method of ClassB");
    }
}
public class ClassC extends ClassB
{
    public void dispC()
    {
        System.out.println("disp() method of ClassC");
    }
    public static void main(String args[])
    {
        //Assigning ClassC object to ClassC reference
ClassC c = new ClassC();
        //call dispA() method of ClassA
c.dispA();
        //call dispB() method of ClassB
c.dispB();
        //call dispC() method of ClassC
c.dispC();
    }
}
```

Hierarchical Inheritance:

When one single class is inherited by multiple subclasses is known as hierarchical inheritance.

As per the below example ClassA will be inherited by ClassB, ClassC and ClassD. ClassA will be acting as a parent class for ClassB, ClassC and ClassD.



Syntax

```
class classA
{
 // members
}
class classB extends classA
{
```

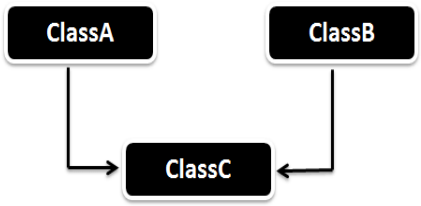
```
 //members
}
class classC extends classA
{
 //members
}
```

Example

```
class Laptop
{
   void display()
   {
      System.out.println("Working...");
   }
class Dell extends Laptop
{
   void print()
   {
      System.out.println("Dell Inspiron");
   }
}
class Lenovo extends Laptop
{
   void show()
   {
      System.out.println("Lenovo YOGA");
   }
}
class TestDemo
{
   public static void main(String args[]){
      Dell d = new Dell();
d.print();
d.display();
      Lenovo l = new Lenovo();
l.show();
l.display();
   }
}
```

Multiple Inheritance

Multiple Inheritance is nothing but one class extending more than one class. Multiple Inheritance is basically not supported by many Object Oriented Programming languages such as Java, Small Talk, C# etc.. (C++ Supports Multiple Inheritance). As the Child class has to manage the dependency of more than one Parent class. But you can achieve multiple inheritance in Java using Interfaces.



47. What is an interface? How interface helps to implement multiple inheritance in java.

OR

48. How multiple inheritance can be implemented in java?

we cannot achieve multiple inheritance in the case of class because of the ambiguity problem. but using the interface we can achieve multiple inheritance in java.

An interface in java is syntactically similar to classes, but they do not have instance variable and their method are declared without any body.

The methods declared inside an interface are by default abstract(only method signature, not implementation) and the variables declared inside an interface are public, static, and final by default.

The interface in java is a mechanism to achieve abstraction. Once an interface is defined, any number of classes can implement this interface. Also, one class can implement any number of interfaces.

define an Interface.

An Interface is defined much like a class. It can have methods and variables like class. But in Interface, all the methods are by default abstract and all the variables are by default public static, and final. This is the general form of an interface:

```
interface name{
  return-type method-name1(parameter-list);
  return-type method-name2(parameter-list);
  type varname1 = value;
  type varname2 = value;
  //.....
  return-type method-nameN(parameter-list);
  type varnameN = value;
}
```

to implement an Interface.

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and create the methods defined by the interface. The general form of a class that includes the implements clauses look like this:

```
class classname implements interface-name{
   // class body
}
```

Example

```
interface Area
{
 float p=3.14;
void compute();
}
class circle implements Area
{
 float r=2.2F;
 public void compute()
{double c=p*r*r;
 System.out.println("area of circle="+c);
}
}
class sphere implements Area
{
 float r=3.7;
public void compute()
{double s=4*p*r*r;
        system.out.println("area of sphere="+s);
}
}
class inter
{
public static void main(String args[])
{
cirle c1=new circle();
 sphere s1=new sphere();
```

```
Area obj;
obj=c1;
obj.compute();
obj=s1;
obj.compute();
}
}
```

49. Explain about the visibility control in java.

OR

50. Discuss the different levels of access protection available in java.

The class members are accessible everywhere in the program and the subclass can inherit all the variables and methods of the superclass by using the keyword extends.

There might be certain situations when you want to restrict the accessibility of members of a class for various reasons.

Java provides three types of visibility controls, namely public, private and protected. They are also known as access modifiers.

Public: When a member of a class is declared as public, it can be accessed everywhere in the program.

Private: A member declared as private can be accessed only within a class.

Protected: A member declared as protected is accessible not only to all the classes in the same package but also to subclasses in other packages.

If no visibility control is specified, by default the data member of a class is visible only within the same package.

The following table summarizes the visibility provided by various access modifiers.

	Same Class	Same Package - Sub Class	Same Package - Non Sub Class	Different Package - Sub Class	Different Package - Non Sub Class
public	Yes	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	Yes	No
default(package)	Yes	Yes	Yes	No	No
private	Yes	No	No	No	No

Example

```
class visibility
{
  int x;                 //default variable
  public int y;         //public variable
  private int z;       // private variable
```

```
        int data(int a)
        {       z=a;
                return z;       }
}
Class test
{
 public static void main(String args[])
 {
        visibility obj=new visibility();
        obj.x=10;
        obj.y=20;
        int k=obj.data(30);
System.out.println("x="+obj.x+"y="+obj.y+"z="+k);
  }
}
```

51. What is Package? Explain the steps for creating a package and create a user defined package.

PACKAGE in Java is a collection of classes, sub-packages, and interfaces. It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve code reusability.

Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.

Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each Java package.

Syntax: -

```
package nameOfPackage;
```

The steps of creating a package.

- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

Step 1) Consider the following package program in Java:

```
package p1;
```

```
class c1(){

public void m1(){System.out.println("m1 of c1");}

public static void main(string args[]){

c1 obj = new c1();

obj.m1();}       }
```

Here,To put a class into a package, at the first line of code define package p1.

Next create class c1 with method m1 and call this method using object obj .

Step 2) In next step, save this file as demo.java

Step 3) In this step, we compile the file.

```
javac demo.java
```

The compilation is completed. A class file c1 is created. However, no package is created? Next step has the solution

Step 4) Now we have to create a package, use the command

```
javac –d . demo.java
```

This command forces the compiler to create a package.

The "." operator represents the current working directory.

Step 5) When you execute the code, it creates a package p1. When you open the java package p1 inside you will see the c1.class file.

Step 6) Compile the same file using the following code

```
javac –d .. demo.java
```

Here ".." indicates the parent directory.

Step 7) Now let's say you want to create a sub package p2 within our existing java package p1. Then we will modify our code as

```
package p1.p2;

class c1{

public void m1() {System.out.println("m1 of c1");}
```

}

to Import Package

Syntax

import packageName;

Once imported, you can use the class without mentioning its fully qualified name.

Step 1) Copy the code into an editor.

```
package p3;

import p1.*; //imports classes only in package p1 and NOT  in the sub-package p2

class c3{

 public   void m3(){ System.out.println("Method m3 of Class c3");  }

 public static void main(String args[]){

  c1 obj1 = new c1();

  obj1.m1();

 }}
```

Step 2) Save the file as Demo2.java. Compile the file using the command

javac –d . Demo2.java

Step 3)Execute the code using the command java p3.c3

52. Write a note on java API packages

Java APl(Application Program Interface) provides a large numbers of classes grouped into different packages according to functionality. Most of the time we use the packages available with the the Java API.

Java System Packages and Their Classes

java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.

java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

53. Differentiate method overloading and method overriding with examples for each.
    OR
54. With suitable examples explain the difference between method overriding and method overloading.

Method Overloading	Method Overriding
Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.

In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

Java Method Overloading example

```
class OverloadingExample{

static int add(int a,int b){return a+b;}
static int add(int a,intb,int c){return a+b+c;}
}
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```