Unit-1

1.Discuss about brief History of Java.

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems (WORA-Write Once Run Anywhere). The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called *OpenJDK*.

A Java distribution typically comes in two flavors, the Java Runtime Environment (JRE) and the Java Development Kit (JDK).

The JRE consists of the JVM and the Java class libraries. Those contain the necessary functionality to start Java programs.
It is Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries.

2. What are the Types of Java Applications ?
There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application
Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application
An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application
An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.
Examples:
- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games
- Big Data Applications
- AI
- IOT applications

3. List various Editions and versions of Java

1)Java Micro Edition was created to support applications running on mobile and on embedded devices.
2)Java Standard Edition and Java Enterprise Edition are heavily used worldwide. Together, they are used in various kinds of solutions like web applications, applications servers, big data technologies and so on.
3) Java EE (Java Enterprise Edition)
It is an enterprise platform which is mainly used to develop web and enterprise applications. It is built on the top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc
4) JavaFX: It is used to develop rich internet applications. It uses a light-weight user interface API.

Java Versions

JDK Alpha and Beta (1995)
The Java Alpha and Beta was the first releases but they have highly unstable APIs and ABIs. The supplied Java web browser was named WebRunner.

JDK 1.0 (January 23, 1996)
It was the first stable released version of Java. Its codename was Oak.The first stable version of JDK was JDK 1.0.2 and it was called Java 1.Up to JDK 1.0.1, private and protected keywords could be used together to create yet another form of protection which used to act as a restriction to methods or variables mainly to subclasses of a given class. In JDK 1.0.2, this capability has been removed.
JDK 1.1 (February 19, 1997)
Some additions were included to this version. i.e.
- The concept of Inner Class

- JavaBeans
- JDBC
- RMI
- AWT event model was totally reshaped.
- Reflection(which supported Introspection only, modification was not possible at runtime).
- JIT(Just In Time) compiler on Microsoft Windows platforms, produced for JavaSoft by Symantec

J2SE 1.2 (December 8, 1998)

Its codename was Playground. First time, it was called J2SE (Java 2 Platform, Standard Edition) .It replaced JDK to recognize the base platform from J2EE (Java 2 Platform, Enterprise Edition) and J2ME(Java 2 Platform, Micro Edition) .It was a very important java release as it tripled the size of the Java platform to 1520 classes in 59 packages.
Some additions were included to this version. i.e.

- Java plug-in
- Java IDL, an IDL implementation for CORBA interoperability
- Collections framework
- the Swing graphical API was integrated into the core classes
- Sun's JVM was equipped with a JIT compiler for the first time

J2SE 1.3 (May 8, 2000)

Its codename was Kestrel. Some additions were included to this version. i.e.

- HotSpot JVM included.
- RMI was modified to support optional compatibility with CORBA.
- JNDI (Java Naming and Directory Interface).
- Java Platform Debugger Architecture (JPDA) included.
- JavaSound.
- Synthetic proxy classes.

J2SE 1.4 (February 6, 2002)

Its codename was Merlin. It was the first Java platform which was released under the Java Community Process.

- Improved libraries.
- Perl regular expressions included.
- Provided exception chaining (It allows an exception to encapsulate original lower-level exception).
- IPv6 support (Internet Protocol version 6).
- Logging API (Specified in JSR 47.)
- Image I/O API for reading and writing images in formats like JPEG and PNG.
- XML parser and XSLT processor integrated.
- Security and cryptography extensions (JCE, JSSE, JAAS) integrated.
- Support and security updates for Java 1.4 ended in October 2008.

J2SE 5.0 (September 30, 2004)
Its codename was Tiger. It was originally numbered 1.5, which is still used as the internal version number. So, it was changed to 5.0 to "better reflect the level of maturity, stability, scalability and security of the J2SE". This process also was released under the Java Community Process.
J2SE 5.0added some significant new language features:

It provided compile-time (static) type safety for collections and eliminates the need for most typecasts.

- Used Metadata or annotations.
- Autoboxing/unboxing.
- Enumerations.
- Enhanced for each loop.
- Improved semantics of execution for multi-threaded Java programs.
- Static imports.
- Scanner class for parsing data from various input streams and buffers.

Java SE 6 (December 11, 2006)
Its codename was Mustang. After the release of this version, Java replaced the name J2SE to Java SE and dropped the .0 from the version number.
Some additions were included to this version. i.e.

- Dropped the support for older Win9x versions.
- Scripting Language Support.
- Generic API for tight integration with scripting languages.
- Improved Web Service support.
- JDBC 4.0 support.

Use a Java Compiler API to invoke a Java Compiler programmatically.
After the release of Java 6, Sun released many updates to fix bugs.
Java SE 7 (July 28, 2011)
Its codename was Dolphin. It was launched on 7, July 2011 but was made available for developers on July 28, 2011.
Some additions were included to this version. i.e.

- JVM support for dynamic languages.
- Compressed 64-bits pointer.
- Strings added in switch.
- Automatic resource management in try-statement.
- Underscores allowed in numeric literals.
- Binary integer literals.
- Improved type interface for creating generic instance. (also called diamond operator <>)
- Improved catching and throwing. (catch multiple exceptions and rethrow with improved type checking)
- Provided Java Deployment rulesets.

- It was the default version to download on java.com from April 2012 up to the release of Java 8.

Java SE 8 (March 18, 2014)
Its codename was Spider. Although, codenames have been discontinued, but the codename Spider is common among java developers.

It includes some features which were proposed for Java SE 7 but added in Java SE 8.
- Language-level support for Lambda expressions.
- Allowed developers to embed JavaScript code within applications.
- Annotation of Java Types.
- Provided Date and Time API.
- Repeating Annotations.
- Launching of JavaFX applications.
- Removal of permanent generation.

Java SE 8 is not supported in Windows XP but after JDK 8 update 25, we can install and run it under Windows XP.

Java 8 is set as a default version to download from java.com from October 2014.

Java SE 9 (September 21, 2017)
Java SE 10 (March, 20, 2018)
Java SE 11 (September, 25th 2018)
Java SE 12(March, 19th 2019)
Java SE 13(September, 17th 2019)
Java SE 14(March, 17th 2020)
Java SE 15(September, 15th 2020)
Java SE 16(March, 16th 2021)
Java SE 17(Expected on Sept. 2021)

4. Discuss about Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java buzzwords.

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

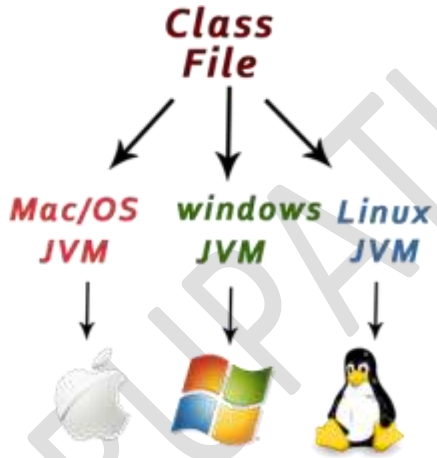
- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Platform Independent

Java is platform independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.



The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

Runtime Environment

- API(Application Programming Interface)
- Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured
Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox
- Classloader: Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects.
- Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Robust
Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral
Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable
Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance
Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed
Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded
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 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).

5. Explain about Java Program Structure
It contains following parts

- Documentation Section
- Package Statement
- Import Statement
- Interface Section
- Class Definition
- Main Method Class

Documentation Section

It is used to improve the readability of the program. It consists of comments in Java which include basic information such as the method's usage or functionality to make it easier for the programmer to understand it while reviewing or debugging the code .The compiler ignores these comments during the time of execution. A Java comment is not necessarily limited to a confined space, it can appear anywhere in the code.

There are three types of comments that Java supports

Single line Comment
Multi-line Comment
Documentation Comment

```
// a single line comment is declared like this
/* a multi-line comment is declared like this
and can have multiple lines as a comment */
/** a documentation comment starts with a delimiter and ends with */
```

Package Statement
There is a provision in Java that allows you to declare your classes in a collection called package. There can be only one package statement in a Java program and it has to be at the beginning of the code before any class or interface declaration. This statement is optional, for example, take a look at the statement below.

```
package student;
```

This statement declares that all the classes and interfaces defined in this source file are a part of the student package. And only one package can be declared in the source file.

import Statement
Many predefined classes are stored in packages in Java, an import statement is used to refer to the classes stored in other packages. An import statement is always written after the package statement but it has to be before any class declaration.

```
import java.util.Date; //imports the date class
import java.applet.*;  //imports all the classes from the java applet package
```

Interface Section

This section is used to specify an interface in Java. It is an optional section which is mainly used to implement multiple inheritance in Java. An interface is a lot similar to a class in Java but it contains only constants and method declarations.

An interface cannot be instantiated but it can be implemented by classes or extended by other interfaces.

```
interface stack{
void push(int item);
void pop();
}
```

Class Definition

A Java program may contain several class definitions, classes are an essential part of any Java program. It defines the information about the user-defined classes in a program.

A class is a collection of variables and methods that operate on the fields. Every program in Java will have at least one class with the main method.

Main Method Class

The main method is from where the execution actually starts and follows the order specified for the following statements. Let's take a look at a sample program to understand how it is structured.

```
public class Example{
//main method declaration
public static void main(String[] args){
System.out.println("hello world");
}
}
```

public static void main

- When **the main method is declared public**, it means that it can be used/called outside of this class as well.(by JVM)

- The word **static** means that we want to access a method without making its objects. As we call the main method without creating any objects.

- The word **void** indicates that it does not return any value. The main is declared as void because it does not return any value.

main is the method, which is an essential part of any Java program , starting point of any java program or program.

- String[] args

It is an array where each element is a string, which is named as args. If you run the Java code through a console, you can pass the input parameter. The main() takes it as an input.

6. What are various data Types and operators available in Java?

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.

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

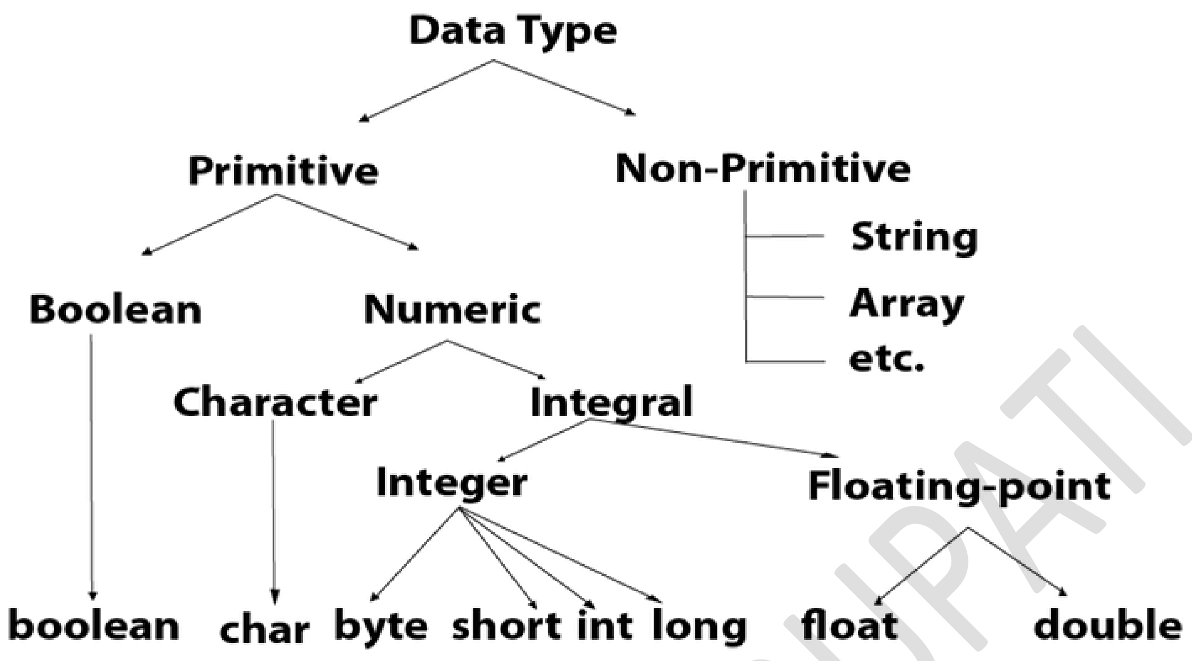
A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

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

3) Static variable

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

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



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits

boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Operators in Java

Operator Type	Category	Precedence
Unary	postfix	`expr++ expr--`
	prefix	`++expr --expr +expr -expr ~ !`
Arithmetic	multiplicative	`* / %`
	additive	`+ -`
Shift	shift	`<< >> >>>`
Relational	comparison	`< > <= >= instanceof`
	equality	`== !=`
Bitwise	bitwise AND	`&`
	bitwise exclusive OR	`^`
	bitwise inclusive OR	`|`
Logical	logical AND	`&&`
	logical OR	`||`
Ternary	ternary	`? :`
Assignment	assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

7. List out various Keywords /Reserved Words

List of keywords in the Java programming language. You cannot use any of the following as **identifiers in your programs**. The keywords const and goto are reserved, even though they are not currently used. true, false, and null might seem like keywords, but they are actually literals; you cannot use them as **identifiers in your programs**.

abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

8. The ? : Operator (ternary or conditional operator)

conditional operator **? :**

(Conditional Exp1) ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions.

To determine the value of the whole expression, initially conditional exp1 is evaluated.

If the value of exp1 is true, then the value of Exp2 will be the value of the whole expression.

If the value of exp1 is false, then Exp3 is evaluated and its value becomes the value of the entire expression.

Ex:- min=(a<b) ? a : b;

9. Explain about various Control Flow Statements in Java

The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

- the **decision-making statements** (if-then, if-then-else, switch),
- the **looping statements** (for, while, do-while), and
- the **branching statements** (break, continue, return) supported by the Java programming language.

I Decision Making Statements

Java If-else Statement
The Java if statement is used to test the condition. It checks boolean condition: true or false. There are various types of if statement in Java.

a) if statement

syntax:

```
if (Boolean_expression) {

  // Statements will execute if the Boolean expression is true
    }
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Ex:-

```
public class Test {
  public static void main(String args[]) {
    int x = 10;
    if( x < 20 ) {
      System.out.print("This is if statement");
    }   }}
```

b) if- else- statement

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

Example:-

```
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");
      }  } }
```

If-else-if statement.

Syntax:

```
if(Boolean_expression 1) {
   // Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2) {
   // Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3) {
   // Executes when the Boolean expression 3 is true
}else {
   // Executes when the none of the above condition is true.
}
```

Switch Statement

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.

Syntax

The syntax of enhanced for loop is −

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

   case value :
      // Statements
      break; // optional

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

```
  // Statements
}
```

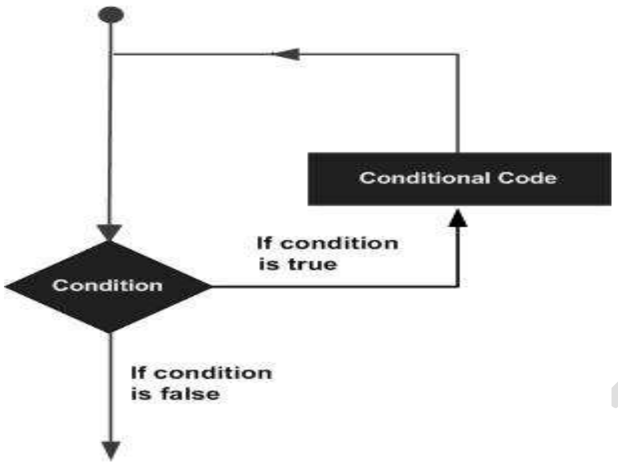
- The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type 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.

Example:-

```
public class Test {

  public static void main(String args[]) {
    // char grade = args[0].charAt(0);
    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);
  }
}
```

II Loop or repetitive Statements

Loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.



Java programming language provides the following types of loop statements to handle looping requirements..

1 **while loop**
2 **for loop**
3 **do… while loop**

1.While Loop
Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Syntax
The syntax of a while loop is −

```
while(Boolean_expression) {
  // Statements
}
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.

When executing, if the boolean_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Example:-

```
public class Test {
   public static void main(String args[]) {
      int x = 10;
      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }}
```

2 for loop

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. A for loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax

The syntax of a for loop is −

```
for(initialization; Boolean_expression; update) {
   // Statements
}
```

Example:

```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x + 1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

3 do...while loop

Like a while statement, except that it tests the condition at the end of the loop body.

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

Following is the syntax of a do...while loop −

```
do {
   // Statements
}while(Boolean_expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested. If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public class Test {
  public static void main(String args[]) {
    int x = 10;
    do {
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }while( x < 20 );
  }
}
```

III branching statements (break, continue, return) supported by the Java programming language.

break; statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }        System.out.println(i);
   }  }  }
```

Java Break Statement with Labeled For Loop

We can use break statement with a label. This feature is introduced since JDK 1.5. So, we can break any loop in Java now whether it is outer loop or inner.

Ex:-

```
aa:
        for(int i=1;i<=3;i++){
                bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        //using break statement with label
                        break aa;
                    }
                    System.out.println(i+" "+j);
                }
            }
```

continue Statement
The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
Ex:-

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
    //for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //using continue statement
            continue;//it will skip the rest statement
        }
        System.out.println(i);
    }
}
}
```

- Java Continue Statement with Labeled For Loop

We can use continue statement with a label. This feature is introduced since JDK 1.5. So, we can continue any loop in Java now whether it is outer loop or inner.

10. Explain about Arrays in Java ?

Java Array is a very common type of data structure which contains all the data values of the same data type. The data items put in the array are called elements and the first element in the array starts with index zero. Arrays inherit the object class and implement the serializable and cloneable interfaces. We can store primitive values or objects in an array.

Array Variables in program are created in following way

1) Declaring your Array
Syntax:
<elementType>[] <arrayName>;
or

<elementType> <arrayName>[];

Ex:-

int intArray[];

// Defines that intArray is an ARRAY variable which will store integer values

int []intArray;

2) Constructing an Array

arrayname = new dataType[]

Example:

intArray = new int[10]; // Defines that intArray will store 10 integer values

Declaration and Construction combined

int intArray[] = new int[10];

3) Initialize an Array

intArray[0]=1; // Assigns an integer value 1 to the first element 0 of the array

intArray[1]=2; // Assigns an integer value 2 to the second element 1 of the array

Declaring and initialize an Array

[] = {};

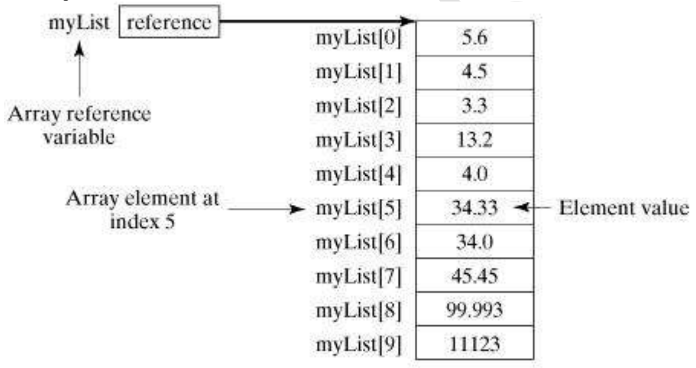
int intArray[] = {1, 2, 3, 4};

Processing Arrays

When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Each element in an array is accessed by using

Arrayname[index]= value;



Example:-

```
class ArrayDemo{
    public static void main(String args[]){
      int array[] = new int[7];
      for (int count=0;count<7;count++){
         array[count]=count+1;
      }
     for (int count=0;count<7;count++){
         System.out.println("array["+count+"] = "+array[count]);
      }
      //System.out.println("Length of Array  =  "+array.length);
      // array[8] =10;
      }}
```

Multidimensional arrays
Multidimensional arrays are actually arrays of arrays.
To declare a multidimensional array variable, specify each additional index using another set of square brackets.

Ex: int twoD[ ][ ] = new int[4][5] ;
When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.

You can allocate the remaining dimensions separately.
Example:

```
public class Guru99 {
public static void main(String[] args) {

// Create 2-dimensional array.
  int[][] twoD = new int[4][4];
  // Assign three elements in it.
  twoD[0][0] = 1;
  twoD[1][1] = 2;
  twoD[3][2] = 3;
  System.out.print(twoD[0][0] + " ");}}
```

The foreach Loops
JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.
Example
The following code displays all the elements in the array myList −

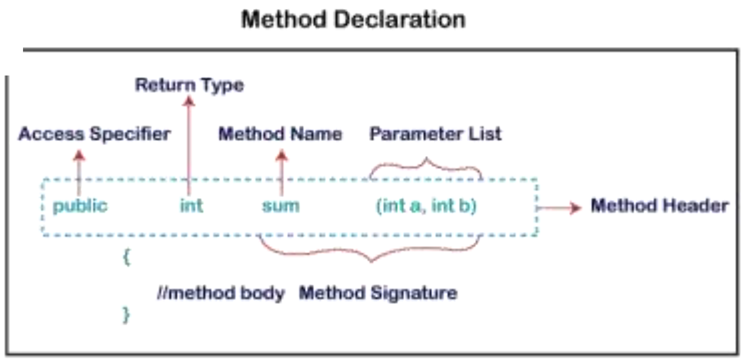
```
public class TestArray {
   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};
      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }   }}
```

11.Discuss about Java Methods(Functions)?

A **method** is a block of code which only runs when it is called.You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as **functions**.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**.



Ex:-

```
public class Main {

  static void myMethod() {

    // code to be executed

  }}
```

Call a Method

To call a method in Java, write the method's name followed by two parentheses () and a semicolon;

```
public class Main {

  static void myMethod() {

    System.out.println("I just got executed!");

  }

  public static void main(String[] args) {

    myMethod();

  }}
```

12. Discuss about Java OOPs Concepts

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion
- Association
- Aggregation
- Composition

Object
Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

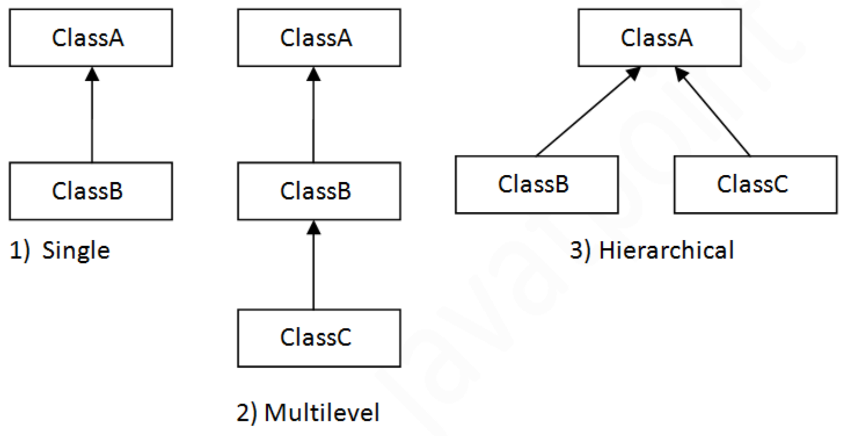
An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

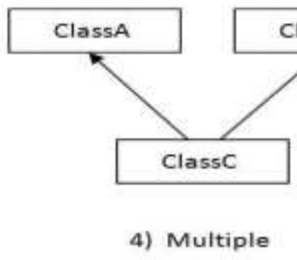
Class
Class are a blueprint or a set of instructions to build a specific type of object. It is a basic concept of Object-Oriented Programming which revolve around the real-life entities. Class in Java determines how an object will behave and what the object will contain. Class doesn't consume any space.

Inheritance
When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism. Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

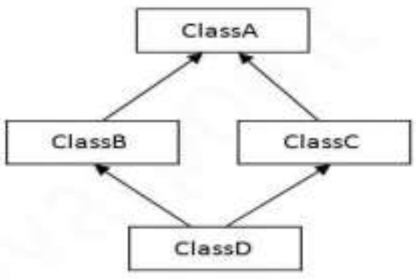
The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
Types of Inheritance



1. Single Level Inheritance
2. Hierarchical Inheritance
3. Multilevel-Inheritance
4. Hybrid Inheritance



4) Multiple



5) Hybrid

Polymorphism
If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction
Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation
Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling
Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cohesion
Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association
Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

Aggregation
Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.
Composition
The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

13.Explain about class and object in Java
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.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

Class in created by class keyword as

```
Access specifer class class_name [extents/implements]  class name or interface(s)
{
        Access-specifier Datatype data member;
        Access-specifier Datatype data member;
        Access-specifier Datatype data member;

        Access specifier  memberfunction-1(arg1,arg2…)
        { }

        Access specifier  memberfunction-2(arg1,arg2…)
        { }


}
```

Instance variable in Java
A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- **Declaration** − A variable declaration with a variable name with an object type.

  Classname object_name1,object_name2;

- **Instantiation** − The 'new' keyword is used to create the object.

  obj = new classname();

- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

  Classname object=new Classname(arg1,arg2…);

```
Ex:- public class Student {
   public Student(String name) {
      // This constructor has one parameter, name.
      System.out.println("Student Name is :" + name );
   }
   public static void main(String []args) {
    // Following statement would create an object myStudent
Student  myStudent = new Student( "John" );
  }
}
```

Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path −

```
/* First create an object */
ObjectReference = new Constructor();
/* Now call a variable as follows */
ObjectReference.variableName;
/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Class Attributes

"variable" for x in the example (as shown below). It is actually an attribute of the class. Or you could say that class attributes are variables within a class:

```
public class Main {
  int x = 5;
  int y = 3;}
```

You can access attributes by creating an object of the class, and by using the dot syntax (.):

14.Discuss about Static keyword.

static keyword is mainly used for memory management. It can be used with variables, methods, blocks and nested classes. It is a keyword which is used to share the same variable or method of a given class. Basically, static is used for a constant variable or a method that is same for every instance of a class. The main method of a class is generally labeled static.

In Java programming language, static keyword is a non-access modifier and can be used for the following:

- Static Block
- Static Variable
- Static Method
- Static Classes

A method that **has static** keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword static before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the main() method.

Example of static method

Display.java

```
public class Display
{
public static void main(String[] args)
{
show();
}
static void show()
{
System.out.println("It is an example of static method.");
}  }
```

Static Block

If you need to do the computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded. Take a look at the below Java program to understand the usage of Static Block.

Ex:

```
import java.util.*;
public class BlockExample{
// static variable
static int j = 10;
static int n;

// static block
static {
System.out.println("Static block initialized.");
n = j * 8;
}

public static void main(String[] args)
{
System.out.println("Inside main method");
System.out.println("Value of j : "+j);
System.out.println("Value of n : "+n);
}
}
```

Static Variable

When you declare a variable as static, then a single copy of the variable is created and divided among all objects at the class level. Static variables are, essentially, global variables. Basically, all the instances of the class share the same static variable. Static variables can be created at class-level only.

```
class Student{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
}
```

15. Explain about Java Constructors

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.
It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
.
Rules for creating Java constructor
There are two rules defined for the constructor.

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized
- Always constructor is public
- Constructor is cannot be called explicitly

There are two types of constructors in Java:
Default Constructor" (no-argument constructor), and
parameterized constructor

A constructor is called "**Default Constructor"** when it doesn't have any parameter.

Syntax of default constructor:
<class_name>(){}
Example:-

```
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

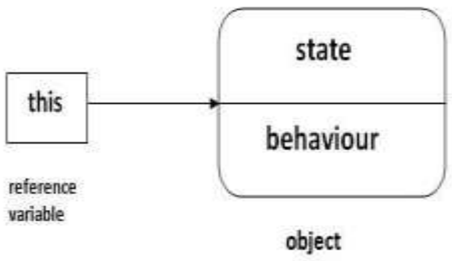
Note:- If there is no constructor in a class, compiler automatically creates a default constructor.

Java Parameterized Constructor
A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```
class Student4{
   int id;
   String name;
   //creating a parameterized constructor
   Student4(int i,String n){
   id = i;
   name = n;
   }
   //method to display the values
   void display(){System.out.println(id+" "+name);}
public static void main(String args[]){
   //creating objects and passing values
   Student4 s1 = new Student4(111,"Karan");
   Student4 s2 = new Student4(222,"Aryan");
   //calling method to display the values of object
   s1.display();
   s2.display();
  }
}
```

16. Write about this keyword in java
There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.



Usage of java this keyword
Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.

2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Ex:-

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
```

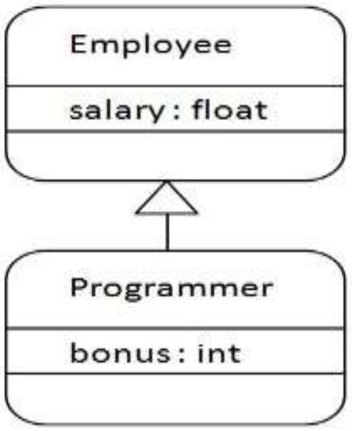
17. Discuss about INHERITANCE in Java.

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

syntax of Java Inheritance

```
class Subclass-name extends Superclass-name

{    //methods and fields

}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass



```
class Employee{

 float salary=40000;

}

class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}  }
```

For more examples refere : https://www.javatpoint.com/inheritance-in-java

- **To reduce the complexity and simplify the language, multiple inheritance is not supported in java**

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee{

int id;

String name;

Address address;//Address is a class

...  }
```

Code reuse is also best achieved by aggregation when there is no is-a relationship.

Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

18. Discuss about Method Overloading ?

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Method overloading increases the readability of the program.Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments
- By changing the data type

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

```
class Adder{

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

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

}

class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
 }}
```

19. Discuss about Method Overiding in JAVA.

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- **Method overriding is used for runtime polymorphism**

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance)

```
//Java Program to illustrate the use of Java Method Overriding

//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

20. Write about Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer immediate parent class object.Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor

```
class Animal{
Animal(String c)
{}
String color="white";
Void show() {}
}
```

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

21. Discuss about Java FINAL keyword.

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Ex:- final double PI=3.14;

2) Java final method

<u>If you make any method as final, you cannot override it.</u>

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}    --→  ERROR
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

final class Bike{}

```
class Honda1 extends Bike{   ---→  ERROR
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

22. What are Abstract classes.Explain?

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 (method with the body).

Abstraction is a process of hiding the implementation details and showing only functionality to the user.Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Class is declared abstract by using keyword abstract:

```
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.

```
abstract void printStatus();//no method body and abstract
```

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

23. Discuss about interfaces in Java.

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

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

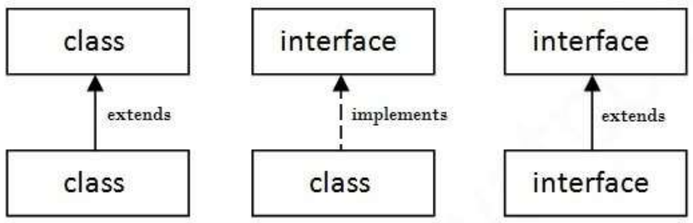
Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class.

Why use Java interface?
There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.
Syntax:

```
interface <interface_name>{
     // declare constant fields
   // declare methods that abstract
   // by default.
}
```



Interface fields are public, static and final by default, and the methods are public and abstract.

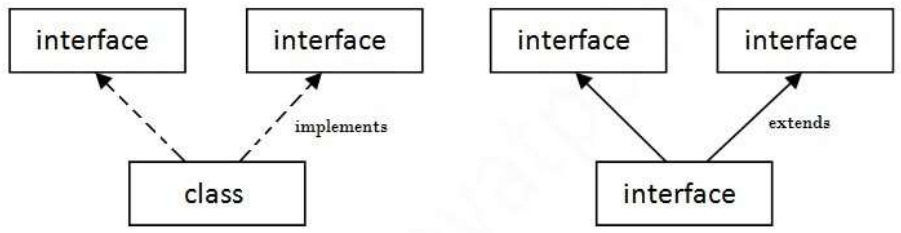
Interface is used with class by using **implements keyword.**

Example:-

```
interface Drawable{
void draw();
}
//Implementation: by second user
```

```
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}  }

class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance**.(in java multiple inheritance is implemented by INTERFACES only)**



Multiple Inheritance in Java

Ex:-

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{  }
```

Ex:-

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{}
```

24. Discuss about Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

- Static Binding (also known as Early Binding).
- Dynamic Binding (also known as Late Binding).

static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding. If there is any private, final or static method in a class, there is static binding.

Ex:-

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}
 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
```

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

<u>Example of dynamic binding:</u>

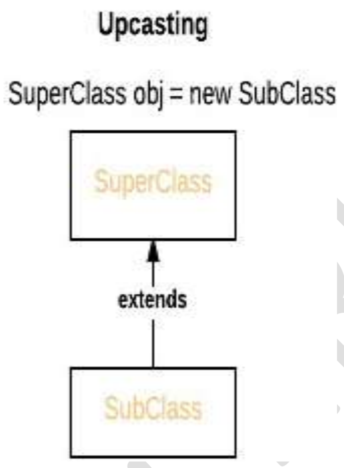
```
class Animal{
 void eat(){System.out.println("animal is eating...");}
}

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

 public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 } }
```

25. Write about Dynamic Method Dispatch or Runtime Polymorphism in Java

Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time



Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Example:-

```
class A
{
    void m1()
    {
        System.out.println("Inside A's m1 method");
    }
}

class B extends A
{
    // overriding m1()
    void m1()
    {
        System.out.println("Inside B's m1 method");
    }
}

class C extends A
{
    // overriding m1()
    void m1()
    {
        System.out.println("Inside C's m1 method");
    }
}

// Driver class
class Dispatch{
    public static void main(String args[])
    {
        // object of type A
        A a = new A();

        // object of type B
        B b = new B();

        // object of type C
        C c = new C();

        // obtain a reference of type A
        A ref;

        // ref refers to an A object
        ref = a;

        // calling A's version of m1()
        ref.m1();

        // now ref refers to a B object
        ref = b;

        // calling B's version of m1()
        ref.m1();

        // now ref refers to a C object
        ref = c;

        // calling C's version of m1()
        ref.m1();
    }
}
```

26. Write about Wrapper classes in JAVA.
The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java
Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

he eight classes of the *java.lang* package are known as wrapper classes in Java. The list of eight wrapper classes are given below:

Primitive Type	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Ex:-
int x=90; ---→ x is an integer variable
Integer obj=new Integer(x); ---→ obj is an integer object. (autoboxing)
Or otherwise by default
Integer j=x;//autoboxing, now compiler will write Integer.valueOf(a) internally

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally

27. write about Java Inner Classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}
public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);  }}
```