Unit 1 : Java Introduction

Que 1: Explain Principles of OBJECT-ORIENTED Languages.

OOP languages follow certain principles like as class, object, **Abstraction, Inheritance, Encapsulation and Polymorphism**

Classes

- A class is defined as the blueprint for an object.
- It serves as a plan or a template.
- The description of a number of similar objects is also called a class.
- An object is not created by just defining a class.
- It has to be created explicitly. Classes are logical in nature.
- For example, furniture does not have any existence but tables and chairs do exist. A class is also defined as a new data type, a user-defined type which contains two things: data members and methods

Objects

- Objects are defined as the instances of a class, e.g. table, chair are all instances of the class Furniture.
- Objects of a class will have same attributes and behaviour which are defined in that class.
- The only difference between objects would be the value of attributes, which may be different.
- Objects (in real life as well as programming) can be physical, conceptual, or software. Objects have unique identity, state, and behaviour.
- There may be several types of objects: l Creator objects: Humans, Employees, Students, Animal l Physical objects: Car, Bus, Plane l Objects in computer system: Monitor, Keyboard, Mouse, CPU, Memory

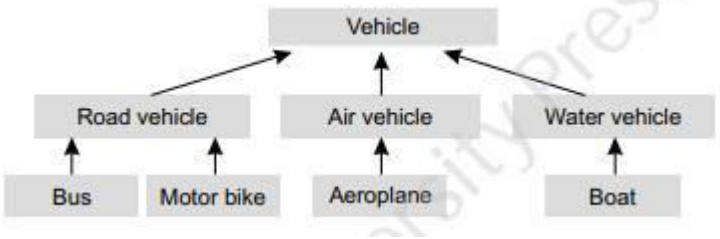
Abstraction

- **Abstraction** is the concept of object-oriented programming that "shows" only essential attributes and "hides" unnecessary information.

- The main purpose of abstraction is hiding the unnecessary details from the users.
- Abstraction is selecting data from a larger pool to show only relevant details of the object to the user.
- It helps in reducing programming complexity and efforts.
- Ex;We drive cars without knowing the internal details about how the engine works and how the car stops on applying brakes.

Inheritance

- Inheritance is the way to adopt the characteristics of one class into another class.
- Here we have two types of classes: base class and subclass.
- There exists a parent–child relationship among the classes.
- When a class inherits another class, it has all the properties of the base class and it adds some new properties of its own.
- We can categorize vehicles into car, bus, scooter, ships, planes, etc.
- The principle of dividing a class into subclass is that each subclass shares common characteristics with the class from where they are inherited or derived.
- Cars, scooters, planes, and ships all have an engine and a speedometer.
- These are the characteristics of vehicles.
- Each subclass has its own characteristic feature, e.g., motorcycles have disk braking system, while planes have hydraulic braking system. A car can run only on the surface, while a plane can fly in air and a ship sails over water



Inheritance aids in reusability. When we create a class, it can be distributed to other programmers which they can use in their programs. This is called reusability.

Encapsulation

- **Encapsulation in Java** is a mechanism to wrap up variables(data) and methods(code) together as a single unit.
- It is the process of hiding information details and protecting data and behaviour of the object.
- It provides us the power to restrict anyone from directly altering the data. Encapsulation is also known as data hiding.
- A class is an **example** of **encapsulation** in computer science in that it consists of data and methods that have been bundled into a single unit.

Polymorphism

- The word polymorphism means having many forms.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- A real-life example of polymorphism, a person at the same time can have different characteristics.
- Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism

Que 2 : Explain JAVA ESSENTIALS

- Java is a platform-independent, object-oriented programming language. Java encompasses the following features:

A High-level Language Java is a high-level language that looks very similar to C and C++ but offers many unique features of its own.

Java Bytecode Bytecode in Java is an intermediate code generated by the compiler, such as Sun's javac, that is executed by the JVM.

Java Virtual Machine (JVM) JVM acts as an interpreter for the bytecode, which takes bytecodes as input and executes it as if it was a physical process executing machine
code.

- Java is designed to be architecturally neutral so that it can run on multiple platforms.

- The same runtime code can run on any platform that supports Java. To achieve its cross-architecture capabilities, the Java compiler generates architecturally neutral bytecode instructions.
- These instructions are designed to be both easily interpreted on any machine and easily translated into native machine code on-the-fly, as shown in Fig. 2.3. Java Runtime Environment (JRE) includes JVM, class libraries, and other supporting files.
- JRE = JVM + Core Java API libraries
  JDK = JRE + development tools like compilers

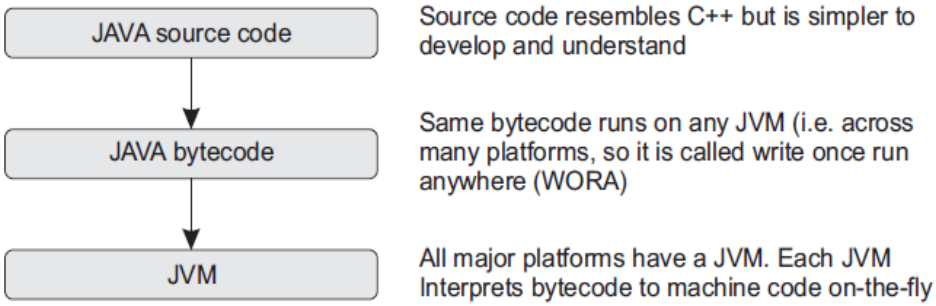


Fig. 2.3 Java Runtime Environment

- Tools such as javac (compiler), java (interpreter), and others are provided in a bundle, popularly known as Java Development Kit (JDK).
- JDK comes in many versions (enhanced in each version) and is different for different platforms such as Windows and Linux.
- A runtime bundle is also provided as a part of JDK (popularly known as Java Runtime Environment).

Que 3 : Features of Java

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

1)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.

2)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.

3) Robust

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

4) Platform Independent

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

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

5) Secure

When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

6) Multi Threading

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

7) Architectural Neutral

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

8) Portable

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

9) High Performance

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

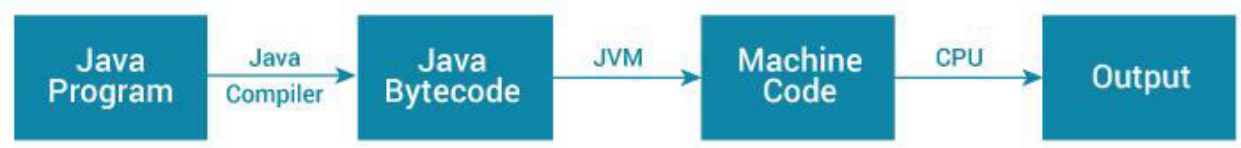
10) Distributed

Java is also a distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++.

Que 4 : Difference between JDK, JRE, and JVM

JVM

- JVM (Java Virtual Machine) is an abstract (theoretical) machine.
- It is called a virtual machine because it doesn't physically exist.
- When you run the Java program, Java compiler first compiles your Java code to bytecode.
- Then, the JVM translates bytecode into native machine code.

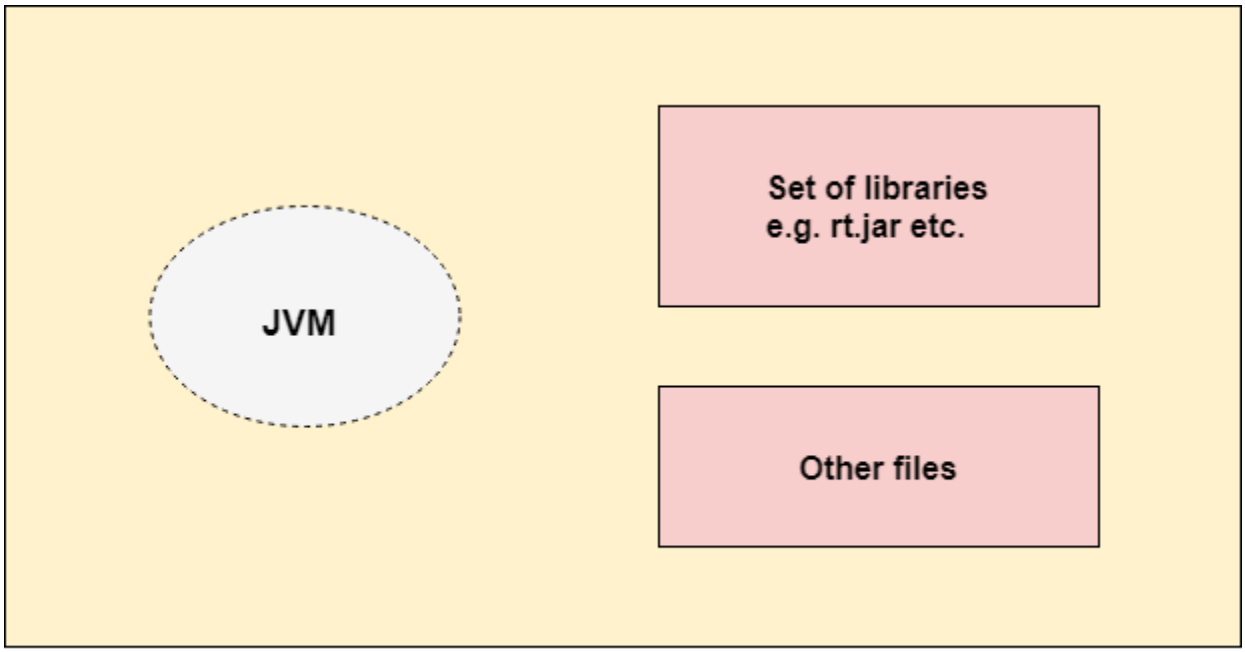


The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

- JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.
- It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment.
- It is the implementation of JVM.
- It physically exists.



JRE

JDK

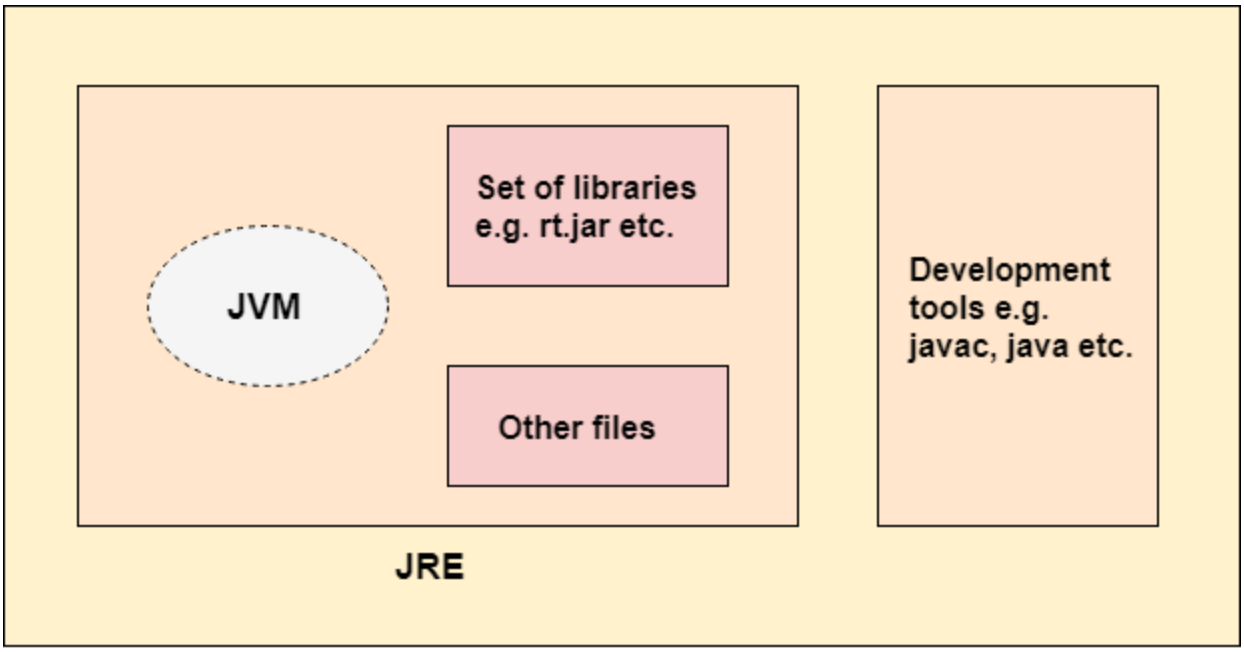
- JDK (Java Development Kit) is a software development kit.
- When you download JDK, JRE is also downloaded with it.
- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets.
- It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform

- Enterprise Edition Java Platform
- Micro Edition Java Platform

- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



JDK

Que 4 : Difference between Java and C++

Next →←

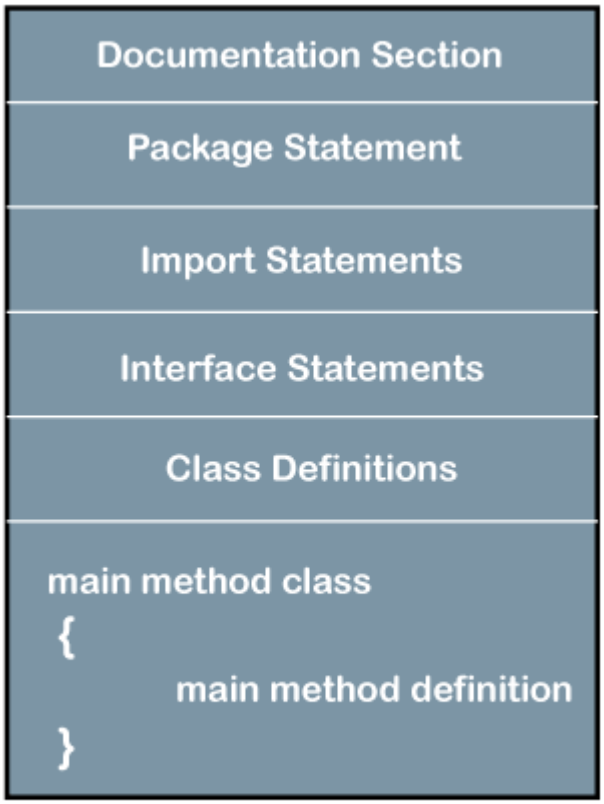
Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.

Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.

Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

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

Que 5 : Program Structure of java



Structure of Java Program

Documentation Section

- It includes **basic information** about a Java program.

- The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program.
- Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program.
- To write the statements in the documentation section, we use **comments**.
- The comments may be **single-line, multi-line,** and **documentation** comments.
- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:

```
//First Java Program
```

- **Multi-line Comment:** It starts with a **/*** and ends with ***/.** We write between these two symbols. For example:

```
/*It is an example of
multiline comment*/
```

- **Documentation Comment:** It starts with the delimiter **(/**)** and ends with ***/**. For example:

```
/**It is an example of documentation comment*/
```

Package Declaration

- The package declaration is optional.
- It is placed just after the documentation section and before any class and interface declaration.
- It is necessary because a Java class can be placed in different packages and directories based on the module they are used.
- For all these classes package belongs to a single parent directory.
- In this section, we declare the **package name** in which the class is placed.
- Note that there can be **only one package** statement in a Java program.
- We use the keyword **package** to declare the package name.
  For example:

```
package javatpoint; //where javatpoint is the package name
```

package com.javatpoint; //where com is the root directory and javatpoint i s the subdirectory

Import Statements

- The package contains the many predefined classes and interfaces.
- If we want to use any class of a particular package, we need to import that class.
- The import statement represents the class stored in the other package.
- We use the **import** keyword to import the class.
- It is written before the class declaration and after the package statement.
- We use the import statement in two ways, either import a specific class or import all classes of a particular package.
- In a Java program, we can use multiple import statements.
  For example:

import java.util.Scanner; //it imports the Scanner class only
import java.util.*; //it imports all the class of the java.util package

Interface Section

- It is an optional section.
- We can create an **interface** in this section if required.
- We use the **interface** keyword to create an interface.
- An interface is a slightly different from the class.
- It contains only **constants** and **method** declarations.
- Another difference is that it cannot be instantiated (call a constructor of a Class).
- We can use interface in classes by using the **implements** keyword. An interface can also be used with other interfaces by using the **extends** keyword.
  For example:

```
interface car
{
void start();
void stop();
}
```

Class Definition

- In this section, we define the class.
- Without the class, we cannot create any Java program.
- A Java program may contain more than one class definition.
- We use the **class** keyword to define the class.
- The class is a blueprint of a Java program.
- It contains information about user-defined methods, variables, and constants.
- Every Java program has at least one class that contains the main() method.
  For example:

```
class Student //class definition
{
}
```

Class Variables and Constants

- In this section, we define variables and **constants** that are to be used later in the program.
- In a Java program, the variables and constants are defined just after the class definition.
- The variables and constants store values of the parameters.
- It is used during the execution of the program.
- We can also decide and define the scope of variables by using the modifiers.
- It defines the life of the variables. For example:

```
class Student //class definition
{
String sname;  //variable
int id;
double percentage;
}
```

Main Method Class

- In this section, we define the **main() method.**
- It is important for all Java programs.
- Because the execution of all Java programs starts from the main() method.
- It must be inside the class.

- Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

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

example:

```
public class Student //class definition
{
public static void main(String args[])
{
//statements
}
}
```

Methods and behavior

- The methods are the set of instructions that we want to perform.
- These instructions execute at runtime and perform the specified task.
  For example:

```
public class Demo //class definition
{
public static void main(String args[])
{
void display()
{
System.out.println("Welcome to javatpoint");
}
//statements
}
}
```

Que 6 : Explain IDE in JAVA

- IDE stands for Integrated Development Environment.

- It is a programming environment that contains a lot of things in a single package i.e. code editor, compiler, debugger and what you see is what you get (WYCIWYG).
- It is actually a software application that provides full facilities to computer programmers for software development.
- It combines all the basic tools that developers need to write or test software.
- This type of environment allows an application developer to write code while compiling, debugging and executing it at the same place.
- It can be a standalone application or a part of one or more compatible applications.

For example: The IDE for developing .NET applications is Microsoft Visual Studio and IDE for developing Java Application is Eclipse, NetBeans, JDeveloper, MyEclipse, BlueJ, RSA etc.

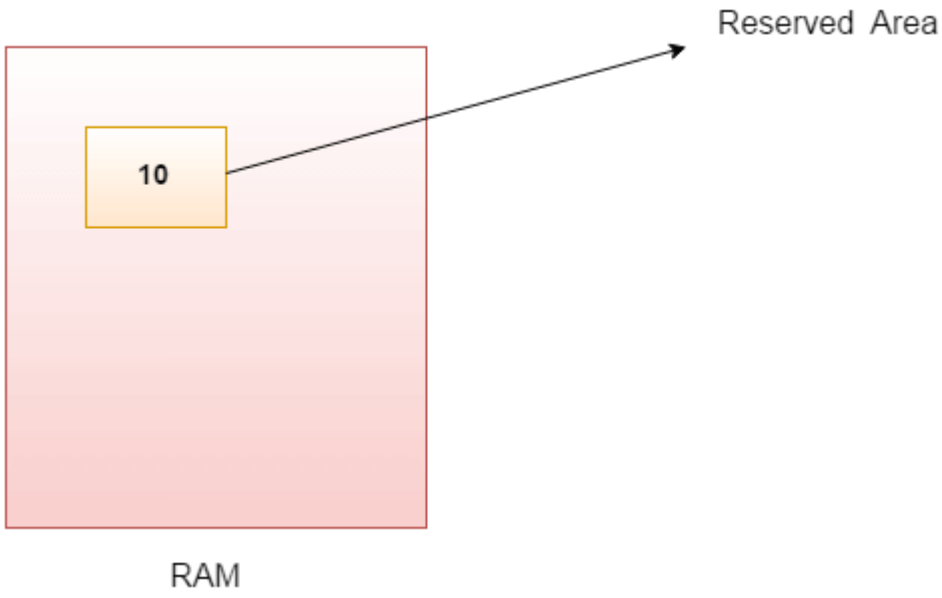
Que 7 : PROCEDURAL LANGUAGE VS OOP

Procedural Languages	**OOP**
l Separate data from functions that operate on them.	l Encapsulate data and methods in a class.
l Not suitable for defining abstract types.	l Suitable for defining abstract types.
l Debugging is difficult.	l Debugging is easier.
l Difficult to implement change.	l Easier to manage and implement change.
l Not suitable for larger programs and applications.	l Suitable for larger programs and applications.
l Analysis and design not so easy.	l Analysis and design made easier.
l Faster.	l Slower.
l Less flexible.	l Highly flexible.
l Data and procedure based.	l Object oriented.
l Less reusable.	l More reusable.

l Only data and procedures are there.	l Inheritance, encapsulation, and polymorphism are the key features.
l Use top-down approach.	l Use bottom-up approach.
l Only a function calls another.	l Object communication is there.
l Example: C, Basic, FORTRAN.	l Example: JAVA, C++, VB.NET, C#.NET.

Que 8 :Explain Variable in java

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location.



Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

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

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

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.

Example

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

Que 9 :Explain Data Types in Java

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

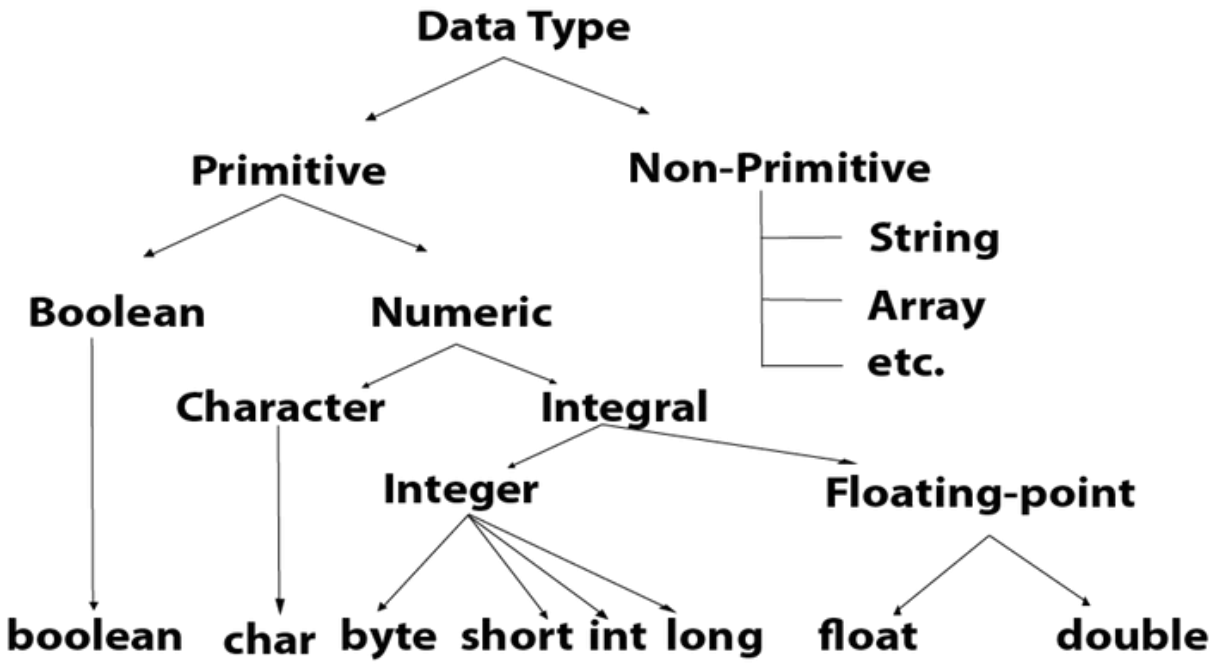
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



Data Type	Default Value	Default size
boolean	false	1 bit

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

Numbers

Primitive number types are divided into two groups:

Integer types:

- It stores whole numbers, positive or negative (such as 123 or -456), without decimals.
- Valid types are byte, short, int and long.
- Which type you should use, depends on the numeric value.

Byte

The byte data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

Example

```
byte myNum = 100;

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

Short

The short data **type can store whole numbers from -32768 to 32767:**

Example

```
short myNum = 5000;

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

Int

- The int data type can store whole numbers from -2147483648 to 2147483647.
- In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;

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

Long

- The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807.
- This is used when int is not large enough to store the value.
- Note that you should end the value with an "L":

Example

```
long myNum = 15000000000L;

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

Floating point types

- represents numbers with a fractional part, containing one or more decimals.
- There are two types: float and double.

Float

- The float data type can store fractional numbers from 3.4e−038 to 3.4e+038.

- Note that you should end the value with an "f":

Example

```
float myNum = 5.75f;

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

Double

- The double data type can store fractional numbers from 1.7e−308 to 1.7e+308.

- Note that you should end the value with a "d":

Scientific Numbers

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

Example

```
float f1 = 35e3f;

double d1 = 12E4d;

System.out.println(f1);

System.out.println(d1);
```

Booleans

A boolean data type is declared with the boolean keyword and can only take the values true or false:

Example

```
boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println(isJavaFun);    // Outputs true

System.out.println(isFishTasty);  // Outputs false
```

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.

Characters

The char data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';

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

Alternatively, you can use ASCII values to display certain characters:

Example

```
char a = 65, b = 66, c = 67;

System.out.println(a);

System.out.println(b);

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

Strings

- The String data type is used to store a sequence of characters (text).

- String values must be surrounded by double quotes:

Example

```
String greeting = "Hello World";

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

Que 10 : Explain Identifiers in Java

- Identifiers in Java are symbolic names used for identification.
- They can be a class name, variable name, method name, package name, constant name, and more.
- However, In Java, There are some reserved words that can not be used as an identifier.

Rules for Identifiers in Java

- There are some rules for declaring the identifiers in Java.
- If the identifiers are not properly declared, we may get a compile-time error.
- Following are some rules and conventions for declaring identifiers:
  - **A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($).**

    for example, @javatpoint is not a valid identifier because it contains a special character which is @.
  - **There should not be any space in an identifier**.

    For example, java t point is an invalid identifier.
  - **An identifier should not contain a number at the starting.**

    For example, 123javatpoint is an invalid identifier.
  - An identifier should be of length 4-15 letters only.
  - However, there is no limit on its length. But, it is good to follow the standard conventions.
  - We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.

- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

Que 11 : Explain Java literals

Literals are data used for representing fixed values. They can be used directly in the code. For example,

```
int a = 1;
float b = 2.5;
char c = 'F';
```

Here, `1`, `2.5`, and `'F'` are literals.
Here are different types of literals in Java.

1. Boolean Literals

In Java, boolean literals are used to initialize boolean data types. They can store two values: true and false. For example,

```
boolean flag1 = false;
boolean flag2 = true;
```

Here, `false` and `true` are two boolean literals.

2. Integer Literals

An integer literal is a numeric value(associated with numbers) without any fractional or exponential part. There are 4 types of integer literals in Java:

1. binary (base 2)
2. decimal (base 10)
3. octal (base 8)
4. hexadecimal (base 16)

For example:

```
// binary
int binaryNumber = 0b10010;
// octal
int octalNumber = 027;

// decimal
int decNumber = 34;

// hexadecimal
int hexNumber = 0x2F; // 0x represents hexadecimal
// binary
int binNumber = 0b10010; // 0b represents binary
```

In Java, binary starts with **0b**, octal starts with **0**, and hexadecimal starts with **0x**.

Note: Integer literals are used to initialize variables of integer types like byte, short, int, and long.

3. Floating-point Literals

A floating-point literal is a numeric literal that has either a fractional form or an exponential form. For example,

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

    double myDouble = 3.4;
    float myFloat = 3.4F;

    // 3.445*10^2
    double myDoubleScientific = 3.445e2;

    System.out.println(myDouble);  // prints 3.4
    System.out.println(myFloat);   // prints 3.4
    System.out.println(myDoubleScientific);   // prints 344.5
```

```
 }
}
```

Note: The floating-point literals are used to initialize `float` and `double` type variables.

4. Character Literals

Character literals are unicode character enclosed inside single quotes. For example,

```
char letter = 'a';
```

Here, `a` is the character literal.
We can also use escape sequences as character literals. For example, **\b** (backspace), **\t** (tab), **\n** (new line), etc.

5. String literals

A string literal is a sequence of characters enclosed inside double-quotes. For example,

```
String str1 = "Java Programming";
String str2 = "Programiz";
```

Here, `Java Programming` and `Programiz` are two string literals.

Que 12 : Explain Java Operators.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators –

Assume integer variable A holds 10 and variable B holds 20, then –

Show Examples

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0

++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then –

Show Examples

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

<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

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

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows –

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators –

Assume integer variable A holds 60 and variable B holds 13 then

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001

~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false, then −

Show Examples

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false

|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Following are the assignment operators supported by Java language –

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C *

		A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators

There are few other operators supported by Java Language.

Conditional Operator (? :)

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as –

variable x = (expression) ? value if true : value if false

```
public class Test {

  public static void main(String args[]) {
    int a, b;
    a = 10;
    b = (a == 1) ? 20: 30;
    System.out.println( "Value of b is : " +  b );

    b = (a == 10) ? 20: 30;
    System.out.println( "Value of b is : " + b );
  }
}
```

This will produce the following result –

Output

```
Value of b is : 30
Value of b is : 20
```

instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as –

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example –

Example

```
public class Test {

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

```
    String name = "James";

    // following will return true since name is type of String
    boolean result = name instanceof String;
    System.out.println( result );
  }
}
```

This will produce the following result –

Output

```
true
```

This operator will still return true, if the object being compared is the assignment compatible with the type on the right. Following is one more example –

Example

```
class Vehicle {}

public class Car extends Vehicle {

  public static void main(String args[]) {

    Vehicle a = new Car();
    boolean result =  a instanceof Car;
    System.out.println( result );
  }
}
```

This will produce the following result –

Output

```
true
```

Que 13 : Explain Expression in Java

- An expression is a combination of operators and/or operands.
- Java expressions are used to create objects, arrays, pass values to methods and call them, assigning values to variables, and so on.
- Expressions may contain identifiers, types, literals, variables, separators, and operators .

For example, int m = 2,n = 3,o = 4; int y = m * n * o; m=2 is an expression which assigns the value 2 to variable m. Similarly, n=3 and o=4 are expressions where n and o are being assigned values 3 and 4. m * n * o is also an expression wherein the values of m, n, and o are multiplied and the result is stored in the variable y.

Que 14 : Explain PRECEDENCE RULES AND ASSOCIATIVITY.

- Operator precedence determines the order in which the operators in an expression are evaluated.
  Example :
  int myInt = 12 - 4 * 2;
- What will be the value of myInt? Will it be (12 - 4)*2, that is, 16? Or it will be 12 - (4 * 2), that is, 4?

- When two operators share a common operand, 4 in this case, the operator with the highest precedence is operated first.

- In Java, the precedence of * is higher than that of -. Hence, the multiplication is performed before subtraction, and the value of myInt will be 4.

Operator Precedence Table

Java Operator Precedence

Operators	Precedence
postfix increment and decrement	++ --
prefix increment and decrement, and unary	++ -- + - ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>

relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

- **Associativity of Operators in Java**
- If an expression has two operators with similar precedence, the expression is evaluated according to its associativity (either left to right, or right to left). Let's take an example.
- a = b = c;
- Here, the value of c is assigned to variable b. Then the value of b is assigned of variable a. Why? It's because the associativity of = operator is from right to left.
- The table below shows the associativity of Java operators along with their associativity.

Java Operator Precedence and Associativity

Operators	Precedence	Associativity
postfix increment and decrement	++ --	left to right
prefix increment and decrement, and	++ -- + - ~ !	right to left

unary		
multiplicative	* / %	left to right
additive	+ -	left to right
shift	<< >> >>>	left to right
relational	< > <= >= instanceof	left to right
equality	== !=	left to right
bitwise AND	&	left to right
bitwise exclusive OR	^	left to right
bitwise inclusive OR	|	left to right
logical AND	&&	left to right
logical OR	||	left to right
ternary	? :	right to left
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=	

Que 14 : Explain Primitive type conversion and casting

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double

- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

Widening (enlarging) Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

```
public class Main

{

 public static void main(String[] args) {

  int myInt = 9;

  double myDouble = myInt; // Automatic casting: int to double



  System.out.println(myInt);     // Outputs 9

  System.out.println(myDouble);  // Outputs 9.0

 }

}
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class Main {

 public static void main(String[] args)
```

```
{
   double myDouble = 9.78;
   int myInt = (int) myDouble; // Manual casting: double to int

   System.out.println(myDouble);  // Outputs 9.78
   System.out.println(myInt);     // Outputs 9
 }
}
```

Que 15: Explain FLOW OF CONTROL

- Control flow statements help programmers make decisions about which statements to execute and to change the flow of execution in a program.
- The Three categories of control flow statements available in Java.

conditional statement, loops, and branch.

Conditional Statements

The two conditional statements provided by Java are:

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.

if statement

The Java if statement tests the condition. It executes the *if block* if condition is true

Syntax:

```
if(condition)
{
//code to be executed

}
```

Example:

```
public class IfExample
 {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

if-else statement
Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Example:

```
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

if-else-if ladder

```
if(condition1)
{
//code to be executed if condition1 is true
}else if(condition2)
{
```

```
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example

```
public class IfElseIfExample
 {
public static void main(String[] args)
{
   int marks=65;

   if(marks<50)
{
      System.out.println("fail");
   }
   else if(marks>=50 && marks<60)
{
      System.out.println("D grade");
   }
   else if(marks>=60 && marks<70)
{
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80)
{
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90)
{
      System.out.println("A grade");
   }else if(marks>=90 && marks<100)
{
      System.out.println("A+ grade");
   }else{
      System.out.println("Invalid!");
   }
```

```
}
}
```

nested if statement

- The nested if statement represents the *if block within another if block*.
- Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
   }
}
```

Example:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
   //Creating two variables for age and weight
   int age=20;
   int weight=80;
   //applying condition on age and weight
   if(age>=18){
      if(weight>50){
         System.out.println("You are eligible to donate blood");
      }
   }
}}
```

Using Ternary Operator

- We can also use ternary operator (? :) to perform the task of if...else statement.
- It is a shorthand way to check the condition.
- If the condition is true, the result of ? is returned.
- But, if the condition is false, the result of : is returned.

Example:

```
public class IfElseTernaryExample {
```

```
public static void main(String[] args) {
   int number=13;
   //Using ternary operator
   String output=(number%2==0)?"even number":"odd number";
   System.out.println(output);
}
}
```

switch-case.

- Java has a shorthand for multiple if statement—the switch-case statement

```
switch(expression)

{

 case x:

   // code block

   break;

 case y:

   // code block

   break;

 default:

   // code block

}
```

Example:

```
public class Main {

  public static void main(String[] args) {

    int day = 4;
```

```
    switch (day) {
      case 1:
       System.out.println("Monday");
       break;
      case 2:
       System.out.println("Tuesday");
       break;
      case 3:
       System.out.println("Wednesday");
       break;
      case 4:
       System.out.println("Thursday");
       break;
      case 5:
       System.out.println("Friday");
       break;
      case 6:
       System.out.println("Saturday");
       break;
      case 7:
       System.out.println("Sunday");
       break;
    }
  }
}
```

Loops

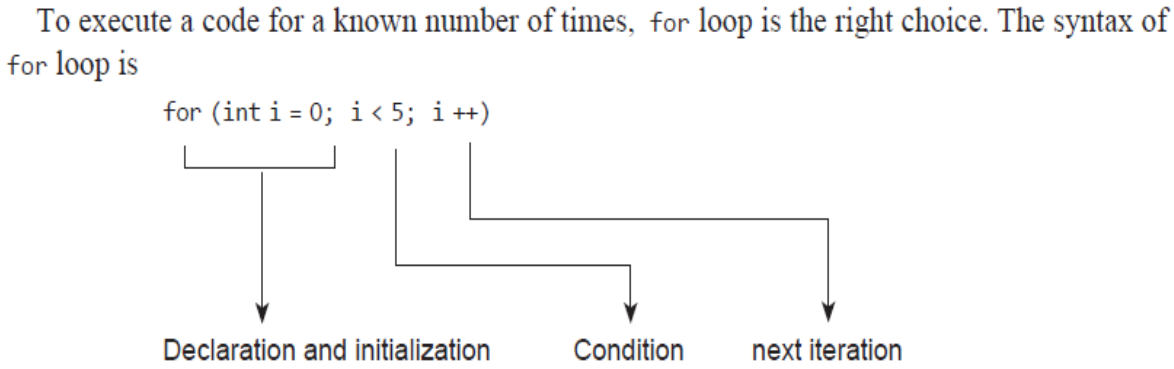
The purpose of loop statements is to execute Java statements many times. There are three types

of loops in Java—for, while, and do-while.

for Loop

The for loop groups the following three common parts together into one statement:
(a) Initialization
(b) Condition
(c) Increment or decrement

To execute a code for a known number of times, for loop is the right choice. The syntax of for loop is



Example

```
class ForDemo
{
public static void main(String args[])
{
 for(int i = 1;i <= 5;i++)
System.out.println("Square of "+i+" is "+ (i*i));
}
}
```

while Loop

The while statement has the following syntax:

```
while (condition)
{
Statements to execute while the condition is true
}
```

Example

```
class WhileDemo
{
public static void main(String args[])
{
int i = 1;
 while(i <= 5)
{
System.out.println("Square of " +i+ "is" +(i*i));
 i++;
}
}
}
```

do-while Loop

Syntax

```
{
Statements to execute once and thereafter while the condition is true
}
while (test);
Next-statement;
```

Example:

```
class DoWhileDemo
{
public static void main(String args[])
{ int i = 1;
do
 {
 System.out.println("Square of" +i+ "is" + (i*i));
i++;
}while(i <= 5);
}
}
```

For-each loop

The Java for-each loop traverses the array or collection until the last element. For each element, it stores the element in the variable and executes the body of the for-each loop.

Syntax:

```
for (type var : arr)
{
// Statements to repeat
}
```

Example :

```
class ForEachExample1{
  public static void main(String args[]){
   int arr[]={12,13,14,44};

   for(int i:arr){
     System.out.println(i);
   }

 }
}
```

Break

The break statement can also be used to jump out of a **loop**.

Example :

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop

Example :

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

Output:
0
1
2
3
4
6
7
8
9

Que 16: Explain Classes and Objects

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.

Syntax

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

Example

```
public class Main

 {

  int x = 5;

}
```

Objects

- Objects are defined as the instances of a class, e.g. table, chair are all instances of the class Furniture.
- Objects of a class will have same attributes and behaviour which are defined in that class.
- The only difference between objects would be the value of attributes, which may be different.
- Objects (in real life as well as programming) can be physical, conceptual, or software. Objects have unique identity, state, and behaviour.
- There may be several types of objects: l Creator objects: Humans, Employees, Students, Animal l Physical objects: Car, Bus, Plane l Objects in computer system: Monitor, Keyboard, Mouse, CPU, Memory

  Syntax

  ```
  ClassName ReferenceVariable = new ClassName();
  ```

  Example

```
public class Main {

 int x = 5;



 public static void main(String[] args) {

  Main myObj = new Main();

  System.out.println(myObj.x);

 }

}
```

Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

Main.java

Second.java

Main.java

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

Second.java

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Run the Second.java file:

C:\Users\Your Name>java Second

And the output will be:

5

Que 17: Explain Method in java

- A method is a collection of statements that perform some specific task and return the result to the caller.
- A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code.
- Dividing a complex problem into smaller chunks makes your program easy to understand and reusable.

In Java, there are two types of methods:

User-defined Methods: We can create our own method based on our requirements.

Standard Library Methods: These are built-in methods in Java that are available to use.

The syntax to declare a method is:

```
returnType methodName()
{
 // method body
}
```

returnType –

- It specifies what type of value a method returns.
- For example if a method has an int return type then it returns an integer value.
- If the method does not return a value, its return type is void.

methodName –

- It is an identifier that is used to refer to the particular method in a program.

method body –

- It includes the programming statements that are used to perform some tasks.
- The method body is enclosed inside the curly braces { }.

Calling a Method in Java

- For using a method, it should be called.
- There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).
- The process of method calling is simple.
- When a program invokes a method, the program control gets transferred to the called method.
- This called method then returns control to the caller in two conditions, when –
    - the return statement is executed.
    - it reaches the method ending closing brace.

Example

```
class Main {

 // create a method

 public int addNumbers(int a, int b) {

  int sum = a + b;

  // return value

  return sum;

 }

 public static void main(String[] args)

{

  int num1 = 25;



  int num2 = 15;
```

```
    // create an object of Main
    Main obj = new Main();
    // calling method
    int result = obj.addNumbers(num1, num2);
    System.out.println("Sum is: " + result);
  }
}
```

Static Method

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

```
public class Display
{
public static void main(String[] args)
{
show();
}
static void show()
{
```

```
System.out.println("It is an example of static method.");

} }
```

Instance Method

The method of the class is known as an instance method. It is a non-static method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

Example

```
public class InstanceMethodExample
{
public static void main(String [] args)
{
//Creating an object of the class
InstanceMethodExample obj = new InstanceMethodExample();
//invoking instance method
System.out.println("The sum is: "+obj.add(12, 13));
}
int s;
//user-defined method because we have not used static keyword
public int add(int a, int b)
{
s = a+b;
//returning the sum
return s;
}
}
```

Abstract Method

The method that does not has method body is known as abstract method. It always declares in the abstract class.
To create an abstract method, we use the keyword abstract.

Syntax

```
abstract void method_name();
```

Example

```
abstract class Demo //abstract class
{
//abstract method declaration
abstract void display();
}
public class MyClass extends Demo
{
//method impelmentation
void display()
{
System.out.println("Abstract method?");
}
public static void main(String args[])
{
//creating object of abstract class
Demo obj = new MyClass();
//invoking abstract method
obj.display();
}
}
```

Method Overloading

When a class has two or more methods by the same name but different parameters, it is known as method overloading.

```
public class ExampleOverloading {

  public static void main(String[] args) {
    int a = 11;
    int b = 6;
    double c = 7.3;
    double d = 9.4;
    int result1 = minFunction(a, b);

    // same function name with different parameters
    double result2 = minFunction(c, d);
    System.out.println("Minimum Value = " + result1);
```

```
    System.out.println("Minimum Value = " + result2);
  }

  // for integer
  public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;

    return min;
  }

  // for double
  public static double minFunction(double n1, double n2) {
    double min;
    if (n1 > n2)
      min = n2;
    else
      min = n1;

    return min;
  }
}
```

Call by Value and Call by Reference in Java

Call by Value

If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }
```

```
 public static void main(String args[]){
   Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);

 }
}
```

Output:before change 50
after change 50

Call By Reference

Java uses only call by value while passing reference variables as well. It creates a copy of references and passes them as valuable to the methods. As reference points to same address of object, creating a copy of reference is of no harm. But if new object is assigned to reference it will not be reflected.

Example

```
class Operation2{
 int data=50;

 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }


 public static void main(String args[]){
   Operation2 op=new Operation2();

   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

Que 17: Explain CommandLine Arguments .

A command-line argument is an information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to main( ).

Example

```
public class CommandLineExample{
public static void main(String args[]) {
    for(int i = 0; i<args.length; i++) {
      System.out.println("args[" + i + "]: " + args[i]);
    }
  }
}
```

compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo

Que 18 : Explain Constructors in detail

- In Java, a constructor is a block of codes similar to the method.
- 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

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

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();
}
}
```

Example of default constructor that displays the default values

```
//Let us see another example of default constructor
//which displays the default values
class Student3{
int id;
String name;
//method to display the value of id and name
```

```
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

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.

Example

```
class student
{
int a1;
int b1;
int c;
student(int a,int b)
{
a=a1;
b=b1;
}
void display()
{
c=a1+b1;
System.out.println(c);
}
public static void main(String[] args)
{
student s=new student(1,1);
```

```
s.display();
}
}
```

Constructor Overloading in Java

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
//Java program to overload constructors
class Student5{
  int id;
  String name;
  int age;
  //creating two arg constructor
  Student5(int i,String n){
  id = i;
  name = n;
  }
  //creating three arg constructor
  Student5(int i,String n,int a){
  id = i;
  name = n;
  age=a;
  }
  void display(){System.out.println(id+" "+name+" "+age);}
  public static void main(String args[]){
  Student5 s1 = new Student5(111,"Karan");
  Student5 s2 = new Student5(222,"Aryan",25);
  s1.display();
  s2.display();  }}
```

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Que 18: Explain Cleaning Up Unused Objects

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
- In other words, it is a way to destroy the unused objects.
- To do so, we were using **free()** function in C language and delete() in C++.
- But, in java it is performed automatically.
- So, java provides better **memory management**.

Advantage of Garbage Collection

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

- It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

gc() method

- The gc() method is used to invoke the garbage collector to perform cleanup processing.
- The gc() is found in System and Runtime classes.

```
public static void gc()
{
}
```

Finalization

- The finalize() method is invoked each time before the object is garbage collected.
- This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize()
{
}
```