REVA UNIVERSITY

School of Computer Science and Engineering

Question Bank

Academic year : 2023-2024[ODD SEMESTER]		
Sub name : Object Oriented Programming with Java	Sem & Sec: 3rd	Sub Code: **B22EF0402**

Unit 1

Chapter 1

1. A) Difference between system software and application software.

System Software	**Application Software**
The Software that is designed to control, integrate and manage the individual hardware components and application software is known as system software.	A set of computer programs installed in the user's system and designed to perform a specific task is known as application software.
System Software maintains the system resources and gives the path for application software to run.	Application software is built for specific tasks.
Low-level languages are used to write the system software.	While high-level languages are used to write the application software.
It is general-purpose software.	While it's a specific purpose software.
Without system software, the system stops and can't run.	While Without application software system always runs.
System software runs when the system is turned on and stops when the system is turned off.	While application software runs as per the user's request.
Example: System software is an operating system, etc.	Example: Application software is Photoshop, VLC player, etc.
System Software programming is more complex than application software.	Application software programming is simpler in comparison to system software.

System Software	**Application Software**
A system software operates the system in the background until the shutdown of the computer.	Application software runs in the front end according to the user's request.
The system software has no interaction with users. It serves as an interface between hardware and the end user.	Application software connects an intermediary between the user and the computer.
System software runs independently.	Application software is dependent on system software because they need a set platform for its functioning.
Example for System Software includes Android, Mac Operating system, MS Windows, Compiler, Assembler, loader etc.	Examples of Application Software includes MS Word, MS Power Point, Adobe Reader, VLC Player, games, etc.

B) Difference between procedural and object oriented programming.

It is a programming language that is derived from **structure programming** and based upon the concept of calling **procedures**. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models **software design around data or objects** rather than functions and logic.
It is **less** secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is **more** secure than procedural programming.
It follows a **top-down approach**.	It follows a **bottom-up approach**.
In procedural programming, data moves **freely** within the system from one function to another.	In OOP, objects can move and communicate with each other via **member functions.**
It is structure/procedure-oriented.	It is object-oriented.

There are **no** access modifiers in procedural programming.	The access modifiers in OOP are named as **private, public, and protected**.
Procedural programming does **not** have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
There is **no** code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
Overloading is **not** possible in procedural programming.	In OOP, there is a concept of **function overloading and operator overloading.**
It gives importance to **functions over data.**	It gives importance to **data over functions.**
In procedural programming, there are **no** virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
It is **not** appropriate for complex problems.	It is **appropriate** for complex problems.
There is **not** any proper way for data hiding.	There is a **possibility** of data hiding.
In Procedural programming, a program is divided into small programs that are referred to as **functions**.	In OOP, a program is divided into small parts that are referred to as **objects**.
Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

C) Difference between C++ and Java.

C++	**JAVA**
C++ was developed by Bjarne Stroustrup at Bell Labs in 1979. It was developed as an extension of the C language.	Java was developed by James Gosling at Sun Microsystems. Now, it is owned by Oracle.
It has support for both procedural programming and object-oriented programming.	Java has support only for object-oriented programming models.
C++ is platform dependent. It is based on the concept of **Write Once Compile Anywhere**.	Java is platform-independent. It is based on the concept of **Write Once Run Anywhere**.
C++ supports features like operator overloading, Goto statements, structures, pointers, unions, etc.	Java does not support features like operator overloading, Goto statements, structures, pointers, unions, etc.
C ++ is only compiled and cannot be interpreted.	Java can be both compiled and interpreted.
C++ has very limited libraries with low-level functionalities. C++ allows direct calls to native system libraries.	Java, on the other hand, has more diverse libraries with a lot of support for code reusability. In Java, only calls through the Java Native Interface and recently Java Native Access are allowed.
In C++, memory management is manual.	In Java, memory management is System controlled.
C++ is pretty consistent between primitive and object types.	In Java, semantics differs for primitive and object types.
In C++, both global and namespace scopes are supported.	Java has no support for global scope.
In C++, a flexible model with constant protection is available.	In Java, the model is cumbersome and encourages weak encapsulation.

2. Explain the features or buzzwords of Java.(Simple, Object Oriented, Dynamic, robust, Secure,Architecural neutral, Portable, Interpreted, High-Performance, Multithreaded, Platform independent, Distributed)

Ans. Features of Java/Buzz words 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: 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.

Basic concepts of OOPs are: Object, Class, Inheritance. Polymorphism, Abstraction, Encapsulation

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

4. Platform independent

5. 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 JVM 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.

6. 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.

7. 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.

8.Interpreted

9.High Performance

10.Multithreaded

11.Distributed

12.Dynamic

8. Explain OOPS principles.(Classes and Objects, Encapsulation, Inheritance, Abstraction, Polymorphism)

Ans.

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

  An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

  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.

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

  To create an object of **MyClass**, specify the class name, followed by the object name, and use the keyword new:

  Example

  Create an object called "obj" and print the value of x:

```
public class MyClass {
 int x = 5;

 public static void main(String[] args) {
   MyClass Obj = new MyClass();
   System.out.println(Obj.x);
 }
}
```

```
Output :
5
```

- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

  A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Example

```
public class Dog {
   String breed;
   int age;
   String color;

   void barking() {
   }

   void hungry() {
   }

   void sleeping() {
   }
}
```

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

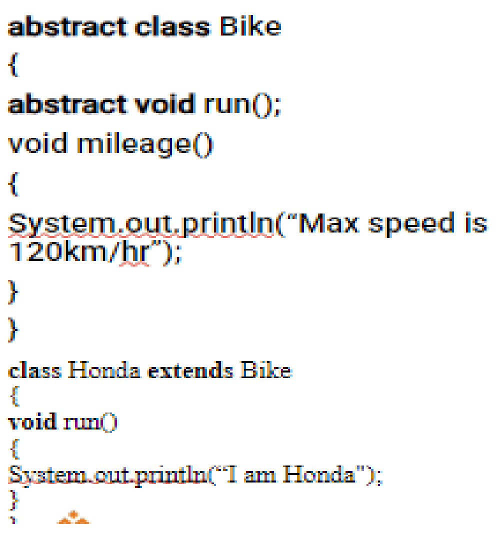
Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java

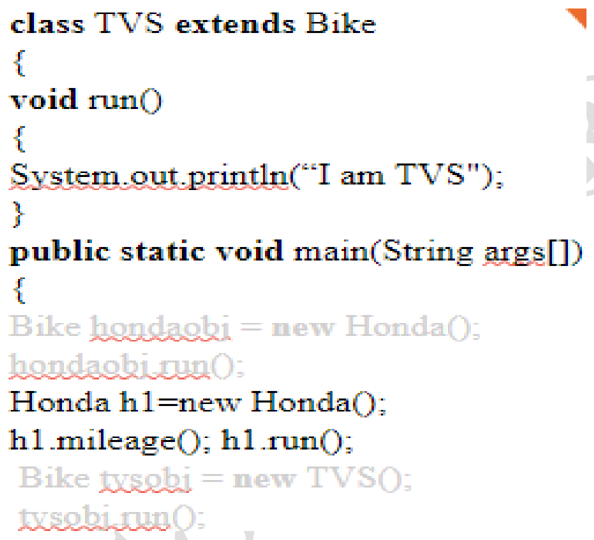
1. **Abstract class**
2. **Interface**

- 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**.
- It can have **final methods, constructors and static methods**



```
abstract class Bike
{
abstract void run();
void mileage()
{
System.out.println("Max speed is
120km/hr");
}
}
class Honda extends Bike
{
void run()
{
System.out.println("I am Honda");
}
```



```
class TVS extends Bike
{
void run()
{
System.out.println("I am TVS");
}
public static void main(String args[])
{
Bike hondaobj = new Honda();
hondaobj.run();
Honda h1=new Honda();
h1.mileage(); h1.run();
 Bike tvsobj = new TVS();
 tvsobj.run();
```

Interface in Java is a mechanism to achieve abstraction.

It has **static constants and abstract methods(not method body)**

It is used to achieve **multiple inheritance** in Java.

It needs to be **implemented**

Syntax:

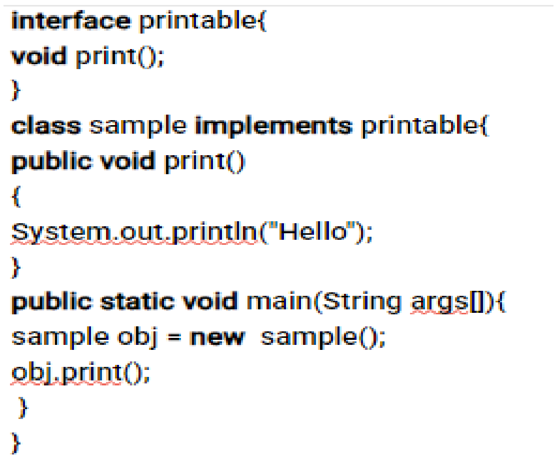
interface <interface_name>{

    // declare constant fields

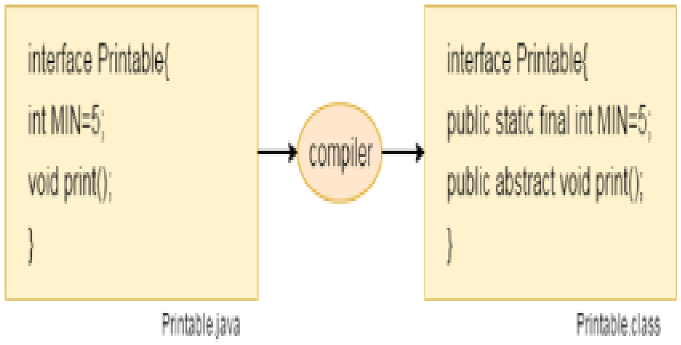
    // declare methods that abstract

    // by default.

}



```
interface printable{
void print();
}
class sample implements printable{
public void print()
{
System.out.println("Hello");
}
public static void main(String args[]){
sample obj = new  sample();
obj.print();
 }
}
```

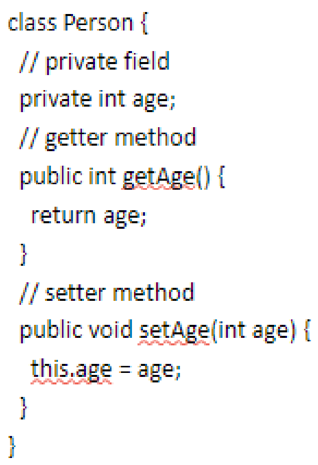


```
interface Printable{
int MIN=5;
void print();
}
```

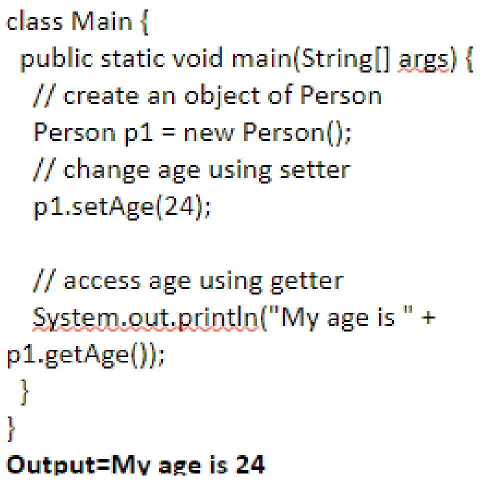
```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

Encapsulation

- Encapsulation refers to the bundling of fields and methods inside a single class.
- It prevents outer classes from accessing and changing fields and methods of a class.
- This also helps to achieve data hiding.



```
class Person {
  // private field
  private int age;
  // getter method
  public int getAge() {
    return age;
  }
  // setter method
  public void setAge(int age) {
    this.age = age;
  }
}
```



```
class Main {
  public static void main(String[] args) {
    // create an object of Person
    Person p1 = new Person();
    // change age using setter
    p1.setAge(24);

    // access age using getter
    System.out.println("My age is " +
p1.getAge());
  }
}
Output=My age is 24
```

INHERITANCE

- Key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).
- The extends keyword is used to perform inheritance in Java

Benefits of Inheritance

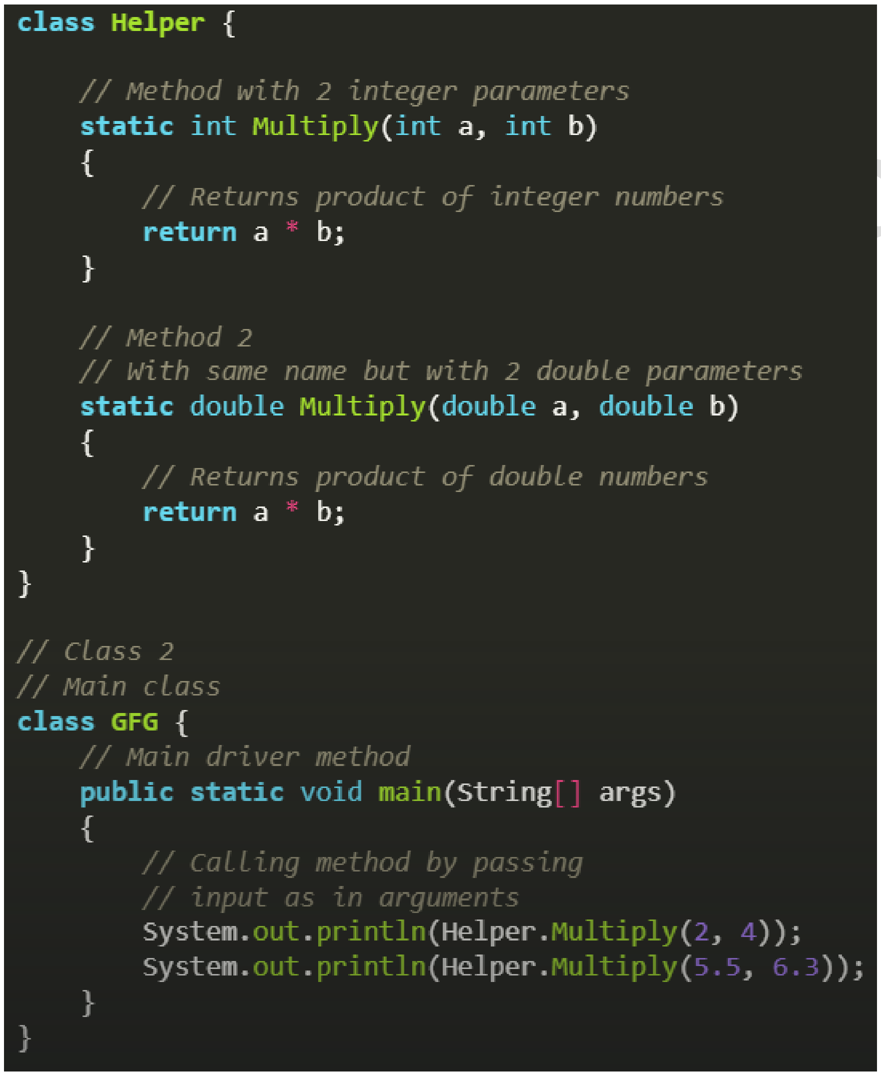
- Inheritance promotes **reusability**.
- Reusability enhanced **reliability**.
- As the existing code is reused, it leads to **less development and maintenance costs.**
- Inheritance makes the **sub classes follow a standard interface**.

- Inheritance helps to reduce code redundancy and supports code extensibility.

TYPES

Single Inheritance
Multi-level Inheritance
Hierarchical Inheritance
Hybrid Inheritance

Polymorphism
It simply means more than one form.
That is, the same entity (method or operator or object) can perform different operations in different scenarios.

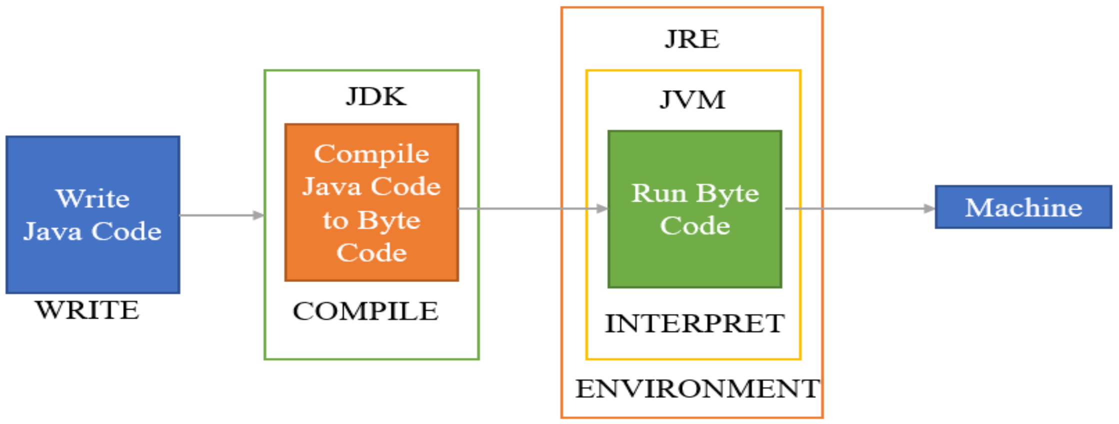


```
class Helper {

    // Method with 2 integer parameters
    static int Multiply(int a, int b)
    {
        // Returns product of integer numbers
        return a * b;
    }

    // Method 2
    // With same name but with 2 double parameters
    static double Multiply(double a, double b)
    {
        // Returns product of double numbers
        return a * b;
    }
}

// Class 2
// Main class
class GFG {
    // Main driver method
    public static void main(String[] args)
    {
        // Calling method by passing
        // input as in arguments
        System.out.println(Helper.Multiply(2, 4));
        System.out.println(Helper.Multiply(5.5, 6.3));
    }
}
```

Chapter 2

9. Explain the JVM architecture(diagram + JDK+ JRE)

Ans.



Java Development Kit (JDK): JDK provides an environment to develop, compile and execute java applications. It is a combination of java compiler (javac) and Java Runtime Environment (JRE). Java compiler accepts a program with .java extension and it will convert the program in high level language to bytecodes or class file (.class extension). Bytecode contains mnemonics or symbols.

Java Runtime Environment (JRE): JRE provides an environment to execute java applications. To execute java applications, JRE contains Java Virtual Machine (JVM).

JVM will accept the bytecodes (.class file), JVM interprets the bytecodes line by line and it will convert the program to machine code (0s and 1s). Java is an example for Interpreted and Compiled language.

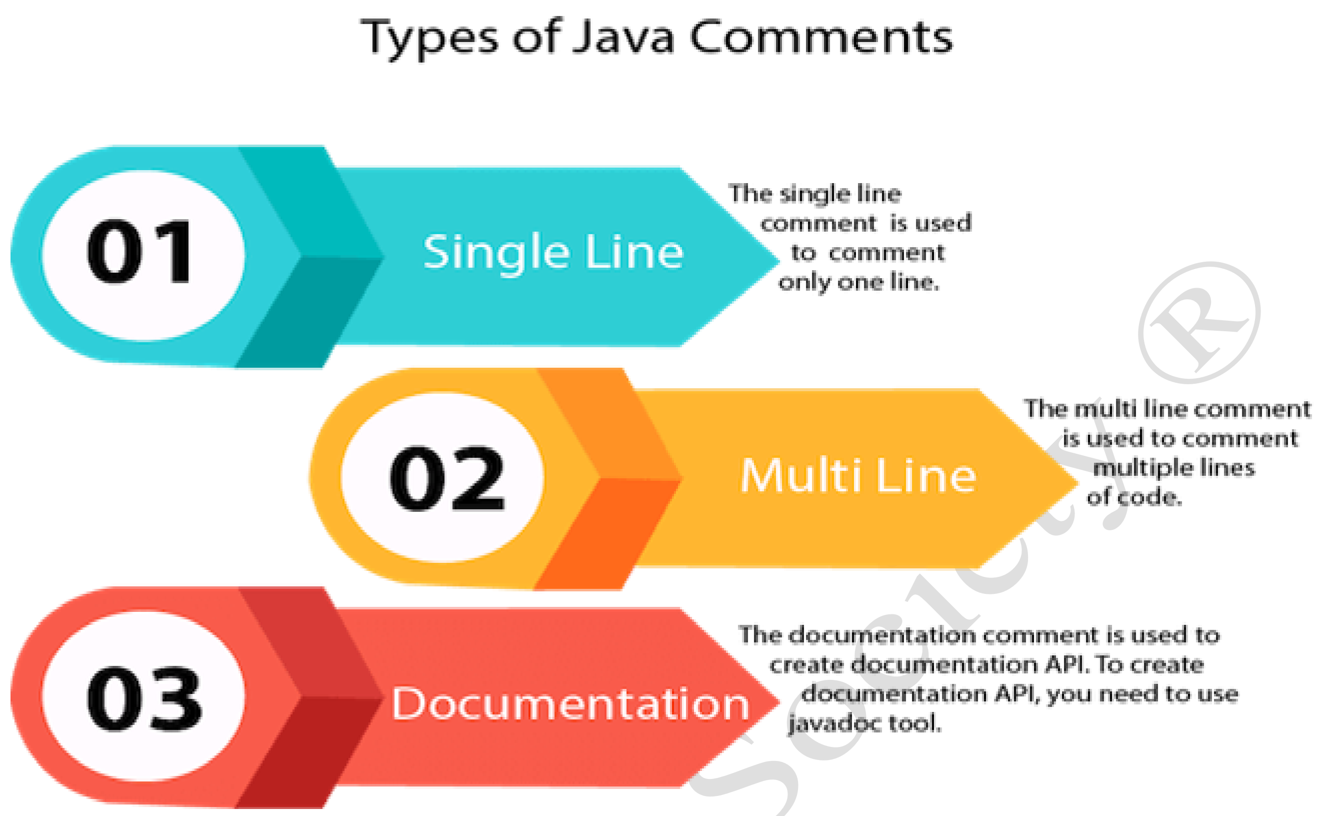
- Once you have installed the JDK, Open a terminal window and run the commands

```
javac HelloWorld.java
java HellWorld
```

- Note that 2 steps are involved to execute the program. The javac command compiles the java source code into an intermediate machine-independent representation called bytecodes, and saves them in class files. The javac compiler creates a file called HelloWorld.class that contains the bytecode version of the program. The java command launches a virtual machine that loads the class files, and executes the bytecodes.
- Once compiled, byte codes can run on any JVM, whether on your desktop computers or on a device in a galaxy far, far away. The promise of “write once, run anywhere” was an important design criterion for java.

10. With example Explain the comments in Java(single line, multiline, documentation)

Ans.


Types of Java Comments
01
Single Line
The single line comment is used to comment only one line.
02
Multi Line
The multi line comment is used to comment multiple lines of code.
03
Documentation
The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

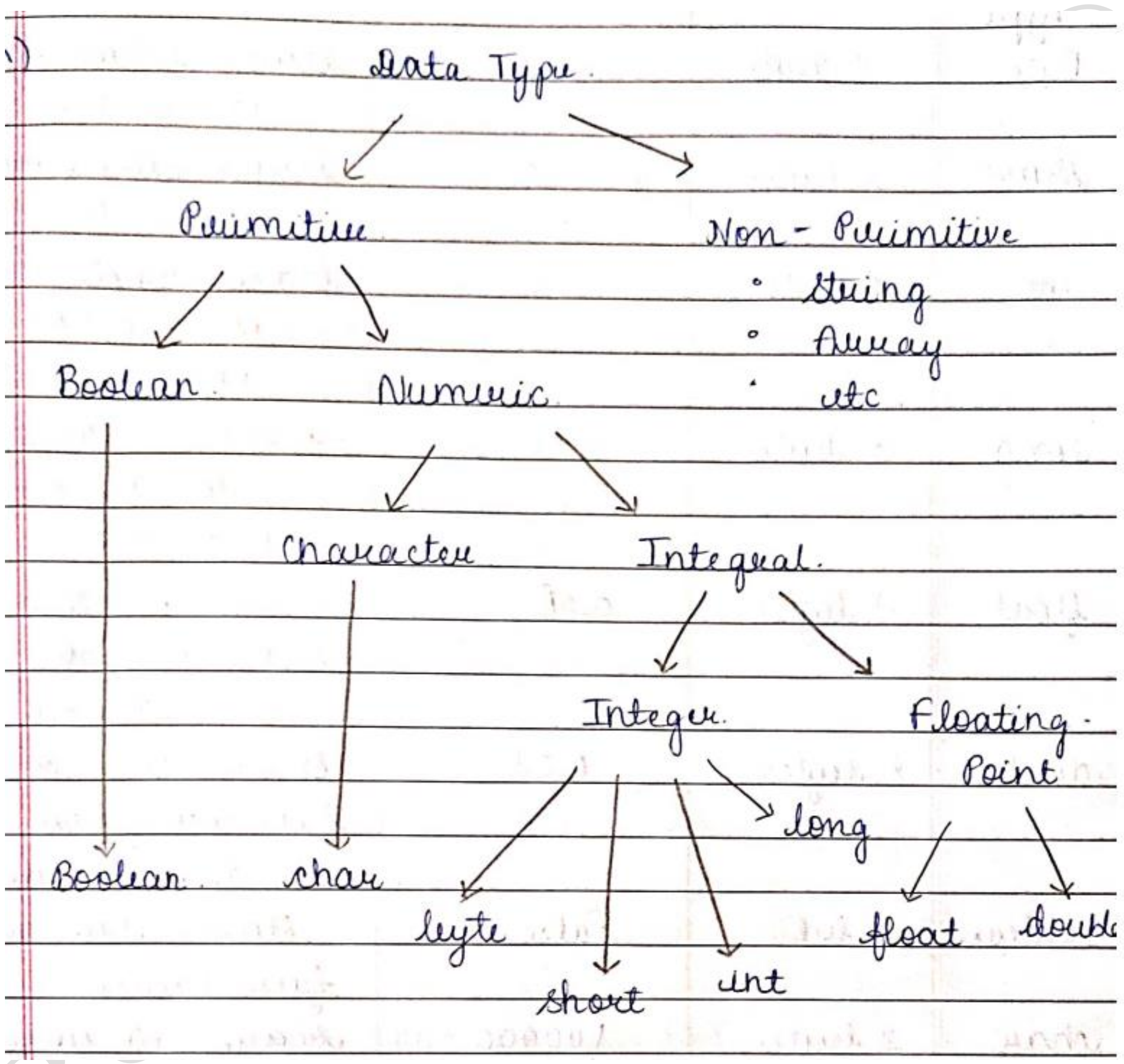
- Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API.
- These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.
- To create documentation API, we need to use the **javadoc tool**.
- The documentation comments are placed between /** and */.

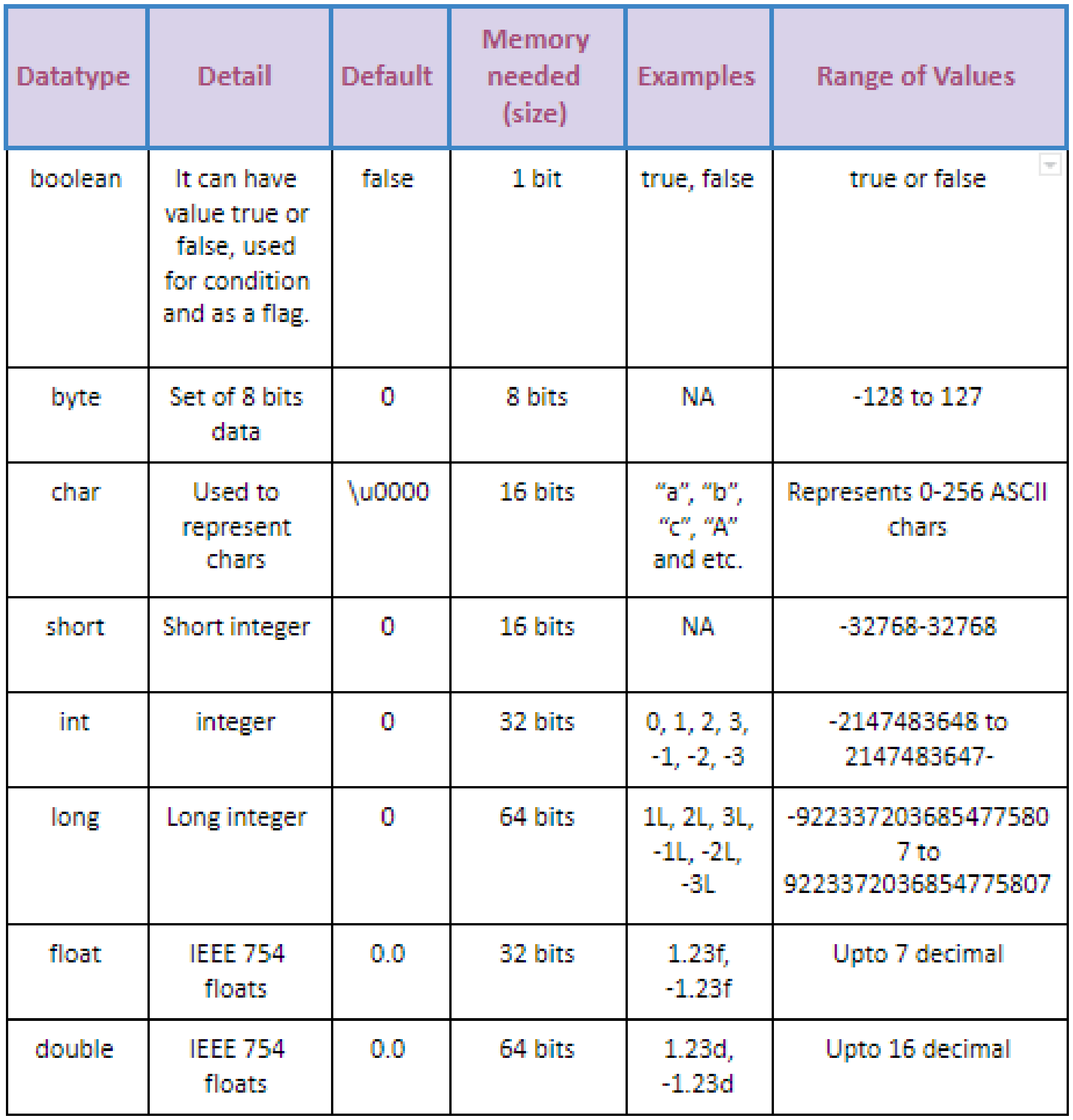
Tag	**Syntax**	**Description**
{@docRoot}	{@docRoot}	to depict relative path to root directory of generated document from any page.
@author	@author name - text	To add the author of the class.

@code	{@code text}	To show the text in code font without interpreting it as html markup or nested javadoc tag.
@version	@version version-text	To specify "Version" subheading and version-text when -version option is used.
@since	@since release	To add "Since" heading with since text to generated documentation.
@param	@param parameter-name description	To add a parameter with given name and description to 'Parameters' section.
@return	@return description	Required for every method that returns something (except void)

```
import java.io.*;

/**
  * This program implements an application
 * to perform operation such as addition of numbers
 * and print the result
 * @author Manasa
 * @version 22.0
 * @since 2024-08-06
 */
```

11. List and Explain the primitive datatypes that are available in java. Write a program using all primitive datatypes.(8 types)



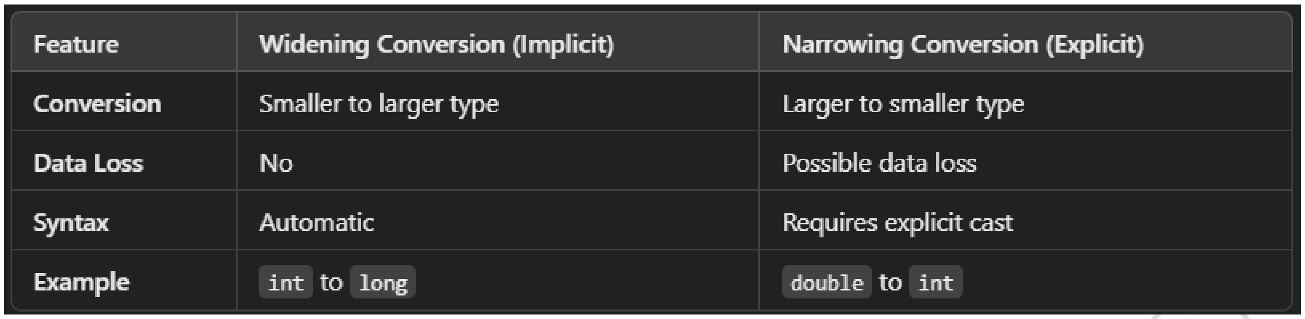


Datatype	Detail	Default	Memory needed (size)	Examples	Range of Values
boolean	It can have value true or false, used for condition and as a flag.	false	1 bit	true, false	true or false
byte	Set of 8 bits data	0	8 bits	NA	-128 to 127
char	Used to represent chars	\u0000	16 bits	“a”, “b”, “c”, “A” and etc.	Represents 0-256 ASCII chars
short	Short integer	0	16 bits	NA	-32768-32768
int	integer	0	32 bits	0, 1, 2, 3, -1, -2, -3	-2147483648 to 2147483647-
long	Long integer	0	64 bits	1L, 2L, 3L, -1L, -2L, -3L	-922337203685477580 7 to 9223372036854775807
float	IEEE 754 floats	0.0	32 bits	1.23f, -1.23f	Upto 7 decimal
double	IEEE 754 floats	0.0	64 bits	1.23d, -1.23d	Upto 16 decimal

12. Explain the different types of Type conversion along with example.(narrowing, widening)
Ans.

Widening (Implicit Conversion): No data loss, happens automatically.
Example: int to long, float to double.

Narrowing (Explicit Conversion): Potential for data loss, requires explicit casting.
Example: double to int, long to byte.



Feature	Widening Conversion (Implicit)	Narrowing Conversion (Explicit)
Conversion	Smaller to larger type	Larger to smaller type
Data Loss	No	Possible data loss
Syntax	Automatic	Requires explicit cast
Example	`int` to `long`	`double` to `int`

Program Demonstrating Both:

```
public class TypeConversion {
   public static void main(String[] args) {
       // Widening (Implicit)
       int intValue = 50;
       double doubleValue = intValue;  // int to double
       System.out.println("Widening: int to double: " + doubleValue);

       // Narrowing (Explicit)
       double anotherDouble = 45.67;
       int narrowedInt = (int) anotherDouble;  // double to int
       System.out.println("Narrowing: double to int: " + narrowedInt);
   }
}
```

```
Output:
Widening: int to double: 50.0
Narrowing: double to int: 45
```

13. Explain the variables, rules to declare variables and their types. (local ,static, instance)

Ans.

1. Local Variable

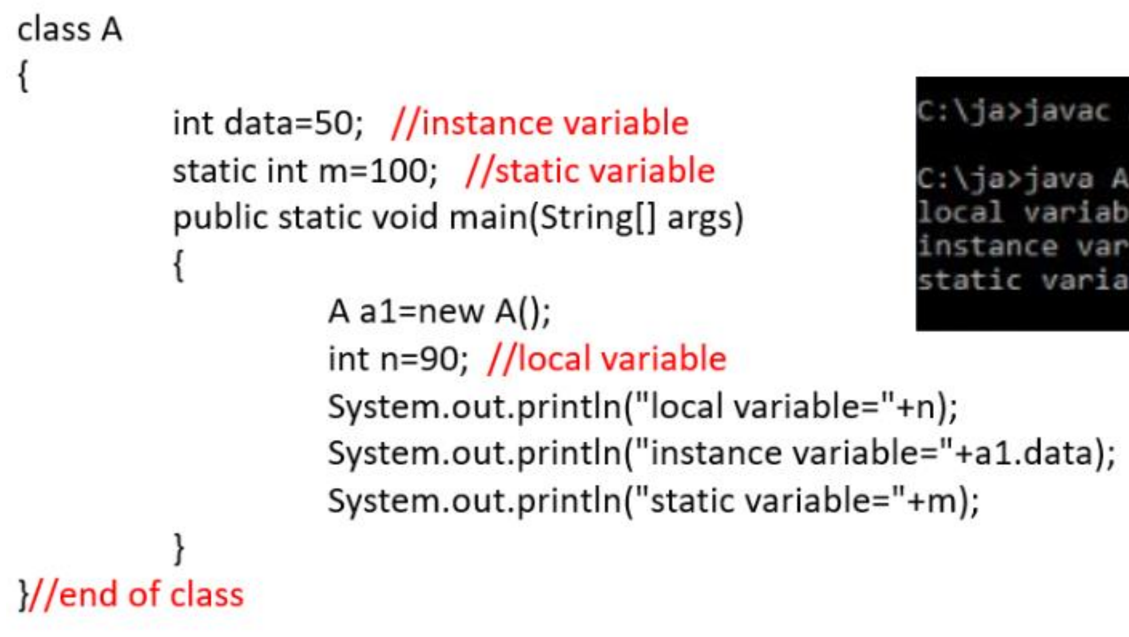
- A variable declared inside the body of the method is called local variable.
- Local Variables are a variable that are declared inside the body of a method.
- A local variable cannot be defined with "static" keyword.

2. Instance Variable

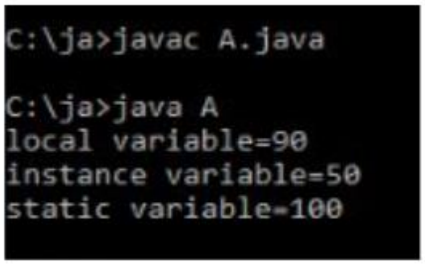
- A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

3. Static variable

- A variable which is declared as static is called static variable.
- It cannot be local.



```
class A
{
        int data=50;   //instance variable
        static int m=100;   //static variable
        public static void main(String[] args)
        {
                A a1=new A();
                int n=90;  //local variable
                System.out.println("local variable="+n);
                System.out.println("instance variable="+a1.data);
                System.out.println("static variable="+m);
        }
}//end of class
```

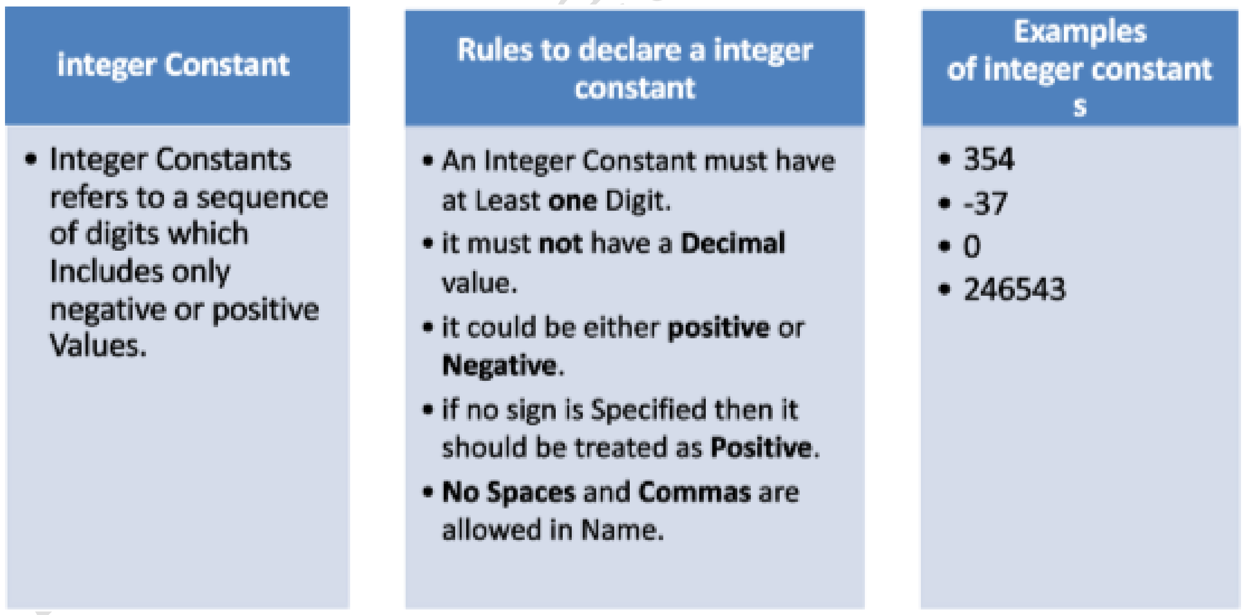


```
C:\ja>javac A.java

C:\ja>java A
local variable=90
instance variable=50
static variable=100
```

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.

14. Explain different constants and how to declare them.(Integer, real, char, string, backslash)

Ans.



integer Constant	Rules to declare a integer constant	Examples of integer constant s
• Integer Constants refers to a sequence of digits which Includes only negative or positive Values.	• An Integer Constant must have at Least **one** Digit. • it must **not** have a **Decimal** value. • it could be either **positive** or **Negative**. • if no sign is Specified then it should be treated as **Positive**. • **No Spaces** and **Commas** are allowed in Name.	• 354 • -37 • 0 • 246543

Real Constants

- A **real constant** is combination of a whole number followed by a decimal point or an exponent.

Rules to declare a real constant

- A Real Constant must have at Least one Digit.
- it must have a Decimal value.
- it could be either positive or Negative.
- if no sign is Specified then it should be treated as Positive.
- No Spaces and Commas are allowed in Name.

Examples of real constants

- 234.890
- 0.0083
- -0.75
- .25

Single char constant

- Character is Single alphabet a single digit or a single symbol that is enclosed within single inverted commas.

Examples of single char constant

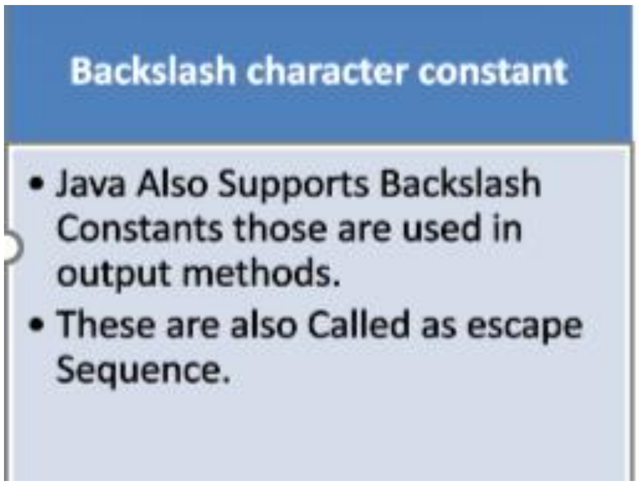
- 's'
- '1'
- 'A'

String constant

- String is a sequence of characters enclosed between double quotes. These characters may be digits ,Alphabets.

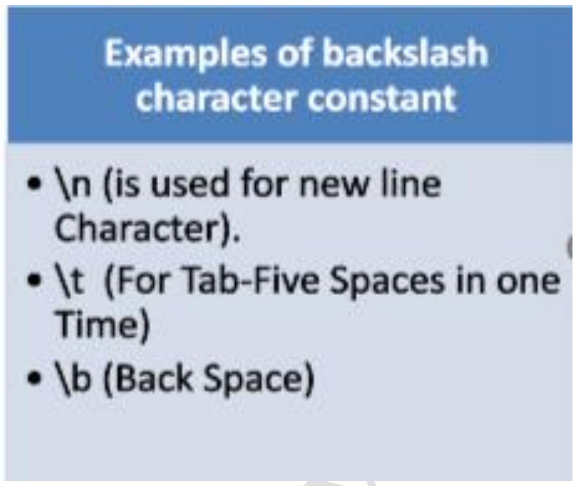
Examples of string constant

- "hello"
- "1234"
- "GOOD"



Backslash character constant

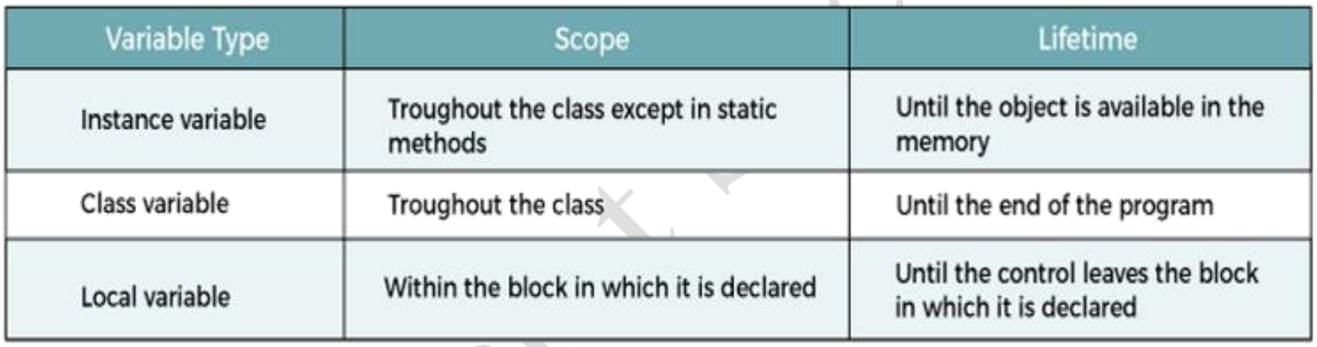
- Java Also Supports Backslash Constants those are used in output methods.
- These are also Called as escape Sequence.



Examples of backslash character constant

- \n (is used for new line Character).
- \t (For Tab-Five Spaces in one Time)
- \b (Back Space)

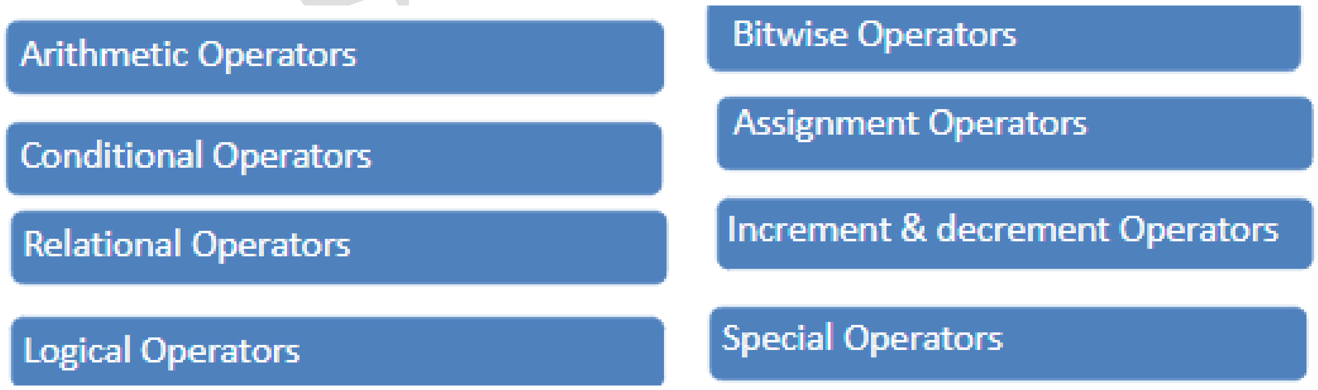
15. Explain scope and lifetime of variables.



Variable Type	Scope	Lifetime
Instance variable	Troughout the class except in static methods	Until the object is available in the memory
Class variable	Troughout the class	Until the end of the program
Local variable	Within the block in which it is declared	Until the control leaves the block in which it is declared

16. List the operators that are available in java along with examples.(8 operators)

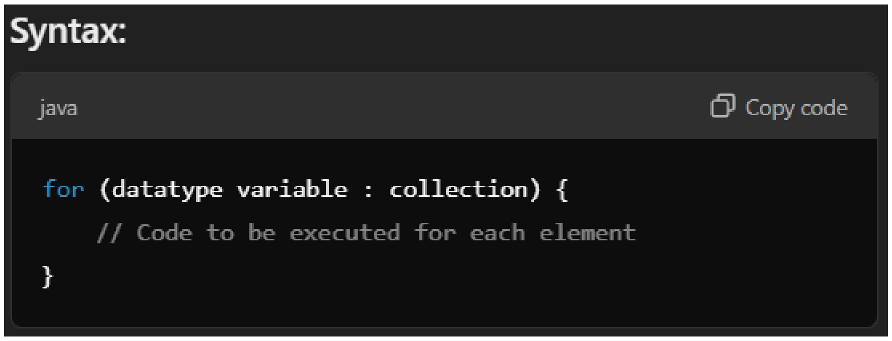
Ans.



- Arithmetic Operators
- Conditional Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Increment & decrement Operators
- Special Operators

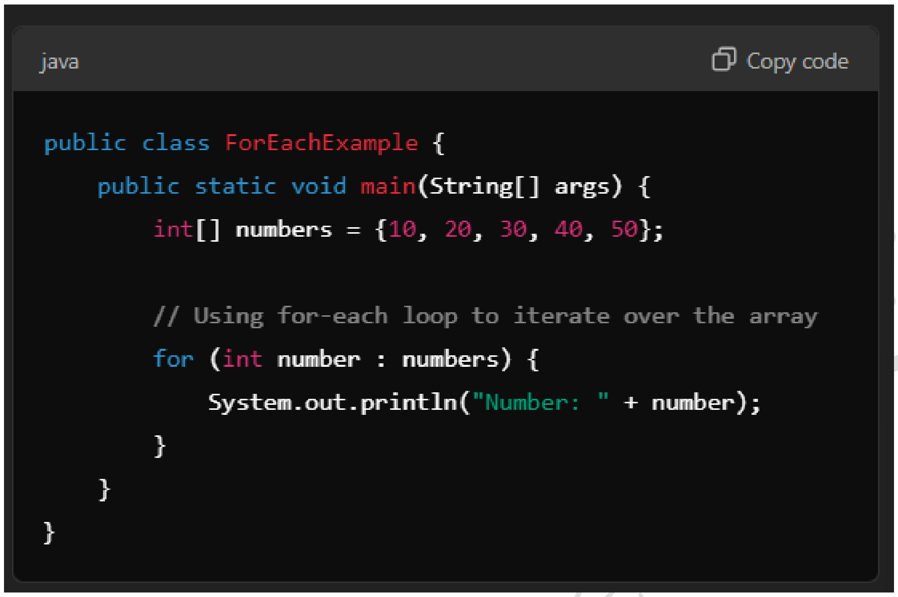
17. Explain the control flow statements that are available in Java.(<u>conditional</u>(if, if else,switch,ifelse if,nested if), <u>iterative</u>(for,for each, while, do while) and <u>jumping</u>(break,continue)

Ans.


Syntax:

java Copy code

```java
for (datatype variable : collection) {
    // Code to be executed for each element
}
```


java Copy code

```java
public class ForEachExample {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};

        // Using for-each loop to iterate over the array
        for (int number : numbers) {
            System.out.println("Number: " + number);
        }
    }
}
```

18. How to declare an array and initialize it? Write a program by declaring 1D,2D,Jaged array.

// write a program to calculate the length of each row of an array

```
class length

{

public static void main(String[] args)

{

int[][] a = {

{11, 12, 33, 45},

{4, 5, 6},

{7},

};

System.out.println("Length of row 1: " + a[0].length); System.out.println("Length of row 2: " + a[1].length);
System.out.println("Length of row 3: " + a[2].length);
}

}
```

19. Define class and explain the general syntax for defining a class in java with example.

Ans.

Definition: Class is a template or a blueprint that describes the behavior/state of an object of its type support.

```
class ClassName {
    // Fields (Attributes)
    datatype variableName;
   // Methods (Behaviors)
    returnType methodName(parameters) {
        // Method code
    }
    public static void main(String[] args) {
        // This is where execution begins
    }
}
```

20. Define object and write a program for creating an object in java.

Ans. **Definition**: An object is an instance of a class with its own state/behavior.

- State
- It represents the data (value) of an object.
- Behavior
- It 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.

21. What are methods, explain the types(Built in, User defined(Instance, Static Method))

Ans.

1. **Predefined Method**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.

It is also known as the standard library method or built-in method.

We can directly use these methods just by calling them in the program at any point. Ex: length(), equals(), compareTo(), sqrt()

2. User-defined Method

The method written by the user or programmer is known as a user-defined method.

These methods are modified according to the requirement.

1. **Instance Method:** Access the instance data using the object name. Declared inside a class.

```
public class student{
int id=10;
String name="abc";
void display() {
System.out.println(id+" "+name);
}
public static void main(String args[]){
Student s1=new Student();
s1.display();
}
```

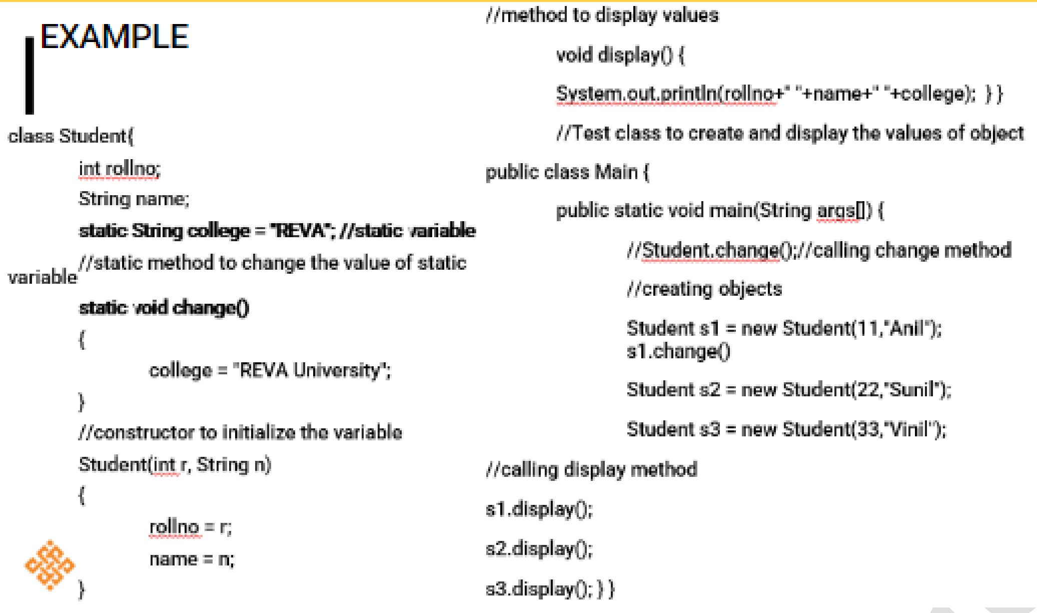
2. Static Methods

Static methods are methods that **do not operate on object.**

A static method **belongs to the class** rather than the object of a class.

A static method can be **invoked without the need for creating an instance of a class.**

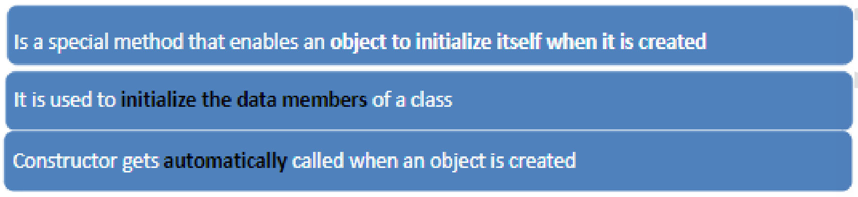
A static method can access static data member and can change the value of it.



EXAMPLE

```
class Student{
        int rollno;
        String name;
        static String college = "REVA"; //static variable
        //static method to change the value of static
variable
        static void change()
        {
                college = "REVA University";
        }
        //constructor to initialize the variable
        Student(int r, String n)
        {
                rollno = r;
                name = n;
        }
//method to display values
        void display() {
        System.out.println(rollno+" "+name+" "+college);  } }
        //Test class to create and display the values of object
public class Main {
        public static void main(String args[]) {
                //Student.change();//calling change method
                //creating objects
                Student s1 = new Student(11,"Anil");
                s1.change()
                Student s2 = new Student(22,"Sunil");
                Student s3 = new Student(33,"Vinil");
//calling display method
s1.display();
s2.display();
s3.display(); } }
```

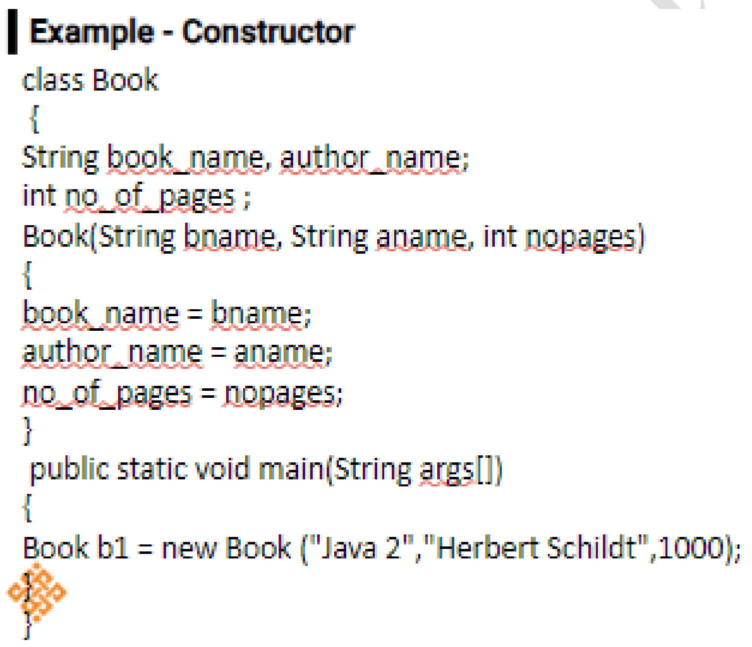
3. Constructors



Is a special method that enables an **object to initialize itself when it is created**

It is used to **initialize the data members** of a class

Constructor gets **automatically** called when an object is created

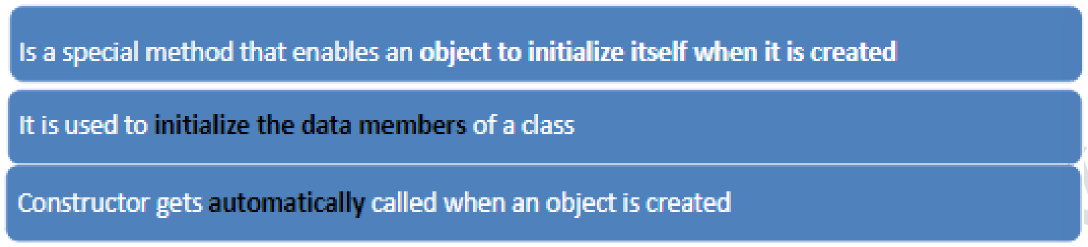


Example - Constructor

```
class Book
 {
String book_name, author_name;
int no_of_pages ;
Book(String bname, String aname, int nopages)
{
book_name = bname;
author_name = aname;
no_of_pages = nopages;
}
 public static void main(String args[])
{
Book b1 = new Book ("Java 2","Herbert Schildt",1000);
}
}
```

22. Define constructor and explain their types (default and parameter) with example.

Ans.



Is a special method that enables an object to initialize itself when it is created

It is used to initialize the data members of a class

Constructor gets automatically called when an object is created

Types of constructors:

1. Default:

Example of default constructor

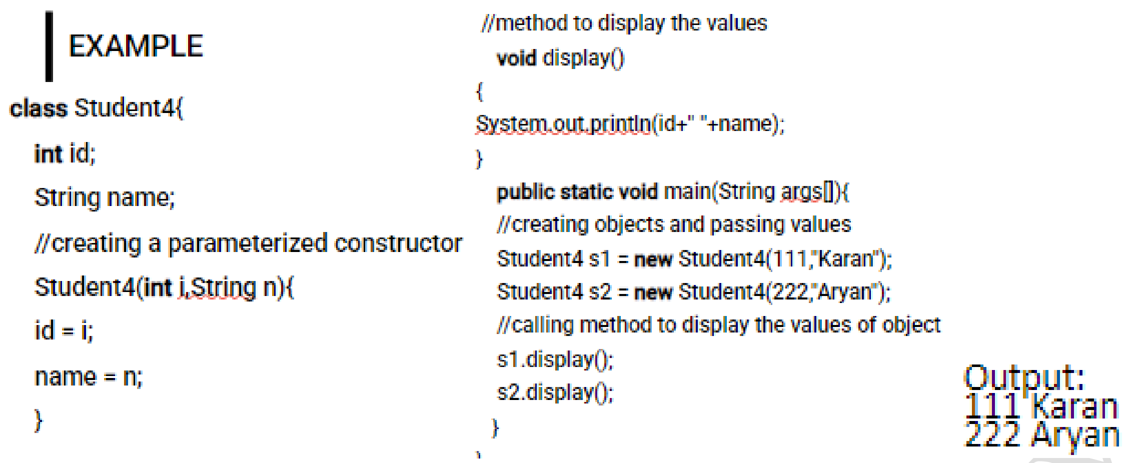
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{
Bike1(){
System.out.println("Bike is created");
}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

2. Parameterized:

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 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();
 }
}
```

```
Output:
111 Karan
222 Aryan
```

Unit 2

23. Define Inheritance and explain different types of inheritance with example.(Single, Multilevel, Hierarchical, Multiple, Hybrid)

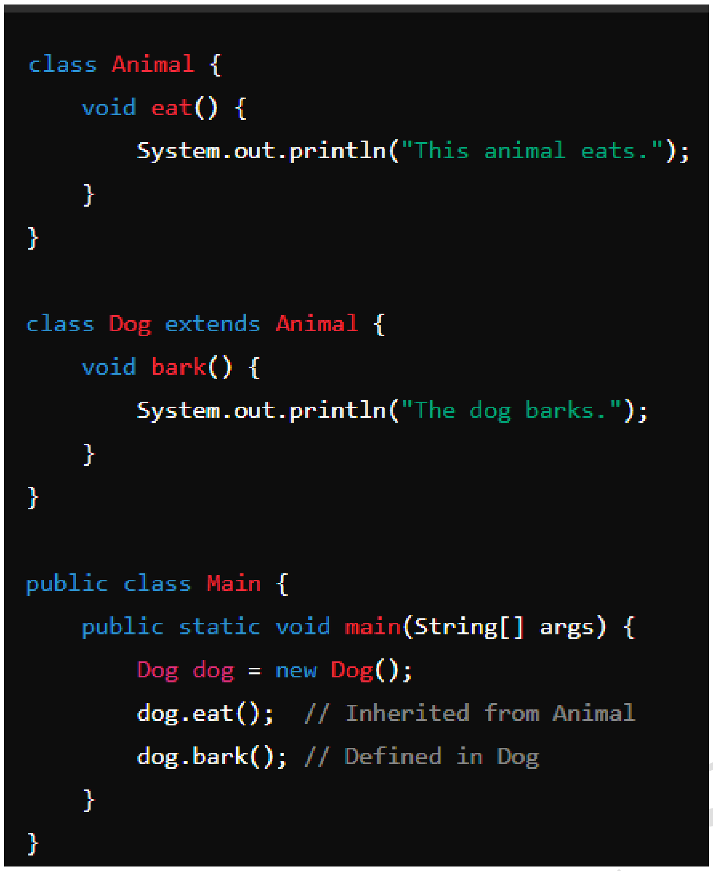
Ans.

Inheritance in Java is a mechanism where one class acquires the properties (fields) and behaviors (methods) of another class. The class that is inherited is called the **superclass** or **parent class**, and the class that inherits is called **the subclass or child class**.

Inheritance allows for code reusability, easier maintenance, and a logical hierarchical classification.

Single Inheritance:

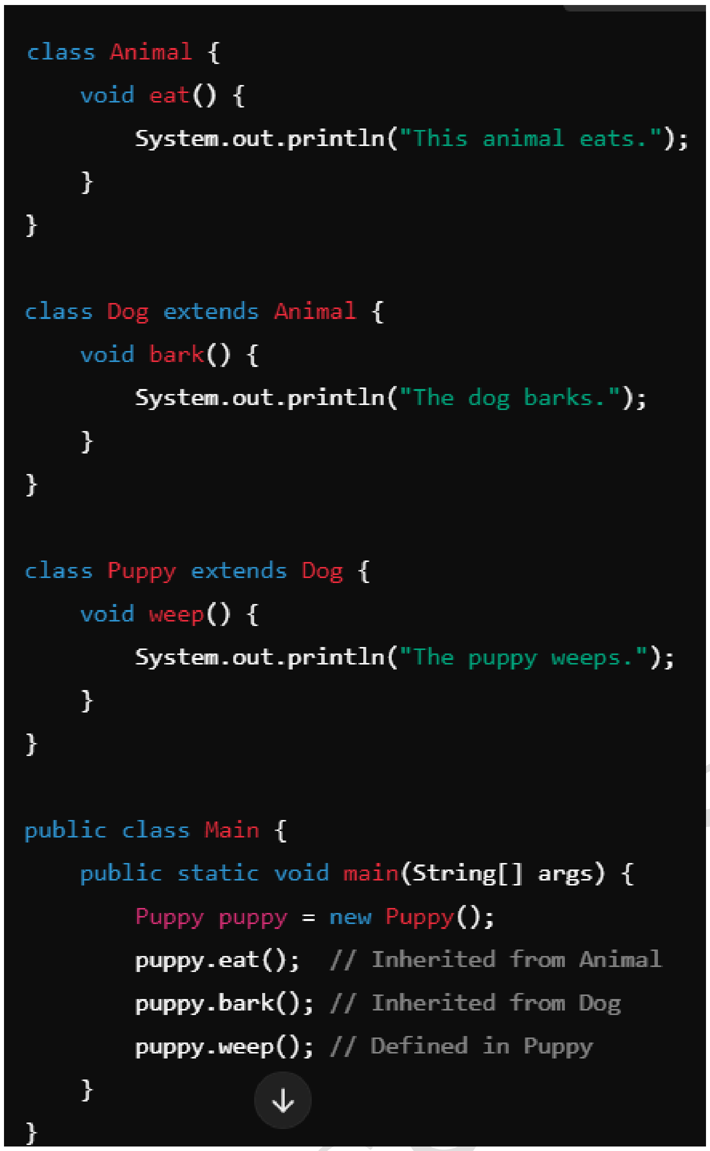
- A subclass inherits from only one superclass.



```java
class Animal {
    void eat() {
        System.out.println("This animal eats.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // Inherited from Animal
        dog.bark(); // Defined in Dog
    }
}
```

Multilevel Inheritance:

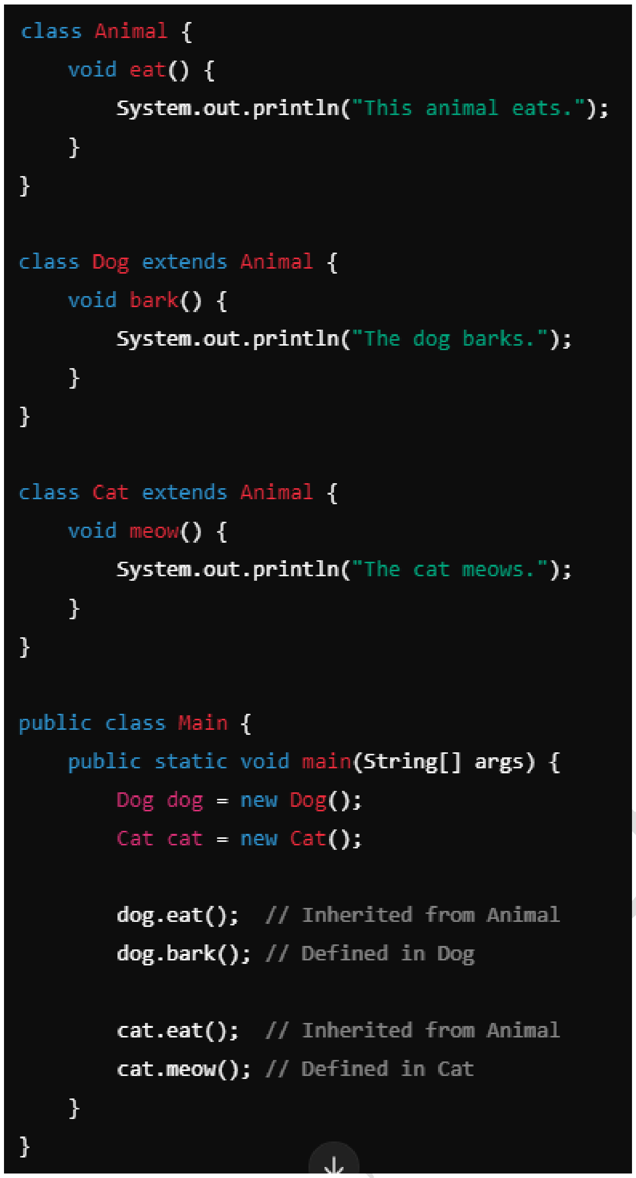
- A class is derived from a class which is also derived from another class, creating a chain.



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

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

class Puppy extends Dog {
    void weep() {
        System.out.println("The puppy weeps.");
    }
}

public class Main {
    public static void main(String[] args) {
        Puppy puppy = new Puppy();
        puppy.eat();  // Inherited from Animal
        puppy.bark(); // Inherited from Dog
        puppy.weep(); // Defined in Puppy
    }
}
```

Hierarchical Inheritance:

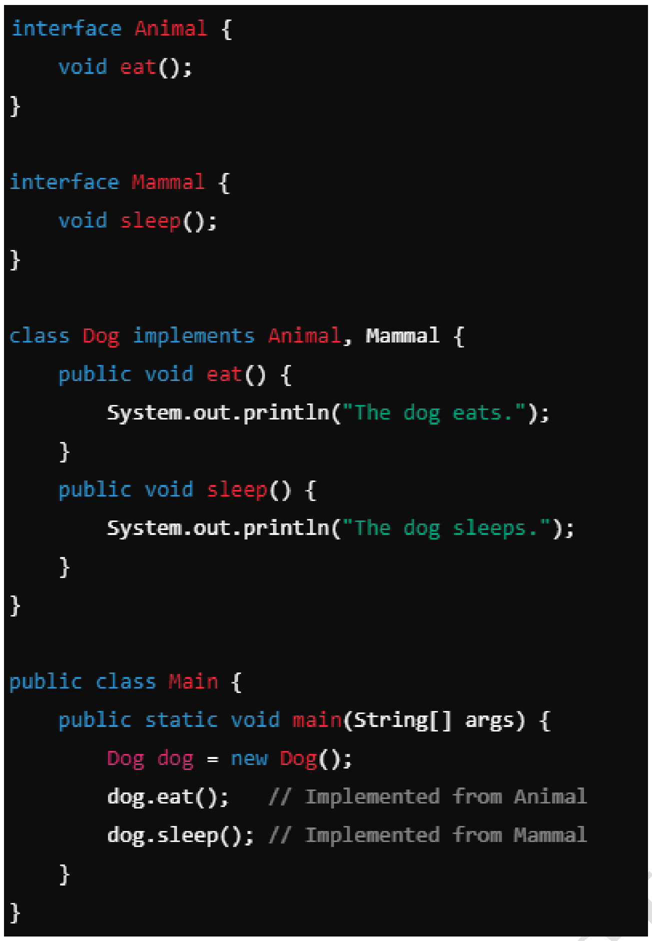
- Multiple classes inherit from a single superclass.



```java
class Animal {
    void eat() {
        System.out.println("This animal eats.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

class Cat extends Animal {
    void meow() {
        System.out.println("The cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();

        dog.eat();  // Inherited from Animal
        dog.bark(); // Defined in Dog

        cat.eat();  // Inherited from Animal
        cat.meow(); // Defined in Cat
    }
}
```

Multiple Inheritance (Not supported directly in Java):

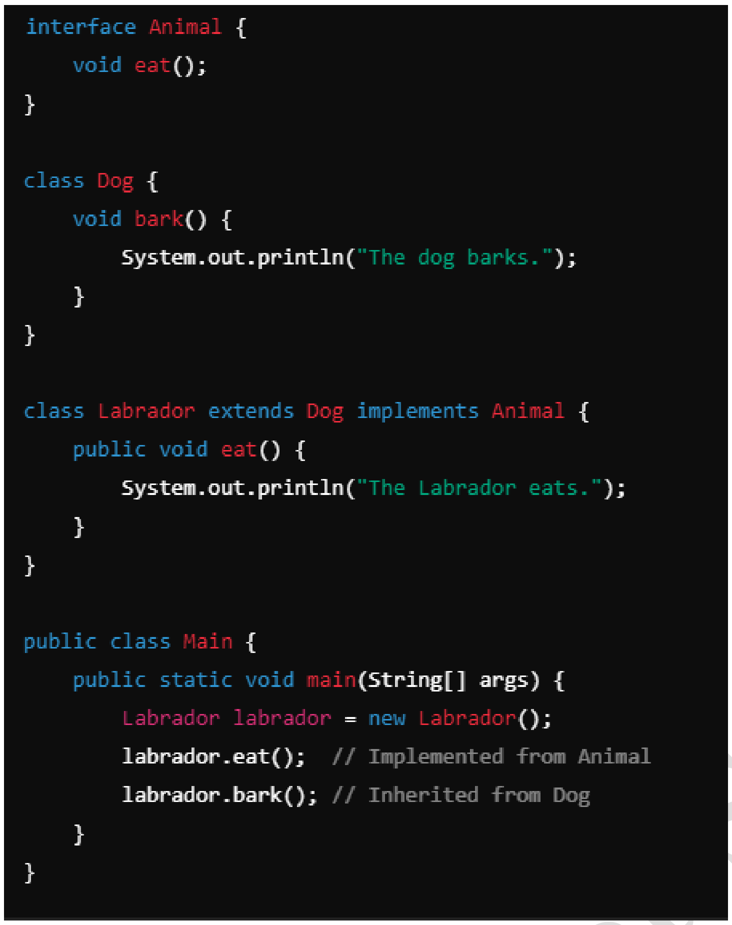
- A class inherits from more than one class.
- **Java does not support multiple inheritance with classes** due to ambiguity issues, but it is achieved using **interfaces**.



```
interface Animal {
    void eat();
}

interface Mammal {
    void sleep();
}

class Dog implements Animal, Mammal {
    public void eat() {
        System.out.println("The dog eats.");
    }
    public void sleep() {
        System.out.println("The dog sleeps.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();   // Implemented from Animal
        dog.sleep(); // Implemented from Mammal
    }
}
```

Hybrid Inheritance (Not directly supported):

- A combination of more than one type of inheritance, typically combining **hierarchical** and **multiple inheritance**.



```java
interface Animal {
    void eat();
}

class Dog {
    void bark() {
        System.out.println("The dog barks.");
    }
}

class Labrador extends Dog implements Animal {
    public void eat() {
        System.out.println("The Labrador eats.");
    }
}

public class Main {
    public static void main(String[] args) {
        Labrador labrador = new Labrador();
        labrador.eat();  // Implemented from Animal
        labrador.bark(); // Inherited from Dog
    }
}
```

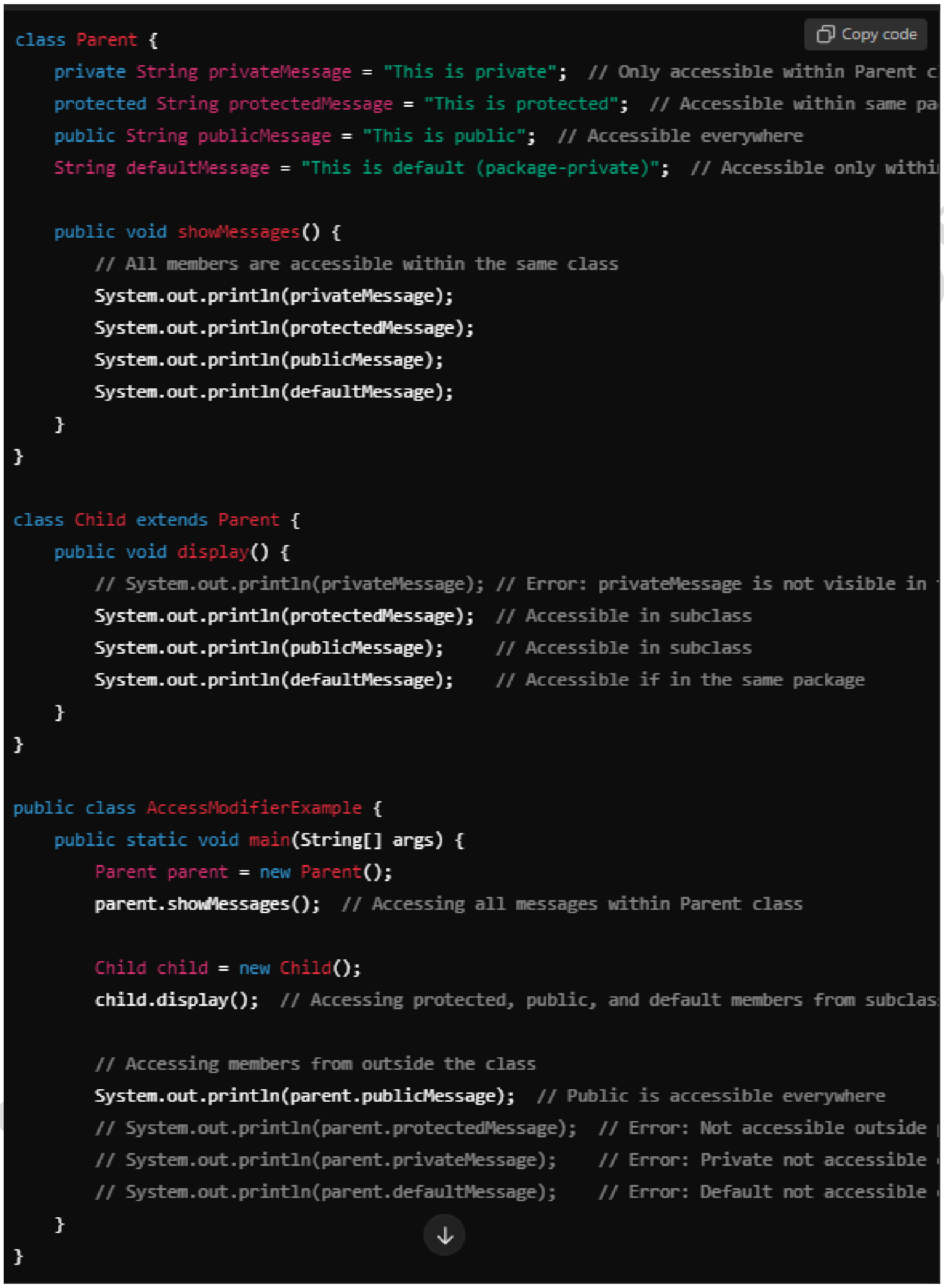
Summary of Types:

- **Single**: One class inherits from one superclass.
- **Multilevel**: A class inherits from a subclass, forming a chain.
- **Hierarchical**: Multiple classes inherit from the same superclass.
- **Multiple**: A class implements multiple interfaces (not multiple classes).
- **Hybrid**: A combination of multiple and other inheritance types using interfaces.

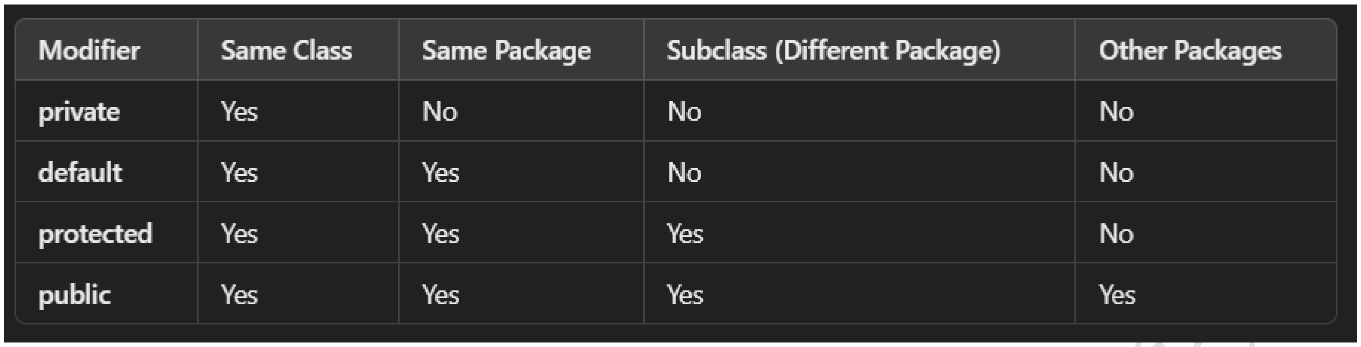
24. Does Java support multiple inheritance. Justify along with the example.
Ans. **Same as 23rd Multiple Inheritence**

25. Explain member access rule with example(Private, Protected, Public, Default)

Ans.



```java
class Parent {
    private String privateMessage = "This is private";  // Only accessible within Parent c
    protected String protectedMessage = "This is protected";  // Accessible within same pa
    public String publicMessage = "This is public";  // Accessible everywhere
    String defaultMessage = "This is default (package-private)";  // Accessible only withi

    public void showMessages() {
        // All members are accessible within the same class
        System.out.println(privateMessage);
        System.out.println(protectedMessage);
        System.out.println(publicMessage);
        System.out.println(defaultMessage);
    }
}

class Child extends Parent {
    public void display() {
        // System.out.println(privateMessage); // Error: privateMessage is not visible in
        System.out.println(protectedMessage);  // Accessible in subclass
        System.out.println(publicMessage);     // Accessible in subclass
        System.out.println(defaultMessage);    // Accessible if in the same package
    }
}

public class AccessModifierExample {
    public static void main(String[] args) {
        Parent parent = new Parent();
        parent.showMessages();  // Accessing all messages within Parent class

        Child child = new Child();
        child.display();  // Accessing protected, public, and default members from subclas

        // Accessing members from outside the class
        System.out.println(parent.publicMessage);  // Public is accessible everywhere
        // System.out.println(parent.protectedMessage);  // Error: Not accessible outside
        // System.out.println(parent.privateMessage);    // Error: Private not accessible
        // System.out.println(parent.defaultMessage);    // Error: Default not accessible
    }
}
```



Modifier	Same Class	Same Package	Subclass (Different Package)	Other Packages
private	Yes	No	No	No
default	Yes	Yes	No	No
protected	Yes	Yes	Yes	No
public	Yes	Yes	Yes	Yes

26. Explain the use of this and super keyword in java along with example.

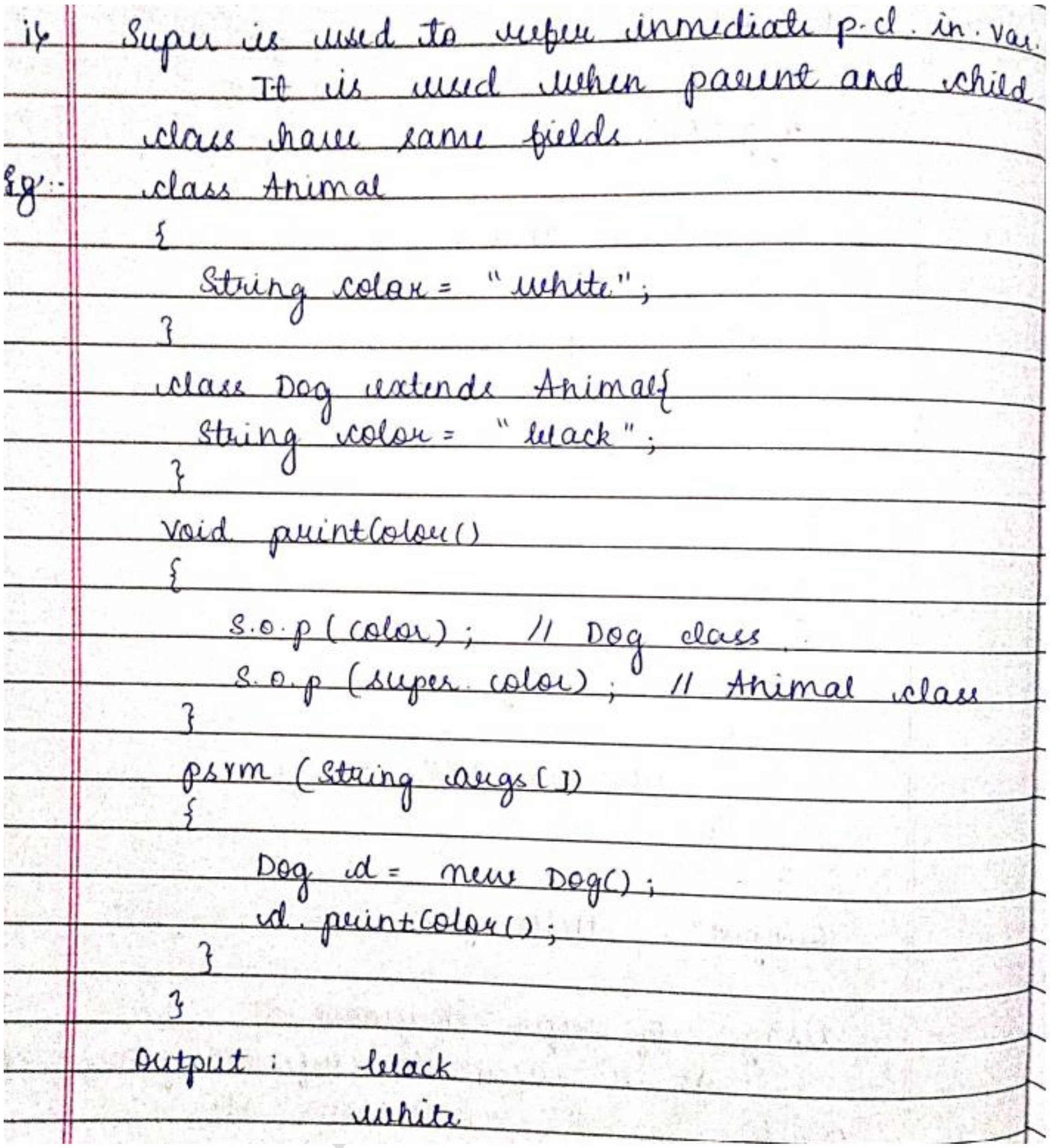
Ans.

In Java, both the this and super keywords are used to reference objects, but they serve different purposes. The **this** keyword refers to the current object instance, while **super** is used to refer to the immediate parent class object.

super Keyword:

The super keyword is used to refer to the **immediate parent class object**. It is used to:

- Access parent class members (methods or variables) that are hidden by child class members.
- Call the parent class constructor.
- Access parent class methods that are overridden in the child class.



1) Super is used to refer immediate p.cl. in. var.
It is used when parent and child class have same fields.

Eg::
```
class Animal
{
   String color = "white";
}
class Dog extends Animal{
   String color = "black";
}
void printColor()
{
     s.o.p(color);  // Dog class.
     s.o.p(super.color);  // Animal class
 }
 psvm (String args[])
 {
      Dog d = new Dog();
      d.printColor();
  }
}
```
Output : black
         white

this Keyword:

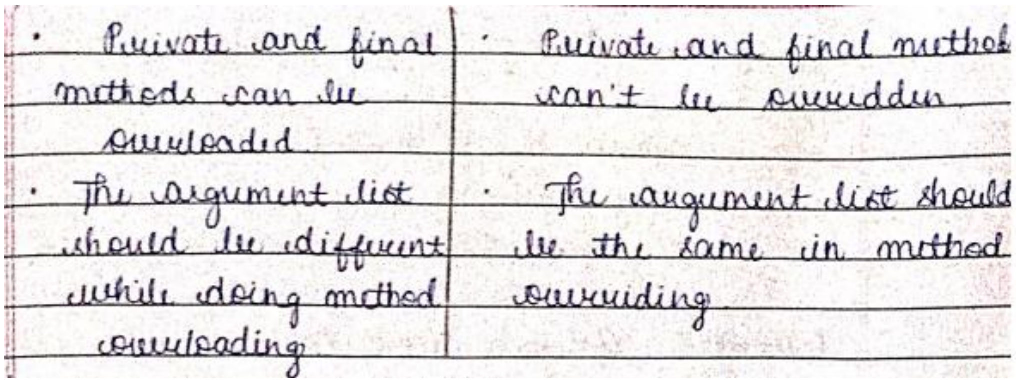
The this keyword refers to the current instance of the class. It is used to:

- Differentiate between instance variables and parameters when they have the same name.
- Invoke other constructors within the same class.
- Pass the current object as a parameter to a method.

```
class Student {
    String name;
    int age;
    // Constructor with the same parameter names as instance variables
    Student(String name, int age) {
        this.name = name;  // 'this' refers to the current instance's variable
        this.age = age;    // 'this' is used to distinguish between instance variable and parameter
    }
    void display() {
        System.out.println("Name: " + this.name + ", Age: " + this.age);
    }
    Student() {
        this("John", 20);  // Using 'this' to call another constructor
    }
}
public class Main {
    public static void main(String[] args) {
        Student student = new Student("Alice", 22);
        student.display();
        Student student2 = new Student(); // Calls the constructor with default values
        student2.display();
        }
}
```

27. Difference between method overloading and method overriding with examples.

Method Overloading	Method Overriding
• Known as compile-time polymorphism	• Known as run-time polymorphism
• Helps to increase the readability of the program.	• Used to grant the specific implementation of the method which is already provided by its parent class or superclass.
• It occurs within the class	• It is performed in two classes with inheritance relationships
• May or may not require inheritance.	• Always needs inheritance
• Methods must have the same name and different signature	• Methods must have th same name and same signature
• The return type can or cannot be the same, but we just have to change the parameter.	• The return type must be the same or co-variant
• Static binding is being used for overloaded methods.	• Dynamic binding is being used for overridin methods.
• Poor performance due to compile time poly.	• It gives better performan The reason behind is that the binding of overridden methods is being done at the runtime.



• Private and final methods can be overloaded.	• Private and final methods can't be overridden
• The argument list should be different while doing method overloading	• The argument list should be the same in method overriding

Progarms

28. Write a java program to create a class called car with the following members color, manufacturing date, vehicle Number and model. Initialize the values of the members of the class using parameterized constructor and display the values of the members.

Ans.

```
class Car {
   private String color;
   private String manufacturingDate;
   private String vehicleNumber;
   private String model;
   public Car(String color, String manufacturingDate, String vehicleNumber, String model) {
      this.color = color;
      this.manufacturingDate = manufacturingDate;
      this.vehicleNumber = vehicleNumber;
      this.model = model;
   }
   public void displayCarDetails() {
      System.out.println("Car Details:");
      System.out.println("Color: " + color);
      System.out.println("Manufacturing Date: " + manufacturingDate);
      System.out.println("Vehicle Number: " + vehicleNumber);
      System.out.println("Model: " + model);
   }
}

public class Main {

    public static void main(String[] args) {

        Car car = new Car("Red", "2023-07-15", "KA05AB1234", "Tesla Model S");

        car.displayCarDetails();

    }

}
```

29. Write a program in java to perform simple mathematical operation by taking the input from user.

Ans.

```
import java.util.Scanner;

public class SimpleMathOperations {
    public static void main(String[] args) {
        // Create a Scanner object for taking user input
        Scanner scanner = new Scanner(System.in);

        // Taking user input for two numbers
        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();

        // Taking user input for the operation
        System.out.println("Choose the operation (+, -, *, /): ");
        char operation = scanner.next().charAt(0);

        // Variable to store the result
        double result = 0;

        // Perform the operation based on user input
        switch (operation) {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 * num2;
                break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                } else {
                    System.out.println("Error! Division by zero is not allowed.");
                    return;
                }
                break;
            default:
                System.out.println("Invalid operation selected!");
                return;
        }

        // Display the result
        System.out.println("The result of the operation is: " + result);

        // Close the scanner
        scanner.close();
    }
}
```

30. Implement stack program with maximum stack size as 5 using class and object concept in Java. Stack class should have push(), pop() and display() methods defined.

Ans.

```
// Stack class definition
class Stack {
    // Maximum size of the stack
    private final int MAX_SIZE = 5;
    private int[] stack = new int[MAX_SIZE];
    private int top = -1;

    // Method to push an element onto the stack
    public void push(int value) {
        if (top >= MAX_SIZE - 1) {
            System.out.println("Stack Overflow! Cannot push " + value);
        } else {
            top++;
            stack[top] = value;
            System.out.println(value + " pushed onto stack.");
        }
    }

    // Method to pop an element from the stack
    public void pop() {
        if (top < 0) {
            System.out.println("Stack Underflow! No elements to pop.");
        } else {
            System.out.println(stack[top] + " popped from stack.");
            top--;
        }
    }

    // Method to display all elements of the stack
    public void display() {
        if (top < 0) {
            System.out.println("Stack is empty.");
        } else {
            System.out.println("Stack elements are:");
            for (int i = top; i >= 0; i--) {
                System.out.println(stack[i]);
            }
        }
    }
}

// Main class to test Stack operations
public class Main {
    public static void main(String[] args) {
        // Create a Stack object
        Stack stack = new Stack();

        // Test stack operations
        stack.push(10);  // Pushing elements onto the stack
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);
        stack.push(60);  // Should display "Stack Overflow!"

        stack.display(); // Display stack elements

        stack.pop();     // Popping elements from the stack
        stack.pop();

        stack.display(); // Display stack elements after pop
    }
}
```

31. A class Box has three instance variables namely Height, Width, Depth, and it also has below four different forms overloaded constructors and a method volume.

    Box ()

    Box (double h, double w, double d)

    Box (double len)

    Box (Box ob)

    Double volume( )

    Create a class Box with above instance variables and all the above constructors and a method defined.

    Create another class BoxWeight which is an inherited class of Box. BoxWeight class also has instance variable weight with different constructors. Using super keyword call constructors of Box class within the constructors of BoxWeight to initialize all instance variables. Print volume by using different objects created.

Ans.

```
// Box class definition
class Box {
    double height, width, depth;

    // Default constructor
    Box() {
        height = width = depth = 0;
    }

    // Parameterized constructor to initialize height, width, and depth
    Box(double h, double w, double d) {
        height = h;
        width = w;
        depth = d;
    }

    // Constructor to initialize all dimensions to the same value (cube)
    Box(double len) {
        height = width = depth = len;
    }

    // Copy constructor to initialize a Box object from another Box object
    Box(Box ob) {
        height = ob.height;
        width = ob.width;
        depth = ob.depth;
    }

    // Method to calculate the volume of the box
    double volume() {
        return height * width * depth;
    }
```

```
}

// BoxWeight class inheriting from Box
class BoxWeight extends Box {
    double weight;

    // Default constructor
    BoxWeight() {
        super();  // Call the default constructor of Box
        weight = 0;
    }

    // Parameterized constructor to initialize height, width, depth, and weight
    BoxWeight(double h, double w, double d, double m) {
        super(h, w, d);  // Call the parameterized constructor of Box
        weight = m;
    }

    // Constructor to initialize a cube box with weight
    BoxWeight(double len, double m) {
        super(len);  // Call the cube constructor of Box
        weight = m;
    }

    // Copy constructor
    BoxWeight(BoxWeight ob) {
        super(ob);  // Call the copy constructor of Box
        weight = ob.weight;
    }

    // Method to display weight
    void displayWeight() {
        System.out.println("Weight: " + weight);
    }
}

// Main class to test Box and BoxWeight
public class Main {
    public static void main(String[] args) {
        // Create various Box and BoxWeight objects

        // Using the default constructor
        Box box1 = new Box();
        System.out.println("Volume of box1: " + box1.volume());

        // Using the parameterized constructor with three arguments
        Box box2 = new Box(2.0, 3.0, 4.0);
        System.out.println("Volume of box2: " + box2.volume());

        // Using the constructor for cube
        Box box3 = new Box(5.0);
        System.out.println("Volume of box3 (cube): " + box3.volume());

        // Using the copy constructor
        Box box4 = new Box(box2);
        System.out.println("Volume of box4 (copied from box2): " + box4.volume());

        // Create BoxWeight objects
        BoxWeight boxWeight1 = new BoxWeight(2.0, 3.0, 4.0, 10.0);
        System.out.println("Volume of boxWeight1: " + boxWeight1.volume());
        boxWeight1.displayWeight();

        // BoxWeight object for cube
        BoxWeight boxWeight2 = new BoxWeight(3.0, 12.0);
        System.out.println("Volume of boxWeight2 (cube): " + boxWeight2.volume());
        boxWeight2.displayWeight();
        // Using the copy constructor for BoxWeight
        BoxWeight boxWeight3 = new BoxWeight(boxWeight1);
        System.out.println("Volume of boxWeight3 (copied from boxWeight1): " + boxWeight3.volume());
        boxWeight3.displayWeight();
    }
}
```