Module : 1

1. Introduction to Java

* Theory :
* History of Java :
* Java is a programming language originally developed by James Gosling at Sun Microsystems.
* and released in 1995 as a core component of Sun Microsystems' Java platform.
* The language derives much of its syntax from C and C++.
* Java is currently one of the most popular programming languages in use, particularly for client server web applications.
* Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.
* Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.
* Java's motto was **"Write Once, Run Anywhere"** (WORA), highlighting its platform independence using the Java Virtual Machine (JVM).
* Oak was renamed to **Java** (the name was inspired by Java coffee).
* Summary :
*  **Creator**: James Gosling and team at Sun Microsystems
*  **First Released**: 1995
*  **Current Owner**: Oracle Corporation
*  **Key Features**: Platform independence, robustness, security, object-oriented
*  **Modern Usage**: Continues to evolve with regular updates and broad community support.
* Features of Java :
* Simple
* Object – Oriented
* Portable
* Distribute
* High Performance
* Multithreaded
* Robust
* Dynamic
* Secure
* Object – Oriented :
* Java is fully Object-oriented programming language. & Object oriented concepts are following.
* Class : A blueprint for creating objects. It defines properties (fields) and behaviors (methods).
* Object : A real-world entity created from a class. It has a state and behavior.
* Inheritance : Bundling data and methods that operate on that data within a class, and hiding internal details from outside access.
* Encapsulation : A mechanism where one class acquires properties and behavior of another class.
* Polymorphism : Ability to take many forms. There are two types:   
   • **Compile-time** (method overloading)   
   • **Runtime** (method overriding)
* Platform Independent :
* **Platform independence** means that a Java program can run on any operating system or hardware platform **without modification**.
* Allows Java applications to run across different operating systems without rewriting code.
* **Source Code Compilation:**
* Java code (.java file) is compiled by the Java compiler (javac) into an intermediate format called **bytecode** (.class file).
* This bytecode is **not** platform-specific.
* **Java Virtual Machine (JVM):**
* The bytecode is executed by the **JVM**, which is platform-specific.
* JVM interprets the bytecode and runs it on the native machine.
* Understanding JVM, JRE, and JDK
* JVM : Java Virtual Machine
* It is a platform-independent execution environment that converts the Java bytecode into machine language and executes it.
* Most programming languages compile source code directly into machine code that is designed to run on a specific operating system, such as Windows or UNIX.
* JVMs are available for many hardware and software platforms.
* The use of the same bytecode for all JVMs on all platforms allows Java to be described as a "compile once, run anywhere" programming language, as opposed to "write once, compile anywhere", which describes crossplatform compiled languages.
* Thus, the JVM is a crucial component of the Java platform.
* JRE : Java Run-Time Environment
* Java Runtime Environment contains JVM, class libraries, and other supporting files.
* It does not contain any development tools such as compiler, debugger, etc.
* Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE.
* . If you want to run any java program, you need to have JRE installed in the system.
* Is a part of the Java Development Kit (JDK) that provides the necessary components to run Java applications.
* It acts as a layer between the Java application and the underlying operating system, handling the execution of Java bytecode.
* JDK : Java Development Kit
* is a comprehensive software development environment used for developing Java applications and applets.
* Java Developer Kit contains tools needed to develop the Java programs
* **Appletviewer:-** (For viewing java applets) this tool can be used to run and debug Java applets without a web browser
* **Javac:-** Javac means Java Compiler, which converts source code into Java bytecode.
* **Java :-** ( java interpreter) the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment.
* The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.
* **Javap :-** ( java disassembler) the class file disassemble.
* **Javah:-** (produce header files) the C header and stub generator, used to write native methods.
* **Java doc:-** (creating html document) the documentation generator, which automatically generates documentation from source code comments.
* **Jdb:-** java debugger.
* Setting up the Java environment and IDE (e.g., Eclipse, IntelliJ)
* Step 1: Install Java Development Kit (JDK)
* **Download and Install:**
* Go to the official [Oracle JDK download page](https://www.oracle.com/java/technologies/javase-downloads.html) or use [OpenJDK](https://jdk.java.net/).
* Download the latest LTS (Long Term Support) version (e.g., Java 17 or Java 21).
* Install it and note the installation path.
* **Set Environment Variables (if not automatically set):**
* **On Windows :**
* Open System Properties > Environment Variables.
* Add a new system variable:
  + **Name:** JAVA\_HOME
  + **Value:** C:\Program Files\Java\jdk-XX
* Edit the Path variable and add: %JAVA\_HOME%\bin
* **Check installation:**
* bash
* java -version
* javac -version

Step 2: Install an IDE

* **Option A: Eclipse IDE**
* **Download & Install:**
* Download from: https://www.eclipse.org/downloads/
* Choose “Eclipse IDE for Java Developers” and install it.
* **First-Time Setup:**
* Launch Eclipse.
* Choose a workspace folder (your projects will be stored here).
* Install any updates it recommends.
* Create a new Java project via:
* File > New > Java Project
* **Option B: IntelliJ IDEA**
* **Download & Install:**
* Download from: <https://www.jetbrains.com/idea/download/>
* Choose the **Community Edition** (free) or **Ultimate** (paid).
* **First-Time Setup:**
* Run IntelliJ and complete the first-time setup wizard.
* Create a new project:
* New Project > Java > Select SDK
* (You can add your installed JDK here.)
* Start coding in the **src** folder.
* IDEs support Maven or Gradle for managing dependencies.
* Java Program Structure (Packages, Classes, Methods)
* **1. Document Section :-** In Documentation section one can write author name, definition of class or a program , description of a program and how this algorithm works and other details, and this is OPTIONAL part and this part write in (/\* \*/)multiline comment.
* **2. Package Statement :-** In java program first statement one can write is a package statement. This statement declares a package name and tells the compiler that the classes included here belongs to this package; the package statement is optional in any java program For ex:-package first; .
* **3. Import Statement:-** Once a package is declared one can write any number of import statements. import statement can be write after package statement and before the class definition. For ex:-import java.util.\*; Import first.\*; .
* **4. Class Declaration :-** In java program there may be more than one class. In this class definition statement we have to provide keyword class along with the class name. A class can include variable as well as methods.
* **5. Main Method :-** In a java program there may be more than one class one class can have main method. This is the essential part of java program. In main method one can create objects of different classes and with the use of there
* **Lab Exercise:**
* Install JDK and set up environment variables.
* Write a simple "Hello World" Java program.
* Compile and run the program using command-line tools (javac, java).
* **Program :**

**public** **class** hello {

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

{

System.***out***.println("hello world");

}

}

1. **Data Types, Variables, and Operators**

* **Theory :**
* **Primitive Data Types in Java (int, float, char, etc.)**
* The primitive data types are predefined data types, which always hold the value of the same data type, and the values of a primitive data type don't share the state with other primitive values.
* These data types are named by a reserved keyword in Java programming language.
* There are eight primitive data types supported by Java programming language:
* **byte :**
* The byte data type is an 8-bit signed two's complement integer. It ranges from -128 to127 (inclusive).
* This type of data type is useful to save memory in large arrays.
* We can also use byte instead of int to increase the limit of the code.
* Syntax : byte b = 5;
* **short:**
* The short data type is a 16-bit signed two's complement integer.
* It ranges from -32,768 to 32,767.short is used to save memory in large arrays.
* **Syntax :** short s = 2;
* **int :** The int data type is used to store the integer values not the fraction values.
* It is a 32-bit signed two's complement integer data type.
* It ranges from -2,147,483,648 to 2,147,483,647 that are more enough to store large number in your program.
* However for wider range of values use long.
* **float :**
* The float data type is a single-precision 32-bit IEEE 754 floating point.
* It ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).
* Use a float (instead of double) to save memory in large arrays.
* **double :**
* This data type is a double-precision 64-bit IEEE 754 floating point.
* It ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative).
* This data type is generally the default choice for decimal values.
* **char :**
* The char data type is a single 16-bit, unsigned Unicode character.
* It ranges from 0 to 65,535.
* They are not integral data type like int, short etc. i.e. the char data type can't hold the numeric values.
* **Boolean :**
* The boolean data type represents only two values: true and false and occupy is 1-bit in the memory.
* These values are keywords in Java and represent the two boolean states: on or off, yes or no.
* We use boolean data type for specifying conditional statements as if, while, do, for.
* Variable Declaration and Initialization
* A **variable** is a name that refers to a memory location used to store data.
* **Instance Variables (Non-static fields):**
* In object oriented programming, objects store their individual states in the "non-static fields" that is declared without the static keyword.
* **Class Variables (Static fields):** These are collectively related to a class and none of the object can claim them its soleproprietor.
* The variables defined with static keyword are shared by all objects.
* **Local Variables:** The variables defined in a method or block of code is called local variables.
* **Parameters:** Parameters or arguments are variables used in method declarations.
* Operators: Arithmetic, Relational, Logical, Assignment, Unary, and Bitwise
* **Operators**
* + Additive operator (also used for String concatenation)
* - Subtraction operator
* \* Multiplication operator
* / Division operator
* % Remainder operator
* **Unary Operators**
* ++ Increment operator; increments a value by 1
* -- Decrement operator; decrements a value by 1
* ! Logical complement operator
* **Relational Operator**
* = = Equal to
* != Not equal to
* > Greater than
* > = Greater than or equal to
* < Less than
* < = Less than or equal to
* **Conditional Operators**
* && Conditional-AND
* || Conditional-OR
* ?: Ternary (shorthand for if-then-else statement)
* **Bitwise and Bit Shift Operators**
* ~ Unary bitwise complement
* << Signed left shift
* >> Signed right shift
* >>> Unsigned right shift
* & Bitwise AND
* ^ Bitwise exclusive OR
* | Bitwise inclusive OR
* Type Conversion and Type Casting
* (Widening / Implicit Casting)
* In Java, **Type Conversion** and **Type Casting** are processes used to convert one data type into another. Though often used interchangeably, they refer to slightly different concepts depending on whether the conversion is done **automatically** or **manually**.
* This is the automatic conversion of a smaller data type to a larger data type. It happens when there is no risk of data loss.
* **Also known as:** Widening Conversion or Implicit Casting
* When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:
* The two types are compatible.
* The destination type is larger than the source type.
* Although the automatic type conversions are helpful, they will not fulfill all needs.
*  Done automatically by the compiler
*  No data loss
*  Safe and preferred when moving from smaller to larger data types
* Type Casting(Narrowing / Explicit Casting)
* This is the manual conversion of a larger data type to a smaller data type. It must be explicitly done by the programmer because it may result in data loss.
* **Also known as:** Narrowing Conversion or Explicit Casting
* **Ex.:**
* double d = 10.5;
* int num = (int) d;
* **Key Points:**
* Requires explicit syntax: (targetType) value
* Can lead to data loss (e.g., truncating decimal values)
* Must be used carefully
* **Lab Exercise:**
* Write a program to demonstrate the use of different data types.

**public** **class** DatatypeDemo {

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

{

**byte** b= 5;

**short** s = 2;

**int** i = 10;

**float** f = 12.1f;

**double** d = 6677.60;

**char** ch = 'c';

**boolean** bl = **false**;

System.**out**.println("This byte : "+b);

System.**out**.println("This short : "+s);

System.**out**.println("This integer : "+i);

System.**out**.println("This float : "+f);

System.**out**.println("This double : "+d);

System.**out**.println("This char : "+ch);

System.**out**.println("This boolean : "+b);

}

}

* Create a calculator using arithmetic and relational operators.

package basicJava;

import java.util.Scanner;

public class SimpleCalculator {

public static void main(String[] args) {

int a,b;

double5 result;

char operator;

Scanner sc = new Scanner(System.in);

System.out.print("Enter A :");

a=sc.nextInt();

System.out.print("Enter B :");

b=sc.nextInt();

System.out.print("choose the operator (+,-,\*,/) :");

operator=sc.next().charAt(0);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

switch(operator)

{

case '+':

result=a+b;

System.out.println("Result :"+result);

break;

case '-':

result=a-b;

System.out.println("Result :"+result);

break;

case '\*':

result=a\*b;

System.out.println("Result :"+result);

break;

case '/':

if(b!=0)

{

result=a/b;

System.out.println("Result :"+result);

}

else

{

System.out.println("This number can not divide by Zero");

}

break;

default:

System.out.println("Invalid operator..");

}

}

}

* Demonstrate type casting (explicit and implicit).

class Conversion {

public static void main(String args[]) {

byte b;

int i = 257;

double d = 323.142;

System.out.println("\nConversion of int to byte.");

b = (byte) i;

System.out.println("i and b " + i + " " + b);

System.out.println("\nConversion of double to int.");

i = (int) d;

System.out.println("d and i " + d + " " + i);

System.out.println("\nConversion of double to byte.");

b = (byte) d;

System.out.println("d and b " + d + " " + b);

}

}

Output :- Conversion of int to byte.

i and b 257 1

Conversion of double to int.

d and i 323.142 323

Conversion of double to byte.

d and b 323.142 67

1. **Control Flow Statements**

* **Theory:**
* If – Else Statement :
* A programming language uses control statements to cause the flow of execution to advance and branch based on changes to the state of a program.
* A common programming construct that is based upon a sequence of nested ifs is the ifelse-if ladder.
* Syntax:
* if(<Boolean\_expression>)

<Statement\_or\_block>

elseif(<Boolean\_expression>)

<Statement\_or\_block>

elseif(<Boolean\_expression>)

<Statement\_or\_block>

else

<Statement\_or\_block>

* Switch Case Statements
* This is an easier implementation to the if-else statements.
* The keyword "switch" is followed by an expression that should evaluates to byte, short, char or int primitive data types , only. In a switch block there can be one or more labeled cases.
* Syntax:
* switch ( expression) {

case : <constant1>:

<statement\_or\_block>\* ;;;;;;;;;;;;;;;;;;;;;

[break;]

case : <constant2>:

<statement\_or\_block>\*

[break;]

default:

<statement\_or\_block>\*

[break;]

* Loops (For, While, Do-While)
* In Java, loops are control flow statements that repeatedly execute a block of code as long as a specified condition is true.
* Loops are fundamental for performing repetitive tasks efficiently.
* **For Loop :**
* **Use case**: When the number of iterations is known beforehand.
* Syntax :
* For(initialization;condition;iteration)
* {
* //body
* }
* When the loop first starts, the initialization portion of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter that controls the loop. It is important to understand that the initialization expression is only executed once. Next, condition is evaluated. This must be a Boolean expression.
* **while Statements**
* The while loop is Java’s most fundamental looping statement.
* It repeats a statement or block while its controlling expression is true.
* While(condition)
* {
* //body
* }
* The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.
* **Do-While statement**
* The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.
* **Break and Continue Keywords**
* In Java, the break statement has three uses.
* First, as you have seen, it terminates a statement sequence in a switch statement.
* Second, it can be used to exit a loop.
* Third, it can be used as a “civilized” form of go to.
* By using break, you can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
* When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.
* **Continue Statement :**
* Sometimes it is useful to force an early iteration of a loop.
* That is, you might want to continue running the loop, but stop processing the remainder of the code in its body for this particular iteration.
* **Lab Exercise:**
* Write a program to find if a number is even or odd using an if-else statement.

Public class IfElase

{

Public Static void main(String[] args)

{

int i;

System.out.println(“enter number :”);

i=sc.nextInt();

if(i%2==0)

{

System.out.println(i+“number is even”);

}

Else

{

System.out.println(i+“number is odd”);

}

}

* Implement a simple menu-driven program using a switch-case.

int day = 5;

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;

default:

System.out.println("Invalid entry");

break;

}

* Write a program to display the Fibonacci series using a loop.

public class FibonacciSeries {

public static void main(String[] args) {

int n;

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of terms for the Fibonacci series: ");

i = scanner.nextInt();

int first = 0, second = 1;

System.out.println("Fibonacci Series up to " + n + " n:");

for (int i = 1; i <= n; i++) {

System.out.print(first + " ");

// Calculate next term

int next = first + second;

first = second;

second = next;

}

}

}

1. **Classes and Objects**

* **Theory:**
* Defining a Class and Object in Java
* A class is a blueprint or prototype from which objects are created.
* This section defines a class that models the state and behavior of a real-world object.
* Syntax :
* <Modifier>\*class<class\_name>
* <attribute\_declaration>
* <constructor\_declaration>
* <method\_declaration>
* **Object :**
* An object is a instance of class. We can also say that object is blue print of the class.
* We use the all member of the class. Using object name But we can not use private member of class.
* Classname objectname = new classname();
* Constructors and Overloading.
* **Constructor**
* It is a special member function whose task is to initialization the objects of it’s classes.
* A Constructor initializes an object immediately upon creation.
* **Constructor**
* It has the same name as the class in which it resides and is syntactically similar to a method.
* They do not have any return type.
* They are invoked automatically when object are created.
* **Types**
* 1. Default(which have no arguments, arg provided by java automatically)
* 2. Parameterized(which have arguments)
* 3. Copy constructor(which have object as arguments)
* **Constructor overloading**
* When there is more than one constructor in a single class there is called constructor overloading.
* Must have different number or types of parameters.
* Helps to create objects in different ways.
* Object Creation, Accessing Members of the Class
* An object is an instance of a class. When you create an object, Java allocates memory and allows you to access the class’s methods and variables.
* **Accessing Members of the Class**
* Once you create an object, you can access the members (variables and methods) of the class using the dot (.)operator.
* **Lab Exercise:**
* Create a class Student with attributes (name, age) and a method to display the details.

class student

{

String name;

int age;

void set(String n,int a){

name =n;

age = a;

}

Void display()

{

System.out.println(“student name :”+n);

System.out.println(“student age is :”+age);

}

public static void main(String[] args)

{

Student s1=new student();

S1.set(“bansari”,20);

S1.display();

}

}

* Create multiple constructors in a class and demonstrate constructor overloading.

class Student {

String name;

int age;

Student() {

name = "Unknown";

age = 0;

}

Student(String name) {

this.name = name;

age = 18;

}

Student(String name, int age) {

this.name = name;

this.age = age;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

// Using default constructor

Student s1 = new Student();

s1.display();

// Using constructor with one parameter

Student s2 = new Student(“bansari”);

S2.display();

Student s3 = new Student(“banari”,20);

S3.display();

}

}

* Implement a simple class with getters and setters for encapsulation.

1. **Methods in Java**

* **Theory**
* Defining Methods
* a method is a block of code that performs a specific task. Methods help you organize and reuse code efficiently.
* Predefined Methods :Already available in Java
* (e.g., System.out.println(),ath.sqrt()).
* User-defined Methods :
* Created by the programmer to perform specific tasks.
* Method Parameters and Return Types
* Method parameters are variables passed to a method when it is called. They allow methods to work with input values.
* **Parameters** - Inputs passed to a method.
* **Return Type** - The data type of the value returned by the method.
* **Void** - Used when no value is returned.
* Method Overloading
* Method Overloading in Java means defining multiple methods in the same class with the same name but different parameters.
* Improves code readability and reusability.
* Allows the same method name to perform different tasks based on arguments.

* Static Methods and Variables
* It is use to declare variable, methods and block
* Allocate the common memory
* static variable - Shared by all objects of a class
* static method - Can be called without creating an object
* Limitation - Can access only static members directly
* **Lab Exercise:**
* Write a program to find the maximum of three numbers using a method.
* class MaximumFinder {

int findMax(int a, int b, int c) {

int max = a;

if (b > max) {

max = b;

}

if (c > max) {

max = c;

}

return max;

}

}

public class Main {

public static void main(String[] args) {

MaximumFinder mf = new MaximumFinder();

// Test values

int num1 = 25, num2 = 42, num3 = 17;

int max = mf.findMax(num1, num2, num3);

System.out.println("The maximum of " + num1 + ", " + num2 + ", and " + num3 + " is: " + max);

}

}

* Implement method overloading by creating methods for different data types.

package basicJava;

public class MethodOverLoding {

void test()

{

System.out.println("test with no argument");

}

void test(int a)

{

System.out.println("text with 1 argument");

}

void test(int a,int b)

{

System.out.println("test with 2 argument");

}

public static void main(String[] args) {

MethodOverLoding m=new MethodOverLoding();

m.test();

m.test(10);

m.test(10,20);

}

}

* Create a class with static variables and methods to demonstrate their use.

public class StaticMethod {

static int a=10;

static int b;

static void math(int x)

{

System.out.println("Math Method called..");

System.out.println("x :"+x);

System.out.println("A :"+a);

System.out.println("B :"+b);

}

static

{

System.out.println("Static block initialized");

b=a\*4;

}

public static void main(String[] args) {

System.out.println("Main method called..");

math(12);

}

}

1. **Object-Oriented Programming (OOPs) Concepts**

* **Theory:**
* **Basics of OOP: Encapsulation, Inheritance, Polymorphism, Abstraction**
* **Encapsulation :** Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
* Encapsulation is the process of wrapping data (variables) and methods (functions) together as a single unit (class), and restricting direct access to some of the object's components.
* Use private variables.
* Use public getter and setter methods to access them.
* **Inheritance :**
* Inheritance allows one class to inherit fields and methods from another class. It supports code reuse.
* The object of one class can aquire the properties of object of another class.
* Creating a new class from an existing class is called inheritance.
* **Polymorphism:**
* Polymorphism means "many forms". It allows one name to be used for different behaviors.
* One name multiple form
* **Type of polymorphism :**
* Method Overloading (compile-time)
* Method Overriding (runtime)
* **Abstraction**
* Abstraction means hiding the internal details and showing only the essential features.
* There are situations in which you will want to define a super class that declares the structure of a given abstraction without providing a complete implementation of every method.
* **Inheritance: Single, Multilevel, Hierarchical**
* **Single Inheritance: -** One class can acquire properties of one super class.
* **Multiple Inheritance: -** One class can acquire properties of more than one classes.
* **Multi-level Inheritance: -** In multilevel inheritance, the ladder of single inheritance increases.
* **Method Overriding and Dynamic Method Dispatch**
* **Method Overriding :** when there is same method prototypes in your both base class and derived class and if you called that method using the object of derived class than only derived class method will be called and say that method of derived class overrides the method of base class.
* Method Overriding is a feature that allows a subclass (child class) to provide a specific implementation of a method that is already defined in its superclass (parent class).
* The method must have the same name, return type, and parameters.
* The method must be inherited from the superclass.
* **Dynamic Method Dispatch :** Dynamic Method Dispatch is the mechanism by which a call to an overridden method is resolved at runtime rather than compile time. It is also known as runtime polymorphism.
* When there is more than one in a single class having the same name but with different number of argument and their datatype there it is called method overloading.
* **Lab Exercise:**
* Write a program demonstrating single inheritance.

class Person {

String name;

int age;

void displayPersonDetails() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

}

}

class Student extends Person {

String course;

void displayStudentDetails() {

System.out.println("Course: " + course);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student();

s1.name = "Alice";

s1.age = 20;

s1.course = "Computer Science";

s1.displayPersonDetails();

s1.displayStudentDetails();

}

* Create a class hierarchy and demonstrate multilevel inheritance.

class Animal {

void eat() {

System.out.println("Animal eats food");

}

}

class Dog extends Animal {

void bark() {

System.out.println("Dog barks");

}

}

class Puppy extends Dog {

void weep() {

System.out.println("Puppy weeps");

}

}

public class Main {

public static void main(String[] args) {

// Create an object of Puppy

Puppy p = new Puppy();

// Call methods from all levels of inheritance

p.eat(); // Inherited from Animal

p.bark(); // Inherited from Dog

p.weep(); // Defined in Puppy

}

}

* Implement method overriding to show polymorphism in action.

class Animal {

void makeSound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

void makeSound() {

System.out.println("Dog barks");

}

}

class Cat extends Animal {

void makeSound() {

System.out.println("Cat meows");

}

}

public class Main {

public static void main(String[] args) {

Animal a;

a = new Dog(); // a refers to Dog

a.makeSound(); // Output: Dog barks

a = new Cat(); // a refers to Cat

a.makeSound(); // Output: Cat meows

}

}

* **Constructors and Destructors**
* **Theory:**
* Constructor Types (Default, Parameterized)
* A Constructor initializes an object immediately upon creation.
* It has the same name as the class in which it resides and is syntactically similar to a method.
* Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes.
* Constructors look a little strange because they have no return type, not even void.
* **Default :** There is always at least one constructor in every class.
* A default constructor is a constructor with no parameters.
* **Parameterized:**
* A parameterized constructor is a constructor that accepts arguments to initialize an object with specific values.
* Copy Constructor (Emulated in Java)
* A copy constructor is a special constructor used to create a new object as a copy of an existing object.
* Emulating a Copy Constructor in Java :
* class Student {

String name;

int age;

Student(String name, int age) {

this.name = name;

this.age = age;

}

Student(Student s) {

this.name = s.name;

this.age = s.age;

}

void display() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

Student s1 = new Student("Alice", 20); // original object

Student s2 = new Student(s1); // copy using copy constructor

s1.display();

s2.display();

}

}

* Constructor Overloading
* Constructor overloading in Java means having more than one constructor in a class with different parameter lists. This allows an object to be initialized in multiple ways.
* Constructors must have the same name as the class
* Each overloaded constructor must differ in the number or type of parameters.
* It improves code flexibility and readability.

* Object Life Cycle and Garbage Collection
* Java is an object-oriented programming language that manages memory automatically through its Garbage Collector (GC).
* The object life cycle refers to the phases an object goes through from creation to destruction:

1. Object Creation

* An object is created using the new keyword.
* The constructor of the class is invoked.
* Memory is allocated on the heap.
* Example:  
  Student s = new Student();

2. Object Usage

* The object’s methods and properties are used as long as a reference exists.
* Example:  
  s.display();

3. Object Becomes Unreachable

* When no references to the object exist, it becomes unreachable and is eligible for garbage collection.
* Example:  
  s = null;

4. Garbage Collection

* The Java Garbage Collector automatically deallocates memory by destroying unreachable objects.
* This helps avoid memory leaks.