**Basic­\_Encapsulation**  
Create a class Person with private fields name and age. Write getter and setter methods for these fields. Instantiate the class and modify the values using the setter methods, then retrieve them using the getter methods.

**Access\_Modifiers\_Practice**  
Write a class Student with a public field name, a private field id, a protected field grade, and a default access field age. Demonstrate accessing these fields from within the same package and a different package.

**Encapsulation\_with\_Validation**  
Create a class BankAccount with private fields accountNumber and balance. Implement getter and setter methods for balance that ensure the balance cannot be negative. Test this behavior in a main method.

**Access\_Modifier\_Effects**  
Write a class Animal with a public method speak(), a protected method eat(), a private method sleep(), and a default access method move(). Create another class in the same package to show which methods can be accessed.

**Encapsulation\_in\_Constructor**  
Implement a class Rectangle with private fields length and width. Use a constructor to initialize these fields and provide getter methods to retrieve their values. Show how to create an instance of this class and access its fields.

**Encapsulation\_with\_Inheritance**  
Create a class Vehicle with private fields make and model. Derive a class Car from Vehicle and add a public method displayInfo() that shows the make and model. Demonstrate how encapsulation affects accessibility in the subclass.

**Access\_Modifiers\_in\_Different\_Packages**  
Write a class Employee with public, protected, private, and default access fields and methods. Create another class in a different package to test access to these members and explain which ones are accessible.

**Encapsulation\_and\_Method\_Overloading**  
Develop a class Circle with private fields radius and color. Provide overloaded methods for setting the radius and color. Use encapsulation to ensure that the radius cannot be negative and the color is a valid color name.

**Encapsulation\_with\_Multiple\_Classes**  
Design a class Library with private fields bookTitle and author. Provide getter and setter methods for these fields. Create a second class LibraryDemo to create an instance of Library and modify its fields using the provided methods.

**Access\_Modifier\_and\_Method\_Overriding**  
Implement a class Person with a public method introduce(), and a class Employee that extends Person and overrides introduce(). Show how the method can be overridden and what access levels are required for the method in the subclass.

**1. Basic Encapsulation**

// Person.java

public class Person {

private String name;

private int age;

// Getter and Setter for name

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// Getter and Setter for age

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public static void main(String[] args) {

Person person = new Person();

person.setName("Alice");

person.setAge(30);

System.out.println("Name: " + person.getName());

System.out.println("Age: " + person.getAge());

}

}

**2. Access Modifiers Practice**

// Student.java

public class Student {

public String name;

private int id;

protected char grade;

int age; // default access

// Getter and Setter for id

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public static void main(String[] args) {

Student student = new Student();

student.name = "John";

student.setId(12345);

student.grade = 'A';

student.age = 20;

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

System.out.println("ID: " + student.getId());

System.out.println("Grade: " + student.grade);

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

}

}

**3. Encapsulation with Validation**

// BankAccount.java

public class BankAccount {

private String accountNumber;

private double balance;

public BankAccount(String accountNumber, double balance) {

this.accountNumber = accountNumber;

setBalance(balance);

}

// Getter for balance

public double getBalance() {

return balance;

}

// Setter for balance with validation

public void setBalance(double balance) {

if (balance >= 0) {

this.balance = balance;

} else {

System.out.println("Balance cannot be negative.");

}

}

public static void main(String[] args) {

BankAccount account = new BankAccount("123456", 1000);

System.out.println("Initial Balance: " + account.getBalance());

account.setBalance(-500); // Should trigger validation

System.out.println("Balance after setting negative value: " + account.getBalance());

account.setBalance(1500);

System.out.println("Updated Balance: " + account.getBalance());

}

}

**4. Access Modifier Effects**

// Animal.java

class Animal {

public void speak() {

System.out.println("Animal speaks.");

}

protected void eat() {

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

}

private void sleep() {

System.out.println("Animal sleeps.");

}

void move() {

System.out.println("Animal moves.");

}

}

// TestAnimal.java (in the same package)

public class TestAnimal {

public static void main(String[] args) {

Animal animal = new Animal();

animal.speak(); // Accessible

animal.eat(); // Accessible

// animal.sleep(); // Not Accessible (private)

animal.move(); // Accessible

}

}

**5. Encapsulation in Constructor**

// Rectangle.java

public class Rectangle {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

// Getter for length

public double getLength() {

return length;

}

// Getter for width

public double getWidth() {

return width;

}

public static void main(String[] args) {

Rectangle rectangle = new Rectangle(5.0, 3.0);

System.out.println("Length: " + rectangle.getLength());

System.out.println("Width: " + rectangle.getWidth());

}

}

**6. Encapsulation with Inheritance**

// Vehicle.java

public class Vehicle {

private String make;

private String model;

public Vehicle(String make, String model) {

this.make = make;

this.model = model;

}

// Getter for make

public String getMake() {

return make;

}

// Getter for model

public String getModel() {

return model;

}

}

// Car.java

public class Car extends Vehicle {

public Car(String make, String model) {

super(make, model);

}

public void displayInfo() {

System.out.println("Make: " + getMake());

System.out.println("Model: " + getModel());

}

public static void main(String[] args) {

Car car = new Car("Toyota", "Corolla");

car.displayInfo();

}

}

**7. Access Modifiers in Different Packages**

// Employee.java (in package com.example)

package com.example;

public class Employee {

public String name;

protected int id;

private double salary;

String department; // default access

public Employee(String name, int id, double salary, String department) {

this.name = name;

this.id = id;

this.salary = salary;

this.department = department;

}

// Getter for salary

public double getSalary() {

return salary;

}

}

// TestEmployee.java (in package com.other)

package com.other;

import com.example.Employee;

public class TestEmployee {

public static void main(String[] args) {

Employee emp = new Employee("Alice", 101, 50000, "HR");

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

// System.out.println("ID: " + emp.id); // Not accessible (protected)

// System.out.println("Salary: " + emp.getSalary()); // Not accessible (private)

// System.out.println("Department: " + emp.department); // Not accessible (default)

}

}

**8. Encapsulation and Method Overloading**

// Circle.java

public class Circle {

private double radius;

private String color;

public Circle() {

this.radius = 1.0;

this.color = "Red";

}

public Circle(double radius) {

this.radius = radius;

this.color = "Red";

}

public Circle(double radius, String color) {

this.radius = radius;

this.color = color;

}

// Setter for radius with validation

public void setRadius(double radius) {

if (radius >= 0) {

this.radius = radius;

} else {

System.out.println("Radius cannot be negative.");

}

}

// Setter for color

public void setColor(String color) {

this.color = color;

}

public double getRadius() {

return radius;

}

public String getColor() {

return color;

}

public static void main(String[] args) {

Circle circle = new Circle();

System.out.println("Default Radius: " + circle.getRadius());

System.out.println("Default Color: " + circle.getColor());

circle.setRadius(5.0);

circle.setColor("Blue");

System.out.println("Updated Radius: " + circle.getRadius());

System.out.println("Updated Color: " + circle.getColor());

}

}

**9. Encapsulation with Multiple Classes**

// Library.java

public class Library {

private String bookTitle;

private String author;

// Getter for bookTitle

public String getBookTitle() {

return bookTitle;

}

// Setter for bookTitle

public void setBookTitle(String bookTitle) {

this.bookTitle = bookTitle;

}

// Getter for author

public String getAuthor() {

return author;

}

// Setter for author

public void setAuthor(String author) {

this.author = author;

}

}

// LibraryDemo.java

public class LibraryDemo {

public static void main(String[] args) {

Library library = new Library();

library.setBookTitle("Java Programming");

library.setAuthor("John Doe");

System.out.println("Book Title: " + library.getBookTitle());

System.out.println("Author: " + library.getAuthor());

}

}

**10. Access Modifier and Method Overriding**

// Person.java

public class Person {

public void introduce() {

System.out.println("I am a person.");

}

}

// Employee.java

public class Employee extends Person {

@Override

public void introduce() {

System.out.println("I am an employee.");

}

public static void main(String[] args) {

Person person = new Person();

person.introduce(); // Calls Person's introduce method

Employee employee = new Employee();

employee.introduce(); // Calls Employee's introduce method

}

}