

**LAB RECORD**

23CSE111 – Object Oriented Programming

***Submitted by***

CH.SC.U4CSE24107– CHEEDELLA JYOTHIRMAI

**BACHELOR OF TECHNOLOGY**

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

## 

**AMRITA VISHWA VIDYAPEETHAM**

**AMRITA SCHOOL OF COMPUTING, CHENNAI**

**BONAFIDE CERTIFICATE**

This is to certify that the Lab Record work for 23CSE111- Object Oriented Programming Subject submitted by ***CH.SC.U4CSE24107 – CHEEDELLA JYOTHIRMAI*** in **“Computer Science and Engineering”** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Chennai.

|  |  |
| --- | --- |
| Internal Examiner 1 | Internal Examiner 2 |

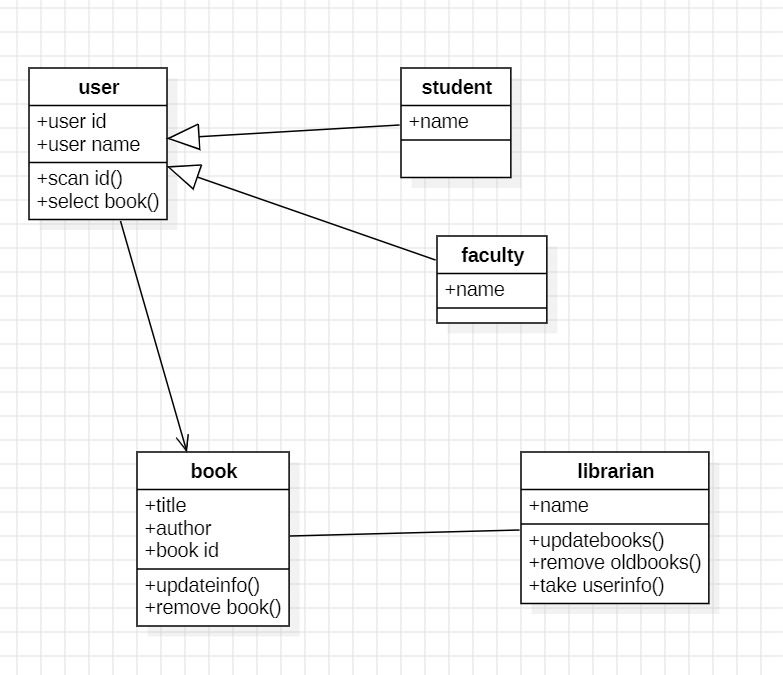
# **Index**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Experiment Name** | **Page Number** |
| 1. | **UML DIAGRAM** | 4 |
|  | Library Management   * Class Diagram * Use Case Diagram * Sequence Diagram * State Activity Diagram * Object Diagram | 4 |
| 2. | Online Shopping   * Class Diagram * Use Case Diagram * Sequence Diagram * State Activity Diagram * Object Diagram | 8 |
| 3. | **Basic Java Programs** | 12 |
| i) | Even or Odd using an if-else statement. | 12 |
| ii) | Maximum of three numbers using if-else statements. | 13 |
| iii) | Calculator(Addition, subtraction, multiplication, and division ) | 14 |
| iv) | Fibonacci series | 15 |
| v) | Reverse a number | 16 |
| vi) | Factorial of a number | 17 |
| vii) | Prime Number Check | 18 |
| viii) | Palindrome Check | 19 |
| xi) | Armstrong Number | 20 |
| x) | Sum Of Natural Number | 21 |
|  | **INHERITANCE** |  |
| 4. | **Hierarchical Inheritance** |  |
| i) | Vehicle Details | 26 |
| ii) | Student Details | 29 |
| 5. | **SINGLE INHERITANCE** |  |
| i) | Student Details | 31 |
| ii) | Bank Details | 32 |
| 6. | **MULTILEVEL INHERITANCE** |  |
| i) | General Details | 34 |
| ii) | Employee Details | 35 |
| 7. | **HYBRID INHERITANCE** |  |
| i) | Assist Details | 36 |
| ii) | Vehicle Details | 38 |
|  | **POLYMORPHISM** |  |
| 8. | **METHOD OVERLOADING** |  |
| i) | Employee Details | 39 |
| ii) | Shapes Details | 41 |
| 9. | **METHOD OVER RIDING** |  |
| i) | Employee Details | 41 |
| ii) | Calculating area,perimeter of shapes | 43 |
| 10. | **CONSTRUCTOR PROGRMS** |  |
|  | Book details(title,no.of pages) | 44 |
| 11. | **CONSTRUCTOR OVERLOADING** |  |
|  | Employee Salary Details | 45 |
|  | **ABSTRACTION** |  |
| 12. | **ABSTRACT CLASS** |  |
| i) | Car Details | 46 |
| ii) | Vehicle (starting,stopping) | 47 |
| iii) | Calculating area of shapes | 49 |
| iv) | Drawing shapes and giving their parameters | 49 |
| 13 | **INTERFACE PROGRAMS** |  |
| i) | Finding areas of shapes | 51 |
| ii) | Smart phone functions | 52 |
| iii) | Payment Systems | 53 |
| iv) | Vehicle( charging,stop,start) | 54 |
| 14. | **ENCAPSULATION** |  |
| i) | Person Details(age,name) | 55 |
| ii) | Car Details(model,speed) | 56 |
| iii) | Bank Account Details | 57 |
| iv) | Playing a song,displaying volume | 58 |
| 15. | **PACKAGES PROGRAMS** |  |
| i) | Simple button | 63 |
| ii) | Person details | 64 |
| iii) | Employee Payroll Details | 65 |
| iv) | Student Report Card | 69 |
| 16. | **EXCEPTION HANDLING PROGRAMS** |  |
| i) | Balance checking | 73 |
| ii) | Age restriction in Train Booking System | 74 |
| iii) | Online shopping details | 75 |
| iv) | Booking system | 79 |
| 17. | **FILE HANDLING PROGRAMS** |  |
| i) | Read File | 82 |
| ii) | WriteFile | 83 |
| iii) | File handling (trycatch) | 84 |
| iv) | ReadWriteWordCount | 86 |

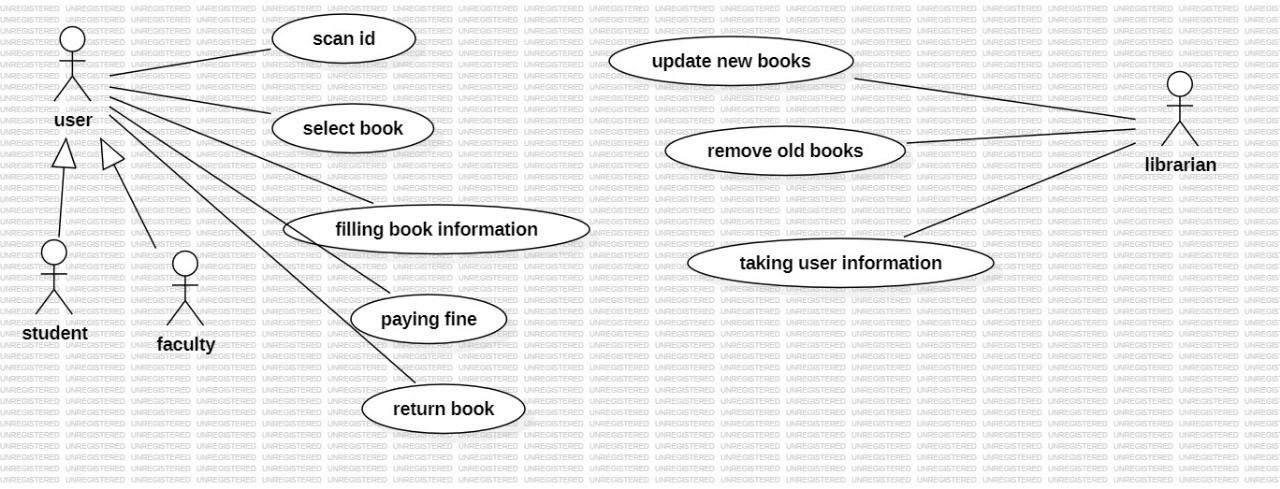
UML DIAGRAM

1.Library Management:

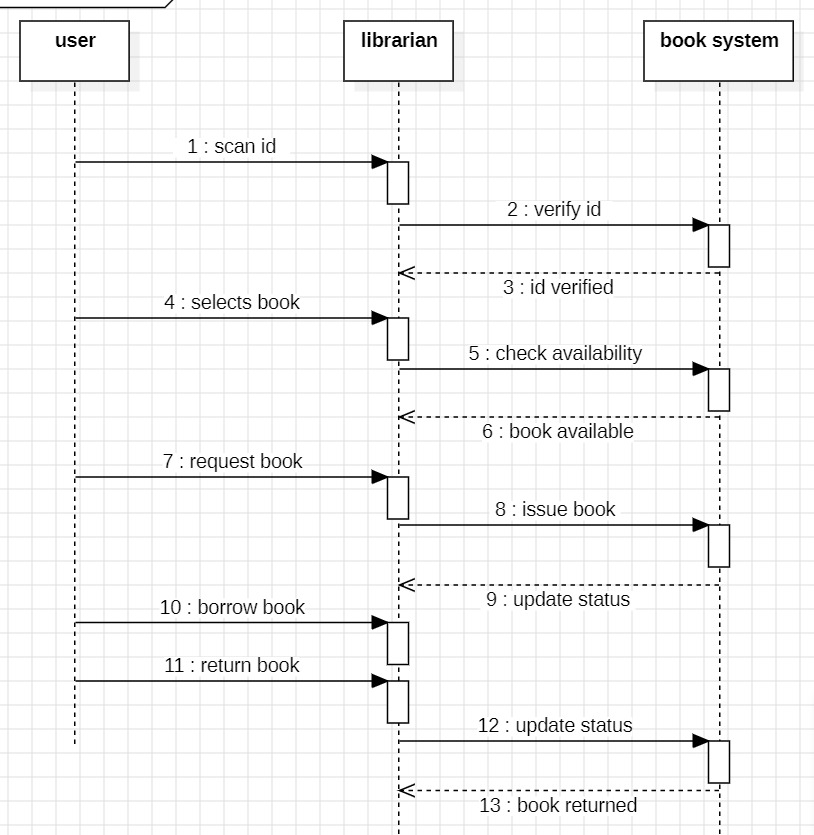
1. Class Diagram:



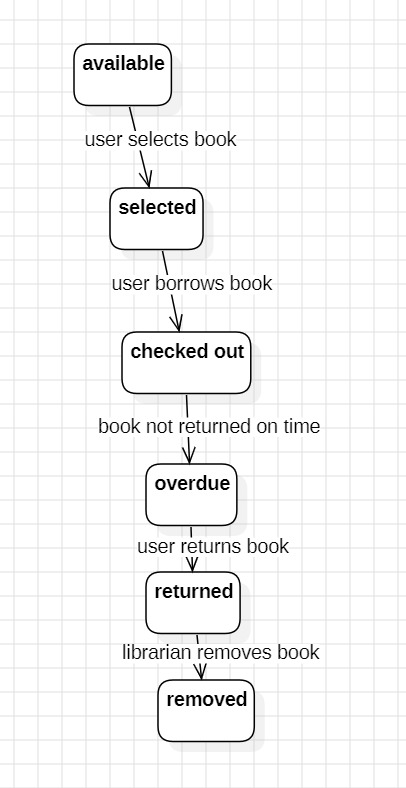
1. Use Case Diagram:



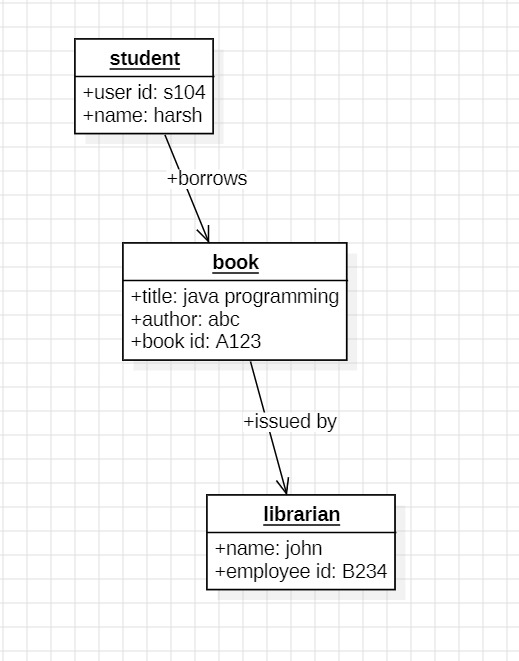
1. Sequence Diagram:



1. State Activity Diagram:

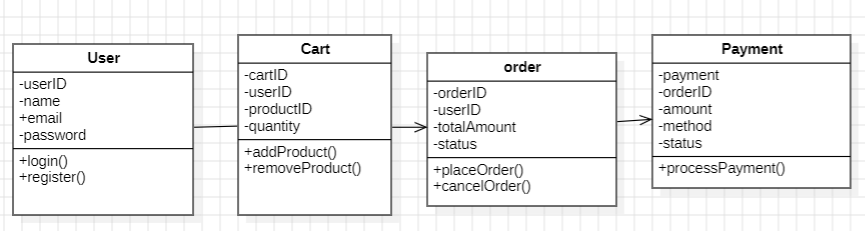


1. Object Diagram:

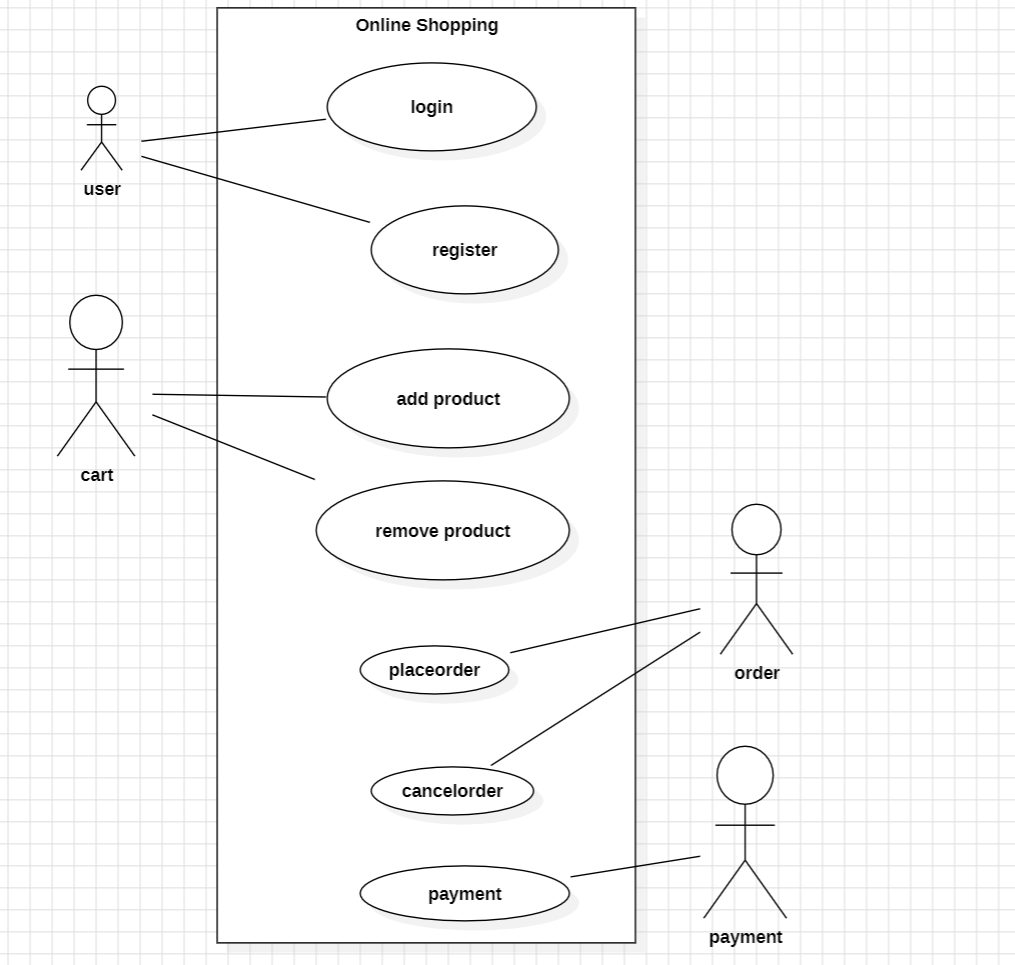


2.Online Shopping

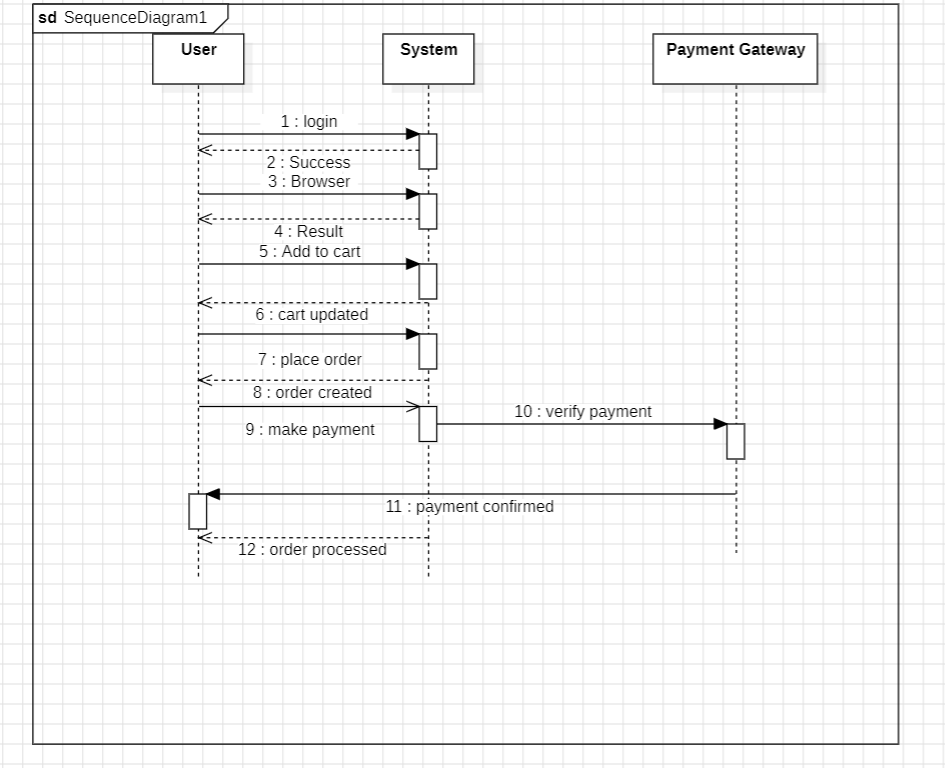
1. Class Diagram:



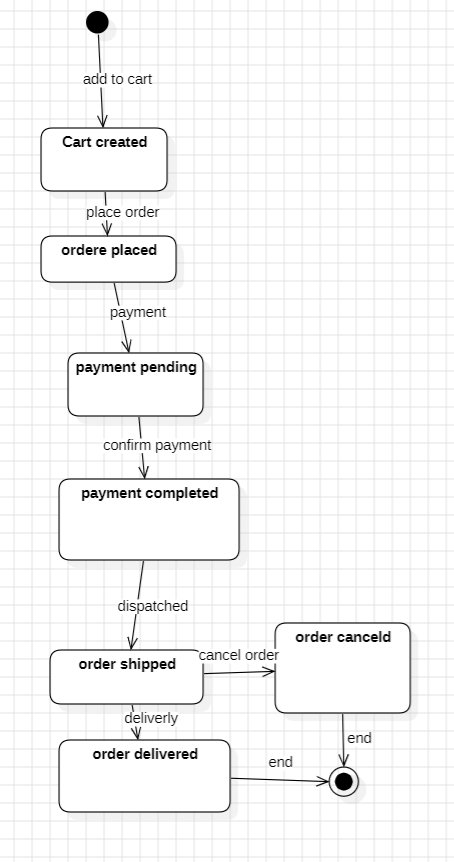
1. Use Case Diagram:



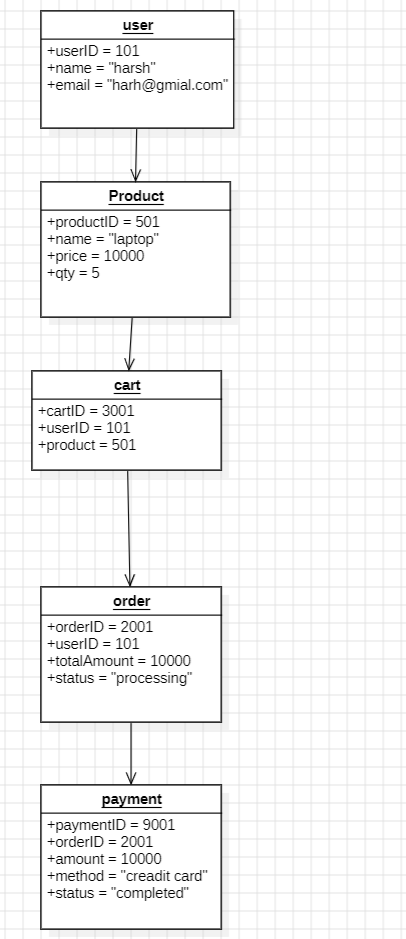
1. Sequence Diagram:



1. State Activity Diagram:



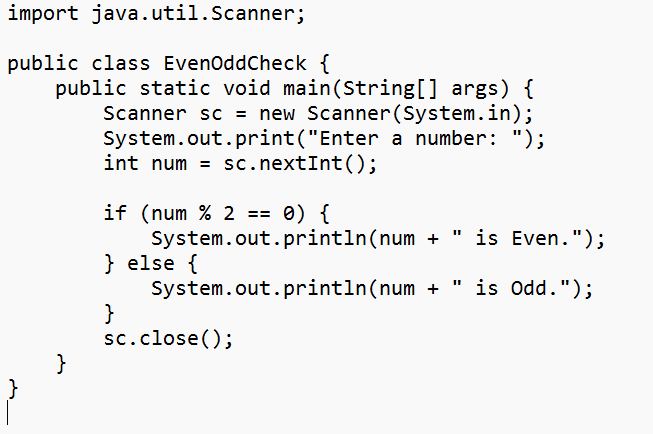
1. Object Diagram:



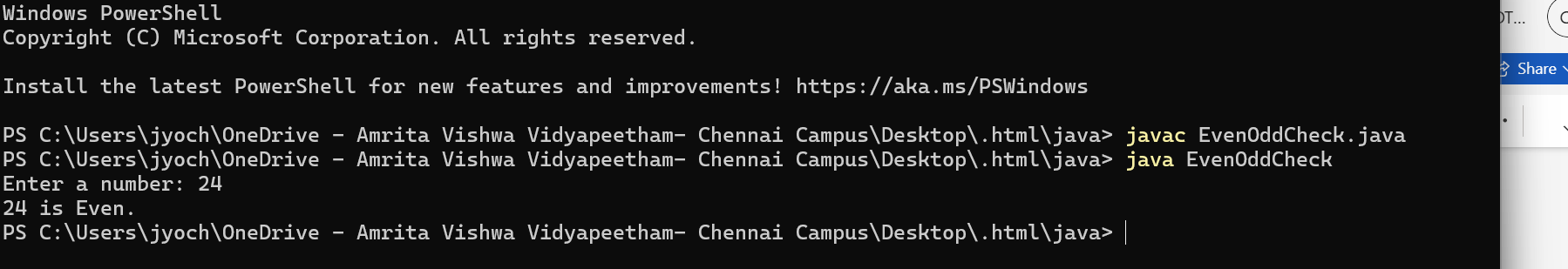
**Basic Java Programs**

1. Even or Odd:

Code:

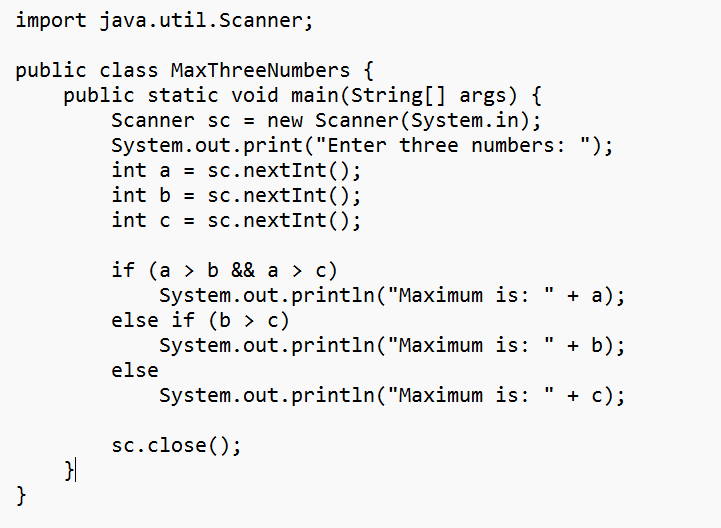


Output:

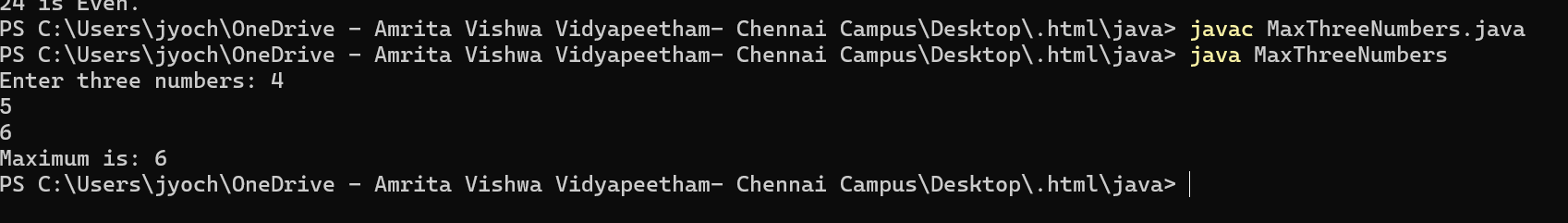


1. Max of 3 numbers:

Code:

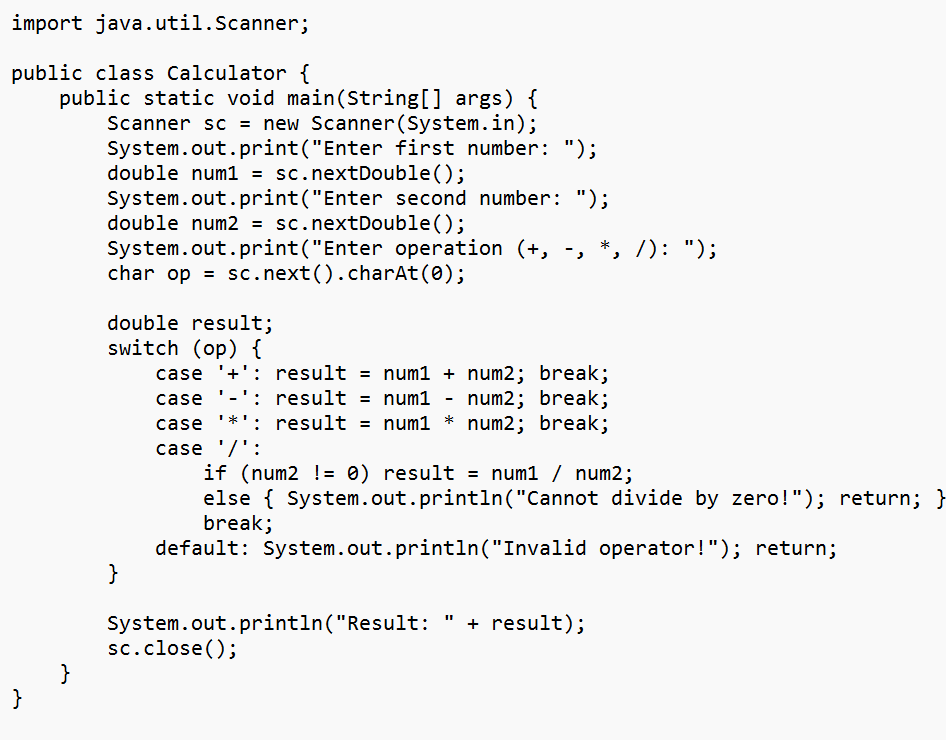


Output:

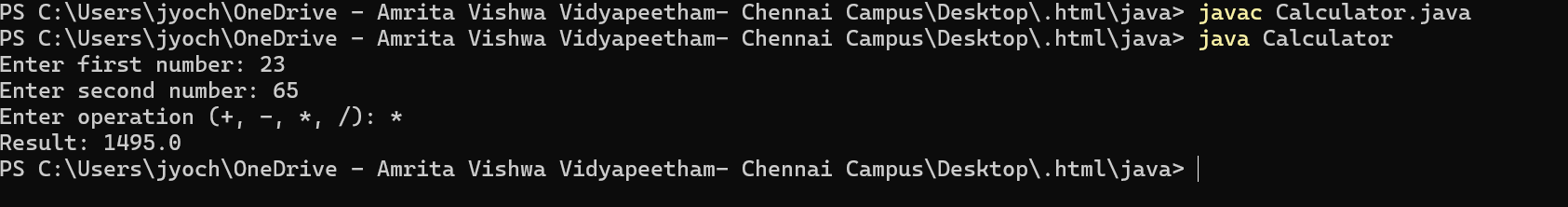


3.Calculator

Code:

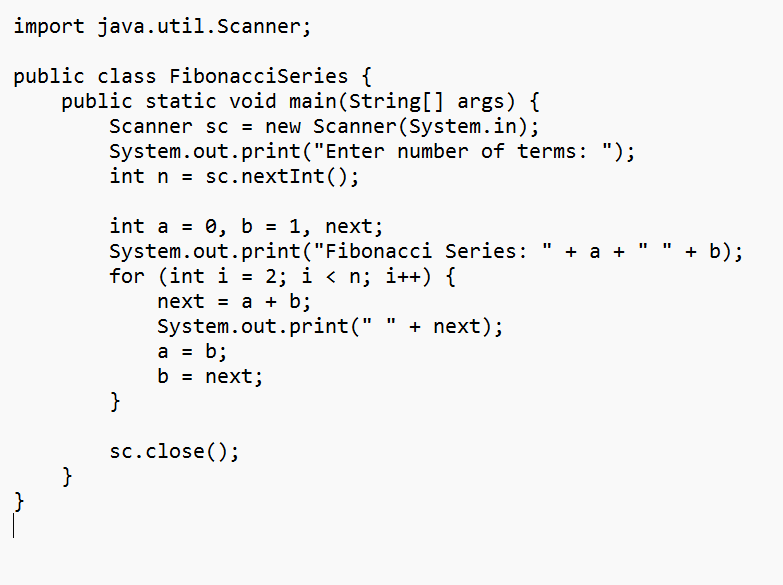


Output:

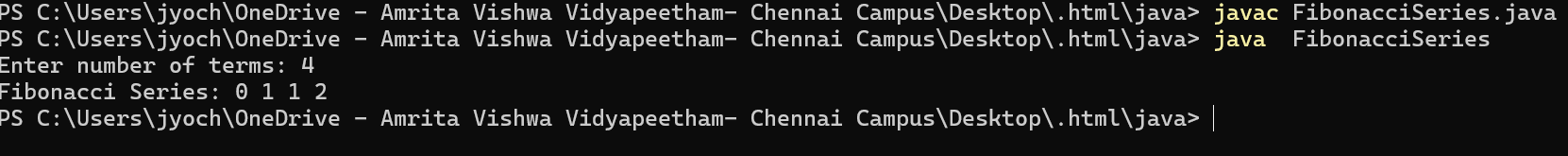


**4.Fibonacci series:**

Code:

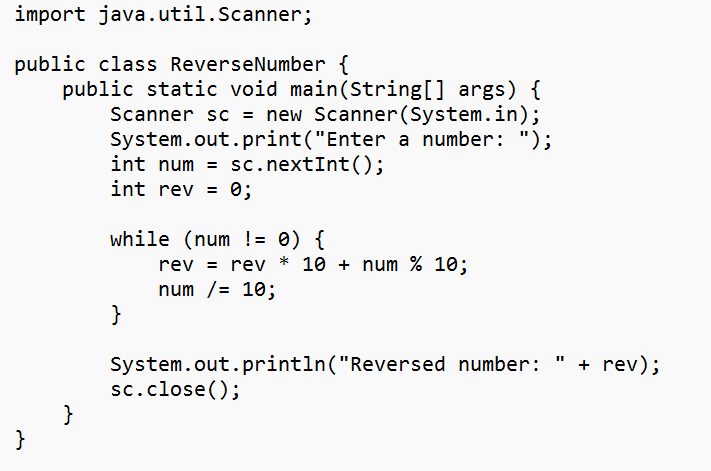


Output:

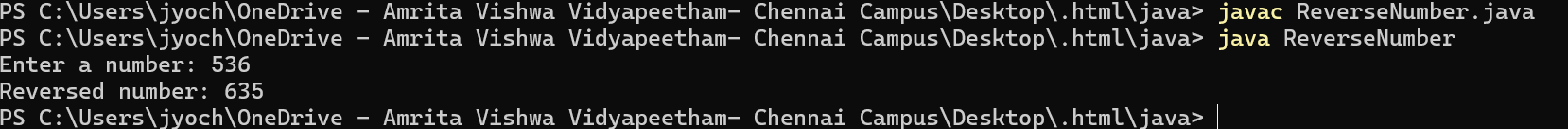


5.Reverse a number:

Code:

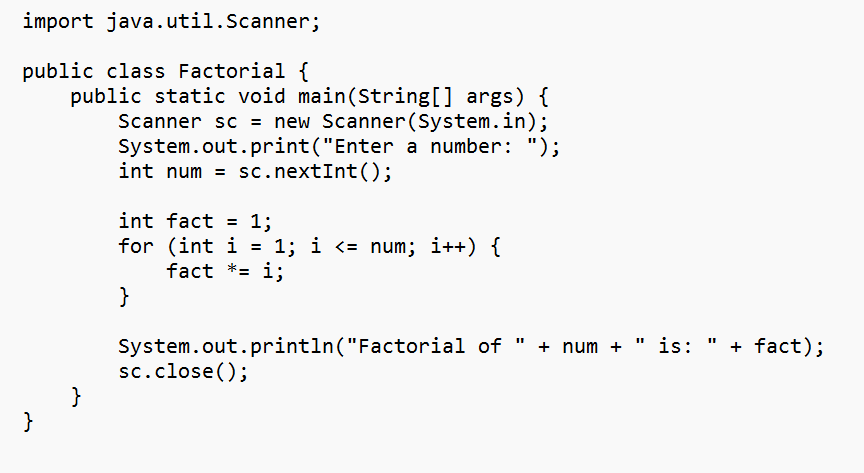


Output:

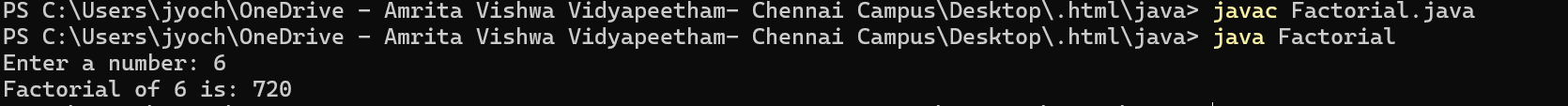


6.Factorial of a number:

Code:

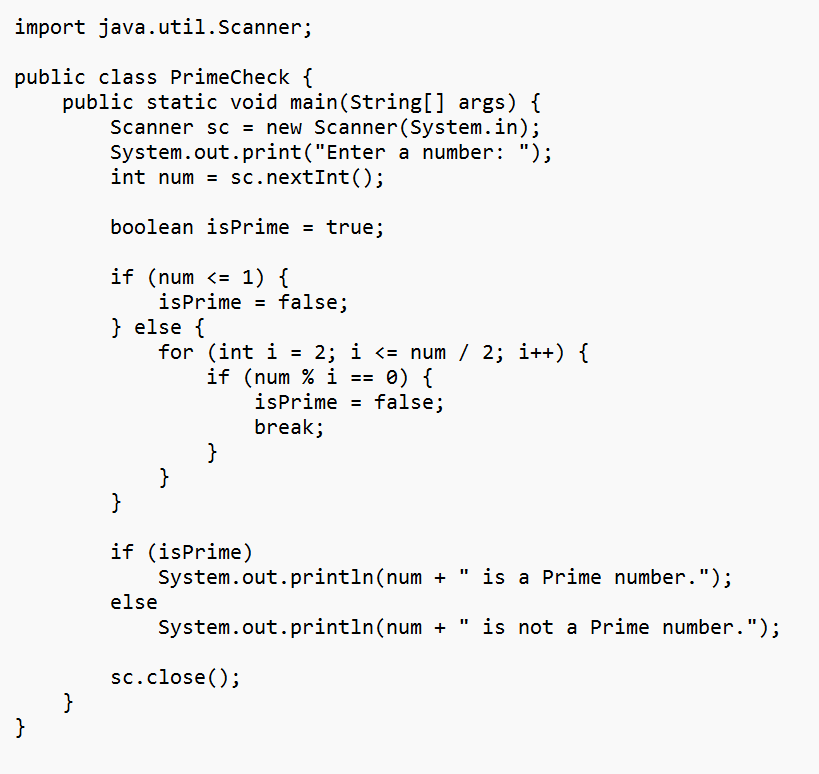


Output:

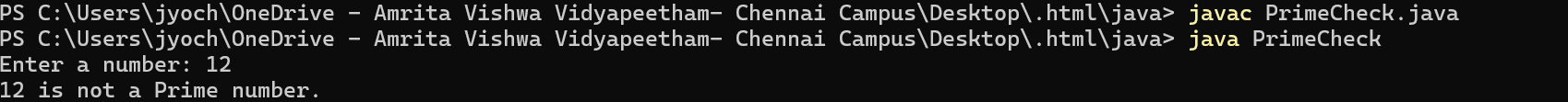


7.Prime Number Check:

Code:

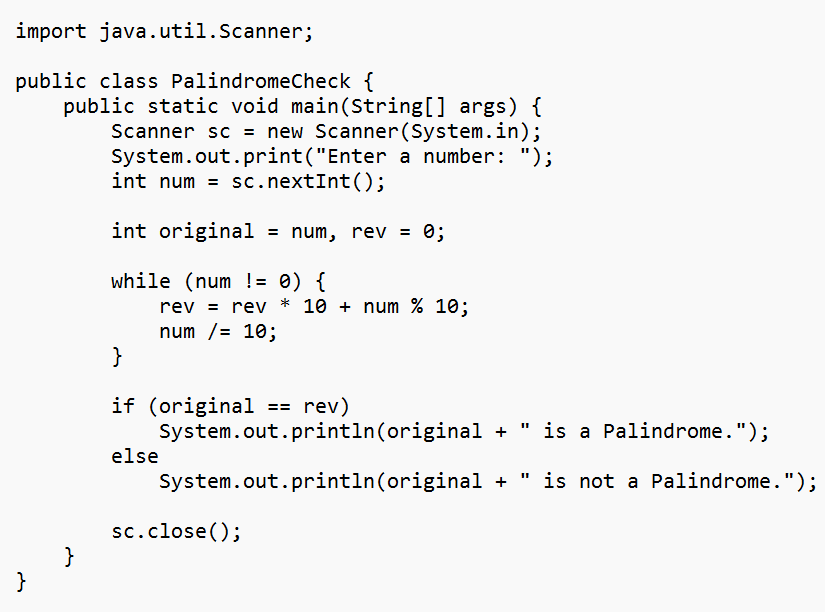


Output:

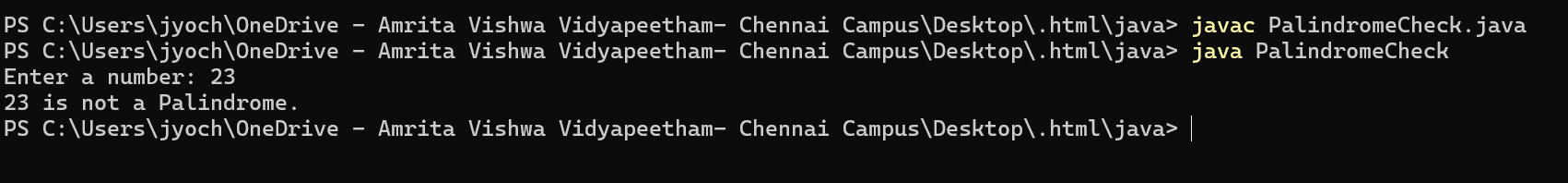


8.Palindrome Check:

Code:

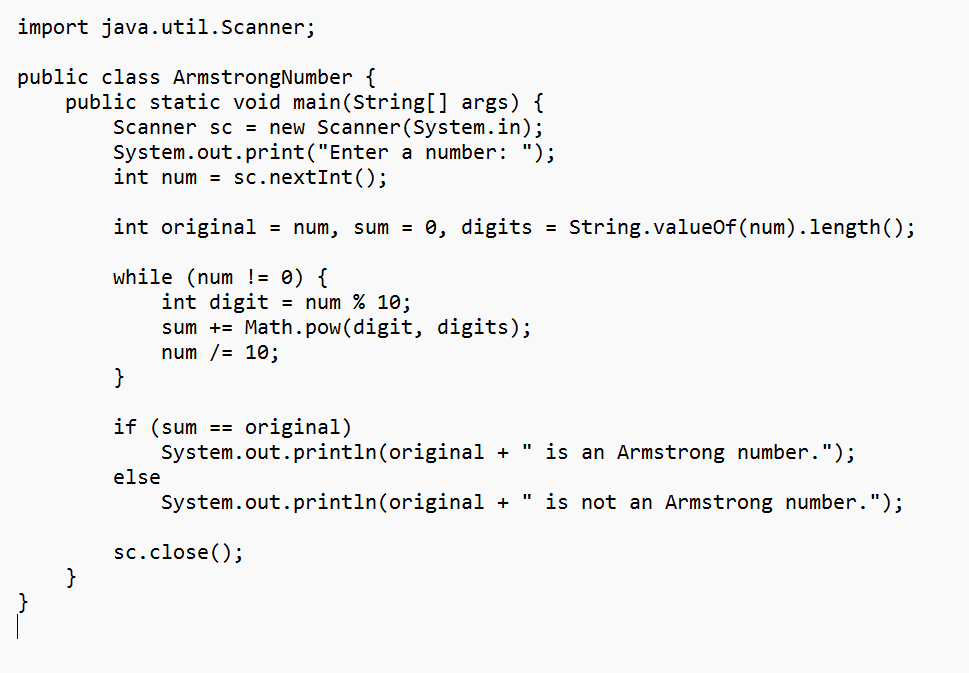


Output:

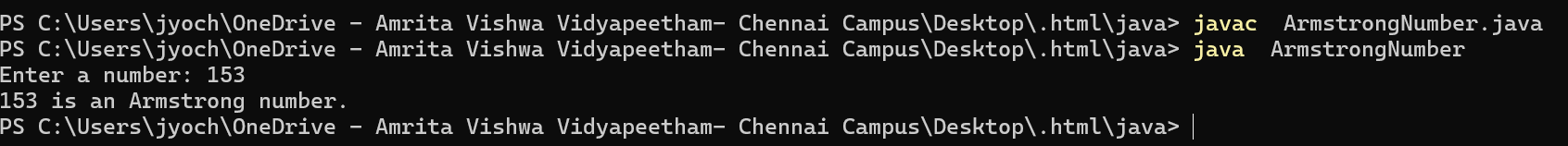


9.Armstrong Number:

Code:

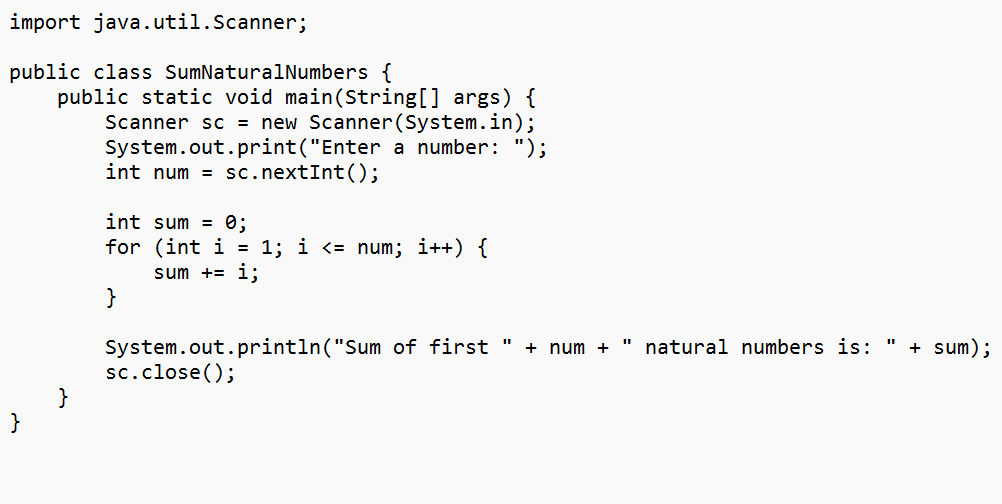


Output:

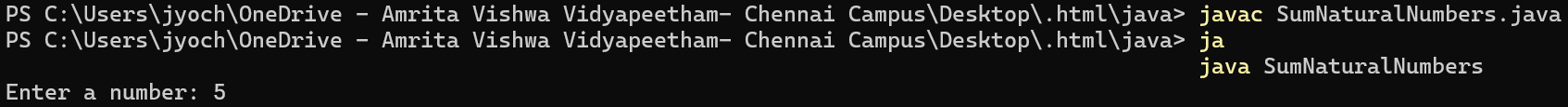


10.Sum of Natural Number:

Code:



Output:



#### **INHERITANCE:**

#### **4)Hierarchical Inheritance:**

**A)**

CODE:

class Vehicle {

private String brand;

private String model;

public Vehicle(String brand, String model) {

this.brand = brand;

this.model = model;

}

public void start() {

System.out.println("Vehicle is starting.");

}

public void stop() {

System.out.println("Vehicle is stopping.");

}

public String getBrand() {

return brand;

}

public String getModel() {

return model;

}

}

class Car extends Vehicle {

private int numberOfDoors

public Car(String brand, String model, int numberOfDoors) {

super(brand, model);

this.numberOfDoors = numberOfDoors;

}

public void drive() {

System.out.println("Car is driving.");

}

public int getNumberOfDoors() {

return numberOfDoors;

}

}

class ElectricCar extends Car {

private int batteryCapacity;

public ElectricCar(String brand, String model, int numberOfDoors, int batteryCapacity) {

super(brand, model, numberOfDoors);

this.batteryCapacity = batteryCapacity;

}

public void charge() {

System.out.println("Electric car is charging.");

}

public int getBatteryCapacity() {

return batteryCapacity;

}

}

class Truck extends Vehicle {

private double cargoCapacity;

public Truck(String brand, String model, double cargoCapacity) {

super(brand, model);

this.cargoCapacity = cargoCapacity;

}

public void loadCargo() {

System.out.println("Truck is loading cargo.");

}

public double getCargoCapacity() {

return cargoCapacity;

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Toyota", "Corolla", 4);

car.start();

car.drive();

car.stop();

System.out.println("Car doors: " + car.getNumberOfDoors());

ElectricCar electricCar = new ElectricCar("Tesla", "Model S", 4, 100);

electricCar.start();

electricCar.drive();

electricCar.charge();

System.out.println("Battery capacity: " + electricCar.getBatteryCapacity());

Truck truck = new Truck("Ford", "F-150", 2000.5);

truck.start();

truck.loadCargo();

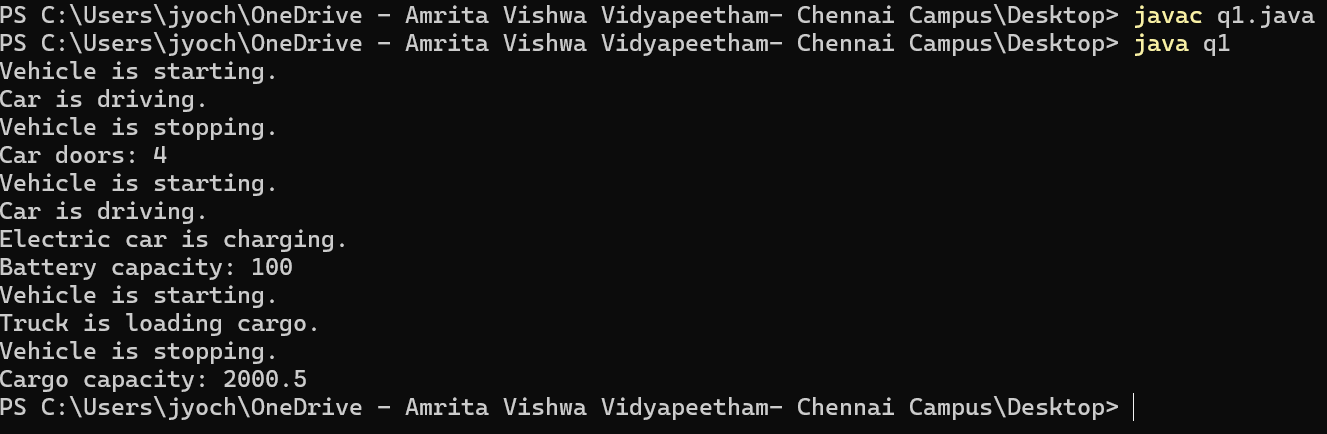
truck.stop();

System.out.println("Cargo capacity: " + truck.getCargoCapacity());

}

}

OUTPUT:



**B)**

CODE:

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public void displayDetails() {

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

}

}

class Student extends Person {

private int studentId;

private String major;

public Student(String name, int age, int studentId, String major) {

super(name, age);

this.studentId = studentId;

this.major = major;

}

public void study() {

System.out.println("Student is studying " + major);

}

public void displayDetails() {

super.displayDetails();

System.out.println("Student ID: " + studentId + ", Major: " + major);

}

}

class Professor extends Person {

private String department;

private String researchArea;

public Professor(String name, int age, String department, String researchArea) {

super(name, age);

this.department = department;

this.researchArea = researchArea;

}

public void teach() {

System.out.println("Professor is teaching in " + department);

}

public void displayDetails() {

super.displayDetails();

System.out.println("Department: " + department + ", Research Area: " + researchArea);

}

}

class TeachingAssistant extends Student {

private String course;

public TeachingAssistant(String name, int age, int studentId, String major, String course) {

super(name, age, studentId, major);

this.course = course;

}

public void assist() {

System.out.println("Teaching assistant is assisting in " + course);

}

public void displayDetails() {

super.displayDetails();

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

}

}

public class q2 {

public static void main(String[] args) {

Student student = new Student("Alice", 20, 101, "Computer Science");

student.displayDetails();

student.study()

Professor professor = new Professor("Dr. Smith", 45, "Computer Science", "AI");

professor.displayDetails();

professor.teach(TeachingAssistant ta = new TeachingAssistant("Bob", 25, 102, "Mathematics", "Calculus");

ta.displayDetails();

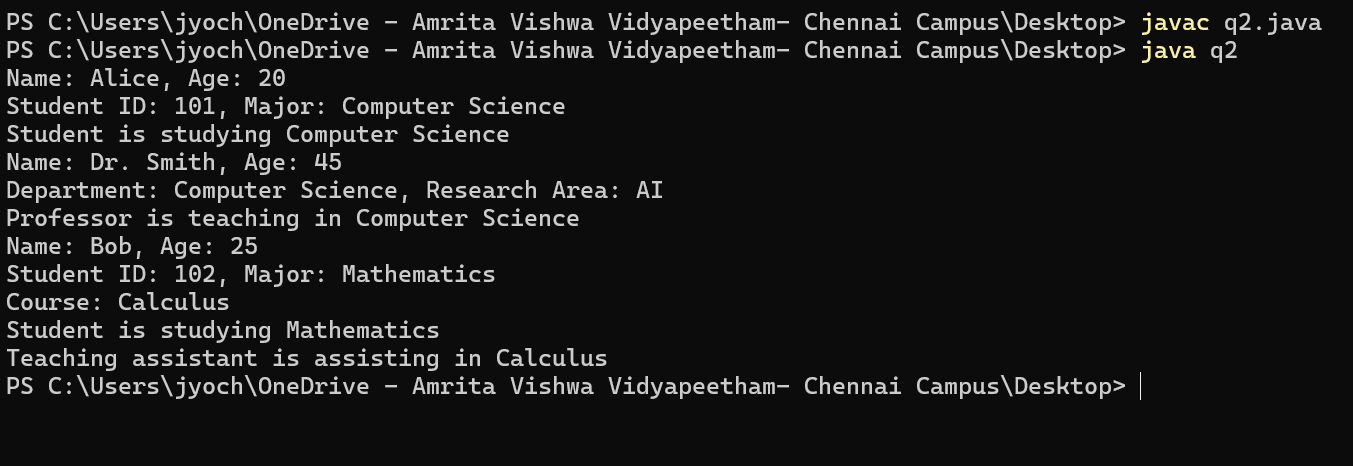
ta.study();

ta.assist();

}

}

OUTPUT:



### **5)SINGLE INHERITANCE:**

**A)**

CODE:

class College {

String collegeName = "AMRITA";

String address = "CHENNAI, India";

void showCollegeDetails() {

System.out.println("College Name: " + collegeName);

System.out.println("Address: " + address);

}

}

class Student extends College {

String studentName;

int rollNumber;

Student(String studentName, int rollNumber) {

this.studentName = studentName;

this.rollNumber = rollNumber;

}

void showStudentDetails() {

System.out.println("Student Name: " + studentName);

System.out.println("Roll Number: " + rollNumber);

}

}

public class SingleInheritanceExample1 {

public static void main(String[] args) {

Student s1 = new Student("Rahul", 101);

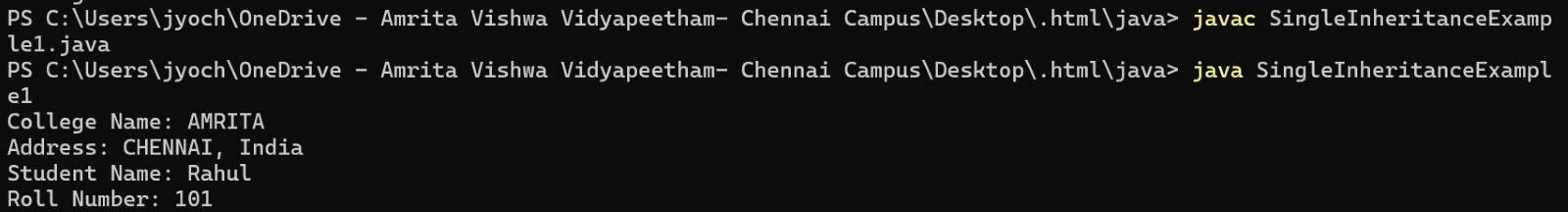
s1.showCollegeDetails();

s1.showStudentDetails();

}

}

OUTPUT:



**B)**

CODE:

class BankAccount {

String accountHolder;

double balance;

BankAccount(String accountHolder, double balance) {

this.accountHolder = accountHolder;

this.balance = balance;

}

void showBalance() {

System.out.println("Account Holder: " + accountHolder);

System.out.println("Balance: $" + balance);

}

}

class SavingsAccount extends BankAccount {

double interestRate = 5.0;

SavingsAccount(String accountHolder, double balance) {

super(accountHolder, balance);

}

void calculateInterest() {

double interest = (balance \* interestRate) / 100;

System.out.println("Annual Interest: $" + interest);

}

}

public class SingleInheritanceExample2 {

public static void main(String[] args) {

SavingsAccount acc1 = new SavingsAccount("John Doe", 5000);

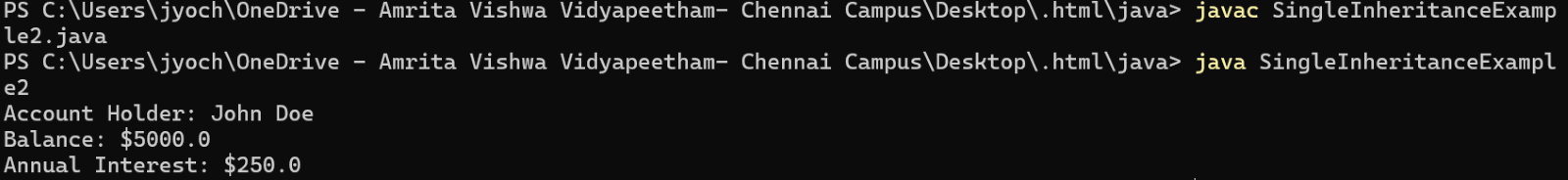
acc1.showBalance();

acc1.calculateInterest();

}

}

OUTPUT:



**6)MULTILEVEL INHERITANCE:**

**A)**

CODE:

class LivingBeing {

void breathe() {

System.out.println("Living beings breathe.");

}

}

class Human extends LivingBeing {

void speak() {

System.out.println("Humans can speak.");

}

}

class Student extends Human {

String name;

int studentID;

Student(String name, int studentID) {

this.name = name;

this.studentID = studentID;

}

void study() {

System.out.println(name + " is studying.");

}

void showDetails() {

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

System.out.println("Student ID: " + studentID);

}

}

public class MultilevelExample1 {

public static void main(String[] args) {

Student s1 = new Student("Rahul", 101);

s1.breathe();

s1.speak();

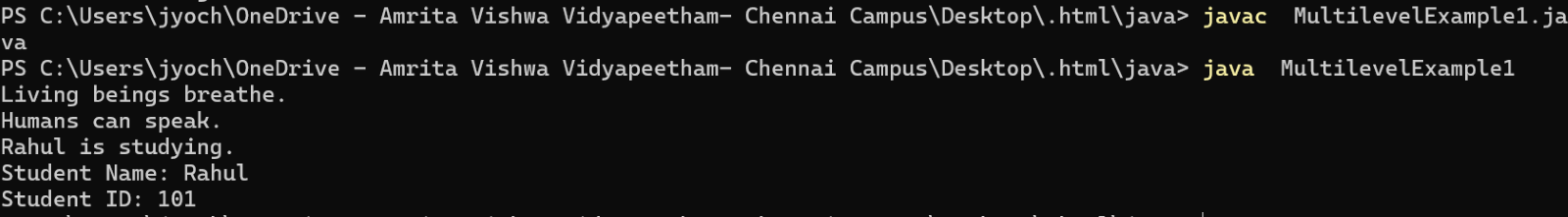
s1.study();

s1.showDetails();

}

}

OUTPUT:



**B)**

CODE:

class Person {

String name;

Person(String name) {

this.name = name;

}

void showPerson() {

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

}

}

class Employee extends Person {

int employeeID;

double salary;

Employee(String name, int employeeID, double salary) {

super(name);

this.employeeID = employeeID;

this.salary = salary;

}

void showEmployee() {

System.out.println("Employee ID: " + employeeID);

System.out.println("Salary: $" + salary);

}

}

class Manager extends Employee {

String department;

Manager(String name, int employeeID, double salary, String department) {

super(name, employeeID, salary);

this.department = department;

}

void showManager() {

System.out.println("Department: " + department);

System.out.println("Role: Manager");

}

}

public class MultilevelExample2 {

public static void main(String[] args) {

Manager m1 = new Manager("Alice", 2001, 75000, "HR");

m1.showPerson();

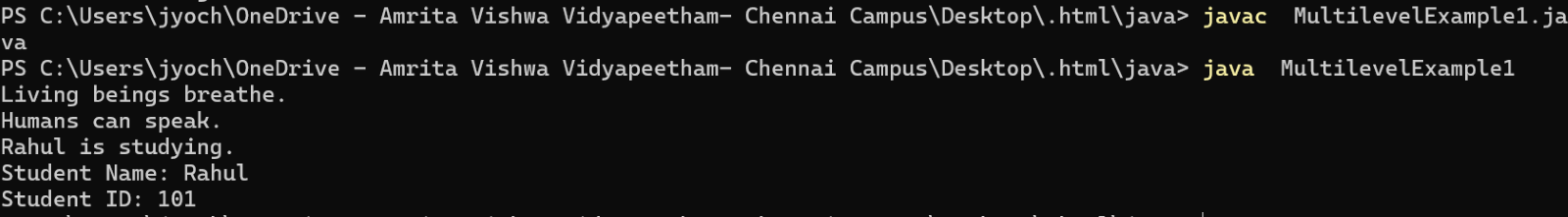
m1.showEmployee();

m1.showManager();

}

}

OUTPUT:



**7)HYBRID INHERITANCE:**

**A)**

CODE:

class Person {

String name;

Person(String name) {

this.name = name;

}

void showDetails() {

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

}

}

class Student extends Person {

int studentID;

Student(String name, int studentID) {

super(name);

this.studentID = studentID;

}

void study() {

System.out.println(name + " is studying.");

}

}

class Teacher extends Person {

String subject;

Teacher(String name, String subject) {

super(name);

this.subject = subject;

}

void teach() {

System.out.println(name + " is teaching " + subject + ".");

}

}

interface Assistant {

void assist();

}

class TeachingAssistant extends Student implements Assistant {

TeachingAssistant(String name, int studentID) {

super(name, studentID);

}

public void assist() {

System.out.println(name + " is assisting in a lab session.");

}

}

public class HybridInheritanceExample1 {

public static void main(String[] args) {

TeachingAssistant ta = new TeachingAssistant("Alex", 101);

ta.showDetails();

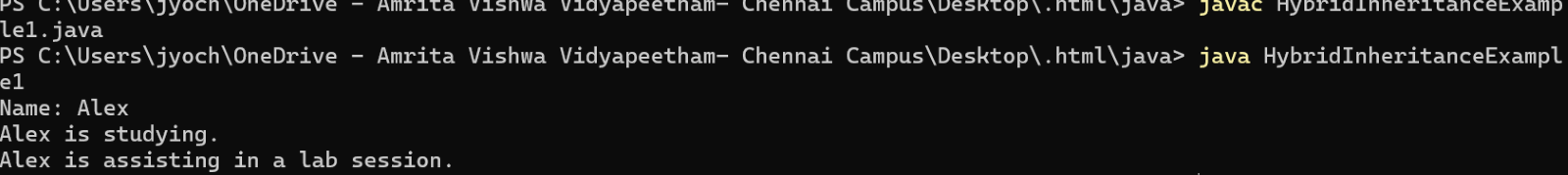
ta.study();

ta.assist();

}

}

OUTPUT:



**B)**

CODE:

class Vehicle {

void startEngine() {

System.out.println("Vehicle engine started.");

}

}

class Car extends Vehicle {

void drive() {

System.out.println("Car is driving.");

}

}

class Boat extends Vehicle {

void sail() {

System.out.println("Boat is sailing.");

}

}

interface Amphibious {

void switchMode();

}

class AmphibiousCar extends Car implements Amphibious {

public void switchMode() {

System.out.println("Switching between land and water mode.");

}

void sail() {

System.out.println("Amphibious car is sailing on water.");

}

}

public class HybridInheritanceExample2 {

public static void main(String[] args) {

AmphibiousCar ac = new AmphibiousCar();

ac.startEngine();

ac.drive();

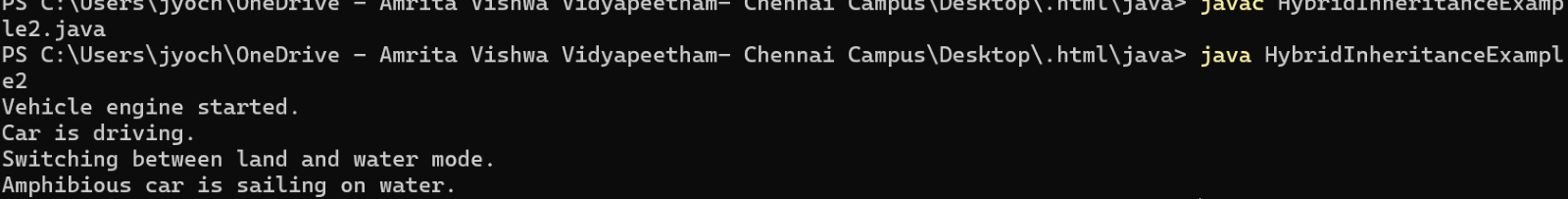
ac.switchMode();

ac.sail();

}

}

OUTPUT:



### **POLYMORPHISM:**

**8)METHOD OVERLOADING**

**A)**

CODE:

class Employee {

private String name;

private int id;

private double salary;

void setDetails(String name, int id) {

this.name = name;

this.id = id;

}

void setDetails(String name, int id, double salary) {

this.name = name;

this.id = id;

this.salary = salary;

}

void setDetails(String name) {

this.name = name;

}

void displayDetails() {

System.out.println("Name: " + name + ", ID: " + id + ", Salary: " + salary);

}

}

public class q3 {

public static void main(String[] args) {

Employee emp1 = new Employee();

emp1.setDetails("Alice", 101); // Calls first method

emp1.displayDetails();

Employee emp2 = new Employee();

emp2.setDetails("Bob", 102, 50000.0); // Calls second method

emp2.displayDetails();

Employee emp3 = new Employee();

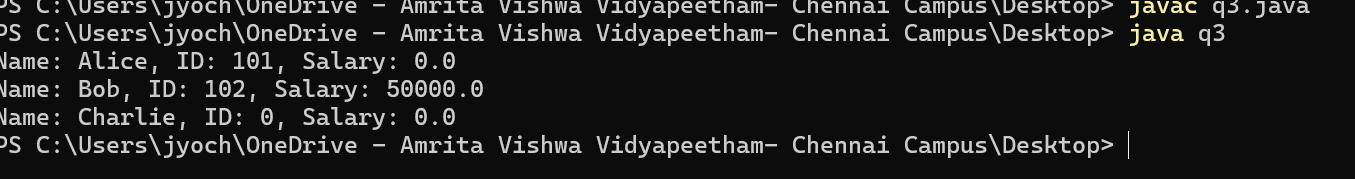
emp3.setDetails("Charlie"); // Calls third method

emp3.displayDetails();

}

}

OUTPUT:



**B)**

CODE:

class Shape {

void draw(double radius) {

System.out.println("Drawing a circle with radius: " + radius);

}

void draw(double length, double width) {

System.out.println("Drawing a rectangle with length: " + length + " and width: " + width);

}

void draw(double side1, double side2, double side3) {

System.out.println("Drawing a triangle with sides: " + side1 + ", " + side2 + ", " + side3);

}

}

public class q4 {

public static void main(String[] args) {

Shape shape = new Shape();

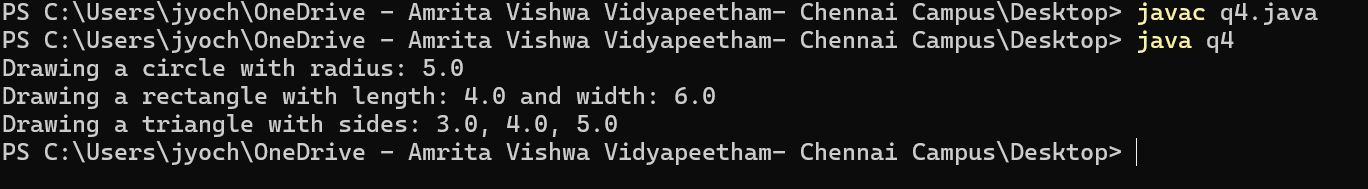
shape.draw(5.0);

shape.draw(4.0, 6.0);

shape.draw(3.0, 4.0, 5.0); }

}

OUTPUT:



**9)METHOD OVER RIDING:**

**A)**

CODE:

abstract class Employee {

protected String name;

protected double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

abstract double calculateBonus();

public void displayDetails() {

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

}

}

class Manager extends Employee {

private static final double BONUS\_PERCENTAGE = 0.20; // 20% bonus

public Manager(String name, double salary) {

super(name, salary);

}

double calculateBonus() {

return salary \* BONUS\_PERCENTAGE;

}

}

class Developer extends Employee {

private static final double BONUS\_PERCENTAGE = 0.10; // 10% bonus

public Developer(String name, double salary) {

super(name, salary);

}

double calculateBonus() {

return salary \* BONUS\_PERCENTAGE;

}

}

public class q5 {

public static void main(String[] args) {

Employee manager = new Manager("John", 100000);

Employee developer = new Developer("Alice", 80000);

System.out.println("Manager Bonus: " + manager.calculateBonus());

System.out.println("Developer Bonus: " + developer.calculateBonus());

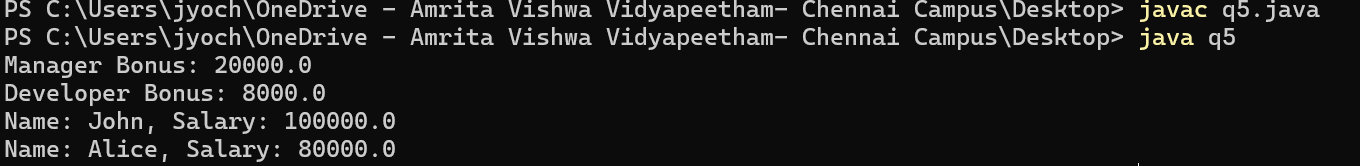
manager.displayDetails();

developer.displayDetails();

}

}

OUTPUT:



**B)**

CODE:

abstract class Shape { abstract double calculateArea(); abstract double calculatePerimeter(); } class Circle extends Shape { private double radius;

public Circle(double radius) {  
 this.radius = radius;  
}  
double calculateArea() {   
 return Math.PI \* radius \* radius;  
}  
double calculatePerimeter() {   
 return 2 \* Math.PI \* radius;  
}

} class Rectangle extends Shape { private double length; private double width;

public Rectangle(double length, double width) {  
 this.length = length;  
 this.width = width;  
}  
double calculateArea() {   
 return length \* width;  
}  
  
double calculatePerimeter() {   
 return 2 \* (length + width);  
}

}

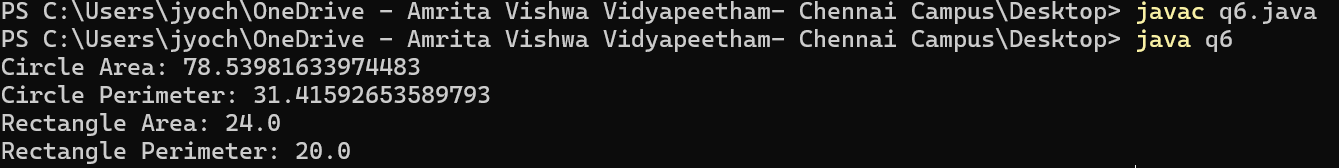
public class q6 { public static void main(String[] args) {

Shape circle = new Circle(5);

Shape rectangle = new Rectangle(4, 6);

System.out.println("Circle Area: " + circle.calculateArea());  
 System.out.println("Circle Perimeter: " + circle.calculatePerimeter());  
  
 System.out.println("Rectangle Area: " + rectangle.calculateArea());  
 System.out.println("Rectangle Perimeter: " + rectangle.calculatePerimeter());  
}  
 }

OUTPUT:



**10)CONSTRUCTOR PROGRMS:**

CODE:

class Book {

String title;

int pages;

Book(String t, int p) {

title = t;

pages = p;

}

Book(Book b) {

title = b.title;

pages = b.pages;

}

void display() {

System.out.println("Book: " + title + ", Pages: " + pages);

}

}

public class ConstructorExample {

public static void main(String[] args) {

Book b1 = new Book("Java Programming", 500);

Book b2 = new Book(b1);

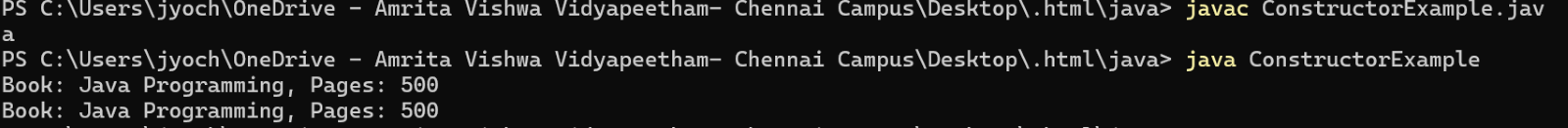
b1.display();

b2.display();

}

}

OUTPUT:



**11)CONSTRUCTOR OVERLOADING:**

CODE:

class Employee {

String name;

int age;

double salary;

Employee() {

name = "Unknown";

age = 18;

salary = 30000;

}

Employee(String n, int a) {

name = n;

age = a;

salary = 40000;

}

Employee(String n, int a, double s) {

name = n;

age = a;

salary = s;

}

void display() {

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

}

}

public class ConstructorOverloadingExample {

public static void main(String[] args) {

Employee e1 = new Employee();

Employee e2 = new Employee("John", 25);

Employee e3 = new Employee("Alice", 30, 60000);

e1.display();

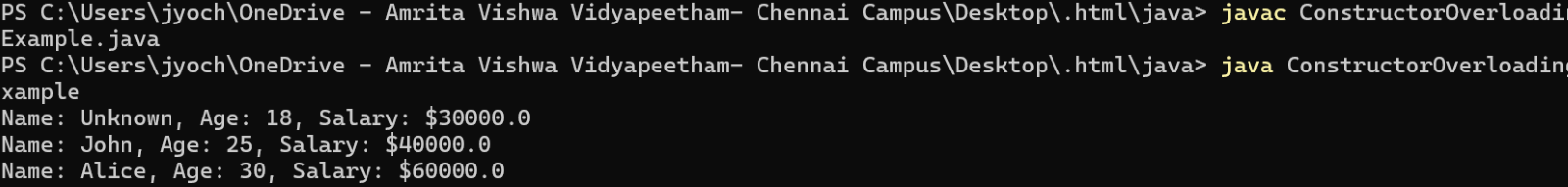
e2.display();

e3.display();

}

}

OUTPUT:



**ABSTRACTION:**

**12)ABSTRACT CLASS:**

**A)**

CODE:

public class abstractclass1{

public static void main(String[] args){

Car c1=new kia();

c1.carName("kia");

c1.carSpeed(122,"kia");

}

}

abstract class Car{

abstract void carName(String name);

abstract void carSpeed(int speed,String name);

abstract void Mileage(int fuel,double mileage);

}

class kia extends Car{

void carName(String name){

System.out.println("Your Car name is "+name);

}

void carSpeed(int speed,String name){

System.out.println(name+" can travel at the speed of "+speed+" km/h");

}

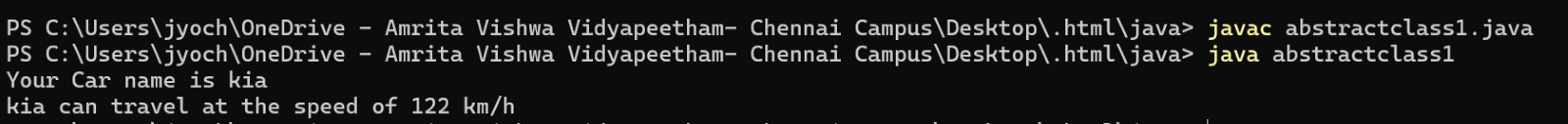
void Mileage(int fuel,double mileage){

System.out.println("kia "+"can travel range of "+(fuel\*mileage));

}

}

OUTPUT:



### **B)**

CODE:

abstract class Vehicle{

abstract void start(String name);

abstract void stop(String name);

}

class Car extends Vehicle{

void start(String name){

System.out.println( name+" is Starting");

}

void stop(String name){

System.out.println(name+" is stopping");

}

}

class Bike extends Vehicle{

void start(String name){

System.out.println( name+" is Starting");

}

void stop(String name){

System.out.println(name+" is stopping");

}

}

public class interface2{

public static void main(String[] args){

Car v1=new Car();

v1.start("Lambo");

v1.stop("lambo");

Bike v2=new Bike();

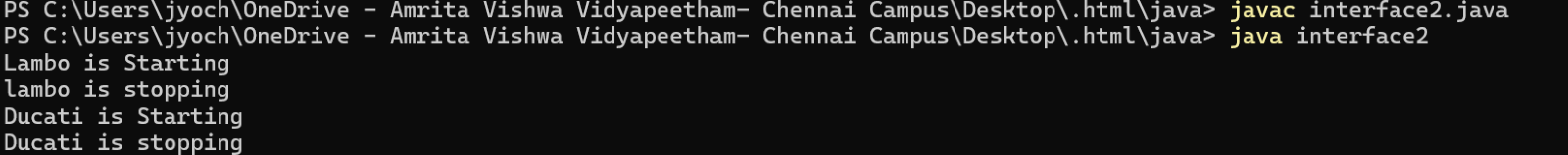
v2.start("Ducati");

v2.stop("Ducati");

}

}

OUTPUT:



**C)**

CODE:

abstract class Shape {

abstract double calculateArea();

}

class Square extends Shape {

private double side;

public Square(double side) {

this.side = side;

}

double calculateArea() {

return side \* side;

}

}

public class abstract3 {

public static void main(String[] args) {

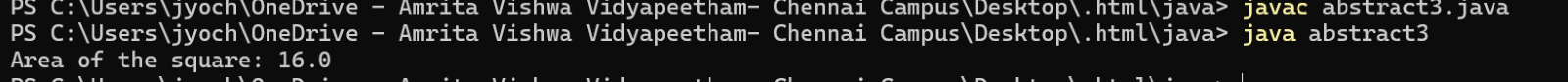
Shape square = new Square(4.0);

System.out.println("Area of the square: " + square.calculateArea());

}

}

OUTPUT:



**D)**

CODE:

abstract class Shape2D{

abstract void draw();

abstract void resize();

}

class Rectangle extends Shape2D{

void draw(){

System.out.println("you are Drawing Rectangle");

}

void resize(){

System.out.println("you can resize the lenght and breadth of rectangle");

}

}

class Circle extends Shape2D{

void draw(){

System.out.println("you are Drawing Circle");

}

void resize(){

System.out.println("you can resize the Radius of circle ");

}

}

public class abstarct4{

public static void main(String[] args){

Shape2D s1=new Rectangle();

s1.draw();

s1.resize();

Shape2D s2=new Circle();

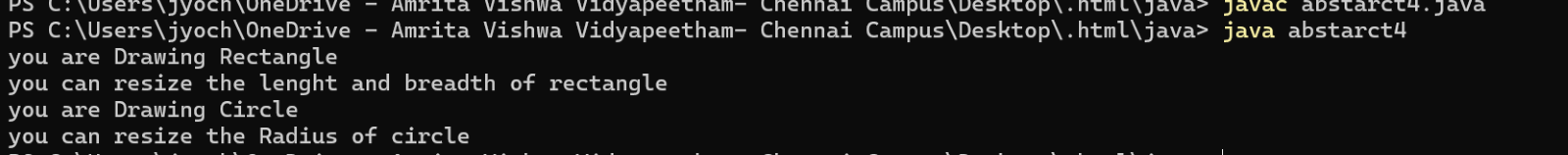
s2.draw();

s2.resize();

}

}

OUTPUT:



**13)INTERFACE PROGRAMS:**

**A)**

CODE:

interface Shape{ void getArea();

}

class Rectangle implements Shape

{ double lenght,breadth;

Rectangle(double lenght, double breadth) {  
 this.lenght = lenght;  
 this.breadth = breadth;  
}  
public void getArea(){  
 System.out.println("The area of rectangle "+(lenght\*breadth));  
}

}

class Circle implements Shape{ double radius; Circle(double radius){

this.radius=radius;

}

public void getArea()

{ System.out.println("The are of circle is "+(3.14\*(radius\*radius)));

}

}

public class Interface1 {

public static void main(String[] args) {

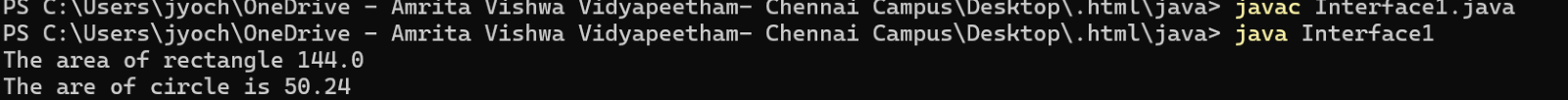
Rectangle r1=new Rectangle(12, 12);

r1.getArea();

Circle c1=new Circle(4);

c1.getArea(); }

}  
OUTPUT:



**B)**

CODE:

interface Camara{

void takePhoto();

}

interface MusicPlayer{

void canPlayMusic();

}

class SmartPhone implements Camara,MusicPlayer{

public void takePhoto(){

System.out.println("SmartPhone can take photo");

}

public void canPlayMusic(){

System.out.println("SmartPhone can play music");

}

}

public class Interface22 {

public static void main(String[] args) {

SmartPhone s1=new SmartPhone();

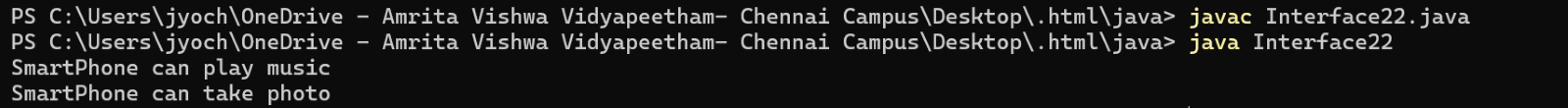
s1.canPlayMusic();

s1.takePhoto();

}

}

OUTPUT:



**C)**

CODE:

interface Payment {

void initiatePayment(double amount);

void getPaymentStatus();

}

class CreditCard implements Payment {

public void initiatePayment(double amount) {

System.out.println("Processing Credit Card payment of $" + amount);

}

public void getPaymentStatus() {

System.out.println("Credit Card payment successful!");

}

}

class PayPal implements Payment {

public void initiatePayment(double amount) {

System.out.println("Processing PayPal payment of $" + amount);

}

public void getPaymentStatus() {

System.out.println("PayPal payment successful!");

}

}

public class Interface3 {

public static void main(String[] args) {

Payment creditCardPayment = new CreditCard();

Payment paypalPayment = new PayPal();

creditCardPayment.initiatePayment(50.0);

creditCardPayment.getPaymentStatus();

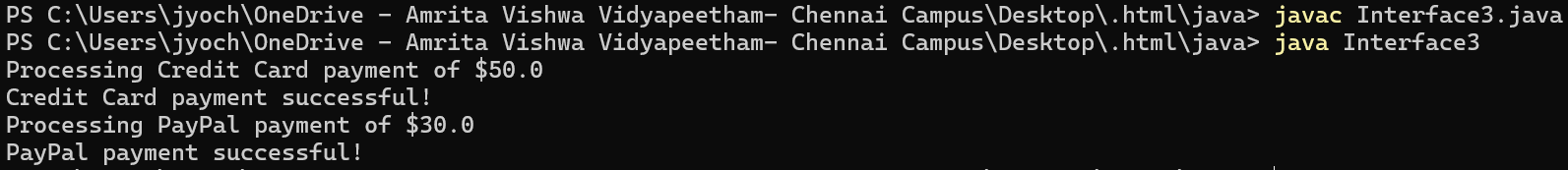
paypalPayment.initiatePayment(30.0);

paypalPayment.getPaymentStatus();

}

}

OUTPUT:



**D)**

CODE:

interface Vehicle{

void start();

void stop();

}

interface ElectricVehicle{

void chargeBattery();

}

interface FuelVehicle{

void refuel();

}

class HybridCar implements Vehicle,ElectricVehicle,FuelVehicle{

public void start(){

System.out.println("Vehicle Started ");

}

public void stop(){

System.out.println("Vehicle Stoped ");

}

public void chargeBattery(){

System.out.println("Vehicle charging battery ");

}

public void refuel(){

System.out.println("Vehicle refilling the fuel tank ");

}

}

public class Interface4 {

public static void main(String[] args) {

HybridCar v1=new HybridCar();

v1.chargeBattery();

v1.start();

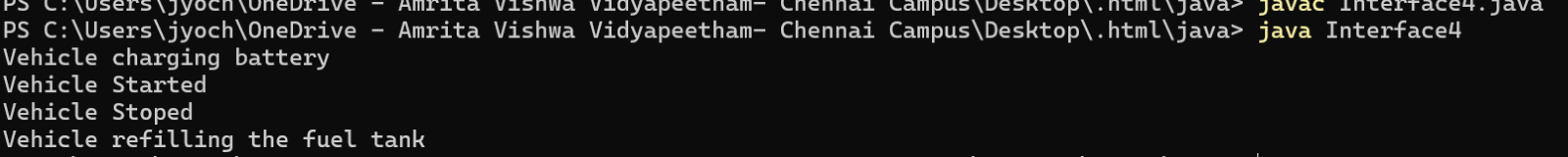
v1.stop();

v1.refuel();

}

}

OUTPUT:



**14)ENCAPSULATION**

**A)**

CODE:

class Person {

private String name;

private int age;

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setAge(int age) {

this.age = age;

}

public int getAge() {

return age;

}

}

public class encap1 {

public static void main(String[] args) {

Person p = new Person();

p.setName("John");

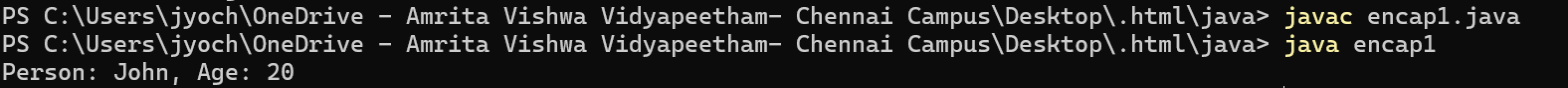
p.setAge(20);

System.out.println("Person: " + p.getName() + ", Age: " + p.getAge());

}

}

OUTPUT:



**B)**

CODE:

class Car {

private String brand;

private int speed;

public void setBrand(String brand) {

this.brand = brand;

}

public String getBrand() {

return brand;

}

public void setSpeed(int speed) {

if (speed > 200) {

this.speed = 200;

} else {

this.speed = speed;

}

}

public int getSpeed() {

return speed;

}

}

public class encap2 {

public static void main(String[] args) {

Car car = new Car();

car.setBrand("Tesla");

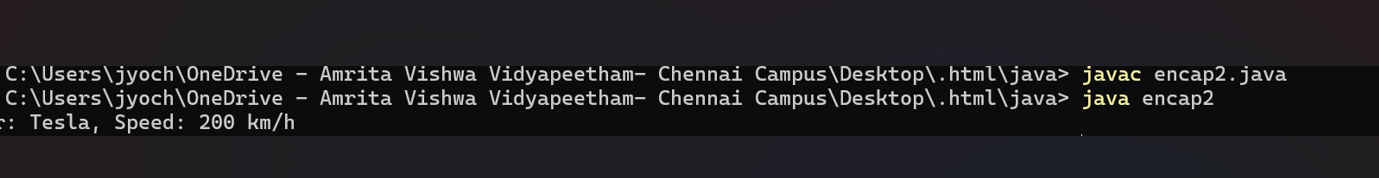
car.setSpeed(250);

System.out.println("Car: " + car.getBrand() + ", Speed: " + car.getSpeed() + " km/h");

}

}

OUTPUT:



**C)**

CODE:

class BankAccount {

private String accountNumber;

private double balance;

public BankAccount(String accountNumber, double balance) {

this.accountNumber = accountNumber;

this.balance = balance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

}

public class encap3 {

public static void main(String[] args) {

BankAccount account = new BankAccount("12345678", 5000);

account.deposit(2000);

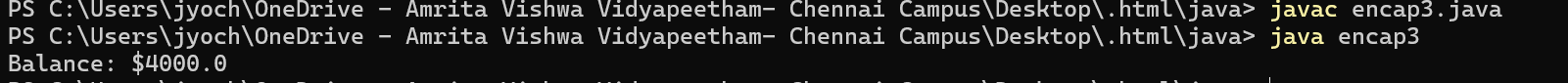
account.withdraw(3000);

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

}

}

OUTPUT:



**D)**

CODE:

import java.lang.\*;

class MusicPlayer {

private String songName;

private int volume;

MusicPlayer(String songName) {

this.songName = songName;

this.volume = 50;

}

String getSongName() {

return songName;

}

void setVolume(int volume) {

if (volume < 0) this.volume = 0;

else if (volume > 100) this.volume = 100;

else this.volume = volume;

}

int getVolume() {

return volume;

}

}

public class encap4 {

public static void main(String[] args) {

MusicPlayer mp = new MusicPlayer("Shape of You");

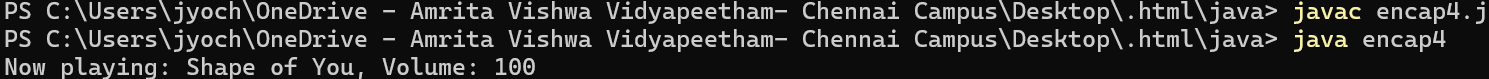
mp.setVolume(120);

System.out.println("Now playing: " + mp.getSongName() + ", Volume: " + mp.getVolume());

}

}

OUTPUT:



**15.PACKAGES:**

**i)**

CODE:

import java.awt.\*;

import java.awt.event.\*;

public class SimpleAWTLabelApp {

SimpleAWTLabelApp() {

Frame frame = new Frame("AWT Label Example");

Label label = new Label("Hello, AWT!");

Button button = new Button("Change Text");

label.setBounds(50, 50, 150, 30);

button.setBounds(50, 100, 100, 30);

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

label.setText("Text Updated!");

}

});

frame.add(label);

frame.add(button);

frame.setSize(300, 200);

frame.setLayout(null);

frame.setVisible(true);

frame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

frame.dispose();

}

});

}

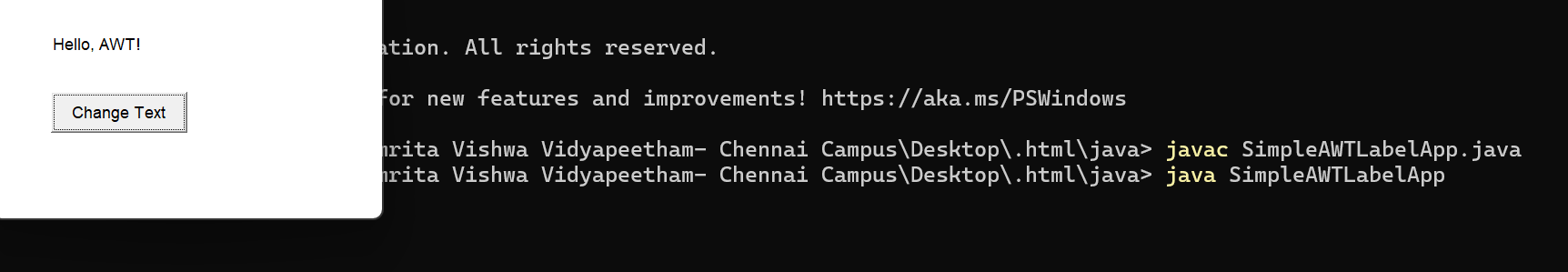
public static void main(String[] args) {

new SimpleAWTLabelApp();

}

}

OUTPUT:



ii)

CODE:

import java.util.ArrayList;

import java.util.HashMap;

public class UtilityExample {

public static void main(String[] args) {

ArrayList<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

System.out.println("Names: " + names);

HashMap<Integer, String> map = new HashMap<>();

map.put(1, "One");

map.put(2, "Two");

System.out.println("HashMap: " + map);

}

}

OUTPUT:



**iii)**

CODE:

package employee;

public class Employee {

private int empId;

private String name;

private double basicSalary;

private double hra;

private double da;

private double tax;

public Employee(int empId, String name, double basicSalary, double hra, double da, double tax) {

this.empId = empId;

this.name = name;

this.basicSalary = basicSalary;

this.hra = hra;

this.da = da;

this.tax = tax;

}

public int getEmpId() { return empId; }

public String getName() { return name; }

public double getBasicSalary() { return basicSalary; }

public double getHra() { return hra; }

public double getDa() { return da; }

public double getTax() { return tax; }

}

package employee;

public class SalaryCalculator {

public static double calculateGrossSalary(Employee emp) {

return emp.getBasicSalary() + emp.getHra() + emp.getDa();

}

public static double calculateNetSalary(Employee emp) {

return calculateGrossSalary(emp) - emp.getTax();

}

}

package employee;

public class PayrollDisplay {

public static void displayPayroll(Employee emp) {

System.out.println("\n----------------------");

System.out.println("EMPLOYEE PAYROLL DETAILS");

System.out.println("----------------------");

System.out.println("Employee ID : " + emp.getEmpId());

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

System.out.println("Basic Salary : ₹" + emp.getBasicSalary());

System.out.println("HRA : ₹" + emp.getHra());

System.out.println("DA : ₹" + emp.getDa());

System.out.println("Gross Salary : ₹" + SalaryCalculator.calculateGrossSalary(emp));

System.out.println("Tax Deduction : ₹" + emp.getTax());

System.out.println("Net Salary : ₹" + SalaryCalculator.calculateNetSalary(emp));

System.out.println("----------------------\n");

}

}

public class PayrollSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Employee ID: ");

int empId = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter Employee Name: ");

String name = scanner.nextLine();

System.out.print("Enter Basic Salary: ₹");

double basicSalary = scanner.nextDouble();

System.out.print("Enter HRA (House Rent Allowance): ₹");

double hra = scanner.nextDouble();

System.out.print("Enter DA (Dearness Allowance): ₹");

double da = scanner.nextDouble();

System.out.print("Enter Tax Deduction: ₹");

double tax = scanner.nextDouble();

Employee emp = new Employee(empId, name, basicSalary, hra, da, tax);

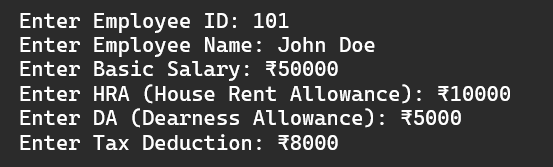
PayrollDisplay.displayPayroll(emp);

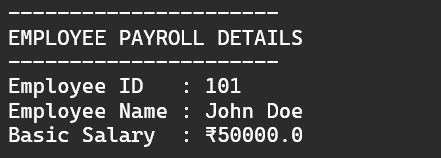
scanner.close();

}

}

OUTPUT:





**iv)**

CODE:

package details;

public class Student {

private int studentId;

private String name;

private double marks1, marks2, marks3;

public Student(int studentId, String name, double marks1, double marks2, double marks3) {

this.studentId = studentId;

this.name = name;

this.marks1 = marks1;

this.marks2 = marks2;

this.marks3 = marks3;

}

public int getStudentId() { return studentId; }

public String getName() { return name; }

public double getMarks1() { return marks1; }

public double getMarks2() { return marks2; }

public double getMarks3() { return marks3; }

}

package details;

public class MarksCalculator {

public static double calculateTotal(Student stu) {

return stu.getMarks1() + stu.getMarks2() + stu.getMarks3();

}

public static char calculateGrade(double total) {

double avg = total / 3;

if (avg >= 90) return 'A';

else if (avg >= 75) return 'B';

else if (avg >= 50) return 'C';

else return 'F';

}

}

package details;

public class Display {

public static void showStudentDetails(Student stu) {

double total = MarksCalculator.calculateTotal(stu);

char grade = MarksCalculator.calculateGrade(total);

System.out.println("\n----------------------");

System.out.println("STUDENT REPORT CARD");

System.out.println("----------------------");

System.out.println("Student ID : " + stu.getStudentId());

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

System.out.println("Total Marks : " + total);

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

System.out.println("----------------------\n");

}

}

import details.Student;

import details.MarksCalculator;

import details.Display;

import java.util.Scanner;

public class StudentApp {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Taking Student Details

System.out.print("Enter Student ID: ");

int studentId = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter Student Name: ");

String name = scanner.nextLine();

System.out.print("Enter Marks in Subject 1: ");

double marks1 = scanner.nextDouble();

System.out.print("Enter Marks in Subject 2: ");

double marks2 = scanner.nextDouble();

System.out.print("Enter Marks in Subject 3: ");

double marks3 = scanner.nextDouble();

// Creating Student Object

Student stu = new Student(studentId, name, marks1, marks2, marks3);

// Displaying Student Report

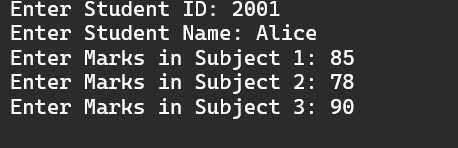
Display.showStudentDetails(stu);

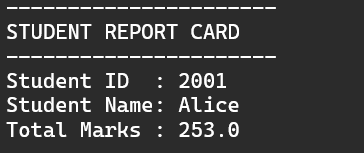
scanner.close();

}

}

OUTPUT:





**16.EXCEPTION HANDLING**

**i)**

CODE:

class InsufficientFundsException extends Exception {

InsufficientFundsException(String message) {

super(message);

}

}

public class ATM {

private static double balance = 5000; // Initial balance

public static void main(String[] args) {

try {

withdraw(6000); // Trying to withdraw more than the balance

} catch (InsufficientFundsException e) {

System.out.println("Transaction failed: " + e.getMessage());

}

}

public static void withdraw(double amount) throws InsufficientFundsException {

if (amount > balance) {

throw new InsufficientFundsException("Insufficient balance! Available: $" + balance);

}

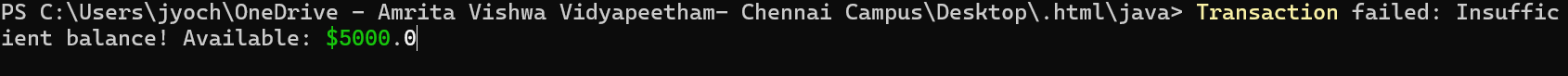
balance -= amount;

System.out.println("Withdrawal successful! Remaining Balance: $" + balance);

}

}

OUTPUT:



**ii)**

CODE:

class AgeRestrictionException extends Exception {

AgeRestrictionException(String message) {

super(message);

}

}

public class TrainBooking {

public static void main(String[] args) {

try {

bookTicket(3); // Booking ticket for a 3-year-old

} catch (AgeRestrictionException e) {

System.out.println("Booking failed: " + e.getMessage());

}

}

public static void bookTicket(int age) throws AgeRestrictionException {

if (age < 5) {

throw new AgeRestrictionException("Children below 5 years cannot travel alone.");

}

System.out.println("Ticket booked successfully!");

}

}

OUTPUT:

**iii)**

CODE:

import java.util.Scanner;

class InvalidCouponException extends Exception {

InvalidCouponException(String message) {

super(message);

}

}

class OutOfStockException extends Exception {

OutOfStockException(String message) {

super(message);

}

}

public class ShoppingCart {

private static double cartTotal = 0;

private static final String validCoupon = "DISCOUNT20";

private static boolean productInStock = true; // Simulating product availability

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

do {

System.out.println("\n===== Shopping Cart =====");

System.out.println("1. Add Item ($50)");

System.out.println("2. Apply Coupon");

System.out.println("3. Checkout");

System.out.println("4. Exit");

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

choice = scanner.nextInt();

try {

switch (choice) {

case 1:

addItem();

break;

case 2:

System.out.print("Enter coupon code: ");

String coupon = scanner.next();

applyCoupon(coupon);

break;

case 3:

checkout();

break;

case 4:

System.out.println("Thank you for shopping with us!");

break;

default:

System.out.println("Invalid choice. Try again.");

}

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

} while (choice != 4);

}

public static void addItem() throws OutOfStockException {

if (!productInStock) throw new OutOfStockException("Product is out of stock!");

cartTotal += 50;

System.out.println("Item added! Cart total: $" + cartTotal);

}

public static void applyCoupon(String coupon) throws InvalidCouponException {

if (!coupon.equals(validCoupon)) throw new InvalidCouponException("Invalid coupon code: " + coupon);

cartTotal \*= 0.8;

System.out.println("Coupon applied! New total: $" + cartTotal);

}

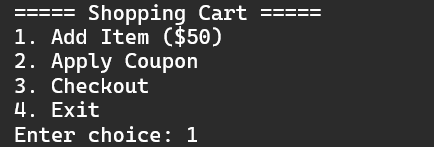
public static void checkout() {

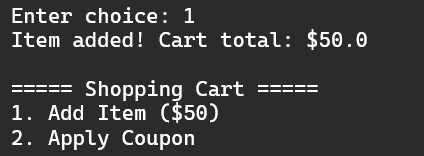
System.out.println("Order placed! Final amount: $" + cartTotal);

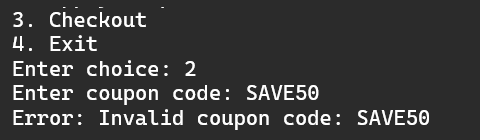
}

}

OUTPUT:







**iv)**

CODE:

import java.util.Scanner;

class AgeRestrictionException extends Exception {

AgeRestrictionException(String message) {

super(message);

}

}

class NoSeatsAvailableException extends Exception {

NoSeatsAvailableException(String message) {

super(message);

}

}

class InvalidInputException extends Exception {

InvalidInputException(String message) {

super(message);

}

}

public class RailwayBooking {

private static int availableSeats = 3; // Initial seat availability

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

do {

System.out.println("\n===== Railway Ticket Booking =====");

System.out.println("1. Book Ticket");

System.out.println("2. Check Available Seats");

System.out.println("3. Exit");

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

choice = scanner.nextInt();

try {

switch (choice) {

case 1:

System.out.print("Enter Passenger Name: ");

String name = scanner.next();

System.out.print("Enter Passenger Age: ");

int age = scanner.nextInt();

bookTicket(name, age);

break;

case 2:

System.out.println("Available Seats: " + availableSeats);

break;

case 3:

System.out.println("Thank you for using our Railway Booking System!");

break;

default:

throw new InvalidInputException("Invalid choice! Please enter a valid option.");

}

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

} while (choice != 3);

}

public static void bookTicket(String name, int age) throws AgeRestrictionException, NoSeatsAvailableException {

if (age < 5) throw new AgeRestrictionException("Children below 5 years cannot travel alone.");

if (availableSeats == 0) throw new NoSeatsAvailableException("No seats available!");

availableSeats--;

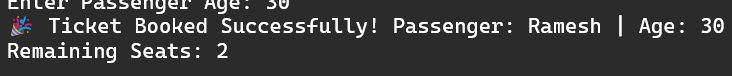
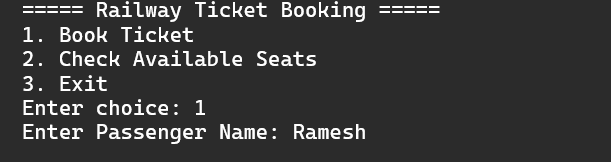
System.out.println("🎉 Ticket Booked Successfully! Passenger: " + name + " | Age: " + age);

System.out.println("Remaining Seats: " + availableSeats);

}

}

OUTPUT:



**17.FILE HANDLING**

i)

CODE:

import java.io.BufferedReader;

import java.io.FileReader;

public class ReadFileExample {

public static void main(String[] args) {

try {

FileReader reader = new FileReader("myfile.txt");

BufferedReader buffer = new BufferedReader(reader);

String line = buffer.readLine();

while (line != null) {

System.out.println(line);

line = buffer.readLine();

}

buffer.close();

} catch (Exception e) {

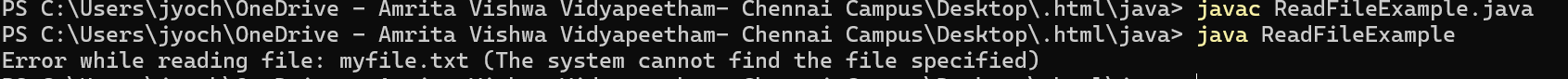
System.out.println("Error while reading file: " + e.getMessage());

}

}

}

OUTPUT:



**ii)**

CODE:

import java.io.BufferedWriter;

import java.io.FileWriter;

public class WriteFileExample {

public static void main(String[] args) {

try {

FileWriter file = new FileWriter("myfile.txt", true); // Append mode

BufferedWriter writer = new BufferedWriter(file);

writer.write("This is a new line.");

writer.newLine();

writer.write("Appending more data.");

writer.close();

System.out.println("Successfully wrote to the file.");

} catch (Exception e) {

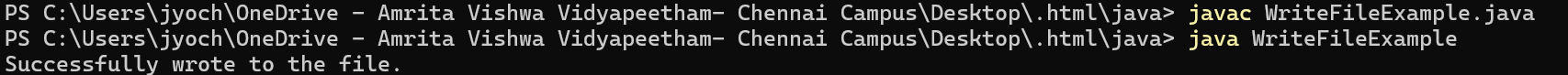
System.out.println("Error while writing to file: " + e.getMessage());

}

}

}

OUTPUT:



**iii)**

CODE:

import java.io.\*;

public class FileHandlingMultipleTryCatch {

public static void main(String[] args) {

String fileName = "example.txt";

try (FileWriter writer = new FileWriter(fileName)) {

writer.write("This is a test file.\n");

writer.write("This is line 2.");

System.out.println("Successfully wrote to the file.");

} catch (IOException e) {

System.out.println("Error writing file: " + e.getMessage());

}

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

System.out.println("\nReading file contents:");

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.out.println("Error reading file: " + e.getMessage());

}

try (FileWriter writer = new FileWriter(fileName, true)) {

writer.write("\nAdding an extra line.");

System.out.println("\nSuccessfully appended to the file.");

} catch (IOException e) {

System.out.println("Error appending to file: " + e.getMessage());

}

try (FileInputStream fis = new FileInputStream(fileName)) {

System.out.println("\nReading using FileInputStream:");

int content;

while ((content = fis.read()) != -1) {

System.out.print((char) content);

}

} catch (IOException e) {

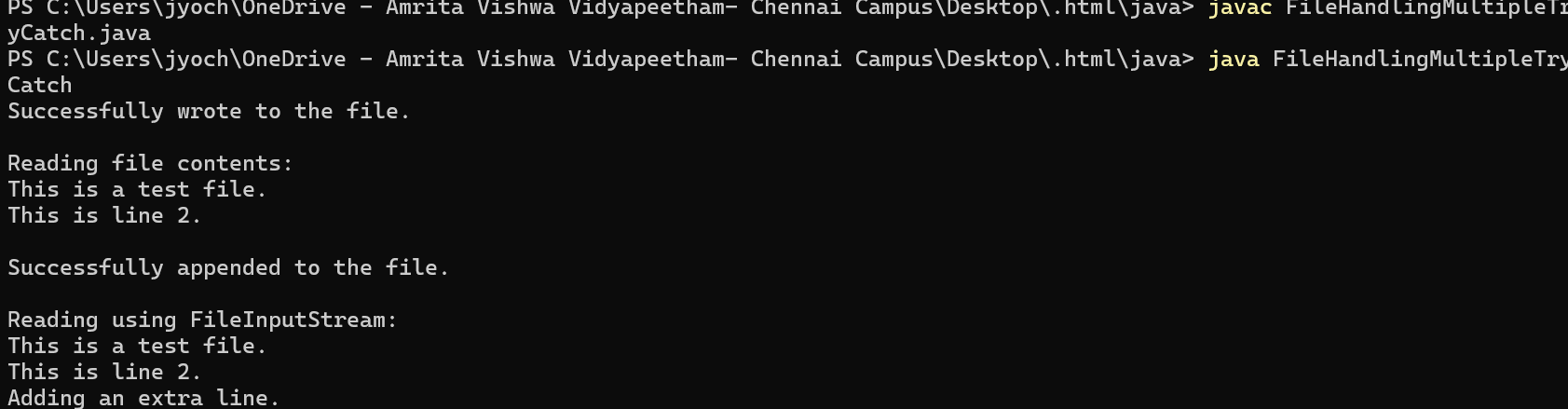
System.out.println("Error reading file: " + e.getMessage());

}

}

}

OUTPUT:



**IV)**

CODE:

import java.io.\*;

public class ReadWriteWordCount {

public static void main(String[] args) {

try (FileReader file = new FileReader("input.txt");

BufferedReader reader = new BufferedReader(file);

FileWriter fileWriter = new FileWriter("output.txt");

BufferedWriter writer = new BufferedWriter(fileWriter)) {

String line;

int wordCount = 0;

while ((line = reader.readLine()) != null) {

System.out.println(line);

writer.write(line);

writer.newLine();

wordCount += line.split("\\s+").length; // Count words

}

System.out.println("Total Words: " + wordCount);

} catch (IOException e) {

System.out.println("Error: " + e.getMessage());

}

}

}

OUTPUT:

