

LAB RECORD

23CSE111- Object Oriented Programming

***Submitted by***

CH.SC.U4CSE24001 -Aaradhana s

# BACHELOR OF TECHNOLOGY

## IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM AMRITA SCHOOL OF COMPUTING

CHENNAI

March - 2025



**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.U4CSE24001 – Aaradhana S*** 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.

This Lab examination held on 08/04/2025

Internal Examiner 1 Internal Examiner 2

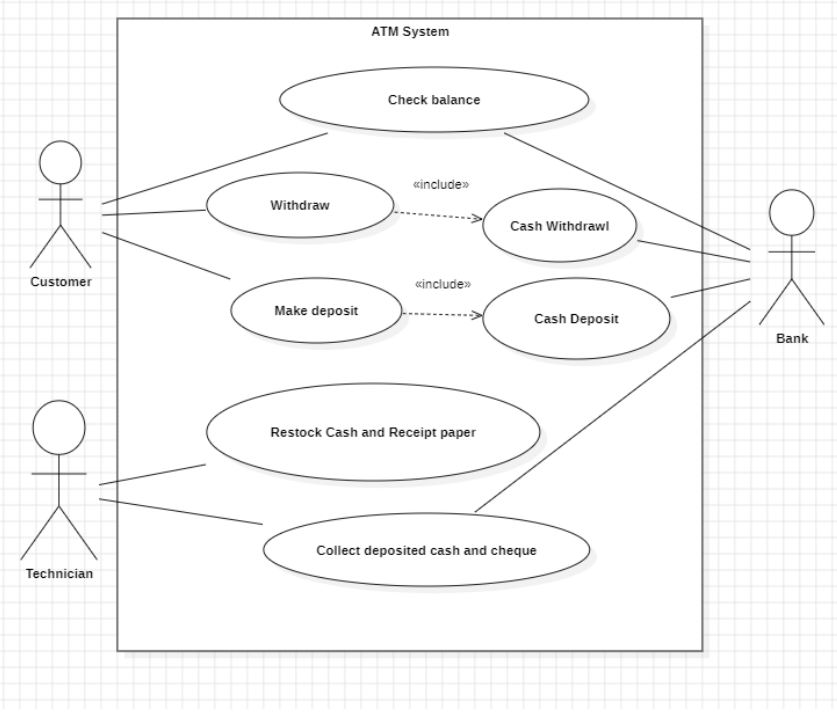
## INDEX

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE.NO** |
| **UML DIAGRAM** | |  |
| 1. | **ATM SYSTEM** |  |
|  | 1.a)Use Case Diagram |  |
|  | 1.b)Class Diagram |  |
|  | 1.c) Sequence Diagram |  |
|  | 1.d)State diagram |  |
|  | 1.e)Deployment diagram |  |
| 2. | **LIBRARY MANAGEMENT SYSTEM** |  |
|  | 2.a) Use Case Diagram |  |
|  | 2.b) Class Diagram |  |
|  | 2.c) Sequence Diagram |  |
|  | 2.d) State diagram |  |
|  | 2.e) Deployment diagram |  |
| 3. | **BASIC JAVA PROGRAMS** |  |
|  | 3.a) |  |
|  | 3.b) |  |
|  | 3.c) |  |
|  | 3.d) |  |
|  | 3.e) |  |
|  | 3.f) |  |
|  | 3.g) |  |
|  | 3.h) |  |
|  | 3.i) |  |
|  | 3.j) |  |
|  | **INHERITANCE** |  |

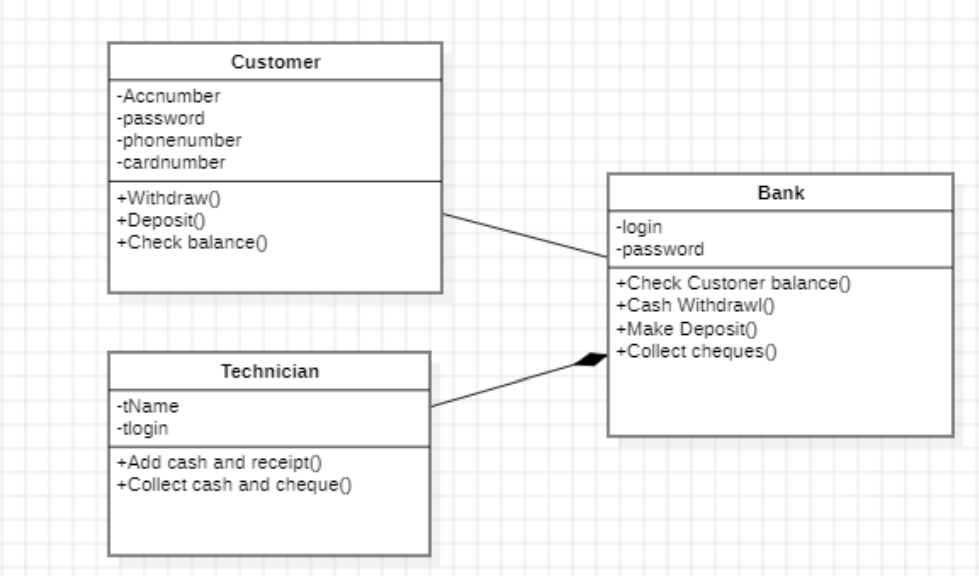
|  |  |  |
| --- | --- | --- |
| 4. | **SINGLE INHERITANCE PROGRAMS** |  |
|  | 4.a) |  |
|  | 4.b) |  |
| 5. | **MULTILEVEL INHERITANCE PROGRAMS** |  |
|  | 5.a) |  |
|  | 5.b) |  |
| 6. | **HIERARCHICAL INHERITANCE PROGRAMS** |  |
|  | 6.a) |  |
|  | 6.b) |  |
| 7. | **HYBRID INHERITANCE PROGRAMS** |  |
|  | 7.a) |  |
|  | 7.b) |  |
|  | **POLYMORPHISM** |  |
| 8. | **CONSTRUCTOR PROGRAMS** |  |
|  | 8.a) |  |
| 9. | **CONSTRUCTOR OVERLOADING PROGRAMS** |  |
|  | 9.a) |  |
| 10. | **METHOD OVERLOADING PROGRAMS** |  |
|  | 10.a) |  |
|  | 10.b) |  |
| 11. | **METHOD OVERRIDING PROGRAMS** |  |
|  | 11.a) |  |
|  | 11.b) |  |
|  | **ABSTRACTION** |  |
| 12. | **INTERFACE PROGRAMS** |  |
|  | 12.a) |  |
|  | 12.b) |  |
|  | 12.c) |  |
|  | 12.d) |  |
| 13. | **ABSTRACT CLASS PROGRAMS** |  |
|  | 13.a) |  |
|  | 13.b) |  |
|  | 13.c) |  |
|  | 13.d) |  |
|  | **ENCAPSULATION** |  |
| 14. | **ENCAPSULATION PROGRAMS** |  |
|  | 14.a) |  |
|  | 14.b) |  |
|  | 14.c) |  |
|  | 14.d) |  |
| 15. | **PACKAGES PROGRAMS** |  |
|  | 15.a)User Defined Packages |  |
|  | 15.b)User Defined Packages |  |
|  | 15.c)Built – in Package(3 Packages) |  |
|  | 15.d)Built – in Package(3 Packages) |  |
| 16. | **EXCEPTION HANDLING PROGRAMS** |  |
|  | 16.a) |  |
|  | 16.b) |  |
|  | 16.c) |  |
|  | 16.d) |  |
| 17. | **FILE HANDLING PROGRAMS** |  |
|  | 17.a) |  |
|  | 17.b) |  |
|  | 17.c) |  |
|  | 17.d) |  |

#### UML DIAGRAM – ATM SYSTEM

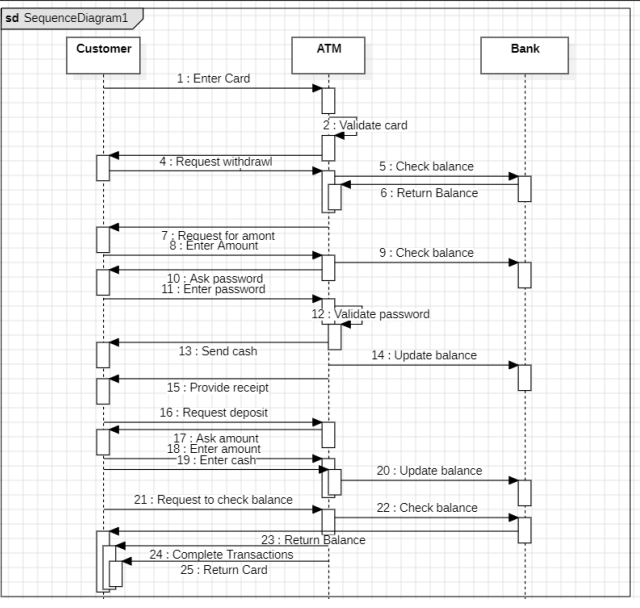
#### USE CASE DIAGRAM



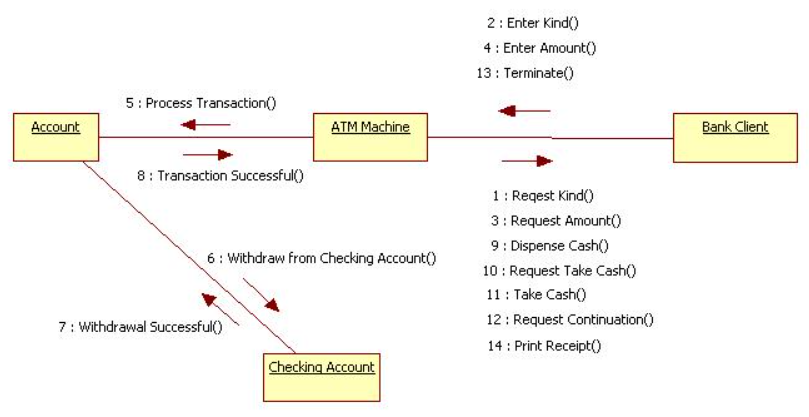
#### CLASS DIAGRAM



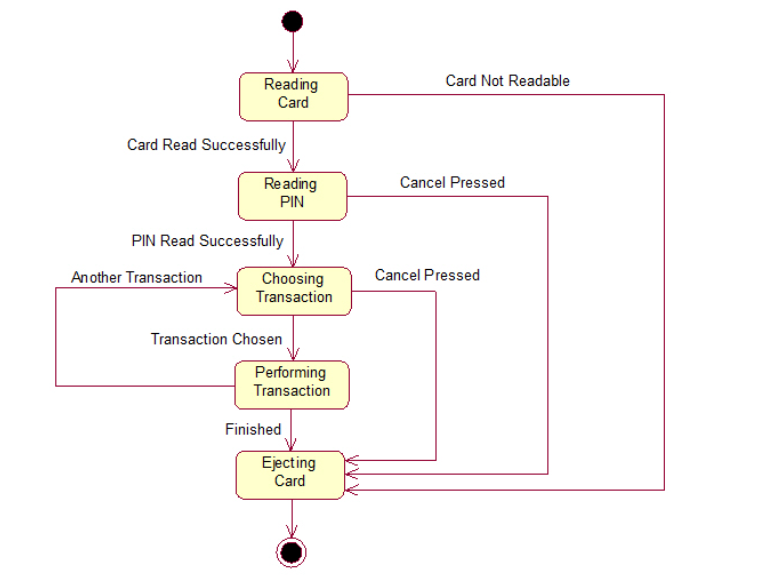
SEQUENCE DIAGRAM



DEPLOYMENT DIAGRAM



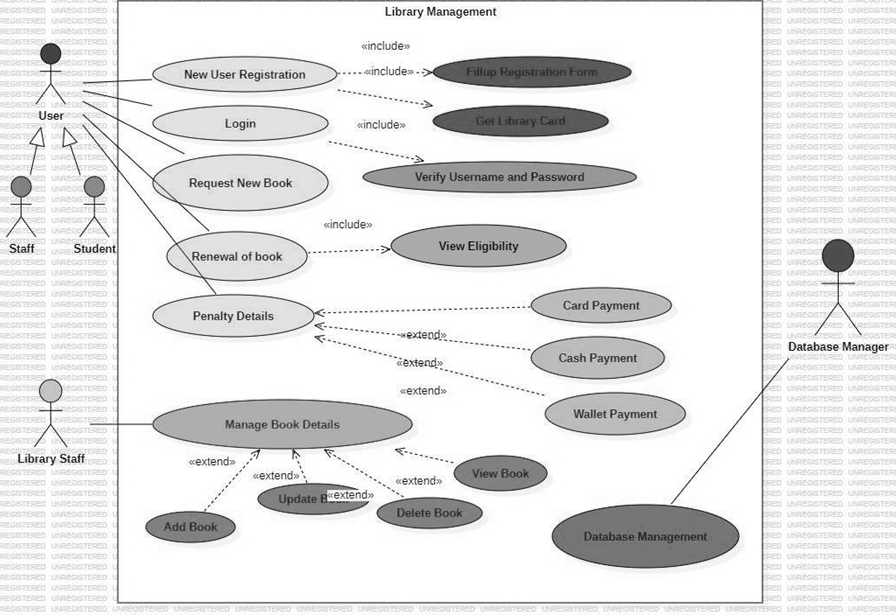
STATE DIAGRAM



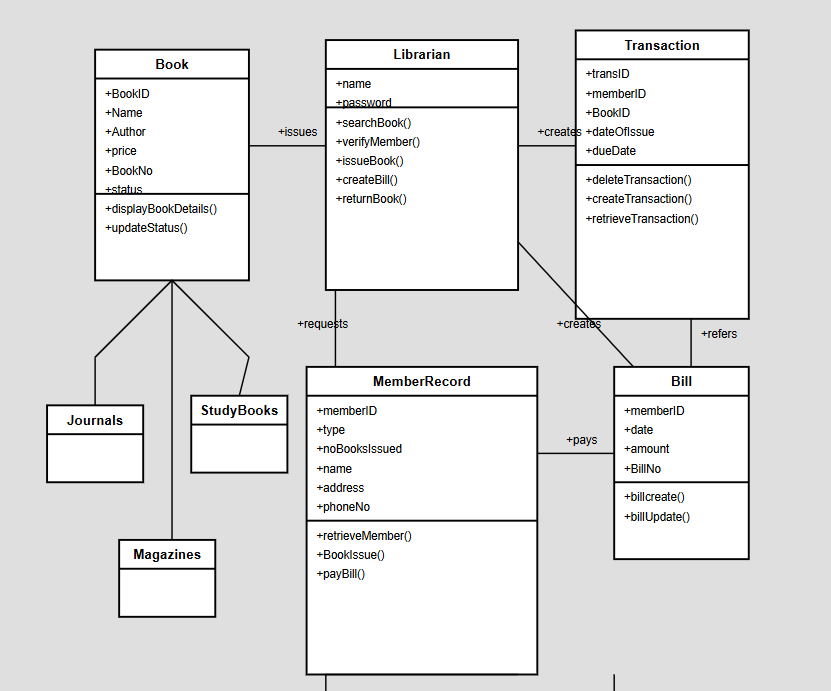
UML DIAGRAM 2

ONLINE SHOPPING MANAGEMENT SYSTEM

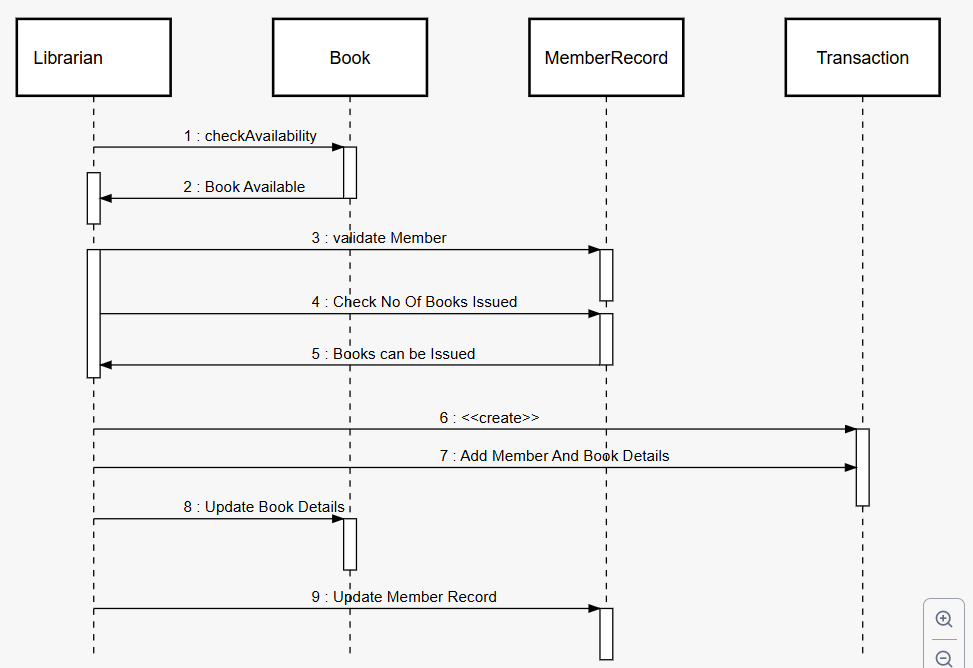
USECASE DIAGRAM



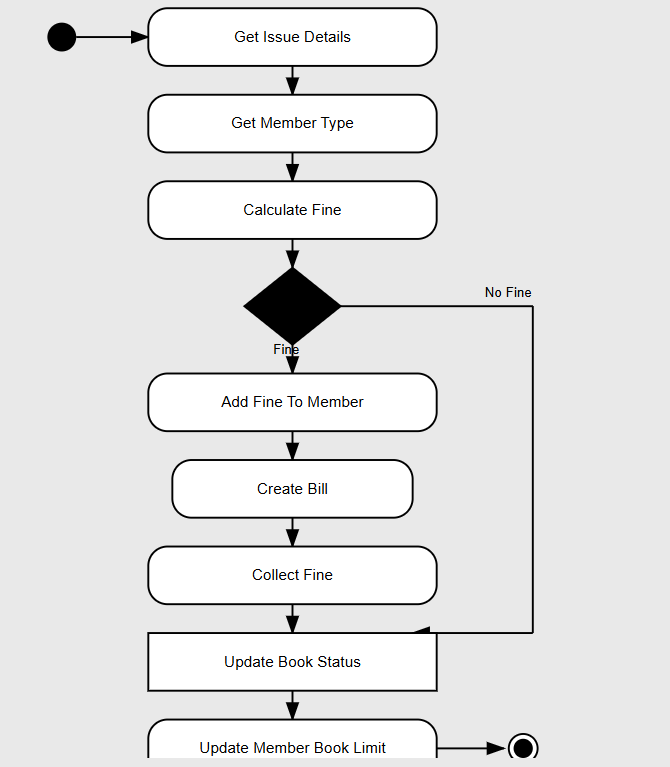
CLASS DIAGRAM



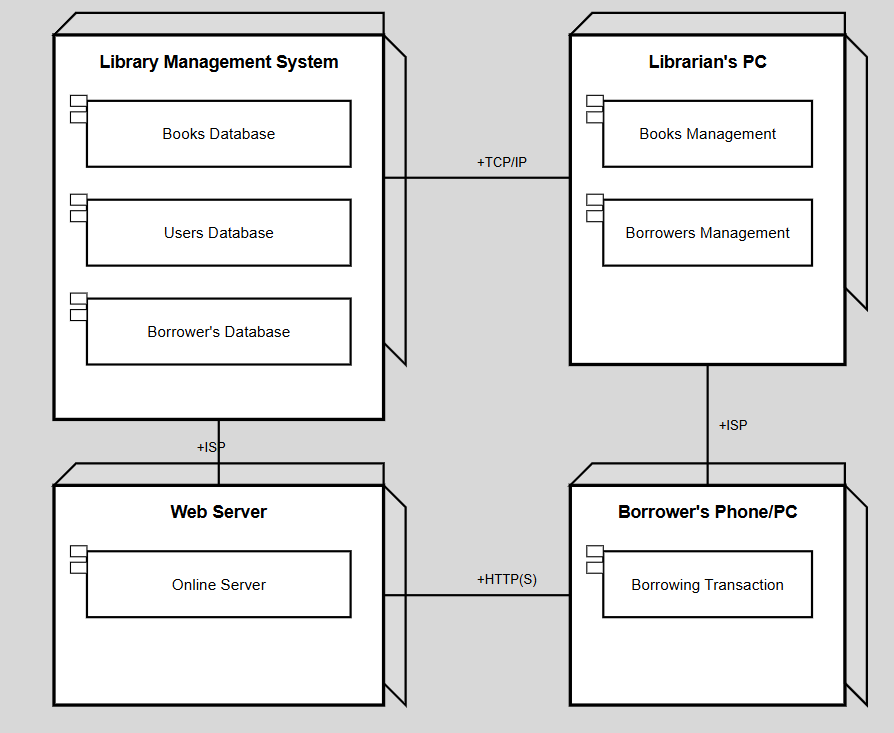
SEQUENCE DIAGRAM



STATE DIAGRAM



DEPLOYMENT DIAGRAM



BASIC PYTHON PROGRAMS

1) import java.util.Scanner;

public class Fibonacci {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int terms = scanner.nextInt();

int num1 = 0, num2 = 1;

System.out.print("Fibonacci Series: ");

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

System.out.print(num1 + " ");

int nextTerm = num1 + num2;

num1 = num2;

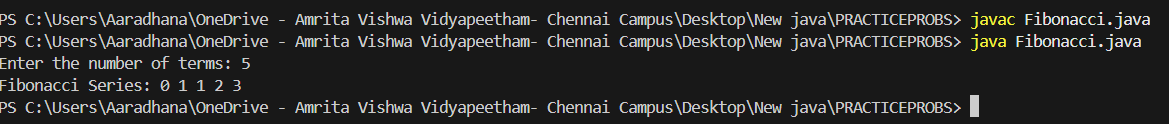
num2 = nextTerm;

}

}

}

OUTPUT:



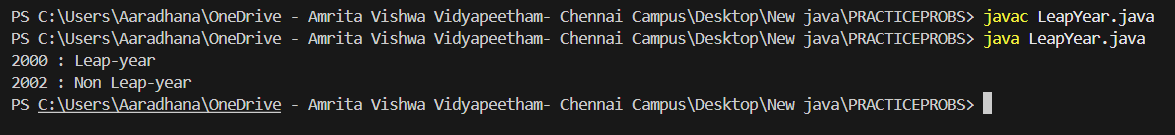
2. import java.io.\*;

public class LeapYear { public static void isLeapYear(int year) { boolean is\_leap\_year = false;

if (year % 4 == 0) {  
 is\_leap\_year = true;  
  
 if (year % 100 == 0) {  
 if (year % 400 == 0)  
 is\_leap\_year = true;  
 else  
 is\_leap\_year = false;  
 }  
 }  
 else  
 is\_leap\_year = false;  
  
 if (!is\_leap\_year)  
 System.out.println(year + " : Non Leap-year");  
 else  
 System.out.println(year + " : Leap-year");  
}  
  
public static void main(String[] args) {  
 isLeapYear(2000);  
 isLeapYear(2002);

}}

**Output:**



3.

public class MaxOfThreeNumbers {

public static void main(String[] args) {

int num1 = 10, num2 = 20, num3 = 15, max;

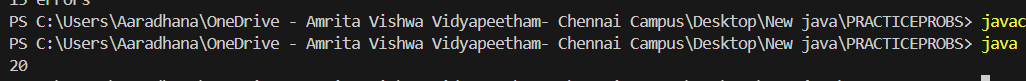
max = (num1 > num2) ? (num1 > num3 ? num1 : num3) : (num2 > num3 ? num2 : num3);

System.out.println(“Maximum of ” + num1 + “, ” + num2 + “, and ” + num3 + ” is: ” + max);

}

}

OUTPUT:



4.class OddNumber {

public static void main(String args[]) {

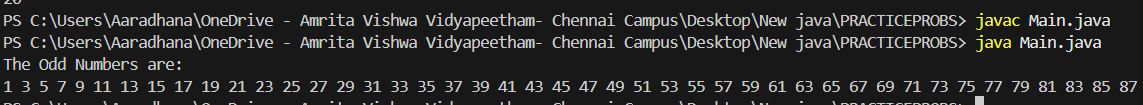
System.out.println("The Odd Numbers are:");

for (int i = 1; i <= 100; i++) {

if (i % 2 != 0) {

System.out.print(i + " ");}}}}

OUTPUT:



5. import java.util.Scanner;

public class ReverseNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

int num = scanner.nextInt();

int reversed = 0;

while (num != 0) {

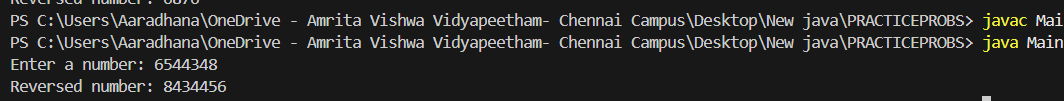
int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;}

System.out.println("Reversed number: " + reversed);}}

OUTPUT:



6. class SumOfNum{

public static void main(String args[]){

int sum = 0;

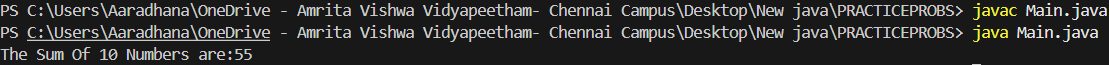
int n=10;

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

{sum = sum + i;}

System.out.println("The Sum Of "+n+" Numbers are:" + sum);}}

OUTPUT:



7.class factorial {

static int factorial(int n) {

int res = 1, i;

for (i = 2; i <= n; i++)

r \*= i;

return r; }

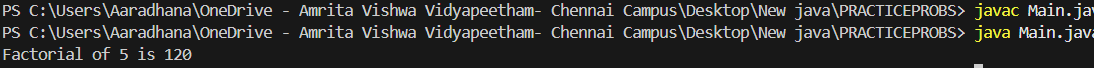
public static void main(String[] args) {

int num = 5;

System.out.println("Factorial of " + num + " is "

+ factorial(5)); }}

OUTPUT:



8.

import java.io.\*;

class lcm {

public static void main(String[] args)

{

int a = 15, b = 25;

int ans = (a > b) ? a : b;

while (true) {

if (ans % a == 0 && ans % b == 0)

break;

ans++;

}

// Printing the Result

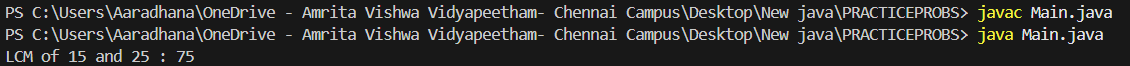
System.out.println("LCM of " + a + " and " + b

+ " : " + ans);

}

}

OUTPUT:



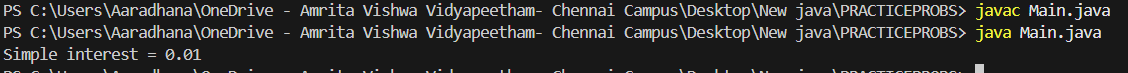
9.

import java.io.\*;

class simint { public static void main(String args[]) { // We can change values here for // different inputs float P = 1, R = 1, T = 1;

/\* Calculate simple interest \*/  
 float SI = (P \* T \* R) / 100;  
 System.out.println("Simple interest = " + SI);  
}}

OUTPUT:



10.import java.io.\*;

class perimrec {

static void perimeter(int length, int breadth){

int perimeter = 2 \* (length + breadth);

System.out.println("The perimeter of the given rectangle of length "

+ length + " and breadth " + breadth + " = "

+ perimeter);}

public static void main(String[] args){

int length = 10;

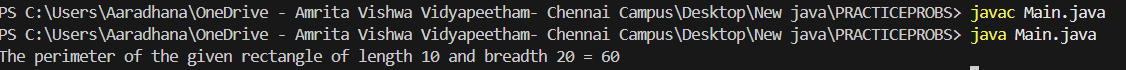
int breadth = 20;

perimeter(length, breadth);

}

}

OUTPUT:



# INHERITENCE

SINGLE INHERITENCE

1.

class Vehicle {

void start() {

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

}}

class Car extends Vehicle {

void drive() {

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

}}

public class SinheritII {

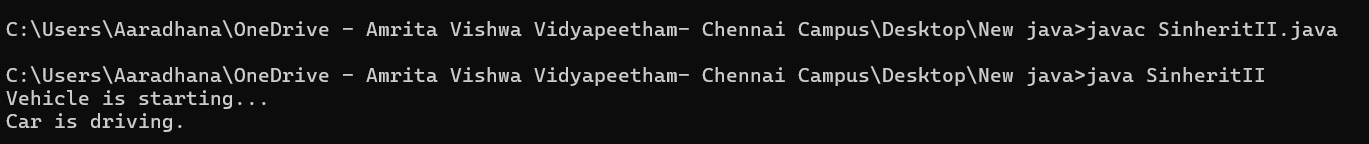
public static void main(String[] args) {

Car car = new Car();

car.start(); // Method from parent class

car.drive(); // Method from child class}}

OUTPUT:



2.

class Animal {

void eat()

{

System.out.println("This animal eats food.");

}

}

class Dog extends Animal

{

void bark() {

System.out.println("The dog barks.");

}

}

public class SinheritI {

public static void main(String[] args) {

Dog dog = new Dog();

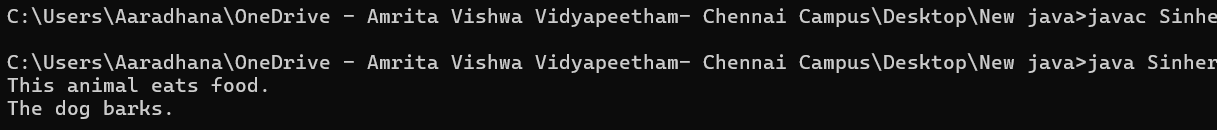
dog.eat(); // Method from parent class

dog.bark(); // Method from child class

}

}

OUTPUT:



MULTILEVEL INHERITENCE

1.

class Animal {

void sound() {

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

class Dog extends Animal {

void bark() {

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

class Puppy extends Dog {

void play() {

System.out.println("Puppy plays.");}}

public class MinheritI {

public static void main(String[] args) {

Puppy puppy = new Puppy();

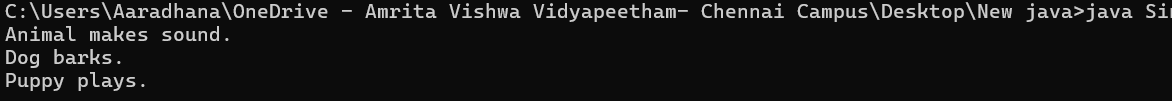
puppy.sound();

puppy.bark();

puppy.play();

}}

OUTPUT:



2.

class Vehicle {

public void start() {

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

}

public void stop() {

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

class Car extends Vehicle {

public void drive() {

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

class ElectricCar extends Car {

public void charge() {

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

public class MinheritII {

public static void main(String[] args) {

ElectricCar electricCar = new ElectricCar();

electricCar.start(); // Inherited from Vehicle

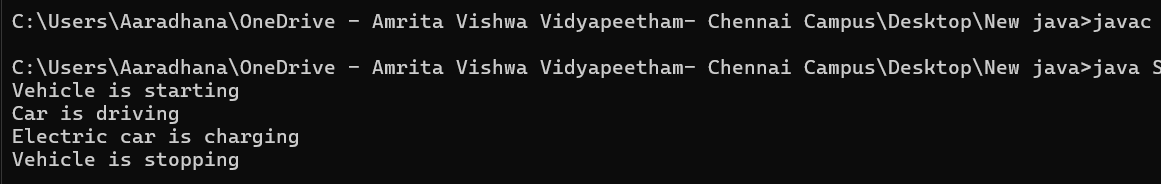
electricCar.drive(); // Inherited from Car

electricCar.charge(); // Defined in ElectricCar

electricCar.stop(); // Inherited from Vehicle

}}

OUTPUT:



HIERARCHICAL INHERITANCE

1.

class Animal {

public void eat() {

System.out.println("Animal is eating");}}

class Dog extends Animal {

public void bark() {

System.out.println("Dog is barking");}}

class Cat extends Animal {

public void meow() {

System.out.println("Cat is meowing");}}

public class HinheritanceI {

public static void main(String[] args) {

Dog dog = new Dog();

dog.eat(); // Inherited from Animal

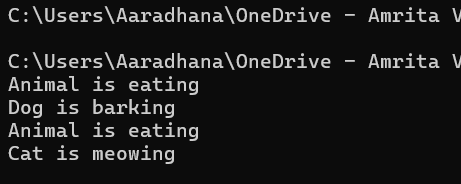
dog.bark(); // Defined in Dog

Cat cat = new Cat();

cat.eat(); // Inherited from Animal

cat.meow(); // Defined in Cat}}

OUTPUT:



2.

class Shape {

public void draw() {

System.out.println("Drawing a shape");}}

class Circle extends Shape {

public void drawCircle() {

System.out.println("Drawing a Circle");}}

class Square extends Shape {

public void drawSquare() {

System.out.println("Drawing a Square");}}

public class HinheritanceII {

public static void main(String[] args) {

Circle circle = new Circle();

circle.draw(); // Inherited from Shape

circle.drawCircle(); // Defined in Circle

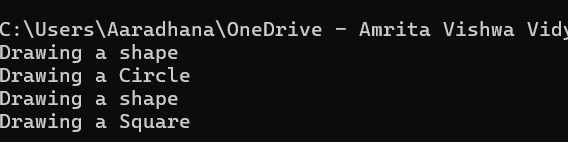
Square square = new Square();

square.draw(); // Inherited from Shape

square.drawSquare(); // Defined in Square

}}

OUTPUT:



HYBRID INHERITANCE

1.

class Person {

void displayInfo() {

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

class Student extends Person {

void study() {

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

}}

class Athlete extends Person {

void play() {

System.out.println("Athlete is playing.");

}}

class EngineeringStudent extends Student {

void code() {

System.out.println("Engineering student writes code.");

}}

public class HybridInheritance3 {

public static void main(String[] args) {

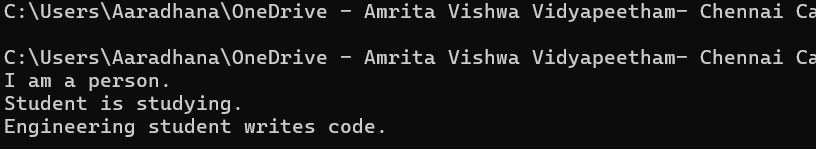
EngineeringStudent es = new EngineeringStudent();

es.displayInfo(); // From Person

es.study(); // From Student

es.code(); // From EngineeringStudent}}

OUTPUT:



2.

class Vehicle {

void move() {

System.out.println("Vehicles can move.");

}}

class Engine {

void engineType() {

System.out.println("Engines can be petrol or diesel.");

}}

class Car extends Vehicle {

void wheels() {

System.out.println("A car has four wheels.");

}}

class SportsCar extends Car {

void speed() {

System.out.println("Sports car is very fast.");

}}

public class HybridInheritance2 {

public static void main(String[] args) {

SportsCar ferrari = new SportsCar();

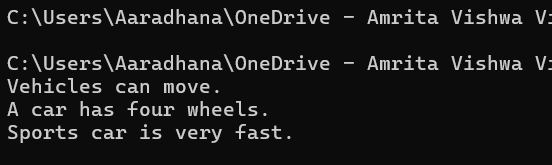
ferrari.move(); // From Vehicle

ferrari.wheels(); // From Car

ferrari.speed(); // From SportsCar

}}

OUTPUT:



POLYMORPHISM

CONSTRUCTOR

1.

class Car {

String model;

int year;

Car(String model, int year) {

this.model = model; // 'this' refers to the current object

this.year = year;

}

void displayDetails() {

System.out.println("Car Model: " + model);

System.out.println("Manufacturing Year: " + year);

}}

public class Const {

public static void main(String[] args) {

Car car1 = new Car("Toyota Corolla", 2020);

Car car2 = new Car("Honda Civic", 2022);

car1.displayDetails();

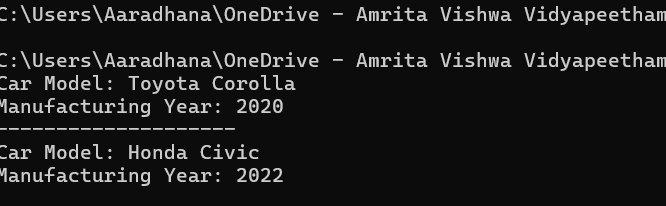
System.out.println("--------------------");

car2.displayDetails();

}

}

OUTPUT:



CONSTRUCTOR OVERLOADING

1.

class Rectangle {

int length, width;

public Rectangle() {

this.length = 1;

this.width = 1;

}

public Rectangle(int length, int width) {

this.length = length;

this.width = width;

}

public void area() {

int area = length \* width;

System.out.println("Area of Rectangle: " + area);

}}

public class COverload {

public static void main(String[] args) {

Rectangle rect1 = new Rectangle();

rect1.area(); // Output: Area of Rectangle: 1

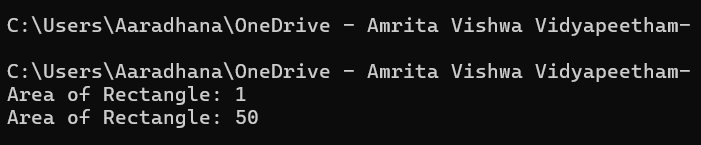
Rectangle rect2 = new Rectangle(5, 10);

rect2.area(); // Output: Area of Rectangle: 50

}

}

OUTPUT:



METHOD OVERLOADING

1.

class MathUtils {

// Method to add two integers

int add(int a, int b) {

return a + b;

}

// Overloaded method to add three integers

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

return a + b + c;

}

}

public class Mo1 {

public static void main(String[] args) {

MathUtils mu = new MathUtils();

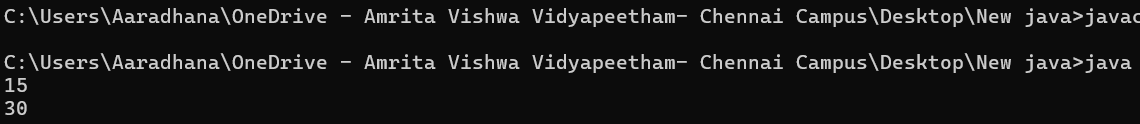
System.out.println(mu.add(5, 10)); // Calls the first add method

System.out.println(mu.add(5, 10, 15)); // Calls the overloaded add method

}

}

OUTPUT:



2.

class Display {

// Method to display integer

void show(int num) {

System.out.println("Integer: " + num);

}

// Overloaded method to display string

void show(String str) {

System.out.println("String: " + str);

}

}

public class Mo2 {

public static void main(String[] args) {

Display d = new Display();

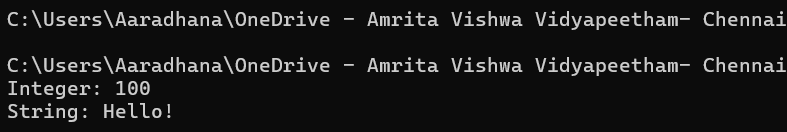
d.show(100); // Calls the integer version

d.show("Hello!"); // Calls the string version

}

}

OUTPUT:



METHOD OVERRIDING

1.

class Animal {

void sound() {

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

}}

class Dog extends Animal {

@Override

void sound() {

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

}}

public class Mor1 {

public static void main(String[] args) {

Animal myAnimal = new Animal();

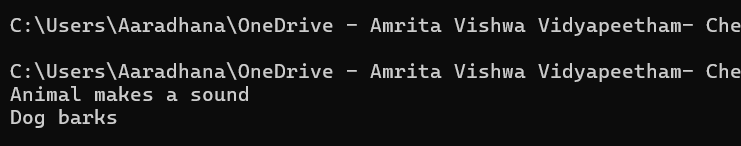
Animal myDog = new Dog(); // Polymorphism in action

myAnimal.sound(); // Calls Animal's sound method

myDog.sound(); // Calls Dog's overridden sound method

}}

OUTPUT:



2.

class Vehicle {

void drive() {

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

}}

class Car extends Vehicle {

// Overriding the drive method

@Override

void drive() {

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

}}

public class Mor2 {

public static void main(String[] args) {

Vehicle v = new Vehicle();

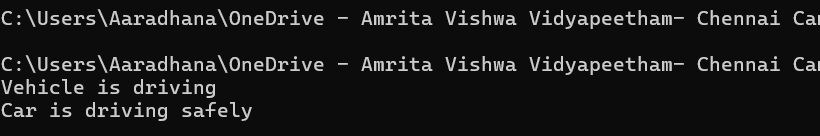
Vehicle c = new Car();

v.drive(); // Calls Vehicle's drive method

c.drive(); // Calls Car's overridden drive method

}}

OUTPUT:



ABSTRACTION

INTERFACE

1.

interface Animal {

void makeSound(); // Abstract method

}

class Dog implements Animal {

public void makeSound() {

System.out.println("Woof! Woof!");

}

}

public class Int1 {

public static void main(String[] args) {

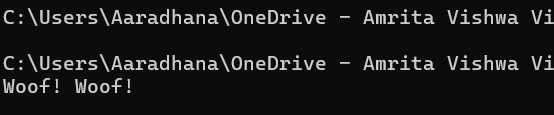
Animal myDog = new Dog();

myDog.makeSound();

}

}

OUTPUT:



2.

interface Vehicle {

void start();}

class Car implements Vehicle {

public void start() {

System.out.println("Car is starting with a key!");

}

}

class Bike implements Vehicle {

public void start() {

System.out.println("Bike is starting with a kick!");

}

}

public class Int2 {

public static void main(String[] args) {

Vehicle myCar = new Car();

Vehicle myBike = new Bike();

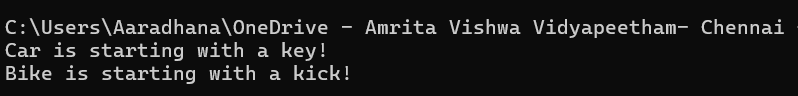
myCar.start();

myBike.start();

}

}

OUTPUT:



3.

interface Shape {

void draw();

double getArea(); }

class Circle implements Shape {

double radius;}

Circle(double r) {  
 this.radius = r;  
}  
public void draw() {  
 System.out.println("Drawing a Circle!");  
public double getArea() {  
 return Math.PI \* radius \* radius;  
}}

public class Int3 {

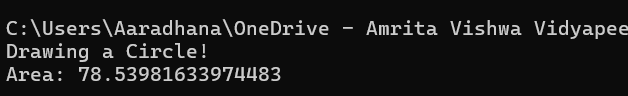
public static void main(String[] args)

{ Shape myCircle = new Circle(5);

myCircle.draw();

System.out.println("Area: " + myCircle.getArea()); } }

OUTPUT:



4.

interface ParentInterface {

void parentMethod();

}

interface ChildInterface extends ParentInterface {

void childMethod();

}

class MyClass implements ChildInterface {

public void parentMethod() {

System.out.println("Parent method implemented.");

}

public void childMethod() {

System.out.println("Child method implemented.");

}}

public class Int4 {

public static void main(String[] args) {

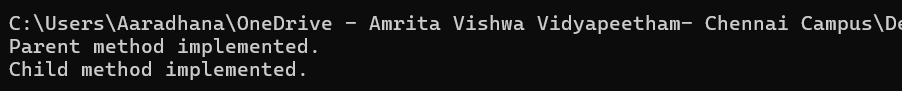
MyClass obj = new MyClass();

obj.parentMethod();

obj.childMethod();

}}

OUTPUT:



ABSTRACT CLASS

1.

abstract class Animal {

abstract void makeSound(); // Abstract method

void sleep() {

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

}}

class Dog extends Animal {

public void makeSound() {

System.out.println("Woof! Woof! ");

}}

public class Abst1{

public static void main(String[] args) {

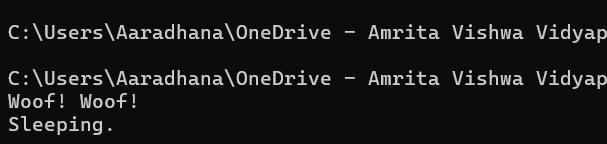
Animal myDog = new Dog();

myDog.makeSound();

myDog.sleep();

}}

OUTPUT:



2.

abstract class Vehicle {

String brand;

Vehicle(String brand) {

this.brand = brand;

}

abstract void start();

void showBrand() {

System.out.println("Brand: " + brand);

}}

class Car extends Vehicle {

Car(String brand) {

super(brand);}

public void start() {

System.out.println(brand + " Car is starting with a key! 🚗");

}}

public class Abst2.java {

public static void main(String[] args) {

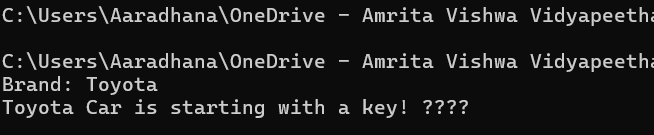
Car myCar = new Car("Toyota");

myCar.showBrand();

myCar.start();

}}

OUTPUT:



3.

abstract class Shape { abstract double getArea();

void display() {  
 System.out.println("This is a shape.");  
}}

class Circle extends Shape { double radius;

Circle(double r) {  
 this.radius = r;  
}  
public double getArea() {  
 return Math.PI \* radius \* radius;  
}}

public class Abst3 {

public static void main(String[] args) {

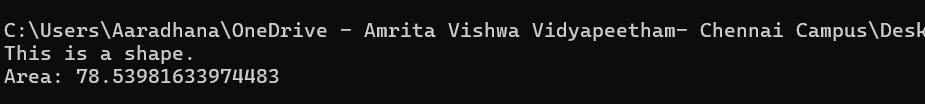
Circle myCircle = new Circle(5);

myCircle.display();

System.out.println("Area: " + myCircle.getArea());

}}

OUTPUT:



4.

abstract class Animal {

abstract void makeSound();

void sleep() {

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

}}

class Dog extends Animal {

void makeSound() {

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

}}

class Cat extends Animal {

void makeSound() {

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

}}

public class AbstractAnimal {

public static void main(String[] args) {

Animal a1 = new Dog();

a1.makeSound();

a1.sleep();

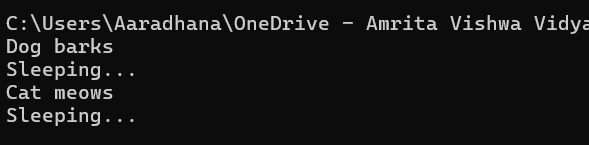
Animal a2 = new Cat();

a2.makeSound();

a2.sleep();

}}

OUTPUT:



ENCAPSULATION

1.

class Person {

private String name;

public String getName() {

return name;

}

public void setName(String newName) {

this.name = newName;

}

}

public class Enc1 {

public static void main(String[] args) {

Person p = new Person();

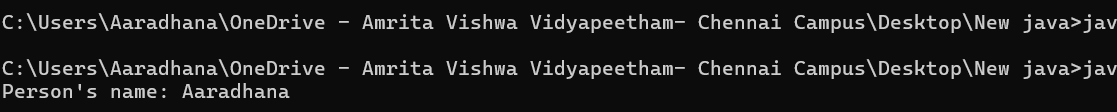
p.setName("Aaradhana");

System.out.println("Person's name: " + p.getName());

}

}

OUTPUT:



2.

class BankAccount {

private double balance;

public double getBalance() {

return balance;}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: " + amount);

} else {

System.out.println("Invalid deposit amount!");}}

public void withdraw(double amount) {

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

balance -= amount;

System.out.println("Withdrawn: " + amount);

} else {

System.out.println("Invalid withdrawal amount!");}}}

public class Enc2 {

public static void main(String[] args) {

BankAccount acc = new BankAccount();

acc.deposit(1000);

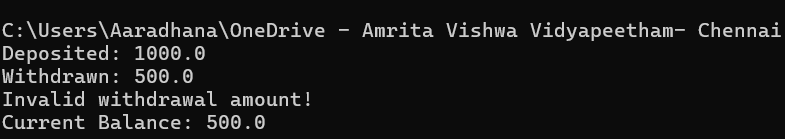
acc.withdraw(500);

acc.withdraw(600); // Should show an error

System.out.println("Current Balance: " + acc.getBalance());

}}

OUTPUT:



3.

class Student {

private int rollNo;

private String name;

public int getRollNo() {

return rollNo;}

public void setRollNo(int rollNo) {

this.rollNo = rollNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}}

public class Enc3 {

public static void main(String[] args) {

Student s = new Student();

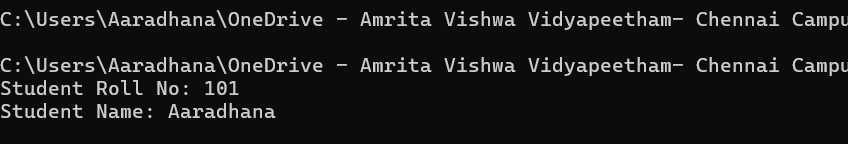
s.setRollNo(101);

s.setName("Aaradhana");

System.out.println("Student Roll No: " + s.getRollNo());

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

OUTPUT:



4.

class Laptop {

private final String brand;

public Laptop(String brand) {

this.brand = brand;

}

public String getBrand() {

return brand;

}

}

public class Enc4 {

public static void main(String[] args) {

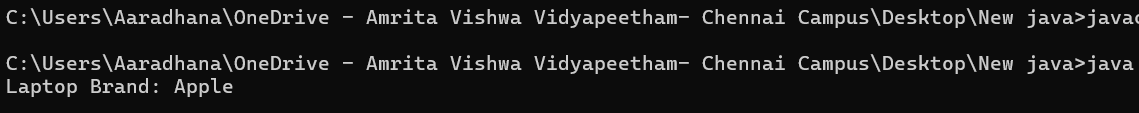
Laptop myLaptop = new Laptop("Apple");

System.out.println("Laptop Brand: " + myLaptop.getBrand());

}

}

OUTPUT:



PACKAGES

1.

import java.util.Scanner;

public class Pkg1 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter your name: ");

String name = input.nextLine();

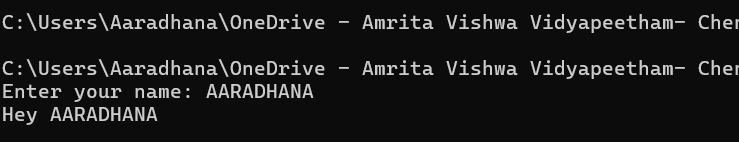
System.out.println("Hey " + name );

input.close();

}

}

OUTPUT:



2.

import java.io.FileWriter;

import java.io.IOException;

public class Pkg2 {

public static void main(String[] args) {

try {

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

writer.write("Java is awesome");

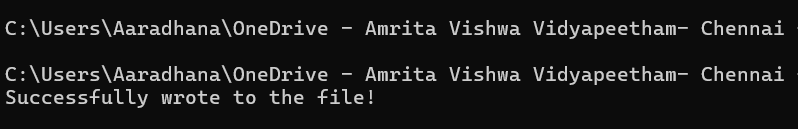
writer.close();

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

} catch (IOException e) {

System.out.println("An error occurred.");}}}

OUTPUT:



3.

public class Pkg3 {

public static void main(String[] args) {

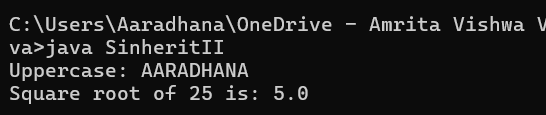
String name = "aaradhana";

System.out.println("Uppercase: " + name.toUpperCase());

double num = 25;

System.out.println("Square root of 25 is: " + Math.sqrt(num));}}

OUTPUT:



4.

import java.util.Random;

public class Pkg4 {

public static void main(String[] args)

{

Random rand = new Random();

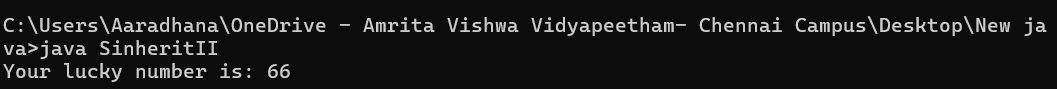
int randomNumber = rand.nextInt(100);

System.out.println("Your lucky number is: " + randomNumber);

}

}

OUTPUT:



EXCEPTION HANDLING

1.

public class Exh1 {

public static void main(String[] args) {

try {

int a = 10, b = 0;

int result = a / b;

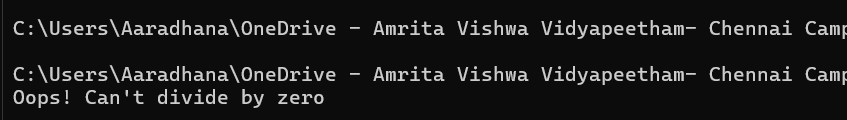
System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Oops! Can't divide by zero ");

}}

OUTPUT:



2.

public class Exh2 {

public static void main(String[] args) {

try {

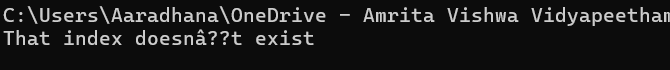
int[] nums = {10, 20, 30};

System.out.println(nums[5]); // Invalid index

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("That index doesn’t exist");}}

OUTPUT:



3.

public class Exh3 {

public static void main(String[] args) {

try {

String str = "hello123";

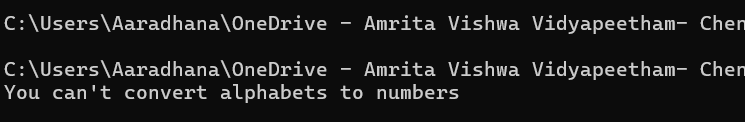
int num = Integer.parseInt(str); // invalid format

System.out.println("Number: " + num);

} catch (NumberFormatException e) {

System.out.println("You can't convert alphabets to numbers");}}

OUTPUT:



4.

public class Exh4 {

public static void main(String[] args) {

try {

int[] arr = new int[3];

System.out.println(10 / 0); // ArithmeticException

System.out.println(arr[4]); // ArrayIndexOutOfBoundsException

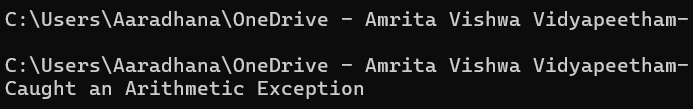
} catch (ArithmeticException e) {

System.out.println("Caught an Arithmetic Exception ");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Caught an Array Index issue");}}}

OUTPUT:

FILE HANDLING

1.

import java.io.FileWriter;

import java.io.IOException;

public class Fl1{

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter("example.txt");

writer.write("This is a file created using Java.");

writer.close();

System.out.println("File written successfully.");

}

catch (IOException e) {

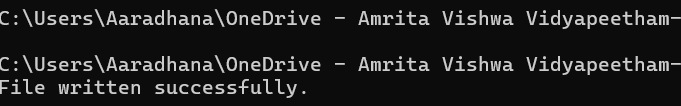
System.out.println("An error occurred.");

}

}

}

OUTPUT:



2.

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class Fl2 {

public static void main(String[] args) {

try {

File file = new File("example.txt");

Scanner reader = new Scanner(file);

while (reader.hasNextLine()) {

String data = reader.nextLine();

System.out.println(data);}

reader.close();

} catch (FileNotFoundException e) {

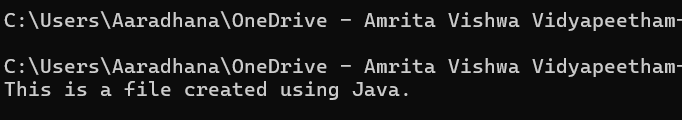
System.out.println("File not found.");

}

}

}

OUTPUT:



3.

import java.io.File;

public class Fl3 {

public static void main(String[] args) {

File file = new File("example.txt");

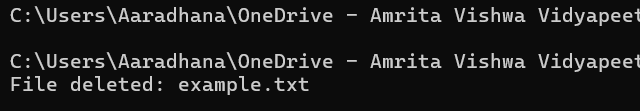
if (file.delete()) {

System.out.println("File deleted: " + file.getName());

} else {

System.out.println("Failed to delete the file.");}}}

OUTPUT:



4.

import java.io.File;

public class Fl4 {

public static void main(String[] args) {

File file = new File("example.txt");

if (file.exists()) {

System.out.println("File name: " + file.getName());

System.out.println("Absolute path: " + file.getAbsolutePath());

System.out.println("Writable: " + file.canWrite());

System.out.println("Readable: " + file.canRead());

System.out.println("File size: " + file.length() + " bytes");

} else {

System.out.println("File does not exist.");}}

OUTPUT:

