LAB RECORD

23CSE111 – OBJECT ORIENTED PROGRAMMING

Submitted by

CH.SC.U4CSE24125 – **M.SHESHANK**

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.U4CSE24125 – M.SHESHANK*** 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.

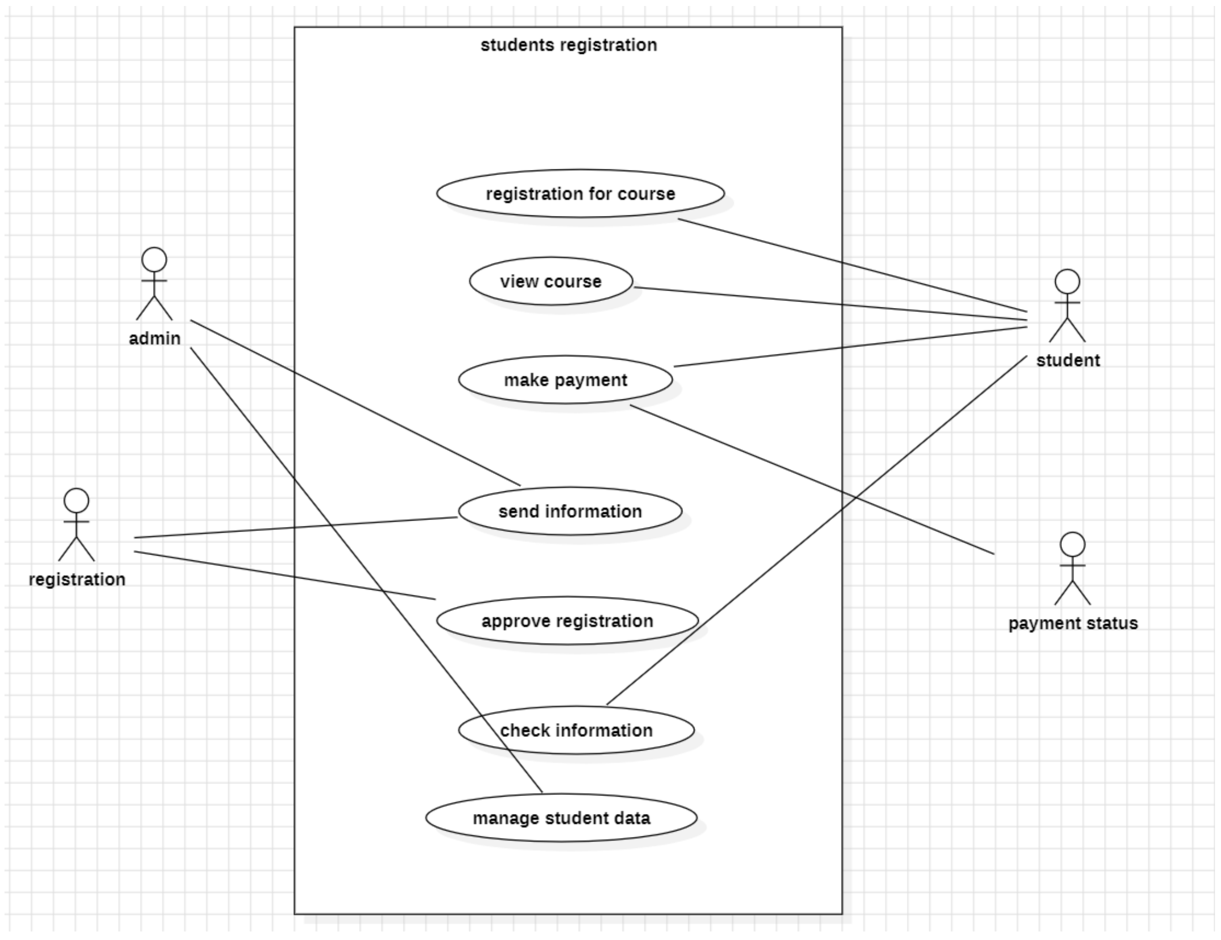
This Lab examination held on 13/03/2025

Internal Examiner 1 Internal Examiner 2

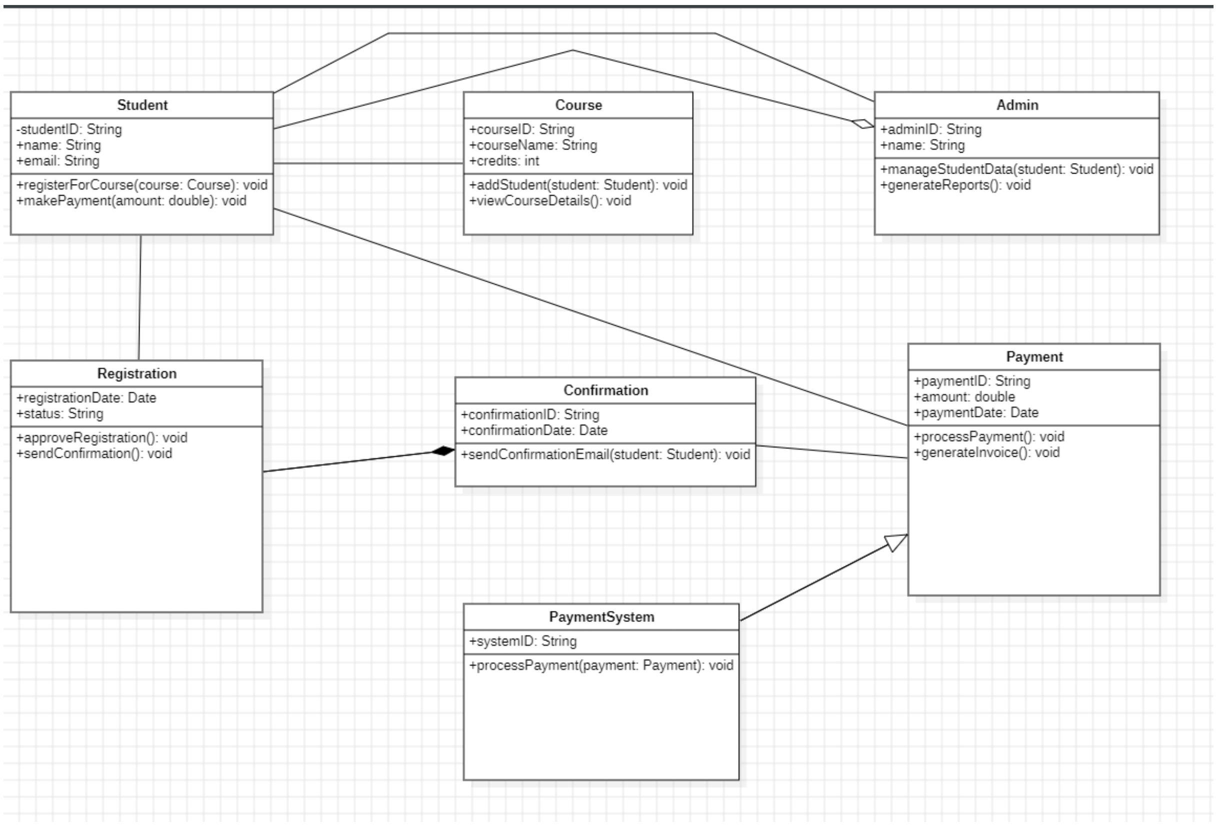
Index

UML DIAGRAMS
STUDENT REGISTRATION

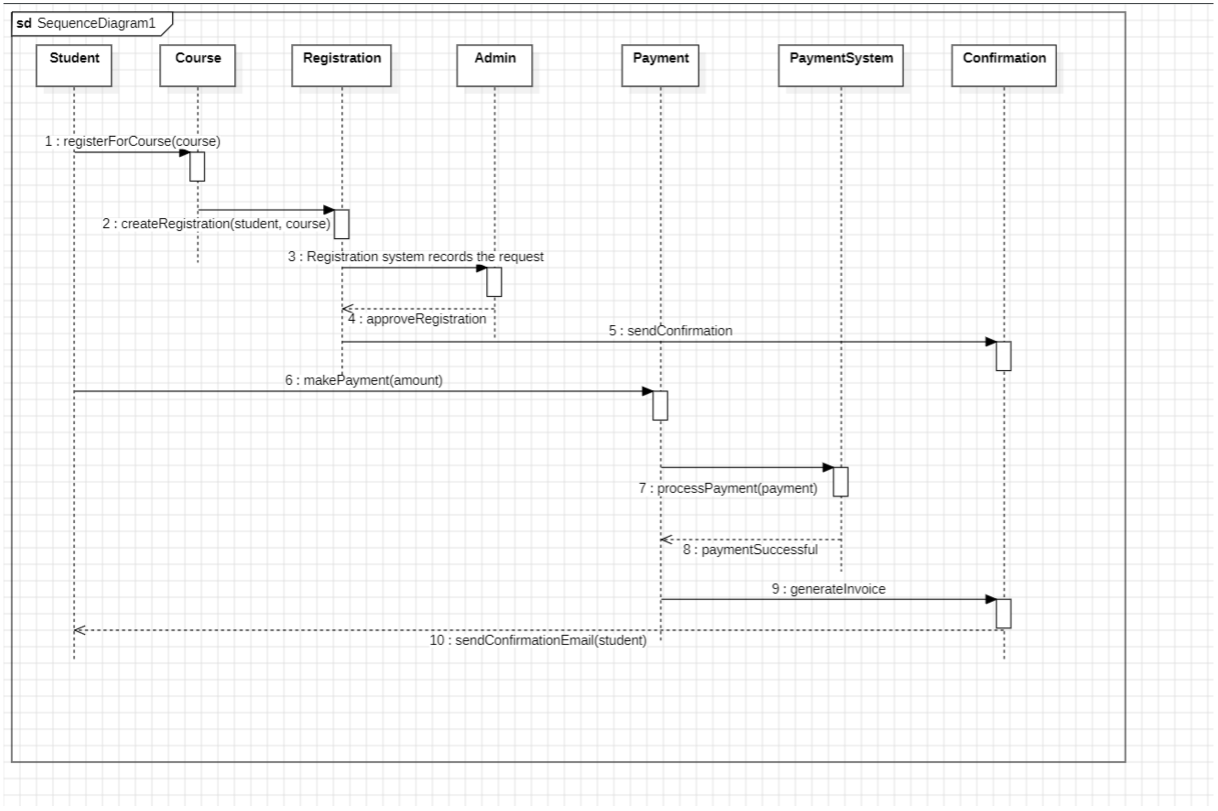
1A.USE CASE DIAGRAM



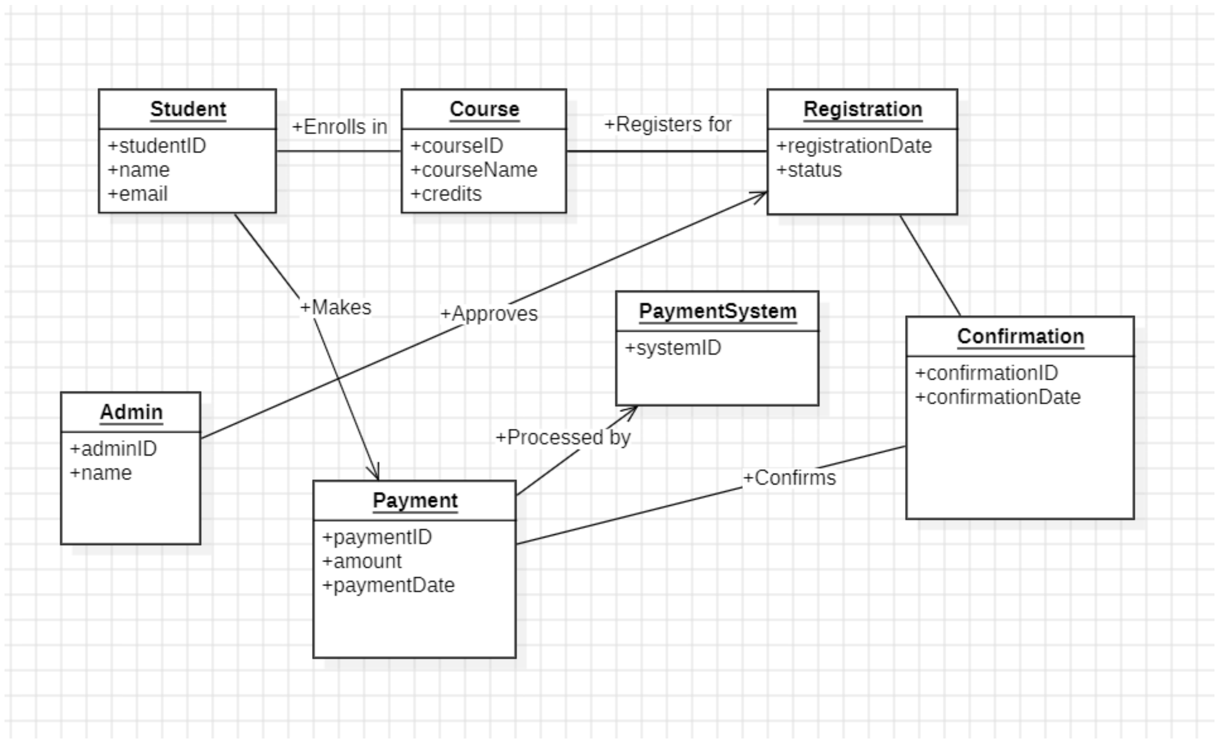
1B.CLASS DIAGRAM



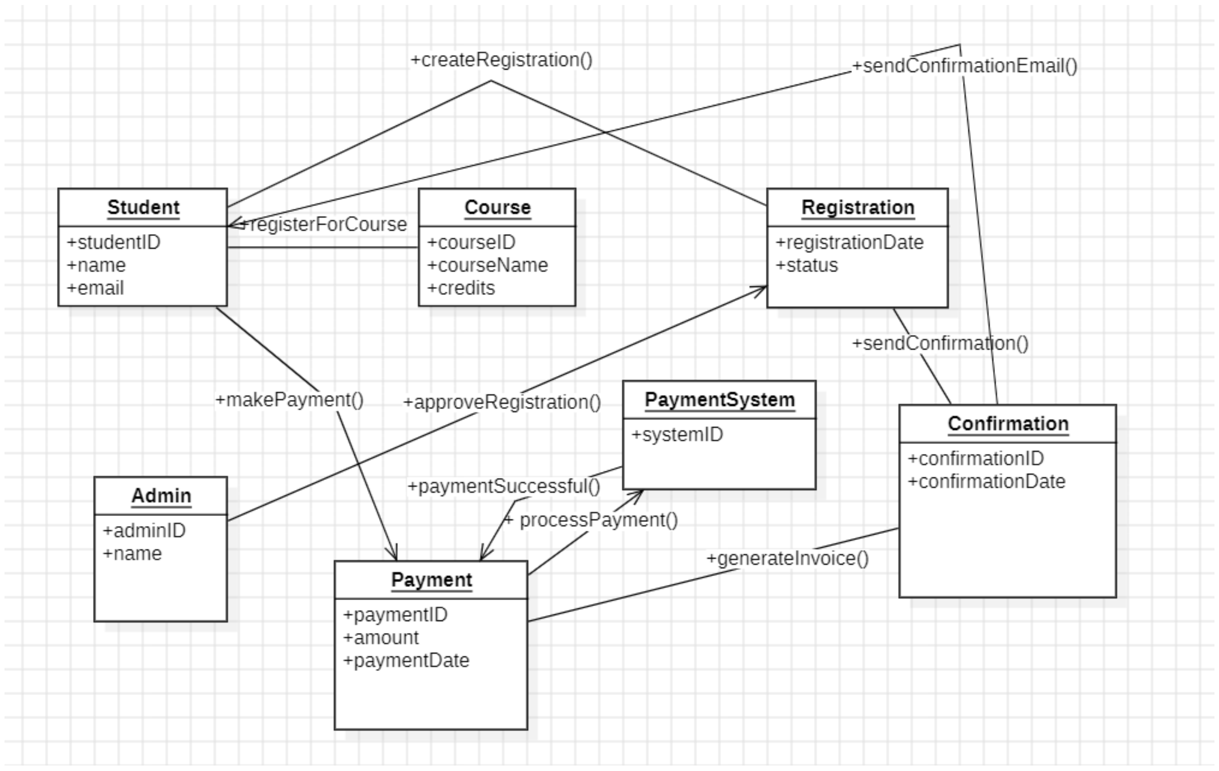
1C.SEQUENCE DIAGRAM:



1D.OBJECT DIAGRAM:

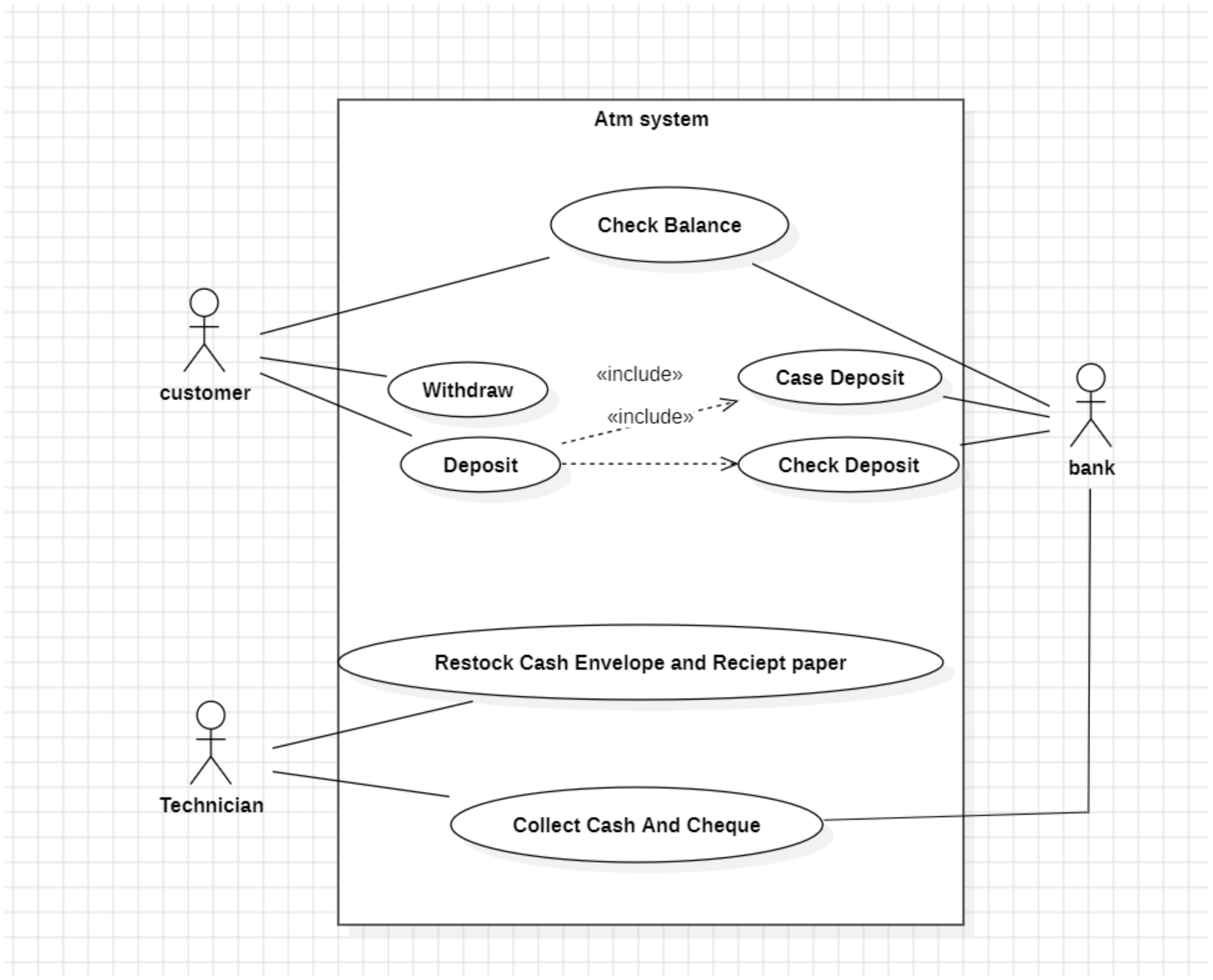


1E.COLLABORATION DIAGRAM:

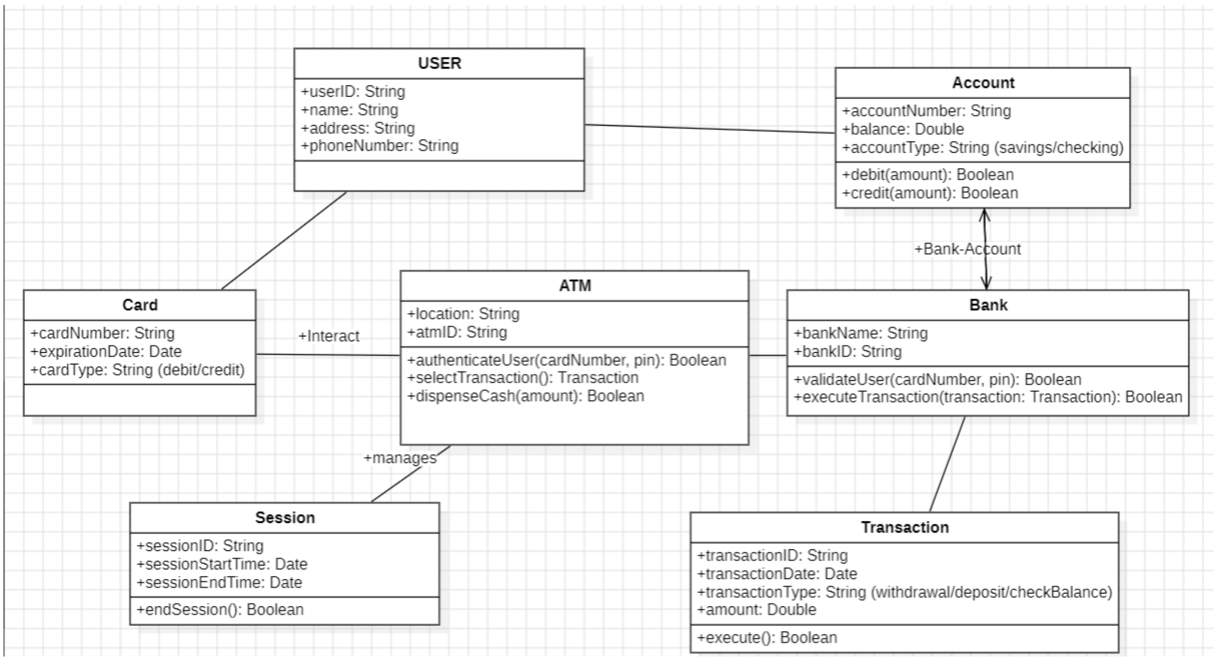


ATM

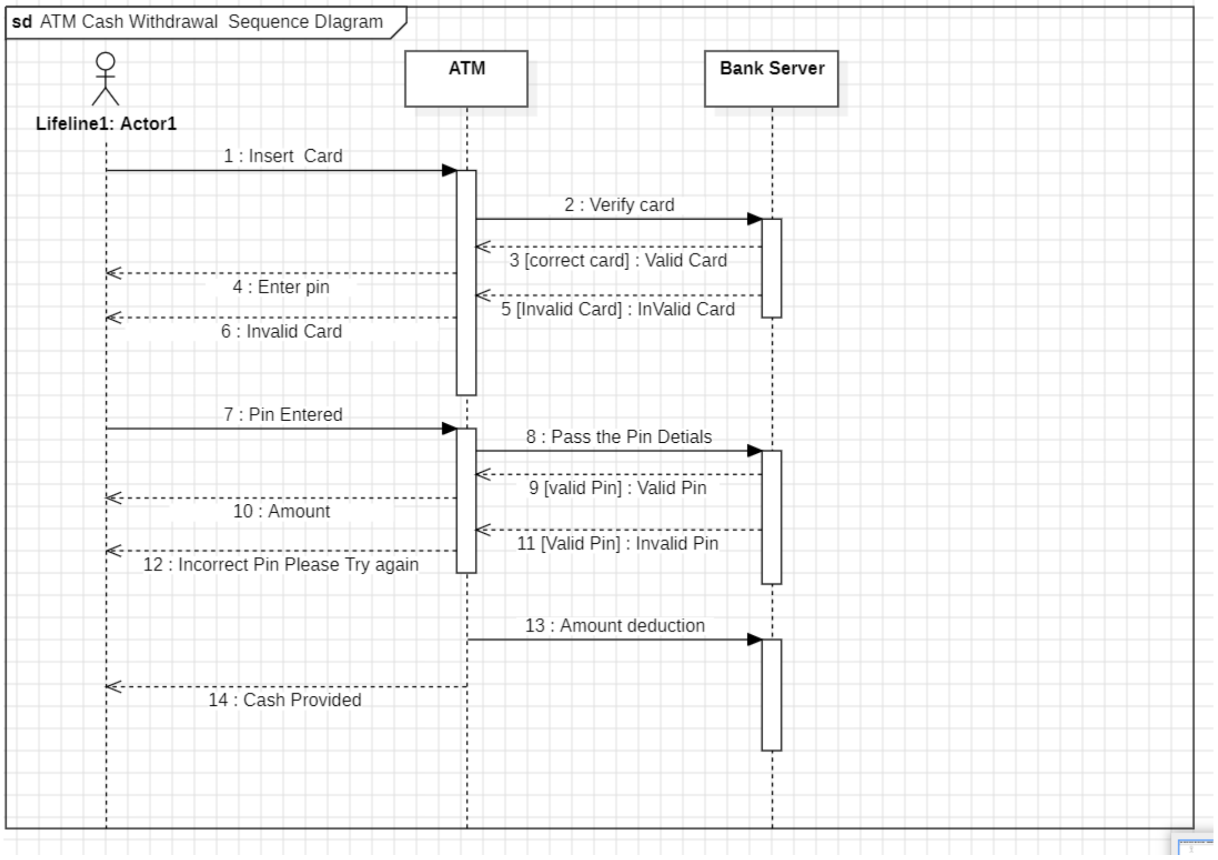
2A.USE CASE DIAGRAM.



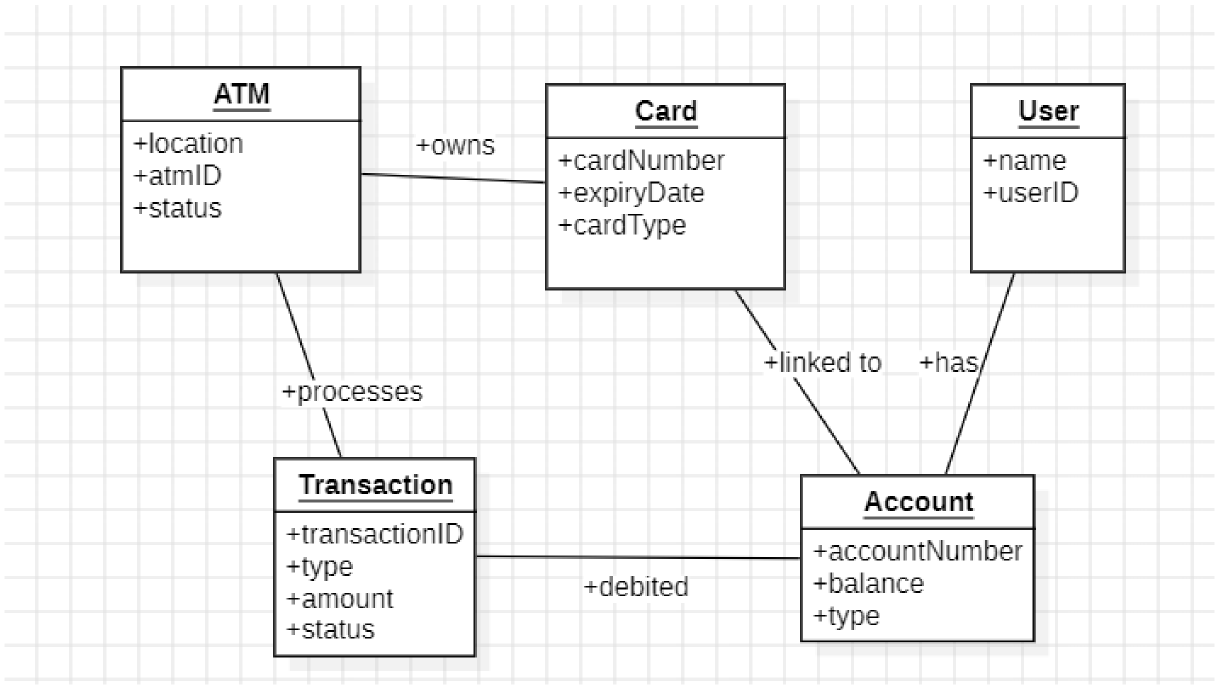
2B.CLASS DIAGRAM.



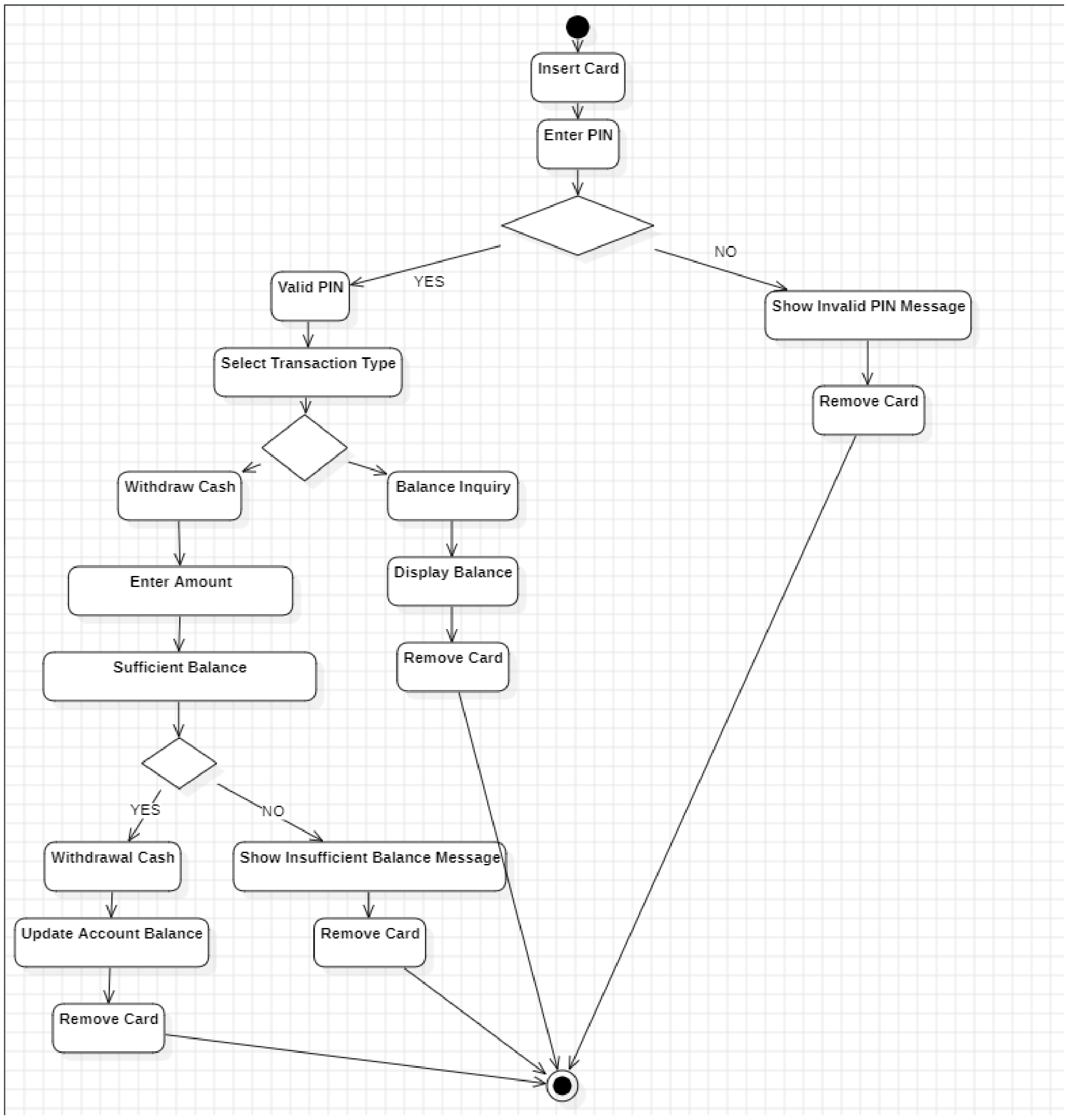
2C.SEQUENCE DIAGRAM.


sd ATM Cash Withdrawal Sequence DIagram
Lifeline1: Actor1
ATM
Bank Server
1 : Insert Card
2 : Verify card
3 [correct card] : Valid Card
4 : Enter pin
5 [Invalid Card] : InValid Card
6 : Invalid Card
7 : Pin Entered
8 : Pass the Pin Detials
9 [valid Pin] : Valid Pin
10 : Amount
11 [Valid Pin] : Invalid Pin
12 : Incorrect Pin Please Try again
13 : Amount deduction
14 : Cash Provided

2D.OBJECT DIAGRAM.



2E.ACTIVITY DIAGRAM.



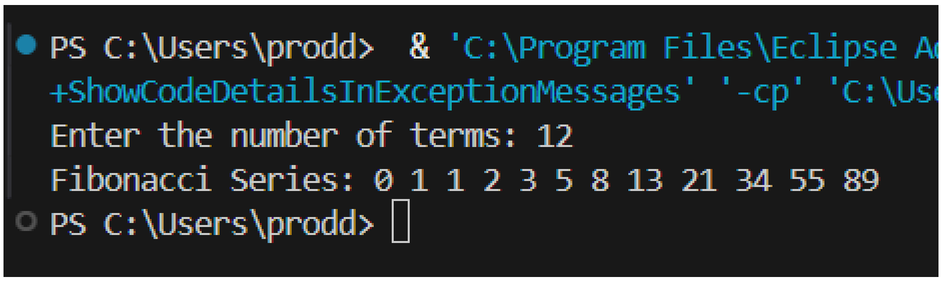
BASIC JAVA PROGRAMS

1)AIM: TO FIND FIBINOCCI SERIES.

CODE:

```
import java.util.Scanner;


public class FibonacciSeries {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of terms: ");
        int n = scanner.nextInt();
        int a = 0, b = 1;
        System.out.print("Fibonacci Series: " + a + " " + b + " ");
        for (int i = 2; i < n; i++) {
            int next = a + b;
            System.out.print(next + " ");
            a = b;
            b = next;
        }
        scanner.close();
    }
}
```

INPUT & OUTPUT:



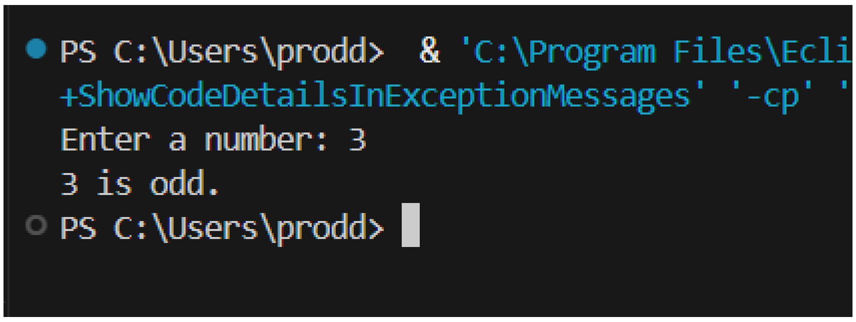
```
PS C:\Users\prodd>  & 'C:\Program Files\Eclipse A
+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Use
Enter the number of terms: 12
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89
PS C:\Users\prodd> 
```

2)AIM:TO FIND THE GIVEN NUMBER IS EVEN OR ODD.

CODE:

```
import java.util.Scanner;
public class EvenOdd {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        if (num % 2 == 0) {
            System.out.println(num + " is even.");
        } else {
            System.out.println(num + " is odd.");
        }
        sc.close();
    }
}
```

INPUT&OUTPUT:



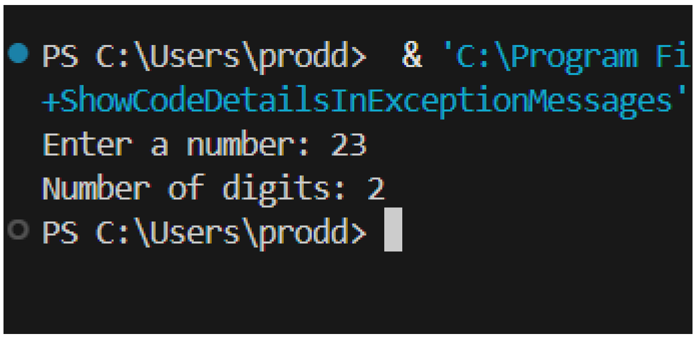
```
● PS C:\Users\prodd>  & 'C:\Program Files\Ecli
  +ShowCodeDetailsInExceptionMessages' '-cp' '
  Enter a number: 3
  3 is odd.
○ PS C:\Users\prodd>
```

3)AIM:TO COUNT THE NUMBER OF DIGITS IN A NUMBER.

CODE:

```
import java.util.Scanner;
public class CountDigits {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        int count = 0;
        while (num != 0) {
            num /= 10;
            count++;
        }
        System.out.println("Number of digits: " + count);
        scanner.close();
    }
}
```

INPUT&OUTPUT:



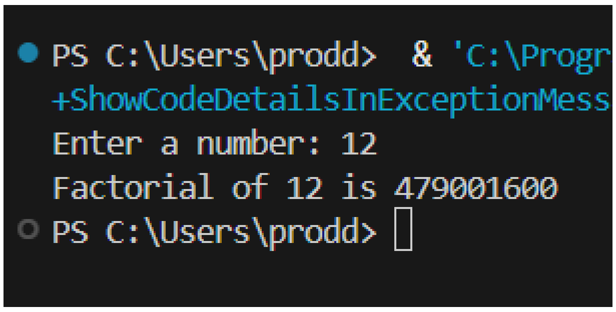
```
PS C:\Users\prodd>  & 'C:\Program Fi
+ShowCodeDetailsInExceptionMessages'
Enter a number: 23
Number of digits: 2
PS C:\Users\prodd>
```

4)AIM:TO FIND THE FACTORIAL OF A NUMBER IN LOOP.

CODE:

```
import java.util.Scanner;
public class FactorialLoop {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        int factorial = 1;
        for (int i = 1; i <= num; i++) {
            factorial *= i;
        }
        System.out.println("Factorial of " + num + " is " + factorial);
        scanner.close();
    }
}
```

INPUT&OUTPUT:



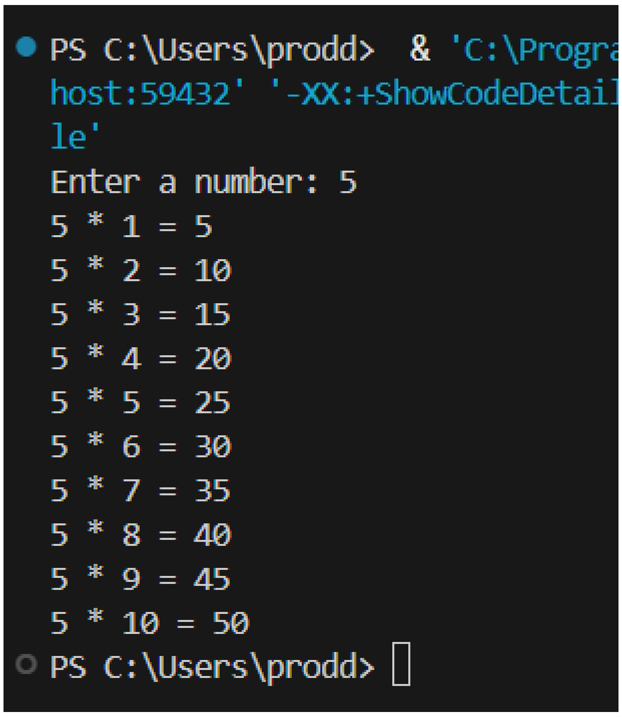
```
PS C:\Users\prodd>  & 'C:\Progr
+ShowCodeDetailsInExceptionMess
Enter a number: 12
Factorial of 12 is 479001600
PS C:\Users\prodd>
```

5)AIM:TO GET A MULTIPLICATION TABLE OF A GIVEN NUMBER.

CODE:

```
import java.util.Scanner;
public class MultiplicationTable {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        int i = 1;
        while (i <= 10) {
            System.out.printf("%d * %d = %d%n", num, i, num * i);
            i++;
        }
        scanner.close();
    }
}
```

INPUT&OUTPUT:



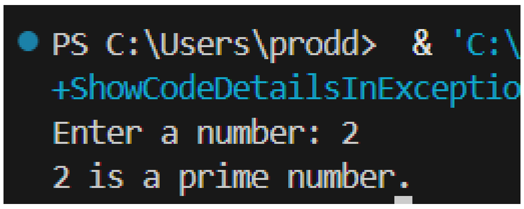
```
PS C:\Users\prodd>  & 'C:\Progra
host:59432' '-XX:+ShowCodeDetail
le'
Enter a number: 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
PS C:\Users\prodd>
```

6)AIM:TO FIND THE GIVEN NUMBER IS PRIME OR NOT.

CODE:

```
import java.util.Scanner;
public class PrimeORNot {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter a number: ");
      int num = scanner.nextInt();
      boolean flag = false;
      if (num < 2) {
         flag = true;
      } else {
         for (int i = 2; i <= num / 2; ++i) {
            if (num % i == 0) {
               flag = true;
               break;
            }
         }
      }
      if (!flag)
         System.out.println(num + " is a prime number.");
      else
         System.out.println(num + " is not a prime number.");
      scanner.close();
   }
}
```

INPUT&OUTPUT:



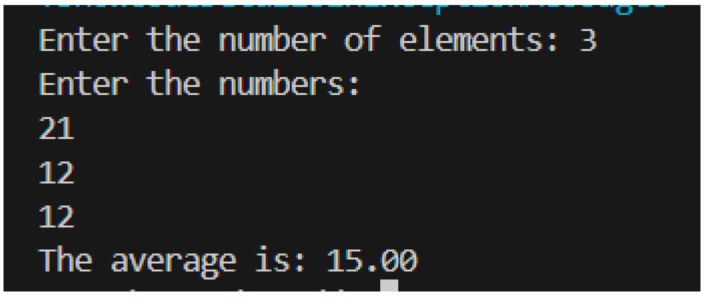
```
PS C:\Users\prodd>  & 'C:\
+ShowCodeDetailsInExceptio
Enter a number: 2
2 is a prime number.
```

7)AIM:TO FIND AVERAGE OF NUMBERS.

CODE:

```
import java.util.Scanner;
public class Average {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();
        double[] numArray = new double[n];
        double sum = 0.0;
        System.out.println("Enter the numbers:");
        for (int i = 0; i < n; i++) {
            numArray[i] = scanner.nextDouble();
            sum += numArray[i];
        }
        double average = sum / n;
        System.out.printf("The average is: %.2f%n", average);
        scanner.close();
    }
}
```

INPUT&OUTPUT:



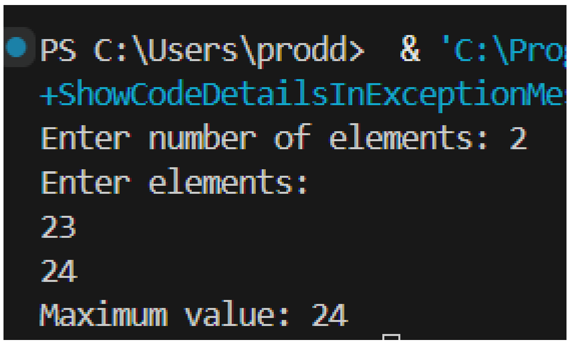
```
Enter the number of elements: 3
Enter the numbers:
21
12
12
The average is: 15.00
```

8)AIM:TO FIND THE LARGEST NUMBER FROM GIVEN NUMBERS.

CODE:

```
import java.util.Scanner;
public class Largest_Number {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter number of elements: ");
int n = scanner.nextInt();
int a[] = new int[n];
System.out.println("Enter elements: ");
for (int i = 0; i < n; i++) {
a[i] = scanner.nextInt();
}
int max = a[0];
for (int i = 0; i < n; i++) {
if (max < a[i]) {
max = a[i];
}
}
System.out.println("Maximum value: " + max);
}
}
```

INPUT&OUTPUT:



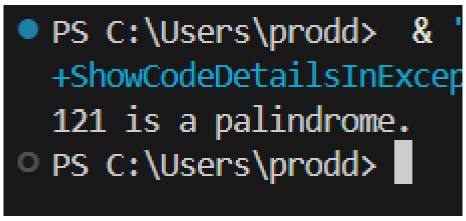
```
PS C:\Users\prodd>  & 'C:\Pro
+ShowCodeDetailsInExceptionMe
Enter number of elements: 2
Enter elements:
23
24
Maximum value: 24
```

9)AIM:TO FIND GIVEN NUMBER IS PALINDROME.

CODE:

```
public class PalindromeCheck {
    public static void main(String[] args) {
        int num = 121; int original = num; int reversed = 0;
        while (num != 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }
        if (original == reversed) {
            System.out.println(original + " is a palindrome.");
        } else {
            System.out.println(original + " is not a palindrome.");
        }
    }
}
```

INPUT&OUTPUT:



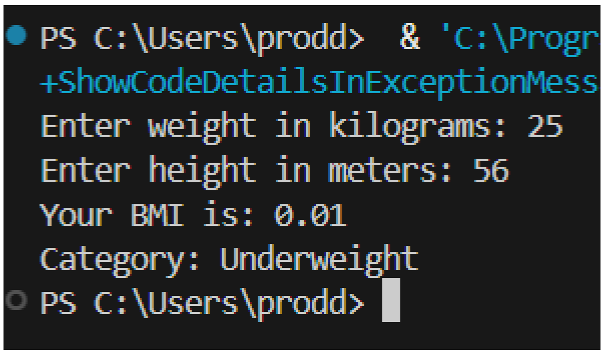
```
PS C:\Users\prodd>  & '
+ShowCodeDetailsInExcep
121 is a palindrome.
PS C:\Users\prodd>
```

10)AIM:TO CALCULATE THE BMI.

CODE:

```
import java.util.Scanner;
public class BMICalculator {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter weight in kilograms: ");
double weight = scanner.nextDouble();
System.out.print("Enter height in meters: ");
double height = scanner.nextDouble();
double bmi = weight / (height * height);
System.out.printf("Your BMI is: %.2f\n", bmi);
if (bmi < 18.5) {
System.out.println("Category: Underweight");
} else if (bmi < 24.9) {
System.out.println("Category: Normal weight");
} else if (bmi < 29.9) {
System.out.println("Category: Overweight");
} else {
System.out.println("Category: Obese");
}
scanner.close();
}
}
```

INPUT&OUTPUT:



```
PS C:\Users\prodd>  & 'C:\Progr
+ShowCodeDetailsInExceptionMess
Enter weight in kilograms: 25
Enter height in meters: 56
Your BMI is: 0.01
Category: Underweight
PS C:\Users\prodd>
```

INHERITANCE

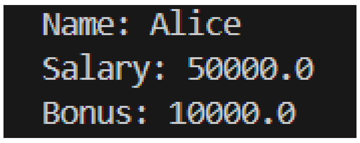
4)SINGLE INHERITANCE:

4a)AIM: Write a Java program where Employee has name and salary, and Manager (child) adds bonus; display all details.

CODE:

```
class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    void display() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}
class Manager extends Employee {
    double bonus;

    Manager(String name, double salary, double bonus) {
        super(name, salary);
        this.bonus = bonus;
    }
    void display() {
        super.display();
        System.out.println("Bonus: " + bonus);
    }
}

public class Main {
    public static void main(String[] args) {
        Manager mgr = new Manager("Alice", 50000, 10000);
        mgr.display();
    }
}
```

OUTPUT:



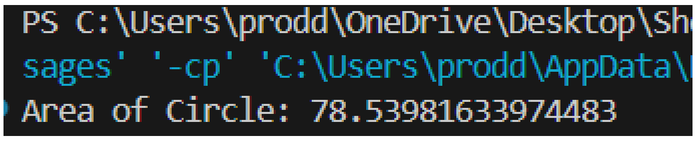
```
Name: Alice
Salary: 50000.0
Bonus: 10000.0
```

4b)AIM: Create a Java program using single inheritance where the parent class Shape has a method area(), and the child class Circle calculates and displays the area of a circle using a given radius.

CODE:

```
class Shape {
   void area() {
      System.out.println("Calculating area...");
   }
}
class Circle extends Shape {
   double radius;
   Circle(double radius) {
      this.radius = radius;
   }
   void area() {
      double result = Math.PI * radius * radius;
      System.out.println("Area of Circle: " + result);
   }
}
public class Main {
   public static void main(String[] args) {
      Circle c = new Circle(5);
      c.area();
   }
}
```

OUTPUT:



```
PS C:\Users\prodd\OneDrive\Desktop\Sh
sages' '-cp' 'C:\Users\prodd\AppData\
Area of Circle: 78.53981633974483
```

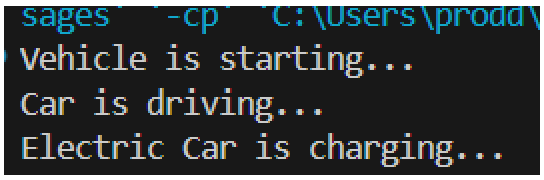
MULTILEVEL INHERITANCE PROGRAMS

5a)AIM: Create a Java program where Vehicle has start(), Car adds drive(), and ElectricCar adds charge(); call all methods using ElectricCar.

CODE:

```
class Vehicle {
   void start() {
      System.out.println("Vehicle is starting...");
   }
}
class Car extends Vehicle {
   void drive() {
      System.out.println("Car is driving...");
   }
}
class ElectricCar extends Car {
   void charge() {
      System.out.println("Electric Car is charging...");
   }
}
public class Main {
   public static void main(String[] args) {
      ElectricCar tesla = new ElectricCar();
      tesla.start();
      tesla.drive();
      tesla.charge();
   }
}
```

OUTPUT:



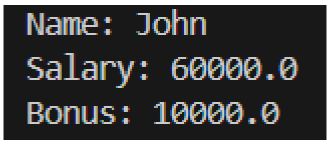
```
sages' '-cp' 'C:\Users\prodd\
Vehicle is starting...
Car is driving...
Electric Car is charging...
```

5b)AIM: Write a Java program where Person has name, Employee adds salary, and Manager adds bonus; display all details using Manager.

CODE:

```
class Person {
    String name;
    Person(String name) {
        this.name = name;
    }
    void display() {
        System.out.println("Name: " + name);
    }
}
class Employee extends Person {
    double salary;
    Employee(String name, double salary) {
        super(name);
        this.salary = salary;
    }
    void display() {
        super.display();
        System.out.println("Salary: " + salary);
    }
}
class Manager extends Employee {
    double bonus;
    Manager(String name, double salary, double bonus) {
        super(name, salary);
        this.bonus = bonus;
    }
    void display() {
        super.display();
        System.out.println("Bonus: " + bonus);
    }
}
public class Main {
    public static void main(String[] args) {
        Manager mgr = new Manager("John", 60000, 10000);
        mgr.display();
    }
}
```

OUTPUT:



```
Name: John
Salary: 60000.0
Bonus: 10000.0
```

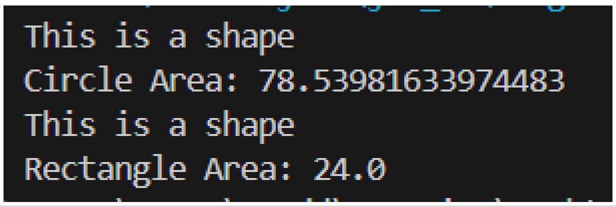
HIERARCHICAL INHERITANCE PROGRAMS

6a)AIM: Create a Java program where Shape has display(), Circle adds area(radius), and Rectangle adds area(length, breadth); call all methods.

CODE:

```
class Shape {
    void display() {
        System.out.println("This is a shape");
    }
}
class Circle extends Shape {
    void area(double radius) {
        System.out.println("Circle Area: " + (Math.PI * radius * radius));
    }
}
class Rectangle extends Shape {
    void area(double length, double breadth) {
        System.out.println("Rectangle Area: " + (length * breadth));
    }
}
public class Main {
    public static void main(String[] args) {
        Circle c = new Circle();
        c.display();
        c.area(5);
        Rectangle r = new Rectangle();
        r.display();
        r.area(4, 6);
    }
}
```

OUTPUT:



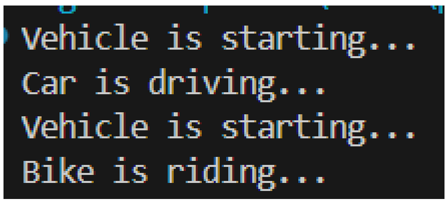
```
This is a shape
Circle Area: 78.53981633974483
This is a shape
Rectangle Area: 24.0
```

6b)AIM: Write a Java program where Vehicle has start(), Car adds drive(), and Bike adds ride(); call all methods using Car and Bike.

CODE:

```
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}
class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving...");
    }
}
class Bike extends Vehicle {
    void ride() {
        System.out.println("Bike is riding...");
    }
}
public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
        car.drive();
        Bike bike = new Bike();
        bike.start();
        bike.ride();
    }
```

OUTPUT:



```
Vehicle is starting...
Car is driving...
Vehicle is starting...
Bike is riding...
```

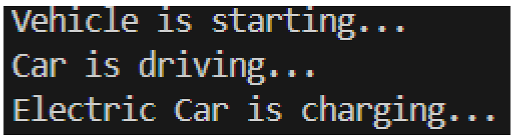
HYBRID INHERITANCE PROGRAMS

7a)AIM: Create a Java program where Vehicle has start(), Car adds drive(), and ElectricCar (using Electric interface) adds charge(); call all methods.

CODE:

```
class Vehicle {
    void start() {
        System.out.println("Vehicle is starting...");
    }
}
interface Electric {
    void charge();
}
class Car extends Vehicle {
    void drive() {
        System.out.println("Car is driving...");
    }
}
class ElectricCar extends Car implements Electric {
    public void charge() {
        System.out.println("Electric Car is charging...");
    }
}
public class Main {
    public static void main(String[] args) {
        ElectricCar tesla = new ElectricCar();
        tesla.start();
        tesla.drive();
        tesla.charge();
    }
}
```

OUTPUT:



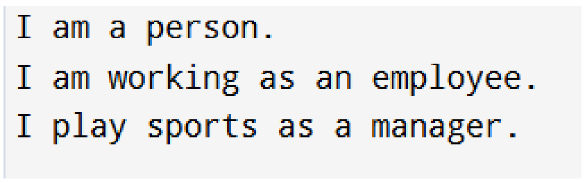
```
Vehicle is starting...
Car is driving...
Electric Car is charging...
```

7b)AIM: Write a Java program where Person has display(), Employee adds work(), and Manager (using Sports interface) adds play(); call all methods.

CODE:

```
interface Sports {
    void play();
}
class Person {
    void display() {
        System.out.println("I am a person.");
    }
}
class Employee extends Person {
    void work() {
        System.out.println("I am working as an employee.");
    }
}
class Manager extends Employee implements Sports {
    public void play() {
        System.out.println("I play sports as a manager.");
    }
}
public class Main {
    public static void main(String[] args) {
        Manager manager = new Manager();
        manager.display();
        manager.work();
        manager.play();
    }
}
```

OUTPUT:



```
I am a person.
I am working as an employee.
I play sports as a manager.
```

POLYMORPHISM

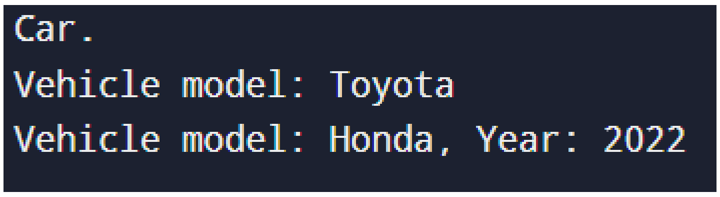
8) CONSTRUCTOR PROGRAM.

8a)AIM: Create a class Vehicle with multiple constructors.

CODE:

```
class Vehicle {
    Vehicle() {
        System.out.println("Car.");
    }
    Vehicle(String model) {
        System.out.println("Vehicle model: " + model);
    }
    Vehicle(String model, int year) {
        System.out.println("Vehicle model: " + model + ", Year: " + year);
    }
}
public class vehicle {
    public static void main(String[] args) {
        Vehicle v1 = new Vehicle();
        Vehicle v2 = new Vehicle("Toyota");
        Vehicle v3 = new Vehicle("Honda", 2022);
    }
}
```

Output:



```
Car.
Vehicle model: Toyota
Vehicle model: Honda, Year: 2022
```

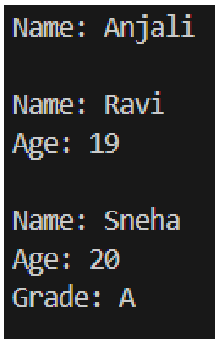
9) CONSTRUCTOR OVERLOADING PROGRAMS:

9a)AIM: Create a class Student with overloaded constructors to initialize only name, name and age, and name, age, and grade.

CODE:

```
class Student {
    String name;
    int age;
    char grade;
    Student(String name) {
        this.name = name;
    }
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    Student(String name, int age, char grade) {
        this.name = name;
        this.age = age;
        this.grade = grade;
    }
    void display() {
        System.out.println("Name: " + name);
        if (age != 0)
            System.out.println("Age: " + age);
        if (grade != '\u0000')
            System.out.println("Grade: " + grade);
        System.out.println();
    }
    public static void main(String[] args) {
        Student s1 = new Student("Anjali");
        Student s2 = new Student("Ravi", 19);
        Student s3 = new Student("Sneha", 20, 'A');
        s1.display();
        s2.display();
        s3.display();
    }
}
```

Screenshot:



```
Name: Anjali

Name: Ravi
Age: 19

Name: Sneha
Age: 20
Grade: A
```

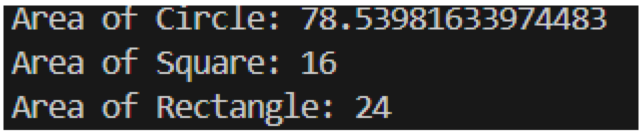
10)METHOD OVERLOADING PROGRAMS

10a)AIM: Calculate Area using Method Overloading.

CODE:

```
class AreaCalculator {
    double area(double radius) {
        return Math.PI * radius * radius;
    }
    int area(int side) {
        return side * side;
    }
    int area(int length, int breadth) {
        return length * breadth;
    }
    public static void main(String[] args) {
        AreaCalculator ac = new AreaCalculator();
        System.out.println("Area of Circle: " + ac.area(5.0));
        System.out.println("Area of Square: " + ac.area(4));
        System.out.println("Area of Rectangle: " + ac.area(4, 6));
    }
}
```

Output:



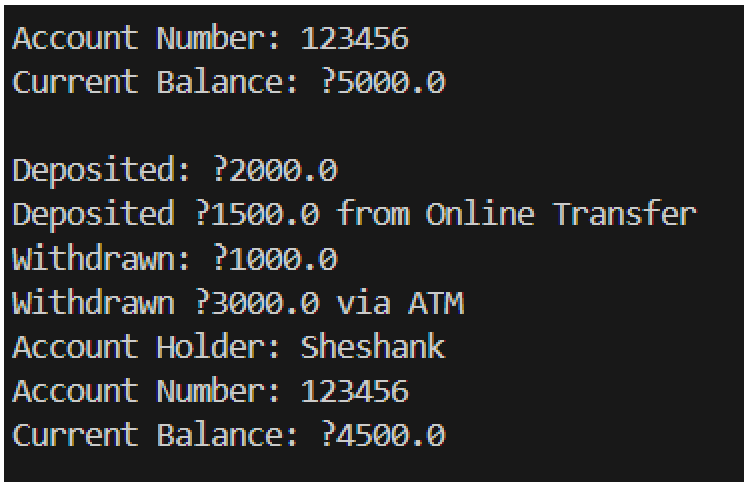
```
Area of Circle: 78.53981633974483
Area of Square: 16
Area of Rectangle: 24
```

10b)AIM: Java program using method overloading in a Banking.

CODE:

```
class BankAccount {
    String accountHolder;
    int accountNumber;
    double balance;

    BankAccount(String name, int accNo, double initialBalance) {
        accountHolder = name;
        accountNumber = accNo;
        balance = initialBalance;
    }

    void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: ₹" + amount);
    }

    void deposit(double amount, String source) {
        balance += amount;
        System.out.println("Deposited ₹" + amount + " from " + source);
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: ₹" + amount);
        } else {
            System.out.println("Insufficient balance!");
        }
    }

    void withdraw(double amount, String mode) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn ₹" + amount + " via " + mode);
        } else {
            System.out.println("Insufficient balance!");
        }
    }
    void showBalance() {
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Current Balance: ₹" + balance);
        System.out.println();
    }

    public static void main(String[] args) {
        BankAccount acc1 = new BankAccount("Sheshank", 123456, 5000.0);

        acc1.showBalance();

        acc1.deposit(2000);
        acc1.deposit(1500, "Online Transfer");

        acc1.withdraw(1000);
        acc1.withdraw(3000, "ATM");
        acc1.showBalance();
    }
}
```

Output:



```
Account Number: 123456
Current Balance: ?5000.0

Deposited: ?2000.0
Deposited ?1500.0 from Online Transfer
Withdrawn: ?1000.0
Withdrawn ?3000.0 via ATM
Account Holder: Sheshank
Account Number: 123456
Current Balance: ?4500.0
```

METHOD OVERRIDING PROGRAMS

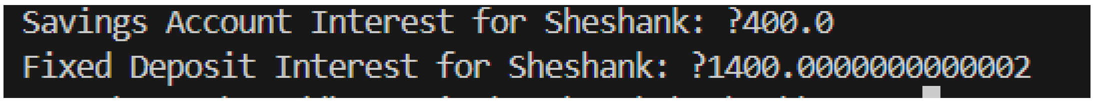
11a)AIM: Bank Account Interest Calculation.

CODE:

```
class BankAccount {
   String accountHolder;
   double balance;

   BankAccount(String name, double balance) {
      this.accountHolder = name;
      this.balance = balance;
   }

   void calculateInterest() {
      System.out.println("Generic account: No specific interest calculation.");
   }
}

class SavingsAccount extends BankAccount {

   SavingsAccount(String name, double balance) {
      super(name, balance);
   }

   @Override
   void calculateInterest() {
      double interest = balance * 0.04;
      System.out.println("Savings Account Interest for " + accountHolder + ": ₹"
+ interest);
   }
}

class FixedDepositAccount extends BankAccount {

   FixedDepositAccount(String name, double balance) {
      super(name, balance);
   }

   @Override
   void calculateInterest() {
      double interest = balance * 0.07;
```

```
        System.out.println("Fixed Deposit Interest for " + accountHolder + ": ₹" +
interest);
    }
}

public class BankTest {
    public static void main(String[] args) {
        BankAccount acc1 = new SavingsAccount("Sheshank", 10000);
        BankAccount acc2 = new FixedDepositAccount("Sheshank", 20000);

        acc1.calculateInterest();
        acc2.calculateInterest();
    }
}
```

Output:



```
Savings Account Interest for Sheshank: ?400.0
Fixed Deposit Interest for Sheshank: ?1400.0000000000002
```

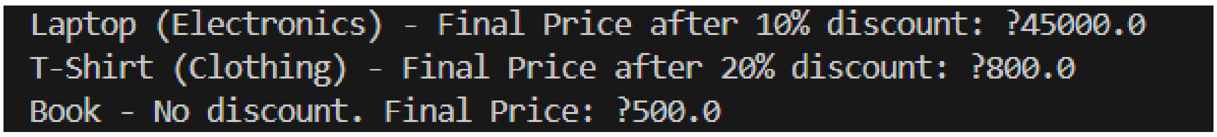
11b)AIM: Shopping cart.

CODE:

```
class Product {
   String name;
   double price;
   Product(String name, double price) {
      this.name = name;
      this.price = price;
   }
   void getDiscountedPrice() {
      System.out.println(name + " - No discount. Final Price: ₹" +
price);
   }
}
class Electronics extends Product {
   Electronics(String name, double price) {
      super(name, price);
   }
   @Override
   void getDiscountedPrice() {
      double discount = 0.10 * price;
      double finalPrice = price - discount;
      System.out.println(name + " (Electronics) - Final Price after 10%
discount: ₹" + finalPrice);
   }
}
class Clothing extends Product {
   Clothing(String name, double price) {
      super(name, price);
   }
   @Override
   void getDiscountedPrice() {
      double discount = 0.20 * price;
      double finalPrice = price - discount;
      System.out.println(name + " (Clothing) - Final Price after 20%
discount: ₹" + finalPrice);
```

```
    }
}
public class ShoppingTest {
    public static void main(String[] args) {
        Product p1 = new Electronics("Laptop", 50000);
        Product p2 = new Clothing("T-Shirt", 1000);
        Product p3 = new Product("Book", 500);
        p1.getDiscountedPrice();
        p2.getDiscountedPrice();
        p3.getDiscountedPrice();
    }
}
```

Output:



```
Laptop (Electronics) - Final Price after 10% discount: ?45000.0
T-Shirt (Clothing) - Final Price after 20% discount: ?800.0
Book - No discount. Final Price: ?500.0
```

ABSTRACTION

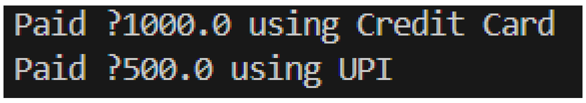
INTERFACE PROGRAMS

12a)AIM: Payment System.

CODE:

```
package Abstraction;
interface Payment {
    void pay(double amount);
}
class CreditCard implements Payment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Credit Card");
    }
}
class UPI implements Payment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using UPI");
    }
}
public class PaymentTest {
    public static void main(String[] args) {
        Payment p1 = new CreditCard();
        Payment p2 = new UPI();
        p1.pay(1000);
        p2.pay(500);
    }
}
```

OUTPUT:



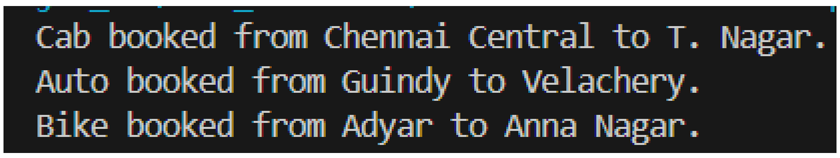
```
Paid ?1000.0 using Credit Card
Paid ?500.0 using UPI
```

12b)AIM: Transport Booking System.

CODE:

```
package Abstraction;
interface Transport {
    void bookRide(String source, String destination);
}
class Cab implements Transport {
    public void bookRide(String source, String destination) {
        System.out.println("Cab booked from " + source + " to " + destination + ".");
    }
}
class Auto implements Transport {
    public void bookRide(String source, String destination) {
        System.out.println("Auto booked from " + source + " to " + destination + ".");
    }
}
class Bike implements Transport {
    public void bookRide(String source, String destination) {
        System.out.println("Bike booked from " + source + " to " + destination + ".");
    }
}
public class TransportBooking {
    public static void main(String[] args) {
        Transport t1 = new Cab();
        Transport t2 = new Auto();
        Transport t3 = new Bike();
        t1.bookRide("Chennai Central", "T. Nagar");
        t2.bookRide("Guindy", "Velachery");
        t3.bookRide("Adyar", "Anna Nagar");
    }
}
```

OUTPUT:



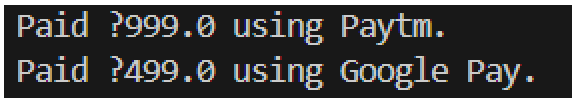
```
Cab booked from Chennai Central to T. Nagar.
Auto booked from Guindy to Velachery.
Bike booked from Adyar to Anna Nagar.
```

12c)AIM: Online Payment System.

CODE:

```
package Abstraction;
interface OnlinePayment {
    void pay(double amount);
}
class Paytm implements OnlinePayment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Paytm.");
    }
}
class GooglePay implements OnlinePayment {
    public void pay(double amount) {
        System.out.println("Paid ₹" + amount + " using Google Pay.");
    }
}
public class PaymentApp {
    public static void main(String[] args) {
        OnlinePayment p1 = new Paytm();
        OnlinePayment p2 = new GooglePay();
        p1.pay(999);
        p2.pay(499);
    }
}
```

OUTPUT:



```
Paid ?999.0 using Paytm.
Paid ?499.0 using Google Pay.
```

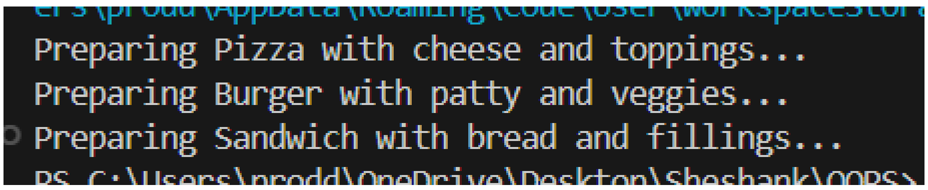
12d)AIM: Restaurant Menu.

CODE:

```
package Abstraction;

interface FoodItem {
   void prepare();
}

class Pizza implements FoodItem {
   public void prepare() {
      System.out.println("Preparing Pizza with cheese and
toppings...");
   }
}

class Burger implements FoodItem {
   public void prepare() {
      System.out.println("Preparing Burger with patty and veggies...");
   }
}

class Sandwich implements FoodItem {
   public void prepare() {
      System.out.println("Preparing Sandwich with bread and
fillings...");
   }
}

public class RestaurantOrder {
   public static void main(String[] args) {
      FoodItem order1 = new Pizza();
      FoodItem order2 = new Burger();
      FoodItem order3 = new Sandwich();

      order1.prepare();
      order2.prepare();
```

```
        order3.prepare();
    }
}
```

OUTPUT:



```
Preparing Pizza with cheese and toppings...
Preparing Burger with patty and veggies...
Preparing Sandwich with bread and fillings...
```

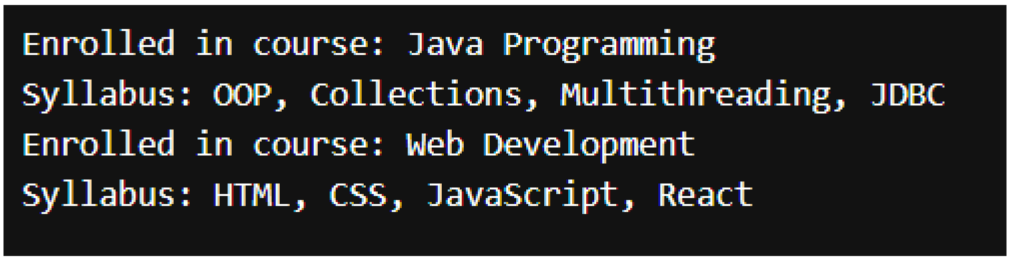
ABSTRACT CLASS PROGRAMS

13a)AIM: Online Courses Platform.

CODE:

```
abstract class Course {
    String title;
    Course(String title) {
        this.title = title;
    }
    abstract void showSyllabus();
    void enroll() {
        System.out.println("Enrolled in course: " + title);
    }
}
class JavaCourse extends Course {
    JavaCourse() {
        super("Java Programming");
    }
    @Override
    void showSyllabus() {
        System.out.println("Syllabus: OOP, Collections, Multithreading,
JDBC");
    }
}
class WebDevCourse extends Course {
    WebDevCourse() {
        super("Web Development");
    }
    @Override
    void showSyllabus() {
        System.out.println("Syllabus: HTML, CSS, JavaScript, React");
    }
}
public class Course_Platform {
    public static void main(String[] args) {
        Course c1 = new JavaCourse();
        Course c2 = new WebDevCourse();
        c1.enroll();
        c1.showSyllabus();
        c2.enroll();
        c2.showSyllabus();
    }
}
```

OUTPUT:



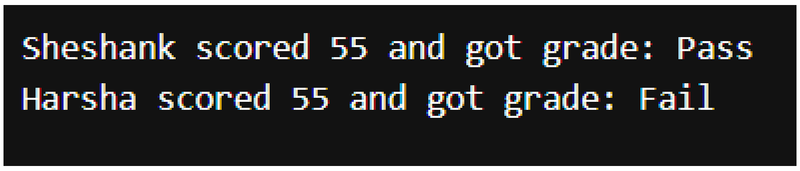
```
Enrolled in course: Java Programming
Syllabus: OOP, Collections, Multithreading, JDBC
Enrolled in course: Web Development
Syllabus: HTML, CSS, JavaScript, React
```

13b)AIM: Student Grading System**.**

CODE:

```
abstract class Student {
    String name;
    int marks;

    Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }

    abstract String getGrade();

    void display() {
        System.out.println(name + " scored " + marks + " and got grade: 
" + getGrade());
    }
}

class UGStudent extends Student {
    UGStudent(String name, int marks) {
        super(name, marks);
    }

    @Override
    String getGrade() {
        return marks >= 50 ? "Pass" : "Fail";
    }
}

class PGStudent extends Student {
    PGStudent(String name, int marks) {
        super(name, marks);
    }

    @Override
    String getGrade() {
        return marks >= 60 ? "Pass" : "Fail";
    }
}

public class GradeSystem {
    public static void main(String[] args) {
        Student ug = new UGStudent("Sheshank", 55);
        Student pg = new PGStudent("Harsha", 55);

        ug.display();
        pg.display();
    }
}
```

OUTPUT:



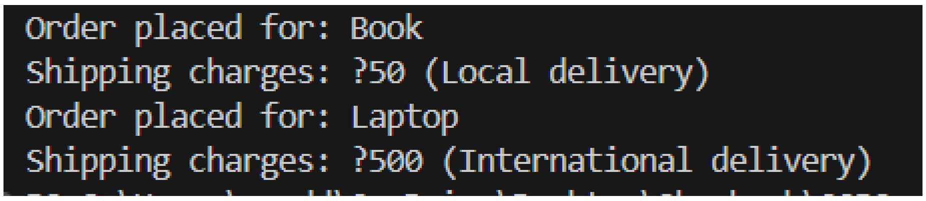
```
Sheshank scored 55 and got grade: Pass
Harsha scored 55 and got grade: Fail
```

13c)AIM: E-Commerce Shipping.

CODE:

```
package Abstraction;

abstract class Order {
    String product;

    Order(String product) {
        this.product = product;
    }

    abstract void calculateShipping();

    void confirmOrder() {
        System.out.println("Order placed for: " + product);
    }
}

class LocalOrder extends Order {
    LocalOrder(String product) {
        super(product);
    }

    @Override
    void calculateShipping() {
        System.out.println("Shipping charges: ₹50 (Local delivery)");
    }
}

class InternationalOrder extends Order {
    InternationalOrder(String product) {
        super(product);
    }

    @Override
    void calculateShipping() {
        System.out.println("Shipping charges: ₹500 (International
delivery)");
    }
}

public class OrderSystem {
    public static void main(String[] args) {
        Order o1 = new LocalOrder("Book");
        Order o2 = new InternationalOrder("Laptop");

        o1.confirmOrder();
        o1.calculateShipping();

        o2.confirmOrder();
        o2.calculateShipping();
    }
}
```

OUTPUT:



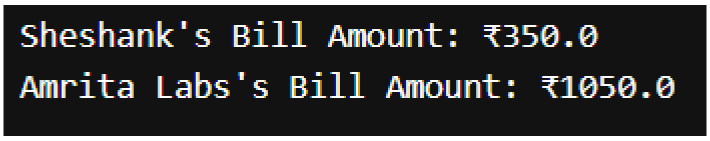
```
Order placed for: Book
Shipping charges: ?50 (Local delivery)
Order placed for: Laptop
Shipping charges: ?500 (International delivery)
```

13d)AIM: Electricity Billing System.

CODE:

```
abstract class ElectricityBill {
    String customerName;
    int units;

    ElectricityBill(String customerName, int units) {
        this.customerName = customerName;
        this.units = units;
    }

    abstract double calculateBill();

    void printBill() {
        System.out.println(customerName + "'s Bill Amount: ₹" +
calculateBill());
    }
}

class DomesticConnection extends ElectricityBill {
    DomesticConnection(String customerName, int units) {
        super(customerName, units);
    }

    @Override
    double calculateBill() {
        return units * 3.5;
    }
}

class CommercialConnection extends ElectricityBill {
    CommercialConnection(String customerName, int units) {
        super(customerName, units);
    }

    @Override
    double calculateBill() {
        return units * 7.0;
    }
}

public class ElectricityMain {
    public static void main(String[] args) {
        ElectricityBill d = new DomesticConnection("Sheshank", 100);
        ElectricityBill c = new CommercialConnection("Amrita Labs",
150);

        d.printBill();
        c.printBill();
    }
}
```

OUTPUT:



```
Sheshank's Bill Amount: ₹350.0
Amrita Labs's Bill Amount: ₹1050.0
```

ENCAPSULATION

ENCAPSULATION PROGRAMS

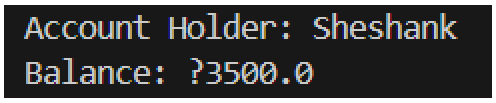
14a)AIM: Bank Account.

CODE:

```
package Encapsulation;

class BankAccount {
    private String accountHolder;
    private double balance;

    public void setAccountHolder(String name) {
        this.accountHolder = name;
    }

    public String getAccountHolder() {
        return accountHolder;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0)
            balance += amount;
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance)
```

```
            balance -= amount;
        else
            System.out.println("Insufficient funds or invalid amount.");
    }
}

public class BankTest {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount();
        acc.setAccountHolder("Sheshank");
        acc.deposit(5000);
        acc.withdraw(1500);

        System.out.println("Account Holder: " + acc.getAccountHolder());
        System.out.println("Balance: ₹" + acc.getBalance());
    }
}
```

OUTPUT:



```
Account Holder: Sheshank
Balance: ?3500.0
```

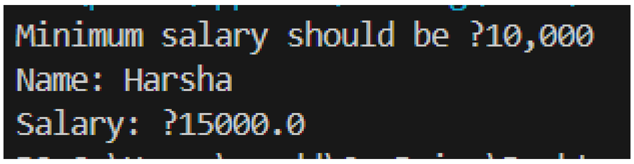
14b)AIM: Employee Salary – Controlled Access

CODE:

```
package Encapsulation;

class Employee {
   private String name;
   private double salary;

   public void setName(String name) {
      this.name = name;
   }

   public void setSalary(double salary) {
      if (salary >= 10000) {
         this.salary = salary;
      } else {
         System.out.println("Minimum salary should be ₹10,000");
      }
   }

   public String getName() {
      return name;
   }

   public double getSalary() {
      return salary;
```

```
    }
}

public class EmployeeTest {
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setName("Harsha");
        emp.setSalary(9000);

        emp.setSalary(15000);

        System.out.println("Name: " + emp.getName());
        System.out.println("Salary: ₹" + emp.getSalary());
    }
}
```

OUTPUT:



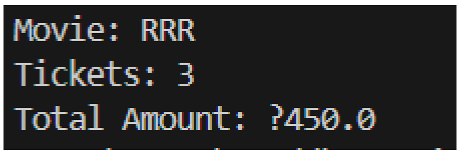
```
Minimum salary should be ?10,000
Name: Harsha
Salary: ?15000.0
```

14c)AIM: Movie Ticket Booking.

CODE:

```
package Encapsulation;
class MovieTicket {
    private String movieName;
    private int numberOfTickets;
    private double ticketPrice;
    public void setMovieName(String name) {
        this.movieName = name;
    }
    public void setNumberOfTickets(int count) {
        if (count > 0) {
            this.numberOfTickets = count;
        } else {
            System.out.println("Ticket count must be positive.");
        }
    }
    public void setTicketPrice(double price) {
        if (price > 0) {
            this.ticketPrice = price;
        }
    }

    public double getTotalAmount() {
        return numberOfTickets * ticketPrice;
    }
    public void printBill() {
        System.out.println("Movie: " + movieName);
        System.out.println("Tickets: " + numberOfTickets);
        System.out.println("Total Amount: ₹" + getTotalAmount());
    }
}
public class MovieTest {
    public static void main(String[] args) {
        MovieTicket ticket = new MovieTicket();
        ticket.setMovieName("RRR");
        ticket.setNumberOfTickets(3);
        ticket.setTicketPrice(150);
        ticket.printBill();
    }
}
```

OUTPUT:



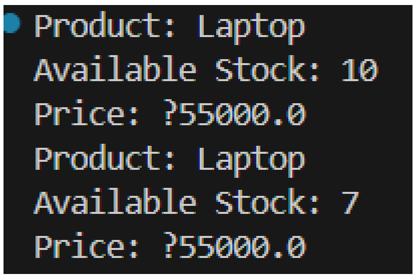
```
Movie: RRR
Tickets: 3
Total Amount: ?450.0
```

14d)AIM: Product Inventory System.

CODE:

```
package Encapsulation;
class Product {
    private String name;
    private int stock;
    private double price;
    public void setName(String name) {
        this.name = name;
    }
    public void setStock(int stock) {
        this.stock = stock;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public void updateStock(int sold) {
        if (sold <= stock) {
            stock -= sold;
        } else {
            System.out.println("Not enough stock available.");
        }
    }
    public void displayProduct() {
        System.out.println("Product: " + name);
        System.out.println("Available Stock: " + stock);
        System.out.println("Price: ₹" + price);
    }
}
public class InventoryApp {
    public static void main(String[] args) {
        Product p = new Product();
        p.setName("Laptop");
        p.setStock(10);
        p.setPrice(55000);
        p.displayProduct();
        p.updateStock(3);  // 3 sold
        p.displayProduct();
    }
}
```

OUTPUT:



```
Product: Laptop
Available Stock: 10
Price: ?55000.0
Product: Laptop
Available Stock: 7
Price: ?55000.0
```

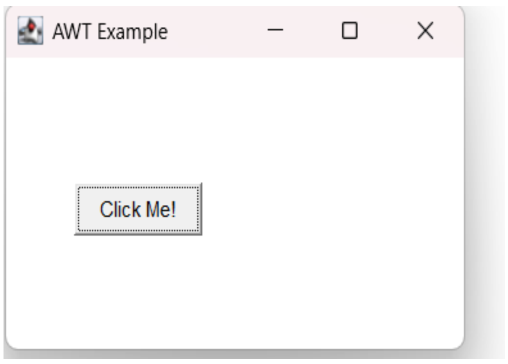
PACKAGES

15a)AIM: Simple App

CODE:

```
import java.awt.*;
import java.awt.event.*;
 public class SimpleAWTApp {
    SimpleAWTApp() {
        Frame frame = new Frame("AWT Example");
        Button button = new Button("Click Me!");
        button.setBounds(50, 100, 80, 30);
        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 SimpleAWTApp();
    }
}
```

OUTPUT:

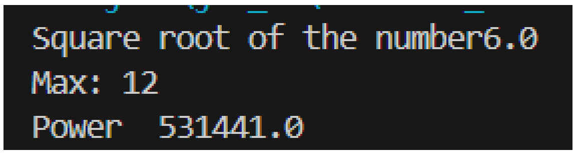


15b)AIM: Math

CODE:

```
import java.lang.*;
public class math {
    public static void main(String[] args) {
        int a=3,b=12;
        System.out.println("Square root of the number" +
Math.sqrt(a*b));
        System.out.println("Max: " + Math.max(a, b));
        System.out.println("Power  " + Math.pow(a, b));
    }


}
```

OUTPUT:



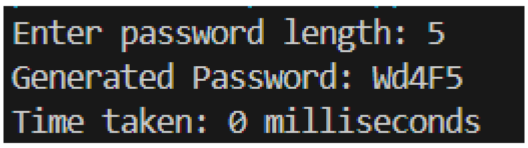
```
Square root of the number6.0
Max: 12
Power  531441.0
```

15c)AIM: Password Generator

CODE:

```
import java.time.Duration;
import java.time.Instant;
import java.util.Random;
import java.util.Scanner;

public class PasswordGenerator {
    public static String generatePassword(int length) {
        String chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw
xyz0123456789@#$%&*!";
        Random random = new Random();
        StringBuilder password = new StringBuilder();

        for (int i = 0; i < length; i++) {
            int index = random.nextInt(chars.length());
            password.append(chars.charAt(index));
        }

        return password.toString();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter password length: ");
        int length = scanner.nextInt();

        Instant start = Instant.now();

        String password = generatePassword(length);

        Instant end = Instant.now();
        Duration timeElapsed = Duration.between(start, end);

        System.out.println("Generated Password: " + password);
        System.out.println("Time taken: " + timeElapsed.toMillis() + "
milliseconds");
        scanner.close();
    }
}
```

OUTPUT:



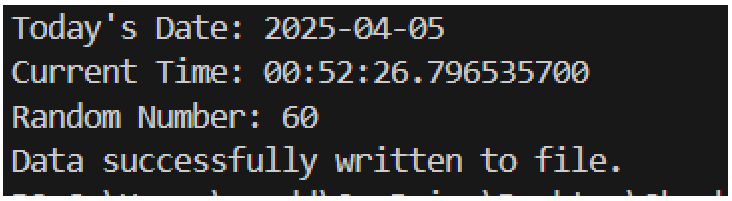
```
Enter password length: 5
Generated Password: Wd4F5
Time taken: 0 milliseconds
```

15d)AIM: Date and time generator.

CODE:

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.Random;
import java.util.Date;
import java.io.FileWriter;
import java.io.IOException;

public class Builtin {
    public static void main(String[] args) {
        LocalDate today = LocalDate.now();
        LocalTime now = LocalTime.now();
        System.out.println("Today's Date: " + today);
        System.out.println("Current Time: " + now);

        Random random = new Random();
        int randNumber = random.nextInt(100);
        System.out.println("Random Number: " + randNumber);

        // File
        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Date: " + today + "\n");
            writer.write("Time: " + now + "\n");
            writer.write("Random Number: " + randNumber + "\n");
            writer.close();
            System.out.println("Data successfully written to file.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



```
Today's Date: 2025-04-05
Current Time: 00:52:26.796535700
Random Number: 60
Data successfully written to file.
```

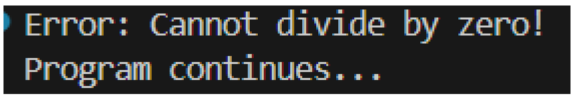
EXCEPTION HANDLING PROGRAMS

16a)AIM: DivideByZero.

CODE:

```
public class DivideByZero {
    public static void main(String[] args) {
        try {
            int num1 = 10, num2 = 0;
            int result = num1 / num2;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }
        System.out.println("Program continues...");
    }
}
```

OUTPUT:



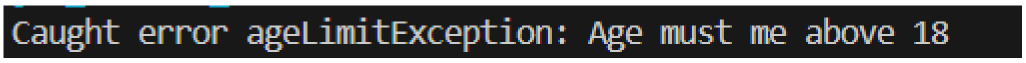
```
Error: Cannot divide by zero!
Program continues...
```

16b)AIM: Age Limit.

CODE:

```
class ageLimitException extends Exception{
ageLimitException(String message){
super(message);
}}
public class main2{
public static void main(String[] args){
try{
ageLimit(17);
}
catch(ageLimitException e){
System.out.println("Caught error "+e);
}
}
public static void ageLimit(int age) throws ageLimitException{
if (age<18){
throw new ageLimitException("Age must me above 18 ");
}
}
}
```

OUTPUT:



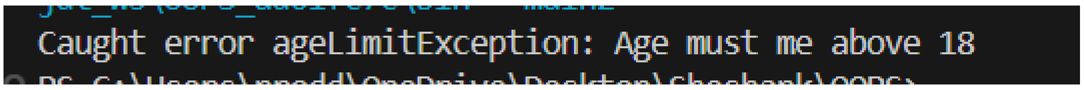
```
Caught error ageLimitException: Age must me above 18
```

16c)AIM: Eligibility of vote.

CODE:

```
public class main3{
public static void main(String[] args){
try{
ageLimit(11);
}
catch(IllegalArgumentException e){
System.out.println("Caught Error "+e);
}


}
public static void ageLimit(int age){
if (age<18){
throw new IllegalArgumentException("Not elligible to vote");
}
System.out.println("Successfully accessed");
}
}
```

OUTPUT:



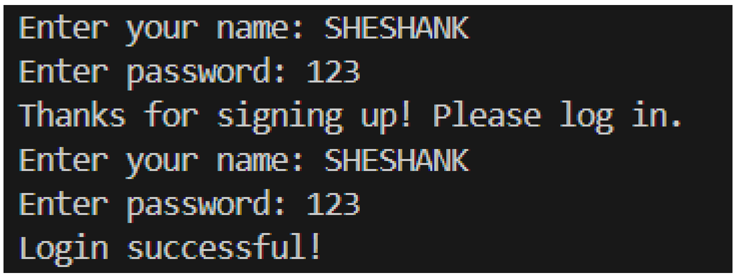
```
Caught error ageLimitException: Age must me above 18
```

16d)AIM: User login

CODE:

```
import java.util.*;
public class user {
    public static void main(String[] args) {
        Scanner n = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String nam = n.nextLine();
        System.out.print("Enter password: ");
        String pass = n.nextLine();
        Login p = new Login();
        p.signup(nam, pass);
    }
}
class SyntaxException extends Exception {
    SyntaxException(String message) {
        super(message);
    }
}
class Login {
    void signup(String nam, String pass) {
        System.out.println("Thanks for signing up! Please log in.");
        try {
            Scanner nn = new Scanner(System.in);
            System.out.print("Enter your name: ");
            String nm = nn.nextLine();
            System.out.print("Enter password: ");
            String pss = nn.nextLine();
            if (!nm.equals(nam) || !pss.equals(pass)) {
                throw new SyntaxException("Wrong username or
password");
            }
            System.out.println("Login successful!");
        } catch (SyntaxException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
Enter your name: SHESHANK
Enter password: 123
Thanks for signing up! Please log in.
Enter your name: SHESHANK
Enter password: 123
Login successful!
```

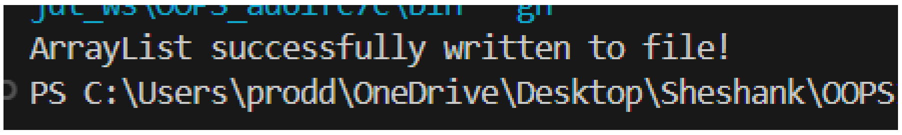
FILE HANDLING PROGRAMS

17a)AIM:Array list.

CODE:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
public class gh {
    public static void main(String[] args) {
        ArrayList<String> names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        names.add("David");
        String filePath = "output.txt";
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            for (String name : names) {
                writer.write(name);
                writer.newLine();
            }
            System.out.println("ArrayList successfully written to file!");
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
ArrayList successfully written to file!
PS C:\Users\prodd\OneDrive\Desktop\Sheshank\OOPS
```

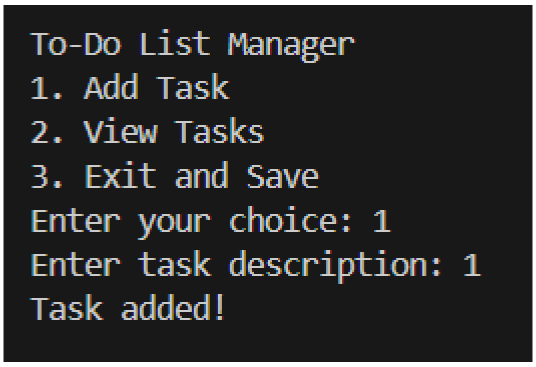
17b)AIM: Task Manager.

CODE:

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class TaskManager {
    private static final String FILE_NAME = "tasks.txt";
    private static List<String> tasks = new ArrayList<>();
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nTo-Do List Manager");
            System.out.println("1. Add Task");
            System.out.println("2. View Tasks");
            System.out.println("3. Exit and Save");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter task description: ");
                    String task = scanner.nextLine();
                    tasks.add(task);
                    System.out.println("Task added!");
                    break;
                case 2:
```

```
                if (tasks.isEmpty()) {
                    System.out.println("No tasks available.");
                } else {
                    System.out.println("\nYour Tasks:");
                    for (int i = 0; i < tasks.size(); i++) {
                        System.out.println((i + 1) + ". " + tasks.get(i));
                    }
                }
                break;
            case 3:
                saveTasks();
                System.out.println("Tasks saved. Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
      }
   }
   private static void loadTasks() {
      try (BufferedReader br = new BufferedReader(new
FileReader(FILE_NAME))) {
         String line;
         while ((line = br.readLine()) != null) {
            tasks.add(line);
         }
      } catch (FileNotFoundException e) {
         System.out.println("No previous tasks found.");
      } catch (IOException e) {
            System.out.println("Error reading tasks: " + e.getMessage());
        }
    }
    private static void saveTasks() {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(FILE_NAME))) {
            for (String task : tasks) {
                bw.write(task);
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving tasks: " + e.getMessage());
        }
    }
}
```

OUTPUT:



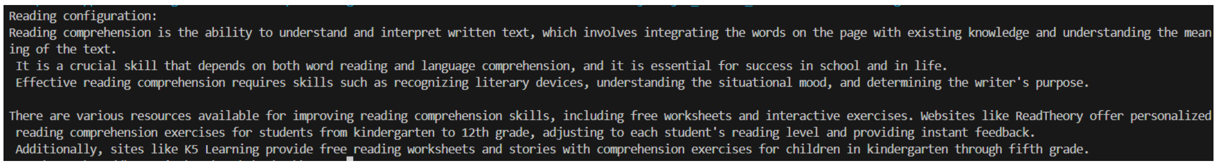
```
To-Do List Manager
1. Add Task
2. View Tasks
3. Exit and Save
Enter your choice: 1
Enter task description: 1
Task added!
```

17c)AIM:Write file

CODE:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class WritingFile {
    public static void main(String[] args) {
        try {
            FileReader fileReader = new
FileReader("C:\\Users\\prodd\\OneDrive\\Desktop\\Sheshank\\OOPS\\File
Handling\\config.txt");
            BufferedReader bufferedReader = new BufferedReader(fileReader);
            String line;
            System.out.println("Reading configuration:");
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
            bufferedReader.close();
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
Reading configuration:
Reading comprehension is the ability to understand and interpret written text, which involves integrating the words on the page with existing knowledge and understanding the mean
ing of the text.
 It is a crucial skill that depends on both word reading and language comprehension, and it is essential for success in school and in life.
 Effective reading comprehension requires skills such as recognizing literary devices, understanding the situational mood, and determining the writer's purpose.

There are various resources available for improving reading comprehension skills, including free worksheets and interactive exercises. Websites like ReadTheory offer personalized
 reading comprehension exercises for students from kindergarten to 12th grade, adjusting to each student's reading level and providing instant feedback.
 Additionally, sites like K5 Learning provide free reading worksheets and stories with comprehension exercises for children in kindergarten through fifth grade.
```

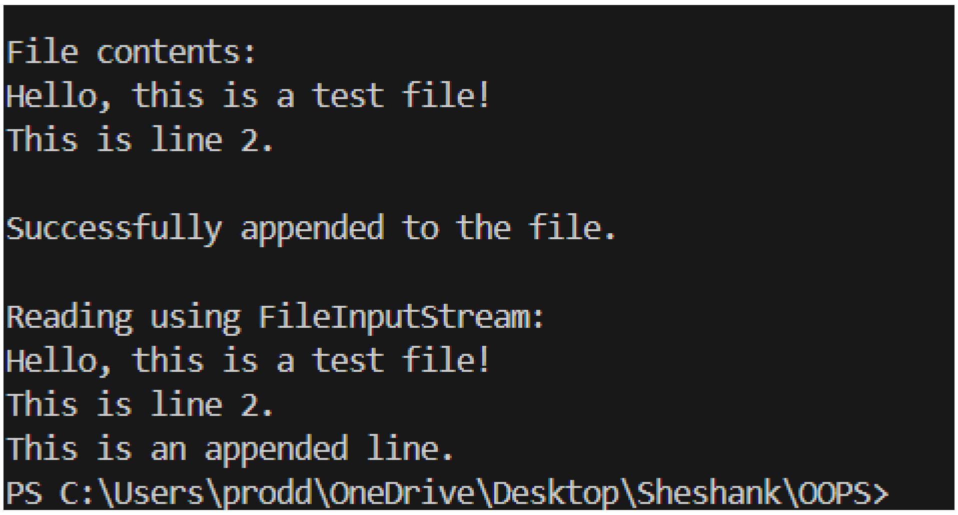
17d)AIM:File writer

CODE:

```
import java.io.*;
public class FileHandlingExample {
   public static void main(String[] args) {
      String fileName = "example.txt";
      try (FileWriter writer = new FileWriter(fileName)) {
         writer.write("Hello, 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 to file: " + e.getMessage());
      }
      try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
         System.out.println("\nFile 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("\nThis is an appended 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:



```
File contents:
Hello, this is a test file!
This is line 2.

Successfully appended to the file.

Reading using FileInputStream:
Hello, this is a test file!
This is line 2.
This is an appended line.
PS C:\Users\prodd\OneDrive\Desktop\Sheshank\OOPS>
```