

LABRECORD

23CSE111-ObjectOrientedProgramming

***Submittedby***

CH.SC.U4CSE24139-S.Preetham Chandran

# BACHELOR OF TECHNOLOGY

## IN

COMPUTER SCIENCE AND ENGINEERING

AMRITAVISHWAVIDYAPEETHAM AMRITA SCHOOL OF COMPUTING

CHENNAI

March-2025



**AMRITA VISHWA VIDYAPEETHAM AMRITASCHOOLOFCOMPUTING,CHENNAI**

**BONAFIDECERTIFICATE**

This is to certify that the Lab Record work for 23CSE111- Object Oriented Programming Subject submitted by ***CH.SC.U4CSE24139 – S.Preetham Chandran*** 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 5/04/2025

InternalExaminer1 InternalExaminer2

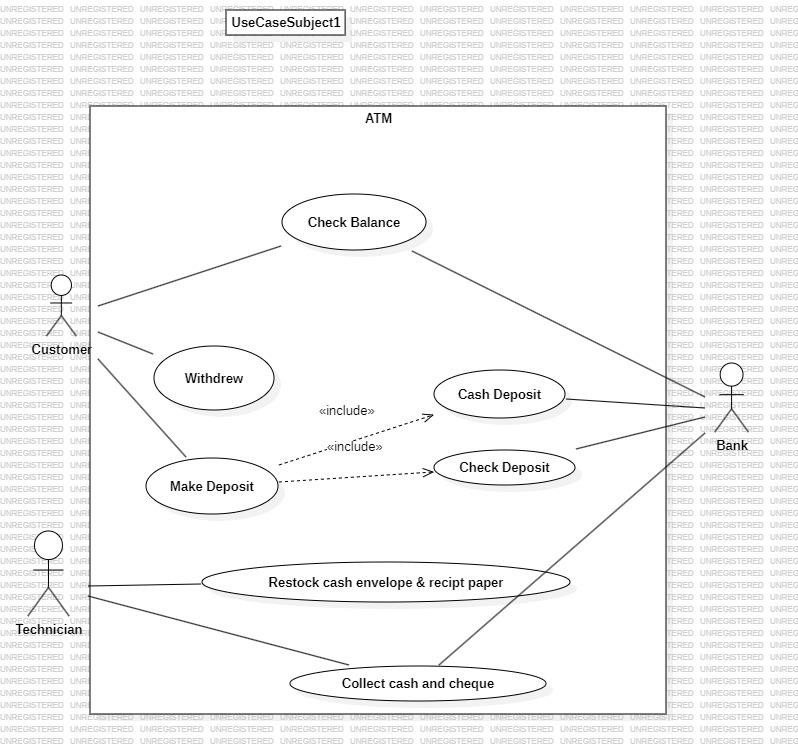
## INDEX

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE.NO** |
| **UMLDIAGRAM** | |  |
| 1. | **TITLE OF UML DIAGRAM -1** |  |
|  | 1.a)Use Case Diagram |  |
|  | 1.b)Class Diagram |  |
|  | 1.c)Sequence Diagram |  |
|  | 1.d) |  |
|  | 1.e) |  |
| 2. | **TITLE OF UML DIAGRAM -2** |  |
|  | 2.a)Use Case Diagram |  |
|  | 2.b)Class Diagram |  |
|  | 2.c)Sequence Diagram |  |
|  | 2.d) |  |
|  | 2.e) |  |
| 3. | **BASICJAVAPROGRAMS** |  |
|  | 3.a) Print from 1 to 10 using while loop in java program |  |
|  | 3.b) Find the factorial of the number using for loop. |  |
|  | 3.c) Check if a Number is Prime using for loop: |  |
|  | 3.d) Fibonacci Series using while loop. |  |
|  | 3.e) Reverse a String using for loop. |  |
|  | 3.f) Find the Largest Number in an Array. |  |
|  | 3.g) Count Vowels and consonants using string. |  |
|  | 3.h) Check if a number is Armstrong or not. |  |
|  | 3.i) Find GCD using while loop. |  |
|  | 3.j) Check if a String is a palindrome. |  |
|  | **INHERITANCE** |  |

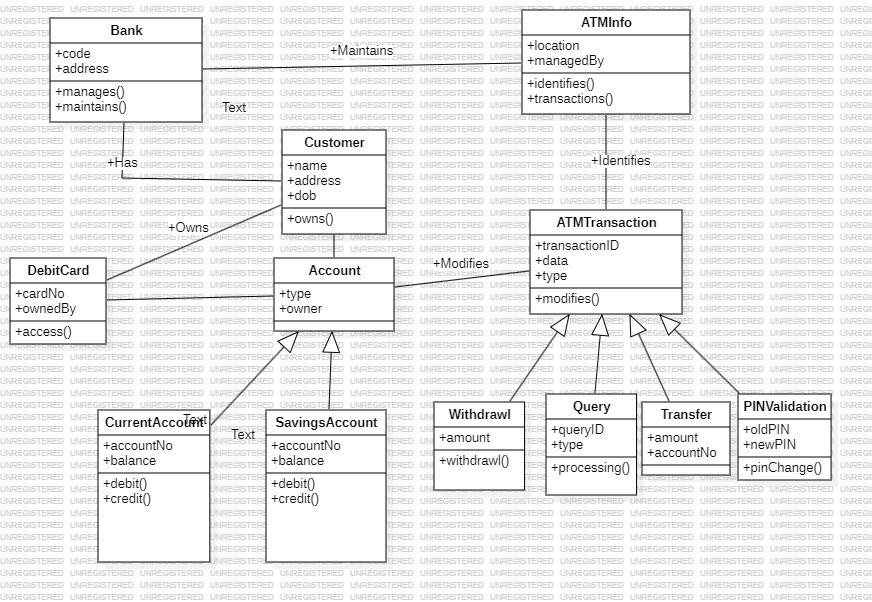
|  |  |  |
| --- | --- | --- |
| 4. | **SINGLE INHERITANCE PROGRAMS** |  |
|  | 4.a)Payment |  |
|  | 4.b)Email Notification |  |
| 5. | **MULTILEVEL INHERITANCE PROGRAMS** |  |
|  | 5.a)Vehicle |  |
|  | 5.b)Employee |  |
| 6. | **HIERARCHICAL INHERITANCE PROGRAMS** |  |
|  | 6.a)Drawing Shape |  |
|  | 6.b)Database Connection |  |
| 7. | **HYBRID INHERITANCE PROGRAMS** |  |
|  | 7.a)University |  |
|  | 7.b)Product |  |
|  | **POLYMORPHISM** |  |
| 8. | **CONSTRUCTOR PROGRAMS** |  |
|  | 8.a)**Bank Account** |  |
| 9. | **CONSTRUCTOR OVERLOADING PROGRAMS** |  |
|  | 9.a)Employee |  |
| 10. | **METHOD OVERLOADING PROGRAMS** |  |
|  | 10.a)Calculator |  |
|  | 10.b)Printer |  |
| 11. | **METHOD OVERRIDING PROGRAMS** |  |
|  | 11.a)Vehicle |  |
|  | 11.b)account |  |
|  | **ABSTRACTION** |  |
| 12. | **INTERFACE PROGRAMS** |  |
|  | 12.a)Smart Device |  |
|  | 12.b)Payment Method |  |
|  | 12.c)Weather Sensor |  |
|  | 12.d)Character Ability |  |
| 13. | **ABSTRACT CLASS PROGRAMS** |  |
|  | 13.a)Social Media Post |  |
|  | 13.b)Delivery |  |
|  | 13.c)Course |  |
|  | 13.d)Coffee Maker |  |
|  | **ENCAPSULATION** |  |
| 14. | **ENCAPSULATION PROGRAMS** |  |
|  | 14.a)Thermostat |  |
|  | 14.b)Inventory Items |  |
|  | 14.c)User Account |  |
|  | 14.d)Vehicle |  |
| 15. | **PACKAGES PROGRAMS** |  |
|  | 15.a)Package Maths |  |
|  | 15.b)Package com.text.utils |  |
|  | 15.c)java time |  |
|  | 15.d)collection analyzer |  |
| 16. | **EXCEPTION HANDLING PROGRAMS** |  |
|  | 16.a)Fund |  |
|  | 16.b)File Processor(java.io) |  |
|  | 16.c)Database(java.sql) |  |
|  | 16.d)Registration |  |
| 17. | **FILE HANDLING PROGRAMS** |  |
|  | 17.a)Manager(java.io) |  |
|  | 17.b)Csv Processor |  |
|  | 17.c)Log Analyzer |  |
|  | 17.d)Employee Record |  |

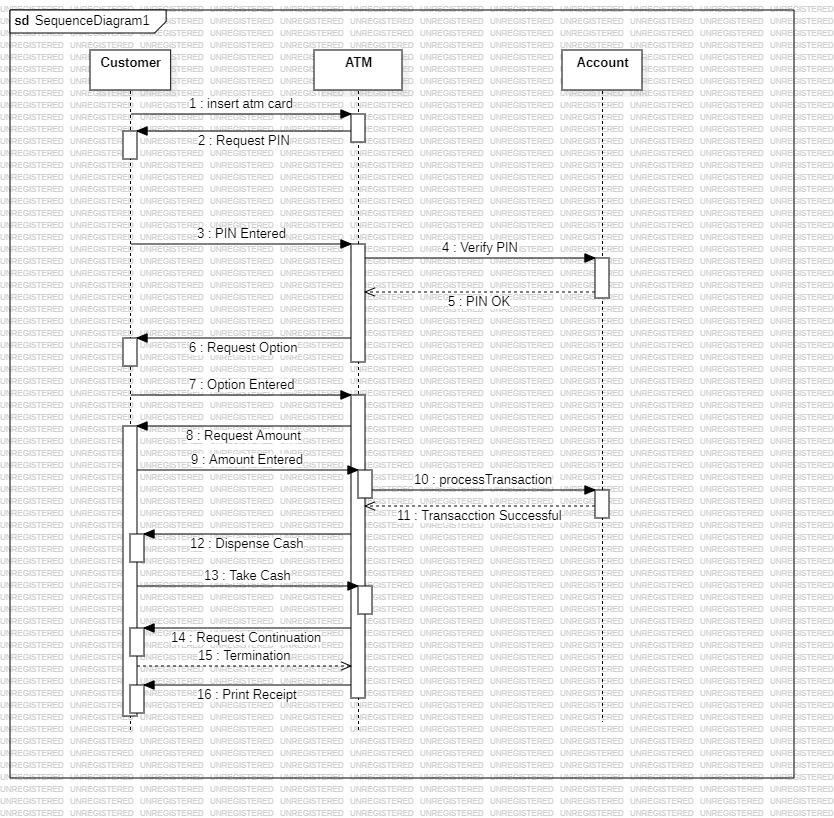
UMLDIAGRAMS

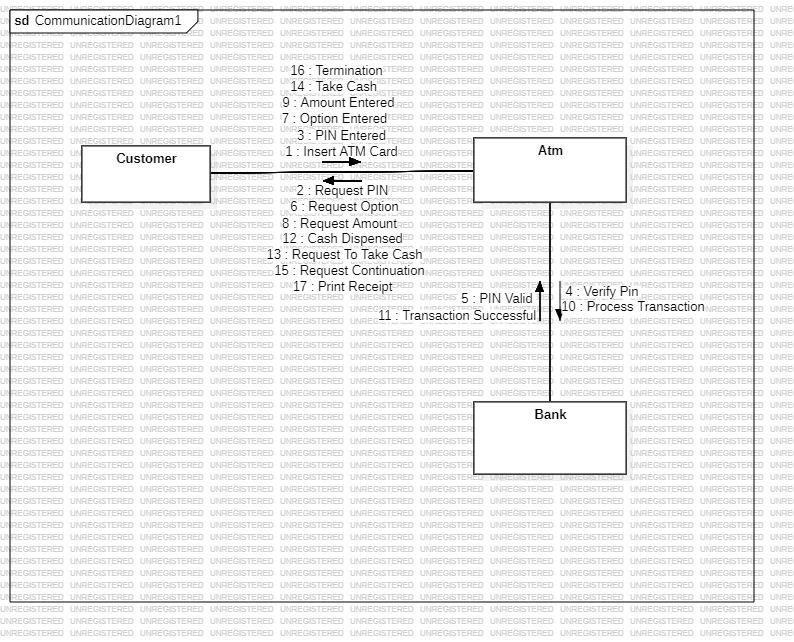
1. ATMWITHDRAWL
2. **Use – Case Diagram :**

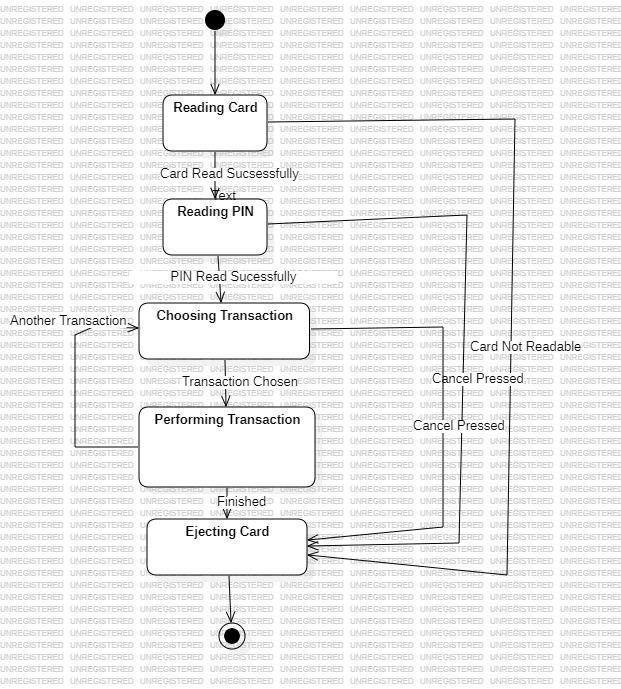


1. **State-Diagram**



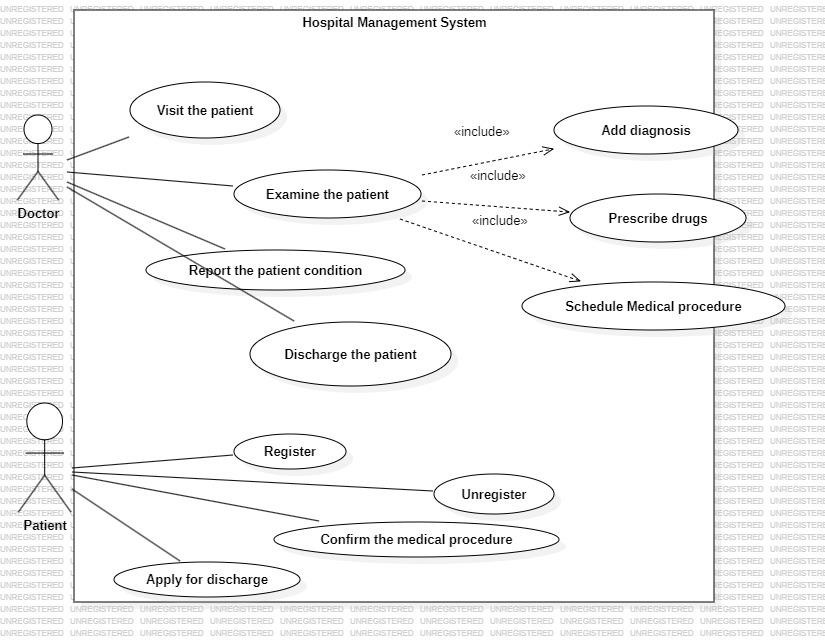
1. **Sequence Diagram ;**
2. **Communication Diagram:**



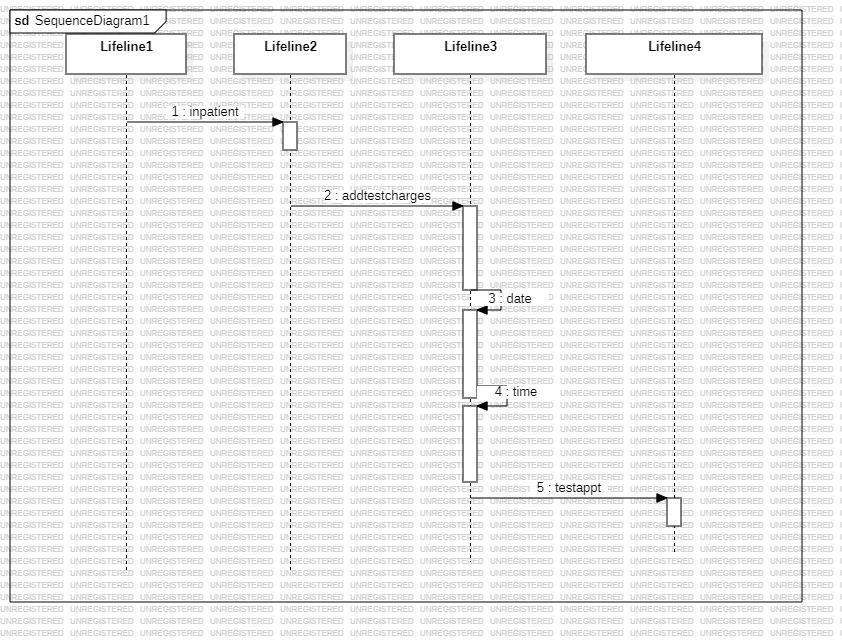
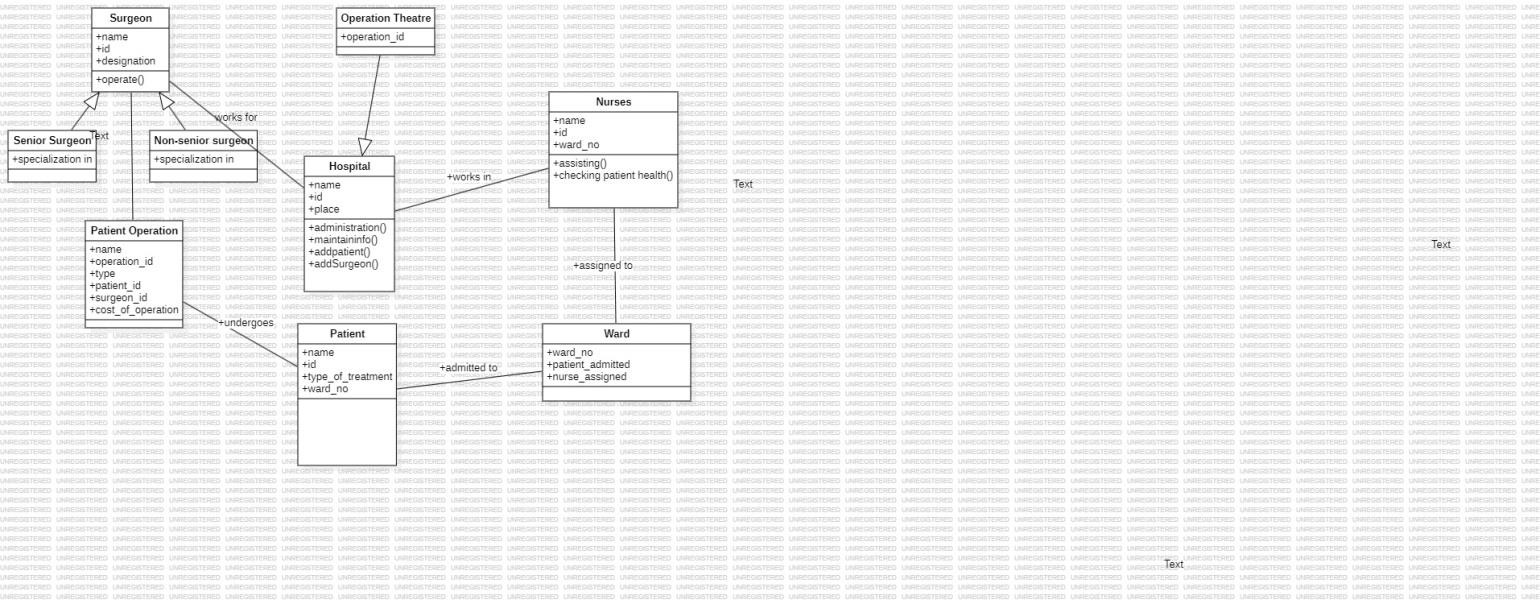
1. **State Diagram :**

Hotel management

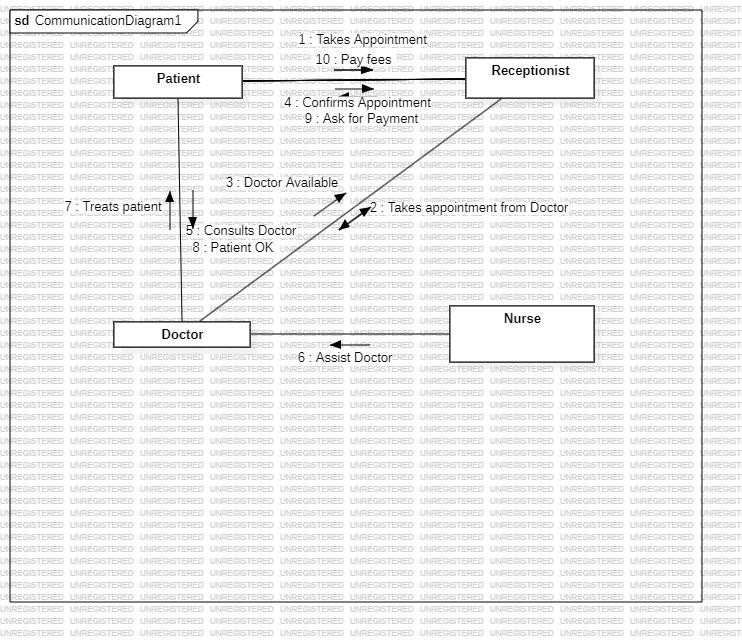
1. **Use-Case Diagram:**



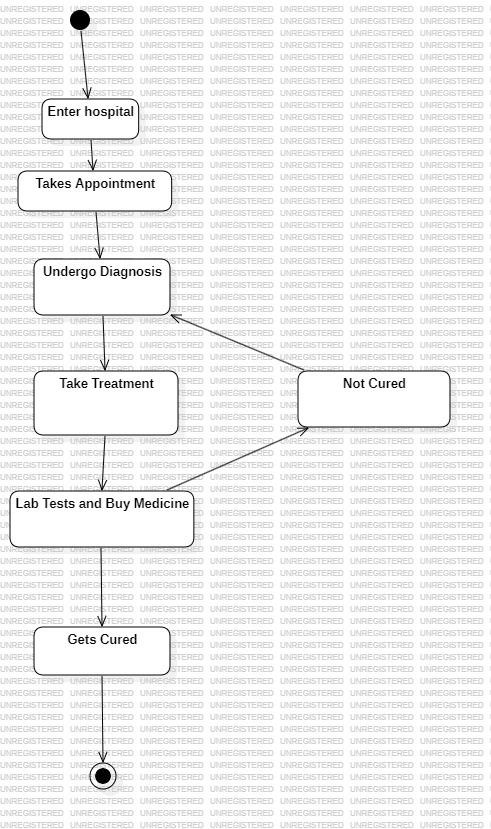
1. **Class & Sequence Diagram :**



1. **Communication Diagram :**



1. **State Diagram :**

****

1.)

**Aim** : Print from 1 to 10 using while loop in java program

**Program Code:**

public class WhileLoop{

public static void main(String[] args) {

int i = 1;

while (i <= 10) {

System.out.println(i);

i++;

}

}

}

**Output :**

1

2

3

4

5

6

7

8

9

10

2.)

**Aim :** Find the factorial of the number using for loop.

**Program Code:**

public class FactorialExample {

public static void main(String[] args) {

int num = 5;

int factorial = 1;

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

factorial \*= i;

}

System.out.println("Factorial of " + num + " is " + factorial);

}

}

**Output :**

Factorial of 5 is 120.

3.)

Aim : Check if a Number is Prime using for loop:

Program Code:

import java.util.Scanner;

public class PrimeCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int num = scanner.nextInt();

boolean isPrime = true;

if (num <= 1) {

isPrime = false;

} else {

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

isPrime = false;

break;

}

}

}

if (isPrime)

System.out.println(num + " is a Prime number");

else

System.out.println(num + " is not a Prime number");

}

}

**Input:**

Enter a number : 17

**Output:**

17 is a Prime number.

4.)

**Aim :** Fibonacci Series using while loop.

**Program Code:**

import java.util.Scanner;

public class FibonacciWhile {

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, i = 1;

while (i <= n) {

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

int temp = a + b;

a = b;

b = temp;

i++;

}

}

}

**Output:**

Enter the number of terms: 7

0 1 1 2 3 5 8

5.)

**Aim:** Reverse a String using for loop

**Program Code:**

import java.util.Scanner;

public class ReverseString {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str = scanner.nextLine();

String reversed = "";

for (int i = str.length() - 1; i >= 0; i--) {

reversed += str.charAt(i);

}

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

}

}

**Output:**

Enter a string: hello

Reversed String: olleh

6.)

**Aim :** Find the Largest Number in an Array.

**Program Code:**

public class LargestInArray {

public static void main(String[] args) {

int[] numbers = {10, 45, 78, 23, 56, 89, 12};

int max = numbers[0];

for (int num : numbers) {

if (num > max) {

max = num;

}

}

System.out.println("Largest number: " + max);

}

}

**Output:**

Largest number: 89

7.)

**Aim :** Count Vowels and consonants using string

**Program Code:**

import java.util.Scanner;

public class VowelConsonantCount {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str = scanner.nextLine().toLowerCase();

int vowels = 0, consonants = 0;

for (char ch : str.toCharArray()) {

if (ch >= 'a' && ch <= 'z') {

if ("aeiou".indexOf(ch) != -1) {

vowels++;

} else {

consonants++;

}

}

}

System.out.println("Vowels: " + vowels);

System.out.println("Consonants: " + consonants);

}

}

**Output :**

Enter a string: Java Programming

Vowels: 5

Consonants: 9

8.)

**Aim :** Check if a number is Armstrong or not.

**Program Code :**

import java.util.Scanner;

public class ArmstrongNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int num = scanner.nextInt();

int originalNum = num, sum = 0, digits = String.valueOf(num).length();

while (num > 0) {

int digit = num % 10;

sum += Math.pow(digit, digits);

num /= 10;

}

if (sum == originalNum)

System.out.println(originalNum + " is an Armstrong number");

else

System.out.println(originalNum + " is not an Armstrong number");

}

}

**Output :**

Enter a number: 153

153 is an Armstrong number

9.)

**Aim :** Find GCD using while loop.

**Program Code:**

import java.util.Scanner;

public class GCDExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter two numbers: ");

int a = scanner.nextInt(), b = scanner.nextInt();

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

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

}

}

**Output :**

Enter two numbers: 48 18

GCD: 6

10.)

**Aim:** Check if a String is a palindrome.

**Program Code:**

import java.util.Scanner;

public class PalindromeCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str = scanner.nextLine();

String reversed = new StringBuilder(str).reverse().toString();

if (str.equals(reversed))

System.out.println(str + " is a Palindrome");

else

System.out.println(str + " is not a Palindrome");

}

}

**Output :**

Enter a string: madam

madam is a Palindrome

4.)

#### a)

#### class Payment {

#### void processPayment(double amount) {

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

#### }

#### }

#### class CreditCardPayment extends Payment {

#### void verifyCard(String cardNumber) {

#### System.out.println("Verifying card: " + cardNumber.substring(12));

#### }

#### }

#### class TestSingle1 {

#### public static void main(String[] args) {

#### CreditCardPayment cc = new CreditCardPayment();

#### cc.verifyCard("1234567812345678");

#### cc.processPayment(99.99);

#### }

#### }

#### Output :

#### 

#### b)

#### class Notification {

#### void send(String message) {

#### System.out.println("Sending: " + message);

#### }

#### }

#### class EmailNotification extends Notification {

#### void setRecipient(String email) {

#### System.out.println("Email to: " + email);

#### }

#### }

#### class TestSingle2 {

#### public static void main(String[] args) {

#### EmailNotification email = new EmailNotification();

#### email.setRecipient("user@example.com");

#### email.send("Your order is confirmed");

#### }

#### }

#### Output:

#### 

#### 5)

#### a)

#### class Vehicle {

#### void startEngine() {

#### System.out.println("Engine started");

#### }

#### }

#### class Car extends Vehicle {

#### void accelerate() {

#### System.out.println("Car accelerating");

#### }

#### }

#### class ElectricCar extends Car {

#### void chargeBattery() {

#### System.out.println("Battery charging");

#### }

#### }

#### class TestMulti1 {

#### public static void main(String[] args) {

#### ElectricCar tesla = new ElectricCar();

#### tesla.startEngine();

#### tesla.accelerate();

#### tesla.chargeBattery();

#### }

#### }

#### Output :

#### 

#### b)

#### class Employee {

#### void work() {

#### System.out.println("Employee working");

#### }

#### }

#### class Manager extends Employee {

#### void conductMeeting() {

#### System.out.println("Manager conducting meeting");

#### }

#### }

#### class SeniorManager extends Manager {

#### void approveBudget() {

#### System.out.println("Approving department budget");

#### }

#### }

#### class TestMulti2 {

#### public static void main(String[] args) {

#### SeniorManager sm = new SeniorManager();

#### sm.work();

#### sm.conductMeeting();

#### sm.approveBudget();

#### }

#### }

#### Output :

#### 

#### 6)

#### a)

#### class Shape {

#### void draw() {

#### System.out.println("Drawing shape");

#### }

#### }

#### class Circle extends Shape {

#### void calculateArea(double radius) {

#### System.out.println("Circle area: " + (3.14 \* radius \* radius));

#### }

#### }

#### class Square extends Shape {

#### void calculateArea(double side) {

#### System.out.println("Square area: " + (side \* side));

#### }

#### }

#### class TestHier1 {

#### public static void main(String[] args) {

#### Circle c = new Circle();

#### Square s = new Square();

#### c.draw();

#### c.calculateArea(5.0);

#### s.draw();

#### s.calculateArea(4.0);

#### }

#### }

#### Output :

#### 

#### b)

#### class DatabaseConnection {

#### void connect() {

#### System.out.println("Connected to database");

#### }

#### }

#### class MySQLConnection extends DatabaseConnection {

#### void queryMySQL() {

#### System.out.println("Executing MySQL query");

#### }

#### }

#### class PostgreSQLConnection extends DatabaseConnection {

#### void queryPostgreSQL() {

#### System.out.println("Executing PostgreSQL query");

#### }

#### }

#### class TestHier2 {

#### public static void main(String[] args) {

#### MySQLConnection mysql = new MySQLConnection();

#### PostgreSQLConnection psql = new PostgreSQLConnection();

#### mysql.connect();

#### mysql.queryMySQL();

#### psql.connect();

#### psql.queryPostgreSQL();

#### }

#### }

#### Output :

#### 

#### 7)

#### a)

#### class UniversityMember {

#### void login() {

#### System.out.println("University member logged in");

#### }

#### }

#### class Student extends UniversityMember {

#### void attendClass() {

#### System.out.println("Student attending class");

#### }

#### }

#### class Professor extends UniversityMember {

#### void teachCourse() {

#### System.out.println("Professor teaching course");

#### }

#### }

#### class ResearchProfessor extends Professor {

#### void conductResearch() {

#### System.out.println("Conducting advanced research");

#### }

#### }

#### class TestHybrid1 {

#### public static void main(String[] args) {

#### Student s = new Student();

#### ResearchProfessor rp = new ResearchProfessor();

#### s.login();

#### s.attendClass();

#### rp.login();

#### rp.teachCourse();

#### rp.conductResearch();

#### }

#### }

#### Output :

#### 

#### b)

#### class Product {

#### void display() {

#### System.out.println("Displaying product");

#### }

#### }

#### class Electronics extends Product {

#### void checkWarranty() {

#### System.out.println("Checking electronics warranty");

#### }

#### }

#### class Clothing extends Product {

#### void checkSize() {

#### System.out.println("Checking clothing size");

#### }

#### }

#### class Smartphone extends Electronics {

#### void installApp() {

#### System.out.println("Installing smartphone app");

#### }

#### }

#### class TestHybrid2 {

#### public static void main(String[] args) {

#### Smartphone phone = new Smartphone();

#### Clothing shirt = new Clothing();

#### phone.display();

#### phone.checkWarranty();

#### phone.installApp();

#### shirt.display();

#### shirt.checkSize();

#### }

#### }

#### Output :

#### 

#### 8)

#### a)

#### class BankAccount {

#### String accountHolder;

#### double balance;

#### 

#### public BankAccount(String name, double initialDeposit) {

#### this.accountHolder = name;

#### this.balance = initialDeposit;

#### System.out.println("Account created for " + name + " with $" + initialDeposit);

#### }

#### 

#### void displayBalance() {

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

#### }

#### }

#### class TestConstructor {

#### public static void main(String[] args) {

#### BankAccount account = new BankAccount("John Doe", 1000.0);

#### account.displayBalance();

#### }

#### }

#### Output :

#### 

#### 9)

#### a)

#### class Employee {

#### String name;

#### int id;

#### String department;

#### 

#### public Employee(String name, int id) {

#### this.name = name;

#### this.id = id;

#### this.department = "General";

#### }

#### 

#### public Employee(String name, int id, String department) {

#### this.name = name;

#### this.id = id;

#### this.department = department;

#### }

#### 

#### void displayInfo() {

#### System.out.println(name + " (ID:" + id + ") - " + department + " Dept.");

#### }

#### }

#### class TestConstructorOverloading {

#### public static void main(String[] args) {

#### Employee emp1 = new Employee("Alice", 1001);

#### Employee emp2 = new Employee("Bob", 1002, "Engineering");

#### emp1.displayInfo();

#### emp2.displayInfo();

#### }

#### }

#### Output :

#### 

#### 10)

#### a)

#### class Calculator {

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

#### return a + b;

#### }

#### 

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

#### return a + b + c;

#### }

#### 

#### double add(double a, double b) {

#### return a + b;

#### }

#### }

#### class TestMethodOverloading1 {

#### public static void main(String[] args) {

#### Calculator calc = new Calculator();

#### System.out.println("Sum (2+3): " + calc.add(2, 3));

#### System.out.println("Sum (2+3+4): " + calc.add(2, 3, 4));

#### System.out.println("Sum (2.5+3.7): " + calc.add(2.5, 3.7));

#### }

#### }

#### Output :

#### 

#### b)

#### class Printer {

#### void print(String text) {

#### System.out.println("Printing: " + text);

#### }

#### 

#### void print(String text, int copies) {

#### for(int i = 0; i < copies; i++) {

#### System.out.println("Copy " + (i+1) + ": " + text);

#### }

#### }

#### 

#### void print(String text, boolean uppercase) {

#### System.out.println("Printing: " + (uppercase ? text.toUpperCase() : text));

#### }

#### }

#### class TestMethodOverloading2 {

#### public static void main(String[] args) {

#### Printer printer = new Printer();

#### printer.print("Hello World");

#### printer.print("Important Notice", 3);

#### printer.print("confidential", true);

#### }

#### }

#### Output :

#### 

#### 11)

#### a)

#### class Vehicle {

#### void start() {

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

#### }

#### 

#### void stop() {

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

#### }

#### }

#### class ElectricCar extends Vehicle {

#### void start() {

#### System.out.println("Electric car starting silently...");

#### }

#### 

#### void stop() {

#### System.out.println("Electric car regenerating brakes...");

#### }

#### 

#### void charge() {

#### System.out.println("Charging battery...");

#### }

#### }

#### class TestMethodOverriding1 {

#### public static void main(String[] args) {

#### Vehicle regular = new Vehicle();

#### ElectricCar tesla = new ElectricCar();

#### 

#### regular.start();

#### regular.stop();

#### 

#### tesla.start();

#### tesla.stop();

#### tesla.charge();

#### }

#### }

#### Output :

#### 

#### b)

#### class Account {

#### void withdraw(double amount) {

#### System.out.println("Withdrawing $" + amount + " from generic account");

#### }

#### 

#### void deposit(double amount) {

#### System.out.println("Depositing $" + amount + " to generic account");

#### }

#### }

#### class SavingsAccount extends Account {

#### void withdraw(double amount) {

#### System.out.println("Withdrawing $" + amount + " from savings (2-day clearance)");

#### }

#### 

#### void deposit(double amount) {

#### System.out.println("Depositing $" + amount + " to savings (earning interest)");

#### }

#### 

#### void applyInterest() {

#### System.out.println("Applying monthly interest");

#### }

#### }

#### class TestMethodOverriding2 {

#### public static void main(String[] args) {

#### Account generic = new Account();

#### SavingsAccount mySavings = new SavingsAccount();

#### 

#### generic.deposit(500);

#### generic.withdraw(100);

#### 

#### mySavings.deposit(1000);

#### mySavings.withdraw(200);

#### mySavings.applyInterest();

#### }

#### }

#### Output :

#### 

#### 12)

#### a) interface SmartDevice {

#### void turnOn();

#### void turnOff();

#### String getStatus();

#### }

#### class SmartLight implements SmartDevice {

#### private boolean isOn = false;

#### 

#### public void turnOn() {

#### isOn = true;

#### System.out.println("Light turned on");

#### }

#### 

#### public void turnOff() {

#### isOn = false;

#### System.out.println("Light turned off");

#### }

#### 

#### public String getStatus() {

#### return "Light is " + (isOn ? "ON" : "OFF");

#### }

#### }

#### class TestSmartHome {

#### public static void main(String[] args) {

#### SmartDevice light = new SmartLight();

#### light.turnOn();

#### System.out.println(light.getStatus());

#### }

#### }

#### Output :

#### 

#### b)

#### interface PaymentMethod {

#### boolean authorize(double amount);

#### void processPayment(double amount);

#### void refund(double amount);

#### }

#### class CryptoPayment implements PaymentMethod {

#### public boolean authorize(double amount) {

#### System.out.println("Authorizing crypto payment...");

#### return true;

#### }

#### 

#### public void processPayment(double amount) {

#### System.out.println("Processing crypto payment of " + amount + " BTC");

#### }

#### 

#### public void refund(double amount) {

#### System.out.println("Refunding " + amount + " BTC");

#### }

#### }

#### public class PaymentDemo {

#### public static void main(String[] args) {

#### PaymentMethod payment = new CryptoPayment();

#### 

#### if(payment.authorize(100.0)) {

#### payment.processPayment(100.0);

#### }

#### 

#### payment.refund(25.0);

#### }

#### }

#### 

#### Output :

#### 

#### c)

#### interface WeatherSensor {

#### double getTemperature();

#### double getHumidity();

#### default String getSensorType() {

#### return "Generic Weather Sensor";

#### }

#### }

#### class OutdoorSensor implements WeatherSensor {

#### public double getTemperature() {

#### return 72.5 + (Math.random() \* 10 - 5); // Random variation

#### }

#### 

#### public double getHumidity() {

#### return 45.8 + (Math.random() \* 20 - 10); // Random variation

#### }

#### 

#### public String getSensorType() {

#### return "Outdoor Weather Sensor";

#### }

#### }

#### public class WeatherStation {

#### public static void main(String[] args) {

#### WeatherSensor sensor = new OutdoorSensor();

#### 

#### System.out.println("Sensor Type: " + sensor.getSensorType());

#### System.out.printf("Current Temperature: %.1f°F%n", sensor.getTemperature());

#### System.out.printf("Current Humidity: %.1f%%%n", sensor.getHumidity());

#### }

#### }

#### Output :

#### 

#### d)

#### interface CharacterAbility {

#### void useAbility();

#### int getCooldown();

#### }

#### class FireballAbility implements CharacterAbility {

#### public void useAbility() {

#### System.out.println("Casting Fireball! Damage: 50");

#### }

#### 

#### public int getCooldown() {

#### return 5;

#### }

#### }

#### public class GameDemo {

#### public static void main(String[] args) {

#### CharacterAbility ability = new FireballAbility();

#### 

#### System.out.println("Testing character ability:");

#### ability.useAbility();

#### System.out.println("Cooldown: " + ability.getCooldown() + " seconds");

#### 

#### System.out.println("\nUsing ability in combat:");

#### for(int i = 0; i < 3; i++) {

#### ability.useAbility();

#### System.out.println("Waiting " + ability.getCooldown() + " seconds...");

#### }

#### }

#### }

#### Output :

#### 

#### 13)

#### a)

#### abstract class SocialMediaPost {

#### protected String author;

#### protected String content;

#### 

#### public SocialMediaPost(String author, String content) {

#### this.author = author;

#### this.content = content;

#### }

#### 

#### abstract void display();

#### 

#### void showAuthor() {

#### System.out.println("Posted by: " + author);

#### }

#### }

#### class TwitterPost extends SocialMediaPost {

#### public TwitterPost(String author, String content) {

#### super(author, content);

#### }

#### 

#### void display() {

#### System.out.println("Tweet: " + content);

#### System.out.println("Like,Share,Subscribe");

#### }

#### }

#### public class SocialMediaDemo {

#### public static void main(String[] args) {

#### SocialMediaPost tweet = new TwitterPost("java\_dev", "Learning Java interfaces today!");

#### 

#### tweet.showAuthor();

#### tweet.display();

#### }

#### }

#### Output:

#### 

#### b)

#### abstract class Delivery {

#### protected String trackingNumber;

#### protected double weight;

#### 

#### public Delivery(String trackingNumber, double weight) {

#### this.trackingNumber = trackingNumber;

#### this.weight = weight;

#### }

#### 

#### abstract double calculateShippingCost();

#### abstract int getDeliveryDays();

#### 

#### void printLabel() {

#### System.out.println("Tracking #: " + trackingNumber);

#### }

#### }

#### class ExpressDelivery extends Delivery {

#### public ExpressDelivery(String trackingNumber, double weight) {

#### super(trackingNumber, weight);

#### }

#### 

#### double calculateShippingCost() {

#### return 10 + (weight \* 2.5);

#### }

#### int getDeliveryDays() {

#### return 1;

#### }

#### }

#### public class LogisticsDemo {

#### public static void main(String[] args) {

#### Delivery express = new ExpressDelivery("EXP123456", 3.5);

#### 

#### express.printLabel();

#### System.out.printf("Shipping cost: $%.2f%n", express.calculateShippingCost());

#### System.out.println("Estimated delivery days: " + express.getDeliveryDays());

#### }

#### }

#### Output :

#### 

#### c)

#### abstract class Course {

#### protected String courseCode;

#### protected String title;

#### protected int creditHours;

#### 

#### public Course(String code, String title, int credits) {

#### this.courseCode = code;

#### this.title = title;

#### this.creditHours = credits;

#### }

#### 

#### abstract void conductClass();

#### abstract void gradeStudents();

#### 

#### void displayCourseInfo() {

#### System.out.println(courseCode + ": " + title + " (" + creditHours + " credits)");

#### }

#### }

#### class ProgrammingCourse extends Course {

#### public ProgrammingCourse(String code, String title, int credits) {

#### super(code, title, credits);

#### }

#### 

#### void conductClass() {

#### System.out.println("Teaching programming concepts with live coding");

#### }

#### 

#### void gradeStudents() {

#### System.out.println("Grading based on projects and exams");

#### }

#### }

#### public class UniversityDemo {

#### public static void main(String[] args) {

#### Course javaCourse = new ProgrammingCourse("CS101", "Introduction to Java", 4);

#### 

#### javaCourse.displayCourseInfo();

#### javaCourse.conductClass();

#### javaCourse.gradeStudents();

#### }

#### }

#### Output :

#### 

#### d)

#### abstract class CoffeeMaker {

#### protected int waterLevel;

#### 

#### public CoffeeMaker(int waterLevel) {

#### this.waterLevel = waterLevel;

#### }

#### 

#### abstract void brewCoffee();

#### abstract void cleanMachine();

#### 

#### void checkWater() {

#### System.out.println("Water level: " + waterLevel + "ml");

#### }

#### }

#### class EspressoMachine extends CoffeeMaker {

#### public EspressoMachine(int waterLevel) {

#### super(waterLevel);

#### }

#### 

#### void brewCoffee() {

#### if (waterLevel >= 30) {

#### System.out.println("Brewing espresso shot");

#### waterLevel -= 30;

#### } else {

#### System.out.println("Not enough water");

#### }

#### }

#### 

#### void cleanMachine() {

#### System.out.println("Running espresso machine cleaning cycle");

#### }

#### }

#### public class CoffeeShop {

#### public static void main(String[] args) {

#### CoffeeMaker machine = new EspressoMachine(100);

#### 

#### machine.checkWater();

#### machine.brewCoffee();

#### machine.brewCoffee();

#### machine.brewCoffee();

#### machine.cleanMachine();

#### }

#### }

#### Output :

#### 

#### 14)

#### a)

#### public class Thermostat {

#### private double currentTemp;

#### private double targetTemp;

#### private boolean isHeating;

#### 

#### public Thermostat(double initialTemp) {

#### this.currentTemp = initialTemp;

#### this.targetTemp = initialTemp;

#### }

#### 

#### public void setTargetTemperature(double temp) {

#### if(temp < 10 || temp > 35) {

#### throw new IllegalArgumentException("Temperature must be between 10°C and 35°C");

#### }

#### this.targetTemp = temp;

#### updateHeatingStatus();

#### }

#### 

#### private void updateHeatingStatus() {

#### this.isHeating = currentTemp < targetTemp;

#### }

#### 

#### public void updateCurrentTemperature(double newTemp) {

#### this.currentTemp = newTemp;

#### updateHeatingStatus();

#### }

#### 

#### public String getStatus() {

#### return String.format("Current: %.1f°C | Target: %.1f°C | Mode: %s",

#### currentTemp, targetTemp, isHeating ? "HEATING" : "IDLE");

#### }

#### public static void main(String[] args) {

#### Thermostat nest = new Thermostat(20.0);

#### System.out.println(nest.getStatus());

#### 

#### nest.setTargetTemperature(22.5);

#### nest.updateCurrentTemperature(21.0);

#### System.out.println(nest.getStatus());

#### 

#### nest.updateCurrentTemperature(23.0);

#### System.out.println(nest.getStatus());

#### }

#### }

#### Output :

#### 

#### b)

#### public class InventoryItem {

#### private String sku;

#### private String name;

#### private int quantity;

#### private double price;

#### 

#### public InventoryItem(String sku, String name, double price) {

#### if(sku == null || sku.isEmpty()) throw new IllegalArgumentException("SKU cannot be empty");

#### this.sku = sku;

#### this.name = name;

#### this.price = price;

#### this.quantity = 0;

#### }

#### 

#### public void restock(int amount) {

#### if(amount <= 0) throw new IllegalArgumentException("Restock amount must be positive");

#### this.quantity += amount;

#### }

#### 

#### public void sell(int amount) {

#### if(amount <= 0) throw new IllegalArgumentException("Sale amount must be positive");

#### if(amount > quantity) throw new IllegalStateException("Insufficient stock");

#### this.quantity -= amount;

#### }

#### 

#### public double getInventoryValue() {

#### return quantity \* price;

#### }

#### 

#### public String getItemInfo() {

#### return String.format("%s (%s) - %d units @ $%.2f", name, sku, quantity, price);

#### }

#### public static void main(String[] args) {

#### InventoryItem laptop = new InventoryItem("LT1001", "Dell XPS 15", 1299.99);

#### laptop.restock(10);

#### System.out.println(laptop.getItemInfo());

#### 

#### laptop.sell(3);

#### System.out.println(laptop.getItemInfo());

#### System.out.printf("Total inventory value: $%.2f%n", laptop.getInventoryValue());

#### }

#### }

#### Output :

#### 

#### c)

#### public class UserAccount {

#### private String username;

#### private String encryptedPassword;

#### private int failedLoginAttempts;

#### private boolean isLocked;

#### 

#### public UserAccount(String username, String password) {

#### if(username == null || username.isEmpty()) throw new IllegalArgumentException("Username required");

#### this.username = username;

#### setPassword(password);

#### }

#### 

#### public void setPassword(String newPassword) {

#### if(newPassword.length() < 8) throw new IllegalArgumentException("Password must be 8+ characters");

#### this.encryptedPassword = encryptPassword(newPassword);

#### }

#### 

#### private String encryptPassword(String password) {

#### return new StringBuilder(password).reverse().toString();

#### }

#### 

#### public boolean login(String attemptedPassword) {

#### if(isLocked) throw new IllegalStateException("Account locked");

#### 

#### boolean success = encryptedPassword.equals(encryptPassword(attemptedPassword));

#### if(!success) {

#### failedLoginAttempts++;

#### if(failedLoginAttempts >= 3) isLocked = true;

#### } else {

#### failedLoginAttempts = 0;

#### }

#### return success;

#### }

#### 

#### public void unlockAccount(String adminKey) {

#### if("ADMIN123".equals(adminKey)) isLocked = false;

#### }

#### public static void main(String[] args) {

#### UserAccount user = new UserAccount("john\_doe", "secure123");

#### 

#### System.out.println("Login attempt 1: " + user.login("wrongpass"));

#### System.out.println("Login attempt 2: " + user.login("stillwrong"));

#### System.out.println("Login attempt 3: " + user.login("secure123"));

#### 

#### UserAccount lockedUser = new UserAccount("test\_user", "password123");

#### lockedUser.login("wrong");

#### lockedUser.login("wrong");

#### lockedUser.login("wrong");

#### System.out.println("Is account locked? " + lockedUser.login("wrong"));

#### 

#### lockedUser.unlockAccount("ADMIN123");

#### System.out.println("After unlock: " + lockedUser.login("password123"));

#### }

#### }

#### Output :

#### 

#### d)

#### public class Vehicle {

#### private String licensePlate;

#### private double speed;

#### private boolean isMoving;

#### private Location currentLocation;

#### 

#### private static class Location {

#### double latitude;

#### double longitude;

#### 

#### Location(double lat, double lon) {

#### this.latitude = lat;

#### this.longitude = lon;

#### }

#### }

#### 

#### public Vehicle(String licensePlate) {

#### this.licensePlate = licensePlate;

#### this.speed = 0;

#### this.isMoving = false;

#### this.currentLocation = new Location(0, 0);

#### }

#### 

#### public void updateLocation(double lat, double lon) {

#### this.currentLocation = new Location(lat, lon);

#### }

#### 

#### public void accelerate(double acceleration) {

#### if(acceleration < 0) throw new IllegalArgumentException("Negative acceleration not allowed");

#### this.speed += acceleration;

#### this.isMoving = speed > 0;

#### }

#### 

#### public void brake(double deceleration) {

#### if(deceleration < 0) throw new IllegalArgumentException("Negative deceleration not allowed");

#### this.speed = Math.max(0, speed - deceleration);

#### this.isMoving = speed > 0;

#### }

#### 

#### public String getStatus() {

#### return String.format("%s - Speed: %.1f km/h - %s at (%.4f, %.4f)",

#### licensePlate, speed, isMoving ? "Moving" : "Stopped",

#### currentLocation.latitude, currentLocation.longitude);

#### }

#### public static void main(String[] args) {

#### Vehicle car = new Vehicle("ABC123");

#### car.updateLocation(40.7128, -74.0060); // New York coordinates

#### 

#### System.out.println("Initial status: " + car.getStatus());

#### 

#### car.accelerate(50.0);

#### car.updateLocation(40.7135, -74.0065);

#### System.out.println("After acceleration: " + car.getStatus());

#### 

#### car.brake(30.0);

#### System.out.println("After braking: " + car.getStatus());

#### 

#### car.brake(30.0);

#### System.out.println("After full stop: " + car.getStatus());

#### }

#### }

#### 

#### Output :

#### 

#### 15)

#### a)

#### package com.math.advanced;

#### public class Statistics {

#### public static double calculateMean(double[] values) {

#### double sum = 0;

#### for(double v : values) sum += v;

#### return sum / values.length;

#### }

#### 

#### public static double calculateStandardDeviation(double[] values) {

#### double mean = calculateMean(values);

#### double sum = 0;

#### for(double v : values) {

#### sum += Math.pow(v - mean, 2);

#### }

#### return Math.sqrt(sum / values.length);

#### }

#### public static void main(String[] args) {

#### double[] data = {12.5, 18.3, 11.2, 19.0, 22.1, 14.7};

#### System.out.printf("Mean: %.2f%n", calculateMean(data));

#### System.out.printf("Standard Deviation: %.2f%n", calculateStandardDeviation(data));

#### }

#### }

#### b)

#### package com.text.utils;

#### public class TextProcessor {

#### public static String removeSpecialChars(String input) {

#### return input.replaceAll("[^a-zA-Z0-9]", "");

#### }

#### 

#### public static int countWords(String text) {

#### return text.trim().isEmpty() ? 0 : text.trim().split("\\s+").length;

#### }

#### public static void main(String[] args) {

#### String testString = "Hello, World! This is a test string.";

#### System.out.println("Original: " + testString);

#### System.out.println("Without special chars: " + removeSpecialChars(testString));

#### System.out.println("Word count: " + countWords(testString));

#### }

#### }

#### Output :

#### 

#### c)

#### import java.time.\*;

#### import java.time.format.DateTimeFormatter;

#### import java.time.temporal.ChronoUnit; // Added this import

#### public class EventScheduler {

#### public static String scheduleEvent(String eventName, int daysFromNow) {

#### LocalDate today = LocalDate.now();

#### LocalDate eventDate = today.plusDays(daysFromNow);

#### return String.format("Event '%s' scheduled for %s",

#### eventName,

#### eventDate.format(DateTimeFormatter.ofPattern("MMMM dd, yyyy")));

#### }

#### 

#### public static long daysUntil(LocalDate futureDate) {

#### return ChronoUnit.DAYS.between(LocalDate.now(), futureDate);

#### }

#### public static void main(String[] args) {

#### System.out.println(scheduleEvent("Java Conference", 30));

#### 

#### LocalDate newYear = LocalDate.of(2024, 1, 1);

#### System.out.println("Days until New Year: " + daysUntil(newYear));

#### }

#### }

#### Output :

#### 

#### d)

#### import java.util.\*;

#### public class CollectionAnalyzer {

#### public static <T> List<T> removeDuplicates(List<T> list) {

#### return new ArrayList<>(new LinkedHashSet<>(list));

#### }

#### 

#### public static <T> Map<T, Integer> countOccurrences(List<T> list) {

#### Map<T, Integer> counts = new HashMap<>();

#### for(T item : list) {

#### counts.put(item, counts.getOrDefault(item, 0) + 1);

#### }

#### return counts;

#### }

#### public static void main(String[] args) {

#### List<String> names = Arrays.asList("Alice", "Bob", "Alice", "Charlie", "Bob");

#### 

#### System.out.println("Original list: " + names);

#### System.out.println("Without duplicates: " + removeDuplicates(names));

#### System.out.println("Counts: " + countOccurrences(names));

#### }

#### }

#### Output :

#### 

#### 16)

#### a)

#### class InsufficientFundsException extends Exception {

#### private double currentBalance;

#### private double requestedAmount;

#### 

#### public InsufficientFundsException(double current, double requested) {

#### super(String.format("Current balance $%.2f is insufficient for $%.2f", current, requested));

#### this.currentBalance = current;

#### this.requestedAmount = requested;

#### }

#### 

#### public double getDeficit() {

#### return requestedAmount - currentBalance;

#### }

#### }

#### class BankAccount {

#### private double balance;

#### 

#### public BankAccount(double initialBalance) {

#### this.balance = initialBalance;

#### }

#### 

#### public void withdraw(double amount) throws InsufficientFundsException {

#### if(amount > balance) {

#### throw new InsufficientFundsException(balance, amount);

#### }

#### balance -= amount;

#### System.out.printf("Withdrawn $%.2f. New balance: $%.2f%n", amount, balance);

#### }

#### }

#### public class Main {

#### public static void main(String[] args) {

#### BankAccount account = new BankAccount(500.00);

#### 

#### try {

#### System.out.println("Initial balance: $500.00");

#### account.withdraw(200.00);

#### account.withdraw(350.00); // This will throw exception

#### } catch (InsufficientFundsException e) {

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

#### System.out.printf("You need $%.2f more to complete this transaction%n", e.getDeficit());

#### }

#### 

#### try {

#### account.withdraw(100.00);

#### } catch (InsufficientFundsException e) {

#### System.out.println("This shouldn't be reached");

#### }

#### }

#### }

#### Output :

#### 

#### b)

#### import java.io.\*;

#### public class FileProcessor {

#### public static void processFile(String filename) {

#### try {

#### String content = readFileContents(filename);

#### System.out.println("File content:\n" + content);

#### } catch (FileNotFoundException e) {

#### System.err.println("Error: File not found - " + e.getMessage());

#### } catch (IOException e) {

#### System.err.println("IO Error: " + e.getMessage());

#### } catch (SecurityException e) {

#### System.err.println("Security restriction: " + e.getMessage());

#### }

#### }

#### 

#### private static String readFileContents(String filename) throws IOException {

#### return new String(java.nio.file.Files.readAllBytes(java.nio.file.Paths.get(filename)));

#### }

#### public static void main(String[] args) {

#### processFile("test.txt");

#### 

#### processFile("nonexistent.txt");

#### }

#### }

#### Output :

#### 

#### c)

#### import java.sql.\*;

#### public class DatabaseAccess {

#### public static void queryDatabase(String url, String user, String password) {

#### String query = "SELECT \* FROM products WHERE price > 100";

#### 

#### try (Connection conn = DriverManager.getConnection(url, user, password);

#### Statement stmt = conn.createStatement();

#### ResultSet rs = stmt.executeQuery(query)) {

#### 

#### while(rs.next()) {

#### System.out.printf("ID: %d, Name: %s, Price: %.2f%n",

#### rs.getInt("id"),

#### rs.getString("name"),

#### rs.getDouble("price"));

#### }

#### } catch (SQLException e) {

#### System.err.println("Database error: " + e.getMessage());

#### }

#### }

#### public static void main(String[] args) {

#### String url = "jdbc:mysql://localhost:3306/mydb";

#### String user = "username";

#### String password = "password";

#### 

#### queryDatabase(url, user, password);

#### }

#### }

#### 

#### Outing :

#### 

#### d)

#### public class ExceptionExample {

#### public static void main(String[] args) {

#### try {

#### // Code that may throw an exception

#### int number = 10;

#### int result = number / 0; // This will cause ArithmeticException

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

#### } catch (ArithmeticException e) {

#### // Handling the exception

#### System.out.println("Error: Cannot divide by zero!");

#### } finally {

#### // This block always runs

#### System.out.println("This block runs no matter what.");

#### }

#### System.out.println("Program continues after exception handling.");

#### }

#### }

#### Output :

#### 

#### 17)

#### a)

#### import java.io.\*;

#### import java.util.Properties;

#### 

#### public class ConfigManager {

#### private Properties properties;

#### private String configFile;

#### 

#### public ConfigManager(String filename) {

#### this.configFile = filename;

#### this.properties = new Properties();

#### loadConfig();

#### }

#### 

#### private void loadConfig() {

#### try (InputStream input = new FileInputStream(configFile)) {

#### properties.load(input);

#### } catch (IOException e) {

#### System.err.println("Error loading config: " + e.getMessage());

#### }

#### }

#### 

#### public String getProperty(String key) {

#### return properties.getProperty(key);

#### }

#### 

#### public void setProperty(String key, String value) {

#### properties.setProperty(key, value);

#### saveConfig();

#### }

#### 

#### private void saveConfig() {

#### try (OutputStream output = new FileOutputStream(configFile)) {

#### properties.store(output, "Updated configuration");

#### } catch (IOException e) {

#### System.err.println("Error saving config: " + e.getMessage());

#### }

#### }

#### public static void main(String[] args) {

#### ConfigManager config = new ConfigManager("config.properties");

#### 

#### 

#### config.setProperty("database.url", "jdbc:mysql://localhost:3306/mydb");

#### config.setProperty("database.user", "admin");

#### 

#### System.out.println("Database URL: " + config.getProperty("database.url"));

#### System.out.println("Database User: " + config.getProperty("database.user"));

#### }

#### }

#### 

#### Output :

#### 

#### b)

#### import java.io.\*;

#### import java.util.\*;

#### public class CsvProcessor {

#### public static List<Map<String, String>> readCsv(String filename) throws IOException {

#### List<Map<String, String>> records = new ArrayList<>();

#### try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

#### String[] headers = br.readLine().split(",");

#### String line;

#### while((line = br.readLine()) != null) {

#### String[] values = line.split(",");

#### Map<String, String> record = new LinkedHashMap<>();

#### for(int i = 0; i < headers.length && i < values.length; i++) {

#### record.put(headers[i].trim(), values[i].trim());

#### }

#### records.add(record);

#### }

#### }

#### return records;

#### }

#### 

#### public static void writeCsv(String filename, List<Map<String, String>> data) throws IOException {

#### if(data.isEmpty()) return;

#### 

#### try (PrintWriter pw = new PrintWriter(new FileWriter(filename))) {

#### String[] headers = data.get(0).keySet().toArray(new String[0]);

#### pw.println(String.join(",", headers));

#### 

#### for(Map<String, String> row : data) {

#### String[] values = new String[headers.length];

#### for(int i = 0; i < headers.length; i++) {

#### values[i] = row.get(headers[i]);

#### }

#### pw.println(String.join(",", values));

#### }

#### }

#### }

#### public static void main(String[] args) {

#### try {

#### // Create sample data

#### List<Map<String, String>> data = new ArrayList<>();

#### Map<String, String> row1 = new LinkedHashMap<>();

#### row1.put("Name", "Alice");

#### row1.put("Age", "30");

#### row1.put("Occupation", "Engineer");

#### data.add(row1);

#### 

#### Map<String, String> row2 = new LinkedHashMap<>();

#### row2.put("Name", "Bob");

#### row2.put("Age", "25");

#### row2.put("Occupation", "Designer");

#### data.add(row2);

#### 

#### writeCsv("people.csv", data);

#### System.out.println("CSV file created successfully");

#### 

#### List<Map<String, String>> records = readCsv("people.csv");

#### System.out.println("\nRecords read from CSV:");

#### for(Map<String, String> record : records) {

#### System.out.println(record);

#### }

#### } catch (IOException e) {

#### System.err.println("Error processing CSV: " + e.getMessage());

#### }

#### }

#### }

#### Output :

#### 

#### c)

#### import java.io.\*;

#### import java.time.\*;

#### import java.time.format.\*;

#### import java.util.regex.\*;

#### public class LogAnalyzer {

#### private static final Pattern LOG\_PATTERN = Pattern.compile(

#### "\\[(.\*?)\\] \\[(.\*?)\\] (.\*)");

#### private static final DateTimeFormatter DATE\_FORMAT =

#### DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

#### 

#### public static void analyzeLogFile(String filename) throws IOException {

#### try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {

#### String line;

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

#### Matcher matcher = LOG\_PATTERN.matcher(line);

#### if(matcher.matches()) {

#### LocalDateTime timestamp = LocalDateTime.parse(matcher.group(1), DATE\_FORMAT);

#### String level = matcher.group(2);

#### String message = matcher.group(3);

#### 

#### System.out.printf("[%s] %s: %s%n",

#### timestamp.format(DateTimeFormatter.ISO\_LOCAL\_TIME),

#### level,

#### message);

#### }

#### }

#### }

#### }

#### public static void main(String[] args) {

#### try (PrintWriter writer = new PrintWriter("app.log")) {

#### writer.println("[2023-11-15 08:30:45] [INFO] Application started");

#### writer.println("[2023-11-15 08:31:20] [WARNING] Low memory detected");

#### writer.println("[2023-11-15 08:32:10] [ERROR] Failed to connect to database");

#### } catch (FileNotFoundException e) {

#### System.err.println("Could not create log file: " + e.getMessage());

#### return;

#### }

#### 

#### try {

#### System.out.println("Analyzing log file:");

#### analyzeLogFile("app.log");

#### } catch (IOException e) {

#### System.err.println("Error analyzing log: " + e.getMessage());

#### }

#### }

#### }

#### Output :

#### 

#### d)

#### import java.io.\*;

#### public class EmployeeRecord implements Serializable {

#### private static final long serialVersionUID = 1L;

#### private String name;

#### private int id;

#### private transient String password; // Won't be serialized

#### 

#### public EmployeeRecord(String name, int id, String password) {

#### this.name = name;

#### this.id = id;

#### this.password = password;

#### }

#### 

#### public String toString() {

#### return "EmployeeRecord [name=" + name + ", id=" + id + "]";

#### }

#### public static void main(String[] args) {

#### EmployeeRecord emp1 = new EmployeeRecord("John Doe", 1001, "secret123");

#### EmployeeRecord emp2 = new EmployeeRecord("Jane Smith", 1002, "password456");

#### 

#### try (ObjectOutputStream oos1 = new ObjectOutputStream(new FileOutputStream("employee1.dat"));

#### ObjectOutputStream oos2 = new ObjectOutputStream(new FileOutputStream("employee2.dat"))) {

#### 

#### oos1.writeObject(emp1);

#### oos2.writeObject(emp2);

#### System.out.println("Employees serialized to files successfully");

#### 

#### } catch (IOException e) {

#### System.err.println("Error during serialization: " + e.getMessage());

#### return;

#### }

#### 

#### try (ObjectInputStream ois1 = new ObjectInputStream(new FileInputStream("employee1.dat"));

#### ObjectInputStream ois2 = new ObjectInputStream(new FileInputStream("employee2.dat"))) {

#### 

#### EmployeeRecord restored1 = (EmployeeRecord) ois1.readObject();

#### EmployeeRecord restored2 = (EmployeeRecord) ois2.readObject();

#### 

#### System.out.println("\nDeserialized Employees:");

#### System.out.println(restored1);

#### System.out.println(restored2);

#### 

#### } catch (IOException | ClassNotFoundException e) {

#### System.err.println("Error during deserialization: " + e.getMessage());

#### }

#### }

#### }

#### Output :

#### 