OOP Problem Statement

1. Create the CaesarCipher class with the following parts:

* Private fields for the alphabet and shiftedAlphabet
* Write a constructor CaesarCipher that has one int parameter key. This method should initialize all the private fields of the class.
* Write an encrypt method that has one String parameter named input. This method returns a String that is the input encrypted using shiftedAlphabet.
* Write a decrypt method that has one String parameter named input. This method returns a String that is the encrypted String decrypted using the key associated with this CaesarCipher object. One way to do this is to create another private field mainKey, which is initialized to be the value of key. Then you can create a CaesarCipher object within decrypt: CaesarCipher cc = new CaesarCipher(26 - mainKey); and call cc.encrypt(input).
* Create the TestCaesarCipher class with the following parts:
* Create a CaesarCipher object with key 18, encrypt the String read in using the Scanner class object, print the encrypted String, and decrypt the encrypted String using the decrypt method.

public class Example1 {

private int shift;

Example1(int key) {

shift = key % 26;

}

public String encrypt(String s) {

StringBuilder res = new StringBuilder();

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

char ch = s.charAt(i);

if (Character.isUpperCase(ch)) {

res.append((char) ('A' + (ch - 'A' + shift) % 26));

} else if (Character.isLowerCase(ch)) {

res.append((char) ('a' + (ch - 'a' + shift) % 26));

} else {

res.append(ch);

}

}

return res.toString();

}

public String decrypt(String s) {

Example1 cc = new Example1(26 - shift);

return cc.encrypt(s);

}

public static void main(String[] args) {

Example1 cc = new Example1(18);

String en = cc.encrypt("Hello");

String de = cc.decrypt(en);

System.out.println("Encrypted String: " + en);

System.out.println("Decrypted String: " + de);

}

}

1. Create the URLFinder class with the following parts:

· Private fields for the url

· Write a constructor URLFinder that has one int parameter url. This method should initialize all the private fields of the class.

* Write an urlChecker method that has one String parameter named inputUrl. This method returns a Boolean “true” for valid url.
* Create the TestURLFinder class with the following parts:
* create a URLFinder object with String read in using the Scanner class

import java.util.\*;

class UrlFinder {

private String url;

public UrlFinder(String s) {

url = s;

}

// more complex url matching can be done using regex

public boolean urlCheck(String s) {

return url.equals(s);

}

}

public class Example2 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the original URL:");

String s = sc.nextLine();

UrlFinder u = new UrlFinder(s);

System.out.println("Enter the URL to check:");

String str = sc.nextLine();

System.out.println("The URLs are the same: " + u.urlCheck(str));

}

}

1. Perform the following operations on given matrix

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

M =

1. Transpose
2. Determinant
3. Inverse

import java.util.Scanner;

public class Matrix3x3Operations {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

double[][] matrix = new double[3][3];

// Input matrix

System.out.println("Enter elements of 3x3 matrix:");

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

for (int j = 0; j < 3; j++)

matrix[i][j] = sc.nextDouble();

System.out.println("\nOriginal Matrix:");

printMatrix(matrix);

// Transpose

double[][] transpose = transpose(matrix);

System.out.println("\nTranspose:");

printMatrix(transpose);

// Determinant

double det = determinant(matrix);

System.out.println("\nDeterminant: " + det);

// Inverse

if (det != 0) {

double[][] inverse = inverse(matrix, det);

System.out.println("\nInverse:");

printMatrix(inverse);

} else {

System.out.println("\nMatrix is not invertible (determinant is zero).");

}

sc.close();

}

public static void printMatrix(double[][] matrix) {

for (double[] row : matrix) {

for (double val : row)

System.out.printf("%.2f\t", val);

System.out.println();

}

}

public static double[][] transpose(double[][] matrix) {

double[][] transposed = new double[3][3];

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

for (int j = 0; j < 3; j++)

transposed[j][i] = matrix[i][j];

return transposed;

}

public static double determinant(double[][] m) {

return m[0][0] \* (m[1][1]\*m[2][2] - m[1][2]\*m[2][1])

- m[0][1] \* (m[1][0]\*m[2][2] - m[1][2]\*m[2][0])

+ m[0][2] \* (m[1][0]\*m[2][1] - m[1][1]\*m[2][0]);

}

public static double[][] inverse(double[][] m, double det) {

double[][] inv = new double[3][3];

inv[0][0] = (m[1][1]\*m[2][2] - m[1][2]\*m[2][1]) / det;

inv[0][1] = (m[0][2]\*m[2][1] - m[0][1]\*m[2][2]) / det;

inv[0][2] = (m[0][1]\*m[1][2] - m[0][2]\*m[1][1]) / det;

inv[1][0] = (m[1][2]\*m[2][0] - m[1][0]\*m[2][2]) / det;

inv[1][1] = (m[0][0]\*m[2][2] - m[0][2]\*m[2][0]) / det;

inv[1][2] = (m[0][2]\*m[1][0] - m[0][0]\*m[1][2]) / det;

inv[2][0] = (m[1][0]\*m[2][1] - m[1][1]\*m[2][0]) / det;

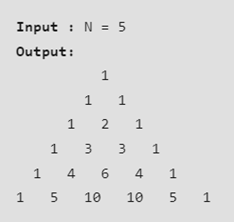
inv[2][1] = (m[0][1]\*m[2][0] - m[0][0]\*m[2][1]) / det;

inv[2][2] = (m[0][0]\*m[1][1] - m[0][1]\*m[1][0]) / det;

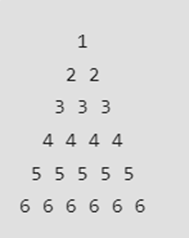
return inv;

}

}

1. A) Write a Program to Print the Pascal’s and Number Triangle in Java

4) B) Write a Java program to print the number triangle



import java.util.\*;

public class Example4A {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

for(int i = 0; i<n; i++){

for(int j = 1; j<=n-i-1; j++){

System.out.print(" ");

}

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

System.out.print(i+1 + " ");

}

System.out.println();

}

}

}

import java.util.Scanner;

public class CenteredNumberTriangle {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int rows = sc.nextInt();

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

// Print leading spaces

for (int s = 1; s <= rows - i; s++) {

System.out.print(" ");

}

// Print number i, i times with space

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

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

}

System.out.println();

}

sc.close();

}

}

5)

1. An ArrayList consists of 1-25 numbers. Write a Java program to remove prime numbers from an ArrayList using an iterator

import java.util.\*;

public class Example5A {

public static boolean isPrime(int n){

if(n<=2) return false;

for(int i = 2; i\*i<=n; i++){

if(n%i==0){

return false;

}

}

return true;

}

public static void main(String[] args){

ArrayList<Integer> list = new ArrayList<Integer>();

for(int i = 1; i<=25; i++){

list.add(i);

}

Iterator<Integer> it1 = list.iterator();

while(it1.hasNext()){

int i = it1.next();

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

}

for(int i = 0; i<list.size(); i++){

if(isPrime(list.get(i))){

list.remove(i);

}

}

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

int i = it.next();

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

}

}

}

1. Write a Java program to

a. Create and traverse (or iterate) an ArrayList using a for-loop, iterator, and advance for-loop.

b. Check if the element(value) exists in the ArrayList?

C. Add element at the particular index of the ArrayList?

import java.util.\*;

public class Example5B {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

ArrayList<Integer> list = new ArrayList<Integer>();

int n = sc.nextInt();

for(int i = 0; i<n; i++){

int x = sc.nextInt();

list.add(x);

}

// for loop

for(int i = 0; i<list.size(); i++){

System.out.print(list.get(i)+" ");

}

System.out.println();

// advanced for loop

for(int i : list){

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

}

System.out.println();

// iterator

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

int i = it.next();

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

}

System.out.println();

int check = sc.nextInt();

System.out.println("The value check exists in the arraylist " + list.contains(check));

int newElm = sc.nextInt();

int idx = sc.nextInt();

System.out.println("list after adding element at given index ");

list.add(idx, newElm);

Iterator<Integer> it1 = list.iterator();

while(it1.hasNext()){

int i = it1.next();

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

}

}

}

6)

* Write a Java program to find the largest and smallest elements in an array of integers.

Implement a function to reverse an array in place.

Given two arrays, write a method to merge them into a single sorted array.  
  
import java.util.\*;

public class Example5B {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

ArrayList<Integer> list = new ArrayList<Integer>();

int n = sc.nextInt();

for(int i = 0; i<n; i++){

int x = sc.nextInt();

list.add(x);

}

// for loop

for(int i = 0; i<list.size(); i++){

System.out.print(list.get(i)+" ");

}

System.out.println();

// advanced for loop

for(int i : list){

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

}

System.out.println();

// iterator

Iterator<Integer> it = list.iterator();

while(it.hasNext()){

int i = it.next();

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

}

System.out.println();

int check = sc.nextInt();

System.out.println("The value check exists in the arraylist " + list.contains(check));

int newElm = sc.nextInt();

int idx = sc.nextInt();

System.out.println("list after adding element at given index ");

list.add(idx, newElm);

Iterator<Integer> it1 = list.iterator();

while(it1.hasNext()){

int i = it1.next();

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

}

}

}

7)

Write a program to check if a given string is a palindrome.

Implement a function to count the occurrences of a specific character in a string.

Write a program to remove all whitespace from a string.  
import java.util.HashMap;

import java.util.Scanner;

public class Example7 {

public static boolean isPalindrome(String s) {

int i = 0;

int j = s.length() - 1;

while (i <= j) {

if (s.charAt(i) != s.charAt(j)) {

return false;

}

i++;

j--;

}

return true;

}

public static void printFreq(String s) {

HashMap<Character, Integer> mp = new HashMap<>();

for (char c : s.toCharArray()) {

mp.put(c, mp.getOrDefault(c, 0) + 1);

}

System.out.println("The frequency of different characters is:");

for (char c : mp.keySet()) {

System.out.println("Key: " + c + " Value: " + mp.get(c));

}

}

public static String removeSpace(String s) {

StringBuilder builder = new StringBuilder();

for (char c : s.toCharArray()) {

if (c != ' ') {

builder.append(c);

}

}

return builder.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string to check palindrome and frequency: ");

String str = sc.nextLine();

System.out.println("The given string is a palindrome: " + isPalindrome(str));

printFreq(str);

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

String strw = sc.nextLine();

System.out.println("String after removing whitespaces: " + removeSpace(strw));

sc.close();

}

}

8) Design a class BankAccount with methods for deposit, withdraw, and check balance. Implement exception handling for insufficient funds during withdrawal.  
  
class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String m) {

super(m);

}

}

class BankAccount {

private int balance;

public BankAccount(int b) {

balance = b;

}

public void deposit(int d) {

balance += d;

}

public void withdraw(int d) throws InsufficientBalanceException {

if (d > balance) {

throw new InsufficientBalanceException("Insufficient balance to carry out transaction");

}

balance -= d;

}

}

public class Example8{

public static void main(String[] args) {

BankAccount b = new BankAccount(1000);

try {

b.withdraw(100);

System.out.println("Withdrew 100 successfully");

b.deposit(1000);

b.withdraw(2000);

System.out.println("Withdrew 2000 successfully");

} catch (InsufficientBalanceException e) {

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

}

}

}

9) Write a program to read data from a text file using IO Stream class and display the number of characters and words on the console.  
  
import java.io.FileReader;

import java.io.IOException;

public class Example9 {

public static void main(String[] args) {

int count = 0;

try(FileReader fr = new FileReader("text.txt")){

int c = fr.read();

while(fr.ready()){

System.out.println((char)c);

if((char)c != ' ') {

count++;

}

c = fr.read();

}

System.out.println("Total characters in the file " + count);

System.out.println();

}catch (IOException e){

System.out.println(e.getMessage());

}

}

}

10)

1. Write a program to find the greatest common divisor (GCD) of two numbers.
2. Write a program to convert a decimal number to binary.

import java.util.Scanner;

public class Example10 {

public int GCD(int a, int b){

if(a==0){

return b;

}

if (a > b) {

GCD(b,a);

}

return GCD(b%a,a);

}

public String decToBin(int n){

int x = n;

StringBuilder s = new StringBuilder();

while(x>0){

if(x%2==1){

s.append('1');

}else{

s.append('0');

}

x /=2;

}

s.reverse();

return s.toString();

}

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int a = sc.nextInt();

int b = sc.nextInt();

Example10 g = new Example10();

System.out.println("The gcd of the given two numbers is " + g.GCD(a,b));

int n = sc.nextInt();

System.out.println("The binary of the given decimal is " + g.decToBin(n));

}

}

11) Create the CarAssembly class which implements Runnable interface with the following parts:

* Private fields for the componentName(String) and timeToPrepare(int)
* Write a constructor CarAssembly that has two parameters componentName and timeToPrepare . This method should initialize all the private fields of the class.
* Write an run method that has sleep method which takes timeToPrepare parameter. The sleep method is invoveked between two print statements componentName is preparing & componentName is ready.
* Components names and their preparation times are as follows
* Engine-3000, Body-4000, Wheels-5000

Create three threads namely engineThread, bodyThread, wheelThread and use Join method for Sysnchronization.

class CarAssembly implements Runnable{

private String name;

private int sleepTime;

public CarAssembly(String n, int time){

name = n;

sleepTime = time;

}

public void run(){

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

try {

Thread.sleep(sleepTime);

}catch(InterruptedException e){

System.out.println("Thread is interrupted " + e.getMessage());

}

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

}

}

public class Example11 {

public static void main(String[] args) {

System.out.println("Car Assembly started");

CarAssembly engine = new CarAssembly("engine", 3000);

CarAssembly body = new CarAssembly("body", 4000);

CarAssembly wheels = new CarAssembly("wheels",5000);

Thread engineThread = new Thread(engine);

Thread bodyThread = new Thread(body);

Thread wheelThread = new Thread(wheels);

engineThread.start();

bodyThread.start();

wheelThread.start();

try{

engineThread.join();

bodyThread.join();

wheelThread.join();

}catch(InterruptedException e){

System.out.println("Thread is interrupted " + e.getMessage());

}

System.out.println("Car Assembly finished");

}

}

12) Design a Java program to manage an ArrayList of integers that supports dynamic insertion at any position, deletion, updating values, and efficiently computing the sum of elements between two given indices after each modification.

|  |  |
| --- | --- |
| Sample Input | Sample Output |
| insert 0 5  insert 1 10  insert 1 15  update 2 20  sum 0 2  delete 1  sum 0 1 | [5]  [5, 10]  [5, 15, 10]  [5, 15, 20]  25  [5, 20]  25 |

import java.util.ArrayList;

import java.util.Scanner;

public class Example12 {

private ArrayList<Integer> arr;

public Example12(){

arr = new ArrayList<Integer>();

}

public void insertElm(int idx, int elm){

arr.add(idx, elm);

}

public void update(int idx, int elm){

arr.set(idx, elm);

}

public int sum(int i, int j){

int sum = 0;

for(int k = i; k<=j; k++){

sum+=arr.get(k);

}

return sum;

}

public void delete(int idx){

arr.remove(idx);

}

public static void main(String[] args){

Scanner sc =new Scanner(System.in);

Example12 a = new Example12();

while(true){

int x;

x = sc.nextInt();

if(x==0){

break;

}else if(x==1){

int idx = sc.nextInt();

int elm = sc.nextInt();

a.insertElm(idx, elm);

}else if(x==2){

int idx = sc.nextInt();

int elm = sc.nextInt();

a.update(idx, elm);

}else if(x==3){

int idx = sc.nextInt();

a.delete(idx);

}else{

int i = sc.nextInt();

int j = sc.nextInt();

a.insertElm(i, j);

}

}

sc.close();

}

}

13) Write a Java program using HashMap to add, remove, and track the frequency of words. You also need to find the most frequent word, and if there’s a tie, return the smallest word alphabetically. The program should handle up to 10^5 operations efficiently.

|  |  |
| --- | --- |
| Sample Input | Sample Output |
| add apple  add banana  add apple  remove apple  query apple  mostFrequent | Frequency of ‘apple’: 1  Most frequent word: banana |

import java.util.HashMap;

import java.util.Scanner;

public class Example13 {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

for(int i = 0; i<n; i++){

String s = sc.nextLine();

mp.put(s, mp.getOrDefault(s, 0)+1);

}

int mxFreq = 0;

String res = "";

for(String s : mp.keySet()){

if(mp.get(s)>mxFreq){

mxFreq = mp.get(s);

res = s;

}

}

System.out.println("Max freq " + mxFreq);

System.out.println("Max Freq elem " + res);

}

}

14) Design and implement a multi-threaded banking system in Java that simulates multiple users performing concurrent transactions on shared bank accounts. Each transaction can be a deposit, withdrawal, or transfer between accounts.  
  
class BankAccountMulti {

private int accountNumber;

private double balance;

public BankAccountMulti(int accountNumber, double openingBalance) {

this.accountNumber = accountNumber;

this.balance = openingBalance;

}

public synchronized void deposit(double amount) {

balance += amount;

System.out.println(Thread.currentThread().getName() + ": Deposited " + amount + " | Balance: " + balance);

}

public synchronized void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

System.out.println(Thread.currentThread().getName() + ": Withdrew " + amount + " | Balance: " + balance);

} else {

System.out.println(Thread.currentThread().getName() + ": Insufficient funds to withdraw " + amount);

}

}

public void transferTo(BankAccountMulti target, double amount) {

synchronized (this) {

synchronized (target) {

if (balance >= amount) {

this.balance -= amount;

target.balance += amount;

System.out.println(Thread.currentThread().getName() + ": Transferred " + amount + " to Account " + target.accountNumber);

} else {

System.out.println(Thread.currentThread().getName() + ": Transfer failed due to insufficient balance.");

}

}

}

}

public double getBalance() {

return balance;

}

}

class TransactionTask implements Runnable {

private BankAccountMulti acc1;

private BankAccountMulti acc2;

public TransactionTask(BankAccountMulti acc1, BankAccountMulti acc2) {

this.acc1 = acc1;

this.acc2 = acc2;

}

@Override

public void run() {

acc1.deposit(1000);

acc1.withdraw(500);

acc1.transferTo(acc2, 300);

}

}

public class Example14 {

public static void main(String[] args) {

BankAccountMulti accountA = new BankAccountMulti(1001, 5000);

BankAccountMulti accountB = new BankAccountMulti(1002, 3000);

BankAccountMulti accountC = new BankAccountMulti(1003, 2000);

Thread t1 = new Thread(new TransactionTask(accountA, accountB), "User-1");

Thread t2 = new Thread(new TransactionTask(accountB, accountC), "User-2");

t1.start();

t2.start();

}

}

15)Design a shopping cart program where users can add items, apply discount codes, and check out. Use custom exceptions to handle scenarios like invalid coupon codes, out-of-stock items, and negative quantity inputs. (Exception handling)

import java.util.ArrayList;

import java.util.List;

class InvalidCouponException extends Exception {

public InvalidCouponException(String message) {

super(message);

}

}

class OutOfStockException extends Exception {

public OutOfStockException(String message) {

super(message);

}

}

class NegativeQuantityException extends Exception {

public NegativeQuantityException(String message) {

super(message);

}

}

class Item {

String name;

double price;

int stock;

public Item(String name, double price, int stock) {

this.name = name;

this.price = price;

this.stock = stock;

}

}

class ShoppingCart {

private List<Item> cartItems;

private double totalAmount;

public ShoppingCart() {

cartItems = new ArrayList<>();

totalAmount = 0.0;

}

public void addItem(Item item, int quantity) throws OutOfStockException, NegativeQuantityException {

if (quantity <= 0) {

throw new NegativeQuantityException("Quantity cannot be zero or negative.");

}

if (quantity > item.stock) {

throw new OutOfStockException("Item " + item.name + " is out of stock.");

}

cartItems.add(item);

totalAmount += item.price \* quantity;

item.stock -= quantity;

}

public void applyCoupon(String couponCode) throws InvalidCouponException {

if (couponCode.equals("DISCOUNT10")) {

totalAmount \*= 0.9;

System.out.println("Coupon applied successfully! 10% discount applied.");

} else if (couponCode.equals("DISCOUNT20")) {

totalAmount \*= 0.8;

System.out.println("Coupon applied successfully! 20% discount applied.");

} else {

throw new InvalidCouponException("Invalid coupon code.");

}

}

public void checkout() {

System.out.println("Checkout complete. Total amount: $" + totalAmount);

}

public void displayCart() {

System.out.println("Items in your cart:");

for (Item item : cartItems) {

System.out.println("Item: " + item.name + ", Price: $" + item.price);

}

}

public double getTotalAmount() {

return totalAmount;

}

}

public class Example15 {

public static void main(String[] args) {

Item item1 = new Item("Laptop", 1000.0, 5);

Item item2 = new Item("Smartphone", 500.0, 3);

Item item3 = new Item("Headphones", 100.0, 10);

ShoppingCart cart = new ShoppingCart();

try {

cart.addItem(item1, 2); // Add 2 Laptops

cart.addItem(item2, 1); // Add 1 Smartphone

cart.addItem(item3, 3); // Add 3 Headphones

cart.displayCart();

cart.applyCoupon("DISCOUNT10");

cart.checkout();

} catch (OutOfStockException | NegativeQuantityException | InvalidCouponException e) {

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

}

}

}

16) Design an interface Vehicle with methods startRide(), endRide(), and calculateFare(int distance). Implement classes like Bike, Auto, and Cab, where each vehicle calculates fares differently. Use a PricingStrategy interface to dynamically adjust fares based on conditions like peak hours and holidays.  
  
interface Vehicle {

void startRide();

void endRide();

float calculateFare(int dist);

}

interface PricingStrategy {

float getMultiplier(int dist);

}

class Auto implements PricingStrategy, Vehicle {

public void startRide() {

System.out.println("Starting Auto");

}

public void endRide() {

System.out.println("Stopping Auto");

}

public float calculateFare(int dist) {

return getMultiplier(dist) \* dist;

}

public float getMultiplier(int dist) {

return (isPeakHours() || isHoliday()) ? 2.5f : 2.0f;

}

private boolean isPeakHours() {

// Simulating peak hours. You can replace this with actual time-based checks.

return true; // Assuming it's peak hour for simplicity

}

private boolean isHoliday() {

// Simulating holiday. You can replace this with actual holiday checking logic.

return false; // Assuming it's not a holiday for simplicity

}

}

class Bus implements PricingStrategy, Vehicle {

public void startRide() {

System.out.println("Starting Bus");

}

public void endRide() {

System.out.println("Stopping Bus");

}

public float calculateFare(int dist) {

return getMultiplier(dist) \* dist;

}

public float getMultiplier(int dist) {

return (isPeakHours() || isHoliday()) ? 1.5f : 0.5f;

}

private boolean isPeakHours() {

// Simulating peak hours.

return false;

}

private boolean isHoliday() {

// Simulating holiday.

return false;

}

}

class Cab implements PricingStrategy, Vehicle {

public void startRide() {

System.out.println("Starting Cab");

}

public void endRide() {

System.out.println("Stopping Cab");

}

public float calculateFare(int dist) {

return getMultiplier(dist) \* dist;

}

public float getMultiplier(int dist) {

return (isPeakHours() || isHoliday()) ? 3.0f : 4.0f;

}

private boolean isPeakHours() {

// Simulating peak hours.

return false;

}

private boolean isHoliday() {

// Simulating holiday.

return false;

}

}

public class Example16 {

public static void main(String[] args) {

// Create instances of each vehicle

Vehicle auto = new Auto();

Vehicle bus = new Bus();

Vehicle cab = new Cab();

// Simulate a ride for each vehicle

int distance = 10; // distance for fare calculation

// Testing Auto

System.out.println("Auto Fare for " + distance + " km: " + auto.calculateFare(distance));

auto.startRide();

auto.endRide();

// Testing Bus

System.out.println("Bus Fare for " + distance + " km: " + bus.calculateFare(distance));

bus.startRide();

bus.endRide();

// Testing Cab

System.out.println("Cab Fare for " + distance + " km: " + cab.calculateFare(distance));

cab.startRide();

cab.endRide();

}

}

17) Design a University Staff Management System using a base class Staff and derived classes Professor, AdministrativeStaff, and MaintenanceStaff.  
 Override methods like displayDetails() and calculateBonus() differently in each subclass using polymorphism.  
 Use a list of base class pointers or references to manage multiple staff objects and demonstrate runtime polymorphism.  
 (Advanced) Implement a promote() method with different behaviors in each subclass.

import java.util.\*;

// Base class

abstract class Staff {

protected String name;

protected int id;

protected double salary;

public Staff(String name, int id, double salary) {

this.name = name;

this.id = id;

this.salary = salary;

}

public abstract void displayDetails();

public abstract double calculateBonus();

public abstract void promote();

}

// Professor class

class Professor extends Staff {

private String subject;

public Professor(String name, int id, double salary, String subject) {

super(name, id, salary);

this.subject = subject;

}

@Override

public void displayDetails() {

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

}

@Override

public double calculateBonus() {

return 0.2 \* salary; // 20% bonus

}

@Override

public void promote() {

salary += 5000;

System.out.println(name + " promoted! New salary: " + salary);

}

}

// AdministrativeStaff class

class AdministrativeStaff extends Staff {

private String department;

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

super(name, id, salary);

this.department = department;

}

@Override

public void displayDetails() {

System.out.println("Admin Staff: " + name + ", ID: " + id + ", Department: " + department + ", Salary: " + salary);

}

@Override

public double calculateBonus() {

return 0.1 \* salary; // 10% bonus

}

@Override

public void promote() {

salary += 3000;

System.out.println(name + " promoted to Senior Admin! New salary: " + salary);

}

}

// MaintenanceStaff class

class MaintenanceStaff extends Staff {

private String shift;

public MaintenanceStaff(String name, int id, double salary, String shift) {

super(name, id, salary);

this.shift = shift;

}

@Override

public void displayDetails() {

System.out.println("Maintenance Staff: " + name + ", ID: " + id + ", Shift: " + shift + ", Salary: " + salary);

}

@Override

public double calculateBonus() {

return 0.05 \* salary; // 5% bonus

}

@Override

public void promote() {

salary += 1500;

System.out.println(name + " promoted to Supervisor! New salary: " + salary);

}

}

// Driver class

public class Example17 {

public static void main(String[] args) {

List<Staff> staffList = new ArrayList<>();

staffList.add(new Professor("Dr. Sharma", 101, 80000, "Physics"));

staffList.add(new AdministrativeStaff("Mr. Mehta", 102, 50000, "Admissions"));

staffList.add(new MaintenanceStaff("Ravi", 103, 30000, "Night"));

// Runtime polymorphism demonstration

for (Staff staff : staffList) {

staff.displayDetails();

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

staff.promote();

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

}

}

}

18)

Create a class Rectangle with attributes length and breadth. Write a constructor that uses this keyword to initialize the attributes. Also, create a method compareArea(Rectangle r) that compares the area of two rectangles.  
  
 public class Example18A {

int l;

int b;

public Example18A(int l, int b){

this.l = l;

this.b = b;

}

public int calculateArea(){

return this.l\*this.b;

}

public boolean compareArea(Example18A r){

return this.calculateArea()>r.calculateArea();

}

public class Main {

public static void main(String[] args) {

Example18A rect1 = new Example18A(10, 5);

Example18A rect2 = new Example18A(7, 8);

System.out.println("Area of rect1: " + rect1.calculateArea());

System.out.println("Area of rect2: " + rect2.calculateArea());

if (rect1.compareArea(rect2)) {

System.out.println("rect1 has a larger area than rect2");

} else {

System.out.println("rect1 does not have a larger area than rect2");

}

}

}

}

1. Create a class MathOperations with:

* A static method square(int n) that returns the square of a number.
* An instance method cube(int n) that returns the cube of a number. Write a Java program that demonstrates calling both static and instance methods properly.

public class Example18B {

public static int square(int n){

return n\*n;

}

public int cube(int n){

return n\*n\*n;

}

public static void main(String[] args) {

int sq = Example18B.square(8);

Example18B m = new Example18B();

int c = m.cube(8);

System.out.println(sq);

System.out.println(c);

}

}

1. Create a Java program for a seating system in a cinema hall represented by a 2D array (rows × columns).

* Mark all seats as available (e.g., 0) initially.
* Allow the user to book seats by marking them as booked (e.g., 1).
* Display the current seat map (matrix form).
* Add functionality to check if a given seat is available before booking.

public class Example19 {

int[][] arr;

int rows;

int cols;

public Example19(int r, int c){

rows = r;

cols = c;

arr = new int[r][c];

for(int i = 0; i<r; i++){

for(int j = 0; j<c; j++){

arr[i][j] = 0;

}

}

}

public void bookSeat(int r, int c){

if(r<=0 || r>rows || c<=0 || c>cols){

System.out.println("Booking failed");

return;

}

r--;

c--;

arr[r][c] = 1;

}

public void display(){

for(int i = 0; i<rows; i++){

for(int j = 0; j<cols; j++){

System.out.print(arr[i][j]+ " ");

}

System.out.println();

}

}

public boolean isBooked(int r, int c){

if(r<=0 || r>rows || c<=0 || c>cols){

System.out.println("Booking failed");

return false;

}

r--;

c--;

return arr[r][c] == 1;

}

public static void main(String[] args) {

Example19 hall = new Example19(5, 5);

System.out.println("Initial Seating Arrangement:");

hall.display();

System.out.println("\nBooking seats at (2, 3), (1, 1), and (5, 5):");

hall.bookSeat(2, 3);

hall.bookSeat(1, 1);

hall.bookSeat(5, 5);

hall.display();

System.out.println("\nAttempting to book invalid seat at (6, 7):");

hall.bookSeat(6, 7);

System.out.println("\nCheck if seats are booked:");

System.out.println("Seat (2, 3): " + hall.isBooked(2, 3));

System.out.println("Seat (3, 3): " + hall.isBooked(3, 3));

System.out.println("Seat (6, 7): " + hall.isBooked(6, 7));

}

}

20)

Develop a Java program that takes a paragraph input from the user and:

* Remove all vowels (a, e, i, o, u) from the paragraph using StringBuilder.
* Efficiently update the paragraph after each deletion.
* Finally, display the transformed paragraph along with the count of characters removed.

import java.util.Scanner;

public class Example20 {

public boolean isVowel(char c){

return c=='A' || c=='E' || c=='I' || c=='O' || c=='U' || c=='a' || c=='e' || c=='i' || c=='o' || c=='u';

}

private int count = 0;

public String remVow(String st) {

StringBuilder s = new StringBuilder();

for (char c : st.toCharArray()) {

if (!isVowel(c)) {

s.append(c);

} else {

count++;

}

}

return s.toString();

}

public int getCount() {

return count;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s = sc.nextLine();

Example20 vowRem = new Example20();

String res = vowRem.remVow(s);

System.out.println(res);

System.out.println(vowRem.getCount());

}

}

1. Create a base class `Employee` with attributes: name, id, and basicSalary, along with methods `displayDetails()` and `calculateSalary()`. Derive two subclasses: `Manager` with an additional bonus attribute and `Developer` with a projectAllowance attribute. Override the `calculateSalary()` method in both subclasses to include their respective additional amounts. In the main method, create objects of Manager and Developer, and call `displayDetails()` for each. Demonstrate polymorphism by invoking `calculateSalary()` using base class references and display the total salary.

class Employee1 {

protected String name;

protected int id;

protected double basicSalary;

public Employee1(String name, int id, double basicSalary) {

this.name = name;

this.id = id;

this.basicSalary = basicSalary;

}

public void displayDetails() {

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

System.out.println("ID: " + id);

System.out.println("Basic Salary: " + basicSalary);

}

public double calculateSalary() {

return basicSalary;

}

}

class Manager extends Employee1 {

private double bonus;

public Manager(String name, int id, double basicSalary, double bonus) {

super(name, id, basicSalary);

this.bonus = bonus;

}

@Override

public double calculateSalary() {

return basicSalary + bonus;

}

@Override

public void displayDetails() {

super.displayDetails();

System.out.println("Bonus: " + bonus);

}

}

class Developer extends Employee1 {

private double projectAllowance;

public Developer(String name, int id, double basicSalary, double projectAllowance) {

super(name, id, basicSalary);

this.projectAllowance = projectAllowance;

}

@Override

public double calculateSalary() {

return basicSalary + projectAllowance;

}

@Override

public void displayDetails() {

super.displayDetails();

System.out.println("Project Allowance: " + projectAllowance);

}

}

public class Example21 {

public static void main(String[] args) {

Employee1 emp1 = new Manager("Alice", 101, 50000, 10000);

Employee1 emp2 = new Developer("Bob", 102, 40000, 7000);

emp1.displayDetails();

System.out.println("Total Salary: " + emp1.calculateSalary());

System.out.println();

emp2.displayDetails();

System.out.println("Total Salary: " + emp2.calculateSalary());

}

}

22)

Create a class BankAccount with the following attributes and methods:

Attributes:

accountNumber (String)

balance (double)

Methods:

A constructor that initializes the accountNumber and balance.

A method withdraw(double amount) that:

Throws an ArithmeticException if the withdrawal amount is greater than the current balance.

Throws an IllegalArgumentException if the withdrawal amount is less than or equal to zero.

In the main() method:

Create an object of the BankAccount class with an initial balance.

Use try-catch blocks to handle the following scenarios:

Catch the ArithmeticException and display a message "Insufficient funds for withdrawal."

Catch the IllegalArgumentException and display a message "Invalid withdrawal amount."

After handling the exception, allow the program to continue running.

Display the current balance after each operation.

class BankAccoun {

// same class as example 8

// just throw arithmetic execption instead of custom exception

public void withdraw(int d) throws ArithmeticException {

if (d > balance) {

throw new ArithmeticException("Insufficient balance to carry out transaction");

}

balance -= d;

}

}

public class Example22 {

public static void main(String[] args) {

// same as example 8

try {

// while taking user input handle invalid argument exception

} catch (ArithmeticException e) {

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

} catch (IllegalArgumentException e){

System.out.println("Exception occurred " + e.getMessage());

} catch (Exception e){

System.out.println("Exception occured "+ e.getMessage());

}

}

}

23)

Create a class Printer with a method printNumbers() that prints the numbers from 1 to 10 with a small delay between each number (use Thread.sleep(500) to simulate the delay).

Create two threads:

One thread will call the printNumbers() method and print numbers from 1 to 10.

The second thread will call the printNumbers() method and also print numbers from 1 to 10.

In the main() method:

Create two Printer objects.

Create two threads and start them to execute the printNumbers() method concurrently.

Ensure that the numbers from both threads are printed without any interruption.

* Additional attribute: department (String)

class Printer{

public synchronized void printNumbers(){

for(int i = 1; i<=10; i++){

try{

Thread.sleep(500);

}catch(InterruptedException e){

System.out.println(e.getMessage());

}

System.out.println(Thread.currentThread().getName()+ " " +i);

}

}

}

class PrinterTask implements Runnable{

private Printer printer;

public PrinterTask(Printer p){

printer = p;

}

@Override

public void run() {

printer.printNumbers();

}

}

public class Example23 {

public static void main(String[] args) {

Printer sharedPrinter = new Printer();

Thread t1 = new Thread(new PrinterTask(sharedPrinter), "thread1");

Thread t2 = new Thread(new PrinterTask(sharedPrinter), "thread2");

System.out.println("printing started");

t1.start();

t2.start();

try{

t1.join();

t2.join();

}catch(InterruptedException e){

System.out.println(e.getMessage());

}

System.out.println("printing finished");

}

}

24) Create a Java program to demonstrate access modifiers (private, public, protected, and default).

1. Class Person:  
   * Attributes:  
     + name (private), age (public), address (protected), phoneNumber (default)
   * Methods:  
     + Constructor to initialize all attributes.
     + displayDetails() (public) to display name, age, and address.
     + updatePhoneNumber() (public) to update phoneNumber.
2. Class Employee (extends Person):  
   * Additional attribute: employeeId (public)
   * Override displayDetails() to include employeeId.
3. In the main() method:  
   * Create an Employee object and demonstrate access to attributes and methods using different access modifiers.

class Person{

private String name;

public int age;

protected String address;

int phoneNumer;

public Person(String n, int a, String adr, int phno){

name = n;

age = a;

address = adr;

phoneNumer = phno;

}

public void displayDetails(){

System.out.println("Details");

}

public void updatePhone(int newPh){

phoneNumer = newPh;

}

}

class Employeee extends Person{

public int empId;

public Employeee(String n, int a, String adr, int phno, int id){

super(n, a, adr, phno);

empId = id;

}

@Override

public void displayDetails(){

System.out.println("Details of employee");

System.out.println(age);

System.out.println(empId);

}

}

public class Example24 {

public static void main(String[] args) {

Person p = new Employeee("abc", 19, "street name", 1234567890, 2);

p.displayDetails();

Person p1 = new Person("abc", 19, "street name", 1234567890);

p1.displayDetails();

}

}

25)

1) Create an application for employee management with the following classes:

a) Create an Employee class with following attributes and behaviors:

i) int empId  
 ii) String empName  
 iii) String email  
 iv) String gender  
 v) float salary  
 vi) void GetEmployeeDetails() -> prints employee details

b) Create one more class EmployeeDB with the following attributes and behaviors:  
 i) ArrayList list;  
 ii) boolean addEmployee(Employee e) -> adds the employee object to the collection  
 iii) boolean deleteEmployee(int empId) -> delete the employee object from the collection with the given empid  
 iv) String showPaySlip(int empId) -> returns the payslip of the employee with the given empId  
  
import java.util.ArrayList;

import java.util.Date;

class Employee{

private int empId;

private float salary;

private String email;

private String gender;

private String empName;

public int getEmpId() {

return empId;

}

public float getSalary() {

return salary;

}

public String getEmail() {

return email;

}

public String getGender() {

return gender;

}

public String getEmpName() {

return empName;

}

public void setEmail(String email) {

this.email = email;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public void setSalary(float salary) {

this.salary = salary;

}

public void setEmpName(String empName) {

this.empName = empName;

}

public void setGender(String gender) {

this.gender = gender;

}

public Employee(int id, String name, String e, String g, float s){

empName = name;

salary = s;

email = e;

empId = id;

gender = g;

}

public void getDetails(){

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

System.out.println("EmpId: " + empId + "\n" + "Name: " + empName + "\n" + "Email: " + email + "\n" + "Gender: " + gender + "\n" + "Salary: " + salary + "\n" );

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

}

public String paySlip(){

return "Amount "+ salary + " paid to " + empName;

}

}

class EmployeeDB{

ArrayList<Employee> arr;

public EmployeeDB(){

arr = new ArrayList<Employee>();

}

public boolean addEmp(Employee e){

try {

arr.add(e);

return true;

} catch (Exception ex) {

System.out.println("Error occured");

ex.printStackTrace();

}

return false;

}

public boolean deleteEmp(int id){

try {

for(int i = 0; i<arr.size(); i++){

if(arr.get(i).getEmpId() == id){

arr.remove(i);

return true;

}

}

} catch (Exception ex) {

System.out.println("Error occured");

ex.printStackTrace();

}

return false;

}

public String showPaySlip(int id){

for(int i = 0; i<arr.size(); i++){

if(arr.get(i).getEmpId()==id){

return arr.get(i).paySlip();

}

}

return "";

}

}

public class Example25 {

public static void main(String[] args) {

Employee e = new Employee(1, "Abc", "abc@gmail.com", "male", 40000);

e.getDetails();

EmployeeDB db = new EmployeeDB();

db.addEmp(e);

System.out.println(db.showPaySlip(1));

}

}

26)

You are given a sorted integer array nums in non-decreasing order. Your task is to remove the duplicate elements in-place such that each element appears only once, and return the new length of the modified array.

You must not allocate extra space for another array; you must do this by modifying the input array in-place with O(1) extra memory.

After removing the duplicates, the first part of the array should contain the unique elements, and the remaining elements can be left as any value (underscores \_ or any arbitrary values).

### Example:

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

import java.util.Scanner;

public class Example26 {

public int moveDups(int[] arr){

int idx = 0;

for(int i = 1; i<arr.length; i++){

if(arr[i]!=arr[idx]){

int temp = arr[i];

arr[i] = arr[idx+1];

arr[idx+1] = temp;

idx++;

}

}

return idx+1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

for(int i = 0; i<n; i++){

arr[i] = sc.nextInt();

}

Example26 mv = new Example26();

int ct = mv.moveDups(arr);

for(int i = ct; i<arr.length; i++){

arr[i] = -1;

}

for(int i : arr){

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

}

}

}

27)

Write a class MathOperation which accepts 5 integers through the command line.  
Create an array using these parameters.  
Loop through the array and obtain the sum and average of all the elements and display the result.

Handle various exceptions that may arise such as:

* ArithmeticException
* NumberFormatException

and other relevant exceptions.  
  
  
public class Example27 {

public static void main(String[] args) {

int sum = 0;

try{

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

sum += Integer.parseInt(args[i]);

}

int avg = sum/args.length;

System.out.println(avg);

}catch(NumberFormatException e){

System.out.println("Invalid number format");

e.printStackTrace();

}catch(ArithmeticException e){

System.out.println("Divide by zero");

e.printStackTrace();

}catch(Exception e){

System.out.println("Exception occurred");

e.printStackTrace();

}

}

}

28)

Create a base class named Fruit with the following attributes:

* name (String)
* taste (String)
* size (String)

Define a method eat() in the Fruit class that prints the name and taste of the fruit.

Now, create two subclasses, Apple and Orange, that inherit from the Fruit class.  
 Override the eat() method in each subclass to display the specific taste of that fruit.

// Base class Fruit

class Fruit {

// Attributes of the fruit

String name;

String taste;

String size;

// Constructor for the base class

public Fruit(String name, String taste, String size) {

this.name = name;

this.taste = taste;

this.size = size;

}

// Method to print the name and taste of the fruit

public void eat() {

System.out.println("Eating " + name + ": It tastes " + taste);

}

}

// Subclass Apple

class Apple extends Fruit {

// Constructor for Apple class

public Apple(String name, String taste, String size) {

super(name, taste, size);

}

// Overriding the eat method for Apple

@Override

public void eat() {

System.out.println("Eating " + name + ": It tastes " + taste + " and is crunchy!");

}

}

// Subclass Orange

class Orange extends Fruit {

// Constructor for Orange class

public Orange(String name, String taste, String size) {

super(name, taste, size);

}

// Overriding the eat method for Orange

@Override

public void eat() {

System.out.println("Eating " + name + ": It tastes " + taste + " and is juicy!");

}

}

public class Example28 {

public static void main(String[] args) {

// Creating an object of Apple

Fruit apple = new Apple("Apple", "sweet", "medium");

apple.eat(); // Calls the overridden eat method in Apple class

// Creating an object of Orange

Fruit orange = new Orange("Orange", "sour", "medium");

orange.eat(); // Calls the overridden eat method in Orange class

}

}

29)

Given a number N, the task is to count the number of unique digits in the given number.

Examples:

*Input: N = 22342 Output: 2*

*Explanation: The digits 3 and 4 occurs only once. Hence, the output is 2.*

*Input: N = 99677 Output: 1  
 Explanation: The digit 6 occurs only once. Hence, the output is 1.*import java.util.\*;

public class Example29 {

public static void main(String[] args) {

int n = 99677;

HashMap<Integer, Integer> mp = new HashMap<Integer, Integer>();

while(n>0){

mp.put(n%10, mp.getOrDefault(n%10, 0)+1);

n = n/10;

}

int count = 0;

for(Integer i : mp.keySet()){

if(mp.get(i)==1){

count++;

}

}

System.out.println(count);

}

} 30)

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1: Input: target = 7, nums = [2,3,1,2,4,3] Output: 2

Example 2: Input: target = 11, nums = [1,1,1,1,1,1,1,1] Output: 0  
  
  
public class Example30

{

public static void main(String[] args) {

int[] arr = {1,1,1,1,1,1,1,1};

int res = Integer.MAX\_VALUE;

int target = 11;

int sum = 0;

for(int l = 0, r = 0; r<arr.length; r++){

sum += arr[r];

while(l<r && sum>target){

sum-=arr[l];

l++;

}

if(sum==target){

res = Math.min(res, r-l+1);

}

}

System.out.println(res==Integer.MAX\_VALUE ? 0 : res );

}

}

31)

Write a Java program to receive an integer number as a command-line argument, and print the binary, octal, and hexadecimal equivalents of the given number.

Given Number : 20

Binary equivalent : 10100

Octal equivalent : 24

Hexadecimal equivalent : 14

public class Example31 {

public static void main(String[] args) {

if (args.length == 0) {

System.out.println("Please provide an integer as a command-line argument.");

return;

}

try {

int num = Integer.parseInt(args[0]); // allowed just to parse input

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

System.out.println("Binary equivalent: " + toBinary(num));

System.out.println("Octal equivalent: " + toOctal(num));

System.out.println("Hexadecimal equivalent: " + toHex(num));

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid integer.");

}

}

static String toBinary(int n) {

return convertToBase(n, 2);

}

static String toOctal(int n) {

return convertToBase(n, 8);

}

static String toHex(int n) {

return convertToBase(n, 16);

}

static String convertToBase(int n, int base) {

if (n == 0) return "0";

String digits = "0123456789ABCDEF";

StringBuilder result = new StringBuilder();

while (n > 0) {

int remainder = n % base;

result.insert(0, digits.charAt(remainder));

n /= base;

}

return result.toString();

}

}

32) Develop an online payment system that supports different payment methods.

Requirements:

1. Create an abstract class Payment with:
   * Attributes: amount, transactionID.
   * Abstract method processPayment().
   * Concrete method showTransactionDetails().
2. Create subclasses:
   * CreditCardPayment (cardNumber, CVV, expiryDate).
   * PayPalPayment (email, password).
   * UPIPayment (UPI ID).
3. Implement the processPayment() method in each subclass to handle payments uniquely.
4. Create a PaymentGateway class to process transactions dynamically.

// Abstract class Payment

abstract class Payment {

private double amount;

private String transactionID;

public void setAmount(double amount) {

this.amount = amount;

}

public void setTransactionID(String transactionID) {

this.transactionID = transactionID;

}

public double getAmount() {

return amount;

}

public String getTransactionID() {

return transactionID;

}

public abstract void processPayment();

public void showTransactionDetails() {

System.out.println("Transaction ID: " + transactionID);

System.out.println("Amount: ₹" + amount);

}

}

// CreditCardPayment subclass

class CreditCardPayment extends Payment {

private String cardNumber;

private String cvv;

private String expiryDate;

public CreditCardPayment(String transactionID, double amount, String cardNumber, String cvv, String expiryDate) {

setTransactionID(transactionID);

setAmount(amount);

this.cardNumber = cardNumber;

this.cvv = cvv;

this.expiryDate = expiryDate;

}

@Override

public void processPayment() {

System.out.println("Processing Credit Card Payment...");

System.out.println("Card Number: " + cardNumber);

showTransactionDetails();

System.out.println("Credit Card Payment Successful.\n");

}

}

// PayPalPayment subclass

class PayPalPayment extends Payment {

private String email;

private String password;

public PayPalPayment(String transactionID, double amount, String email, String password) {

setTransactionID(transactionID);

setAmount(amount);

this.email = email;

this.password = password;

}

@Override

public void processPayment() {

System.out.println("Processing PayPal Payment...");

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

showTransactionDetails();

System.out.println("PayPal Payment Successful.\n");

}

}

// UPIPayment subclass

class UPIPayment extends Payment {

private String upiId;

public UPIPayment(String transactionID, double amount, String upiId) {

setTransactionID(transactionID);

setAmount(amount);

this.upiId = upiId;

}

@Override

public void processPayment() {

System.out.println("Processing UPI Payment...");

System.out.println("UPI ID: " + upiId);

showTransactionDetails();

System.out.println("UPI Payment Successful.\n");

}

}

// PaymentGateway to process all payments

class PaymentGateway {

public void process(Payment payment) {

payment.processPayment();

}

}

// Main class

public class Example32 {

public static void main(String[] args) {

PaymentGateway gateway = new PaymentGateway();

Payment credit = new CreditCardPayment("TXN1001", 5000, "1234-5678-9012-3456", "123", "12/26");

Payment paypal = new PayPalPayment("TXN1002", 2999, "user@example.com", "securePass");

Payment upi = new UPIPayment("TXN1003", 1500, "user@upi");

gateway.process(credit);

gateway.process(paypal);

gateway.process(upi);

}

}

33) Design a simple system to calculate the area of different 2D shapes using interfaces in Java.

Define an interface named Shape with a method:

Create the following classes that implement the Shape interface:

* Circle with a field radius
* Rectangle with fields length and width
* Triangle with fields base and height

Each class must implement the calculateArea() method according to the respective formula:

* Circle: π × radius²
* Rectangle: length × width
* Triangle: 0.5 × base × height

In the main() method, use polymorphism to create an array of Shape references and call calculateArea() on each.

interface Shape {

double calculateArea();

}

class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

class Rectangle implements Shape {

private double length, width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

public double calculateArea() {

return length \* width;

}

}

class Triangle1 implements Shape {

private double base, height;

public Triangle1(double base, double height) {

this.base = base;

this.height = height;

}

public double calculateArea() {

return 0.5 \* base \* height;

}

}

public class Example33 {

public static void main(String[] args) {

Shape[] shapes = new Shape[3];

shapes[0] = new Circle(5); // radius = 5

shapes[1] = new Rectangle(4, 6); // length = 4, width = 6

shapes[2] = new Triangle1(3, 8); // base = 3, height = 8

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

System.out.println("Area of Shape " + (i + 1) + ": " + shapes[i].calculateArea());

}

}

}

34). simple banking system that handles user withdrawals, including proper use of exception handling and custom exceptions.

#### Requirements:

1. Create a class BankAccount with the following:  
   * Field: double balance
   * Constructor to initialize balance
   * Method:  
       
      If the withdrawal amount is greater than the balance, throw a custom exception named InsufficientFundsException.
   * Otherwise, deduct the amount from the balance.

2. Define a custom exception class:

* Include a constructor that accepts a custom error message.

In the main() method:

* Create a BankAccount object with an initial balance.
* Try to withdraw different amounts (some valid, some invalid).
* Catch the exception and display appropriate error messages.

// Custom Exception Class

class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

// BankAccount Class

class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

balance = initialBalance;

}

public void withdraw(double amount) throws InsufficientFundsException {

if (amount > balance) {

throw new InsufficientFundsException("Insufficient funds! You tried to withdraw " + amount + " but only " + balance + " is available.");

} else {

balance -= amount;

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

}

}

public double getBalance() {

return balance;

}

}

// Main Class

public class BankingSystem {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000.00); // Initial balance

double[] withdrawalAttempts = {200.00, 500.00, 400.00}; // Some valid, some invalid

for (double amount : withdrawalAttempts) {

try {

System.out.println("\nAttempting to withdraw: " + amount);

account.withdraw(amount);

} catch (InsufficientFundsException e) {

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

}

}

}

}

35)

The Citizen class should have following attributes name, id, country, sex, maritalStatus, anualIncome, and economyStatus. Validate the fields if the age is below 18 and country is not ‘India’ throw NonEligibleException and give proper message. Use toString method to display the citizen object in proper format. Use separate packages for Exception and application classes

import IneligibleException.IneligibleException;

public class Example35 {

private String name;

private String country;

private int age;

public Example35(String n, String c, int a) throws IneligibleException{

if(!c.equals("India")) throw new IneligibleException("Ineligible country");

if(age<18) throw new IneligibleException("Age must be greater than or equal to 18");

name = n;

country = c;

age = a;

}

public String toString(){

return "Name: "+name+" Age: "+ age+" Country: "+country;

}

public static void main(String[] args){

try{

Example35 c1 = new Example35("Abc", "India", 19);

Example35 c2 = new Example35("xyz", "Not India", 19);

System.out.println(c1.toString());

System.out.println(c2.toString());

}catch (IneligibleException e){

System.out.println("Error occured");

e.printStackTrace();

}

}

}

36)

A. Write a Java program to remove prime numbers between 1 to 25 from ArrayList using an iterator.

B. Write a Java program to

a. create and traverse (or iterate) ArrayList using for-loop, iterator, and advance for-loop.

b. check if element(value) exists in ArrayList?

c. add element at particular index of ArrayList?  
  
import java.util.ArrayList;

import java.util.Iterator;

public class ArrayListOperations {

// Utility method to check if a number is prime

public static boolean isPrime(int num) {

if (num <= 1) return false;

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

if (num % i == 0) return false;

return true;

}

public static void main(String[] args) {

// Part A: Remove prime numbers from ArrayList using Iterator

ArrayList<Integer> numbers = new ArrayList<>();

// Fill ArrayList with numbers from 1 to 25

for (int i = 1; i <= 25; i++) {

numbers.add(i);

}

// Display original list

System.out.println("Original List (1 to 25): " + numbers);

// Use iterator to remove prime numbers

Iterator<Integer> it = numbers.iterator();

while (it.hasNext()) {

int val = it.next();

if (isPrime(val)) {

it.remove(); // Safe removal using iterator

}

}

// List after removing primes

System.out.println("After Removing Prime Numbers: " + numbers);

// Part B: ArrayList Operations

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

// a. Create and traverse ArrayList

names.add("Alice");

names.add("Bob");

names.add("Charlie");

System.out.println("\n--- Traversing using for-loop ---");

for (int i = 0; i < names.size(); i++) {

System.out.println(names.get(i));

}

System.out.println("\n--- Traversing using Iterator ---");

Iterator<String> nameIt = names.iterator();

while (nameIt.hasNext()) {

System.out.println(nameIt.next());

}

System.out.println("\n--- Traversing using enhanced for-loop ---");

for (String name : names) {

System.out.println(name);

}

// b. Check if element exists

String search = "Bob";

if (names.contains(search)) {

System.out.println("\n" + search + " exists in the list.");

} else {

System.out.println("\n" + search + " does not exist in the list.");

}

// c. Add element at specific index

names.add(1, "David");

System.out.println("\nList after adding 'David' at index 1: " + names);

}

}

37).

Write a Java program that handles various types of exceptions while performing different operations. The application should read data from a file specified by the user, handling potential `FileNotFoundException` and `IOException`. It should also allow the user to input values for arithmetic operations and handle division by zero using `ArithmeticException`. Additionally, implement exception handling for `InputMismatchException` when the user provides invalid input, `ArrayIndexOutOfBoundsException` for accessing invalid indices in arrays, and `NullPointerException` when performing operations on `null` values. The program should provide user-friendly error messages and ensure smooth execution even when exceptions occur.  
  
import java.io.\*;

import java.util.\*;

public class Example37 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter filename to read: ");

String filename = scanner.nextLine();

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

System.out.println("File contents:");

String line;

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

System.out.println(line);

}

} catch (FileNotFoundException e) {

System.out.println("Error: File not found. Please check the filename.");

} catch (IOException e) {

System.out.println("Error: An I/O error occurred while reading the file.");

}

try {

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

int numerator = scanner.nextInt();

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

int denominator = scanner.nextInt();

int result = numerator / denominator;

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

} catch (InputMismatchException e) {

System.out.println("Error: Please enter valid integers.");

scanner.nextLine(); // clear the invalid input

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero is not allowed.");

}

try {

int[] arr = {1, 2, 3};

System.out.print("Enter index to access (0-2): ");

int index = scanner.nextInt();

System.out.println("Value at index " + index + ": " + arr[index]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Index out of bounds.");

} catch (InputMismatchException e) {

System.out.println("Error: Please enter a valid index.");

scanner.nextLine();

}

try {

String str = null;

System.out.println("Length of the string: " + str.length());

} catch (NullPointerException e) {

System.out.println("Error: Cannot perform operations on a null reference.");

}

System.out.println("Program executed with exception handling.");

scanner.close();

}

}

38)

Create an abstract class Person with:

* Fields: name, age
* Constructor to initialize the fields
* Abstract method:
* Create a class Student that inherits from Person:  
  + Additional fields: rollNumber, course
  + Override the displayDetails() method to print all student details
* Create another class Teacher that also extends Person:  
  + Additional fields: employeeId, subject
  + Override the displayDetails() method to print all teacher details
* Demonstrate encapsulation by making all fields private and using getter and setter methods.
* In the main() method:  
  + Create an array of Person references (use polymorphism).
  + Store both Student and Teacher objects.
  + Call the displayDetails() method for each object using a loop.

abstract class Person1 {

private String name;

private int age;

public Person1(String name, int age) {

this.name = name;

this.age = age;

}

public abstract void displayDetails();

// Getters and setters

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

class Student extends Person1 {

private int rollNumber;

private String course;

public Student(String name, int age, int rollNumber, String course) {

super(name, age);

this.rollNumber = rollNumber;

this.course = course;

}

@Override

public void displayDetails() {

System.out.println("Student Details:");

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

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

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

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

System.out.println();

}

// Getters and setters

public int getRollNumber() {

return rollNumber;

}

public void setRollNumber(int rollNumber) {

this.rollNumber = rollNumber;

}

public String getCourse() {

return course;

}

public void setCourse(String course) {

this.course = course;

}

}

class Teacher extends Person1 {

private String employeeId;

private String subject;

public Teacher(String name, int age, String employeeId, String subject) {

super(name, age);

this.employeeId = employeeId;

this.subject = subject;

}

@Override

public void displayDetails() {

System.out.println("Teacher Details:");

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

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

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

System.out.println("Subject: " + subject);

System.out.println();

}

// Getters and setters

public String getEmployeeId() {

return employeeId;

}

public void setEmployeeId(String employeeId) {

this.employeeId = employeeId;

}

public String getSubject() {

return subject;

}

public void setSubject(String subject) {

this.subject = subject;

}

}

public class Example38 {

public static void main(String[] args) {

Person1[] people = new Person1[2];

people[0] = new Student("Alice", 20, 101, "Computer Science");

people[1] = new Teacher("Bob", 40, "T102", "Mathematics");

for (Person1 p : people) {

p.displayDetails();

}

}

}

1. A) Write a Java program to create a class called Employee with methods called work() and getSalary(). Create a subclass called HRManager that overrides the work() method and adds a new method called addEmployee().

class Employee3 {

public void work() {

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

}

public double getSalary() {

return 30000.00;

}

}

class HRManager extends Employee3 {

@Override

public void work() {

System.out.println("HR Manager is hiring and managing employees...");

}

public void addEmployee() {

System.out.println("New employee added by HR Manager.");

}

}

public class Example39A {

public static void main(String[] args) {

HRManager hr = new HRManager();

hr.work();

hr.addEmployee();

System.out.println("Salary: " + hr.getSalary());

}

}

1. Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.

class Shape39 {

public void draw() {

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

}

public void erase() {

System.out.println("Erasing a shape...");

}

}

class Circle39 extends Shape39 {

@Override

public void draw() {

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

}

@Override

public void erase() {

System.out.println("Erasing a Circle");

}

}

class Triangle39 extends Shape39 {

@Override

public void draw() {

System.out.println("Drawing a Triangle");

}

@Override

public void erase() {

System.out.println("Erasing a Triangle");

}

}

class Square39 extends Shape39 {

@Override

public void draw() {

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

}

@Override

public void erase() {

System.out.println("Erasing a Square");

}

}

public class Example39B {

public static void main(String[] args) {

Shape39[] shapes = { new Circle39(), new Triangle39(), new Square39() };

for (Shape s : shapes) {

s.draw();

s.erase();

System.out.println();

}

}

}

1. Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.   
     
    abstract class Shape1 {

abstract double calculateArea();

abstract double calculatePerimeter();

}

class Circle1 extends Shape1 {

private double radius;

public Circle1(double radius) {

this.radius = radius;

}

@Override

double calculateArea() {

return Math.PI \* radius \* radius;

}

@Override

double calculatePerimeter() {

return 2 \* Math.PI \* radius;

}

}

class Triangle extends Shape1 {

private double sideA, sideB, sideC;

public Triangle(double a, double b, double c) {

this.sideA = a;

this.sideB = b;

this.sideC = c;

}

@Override

double calculateArea() {

double s = (sideA + sideB + sideC) / 2; // semi-perimeter

return Math.sqrt(s \* (s - sideA) \* (s - sideB) \* (s - sideC)); // Heron's formula

}

@Override

double calculatePerimeter() {

return sideA + sideB + sideC;

}

}

public class Example40 {

public static void main(String[] args) {

Shape1 circle = new Circle1(5); // radius = 5

Shape1 triangle = new Triangle(3, 4, 5); // sides = 3, 4, 5

System.out.println("Circle Area: " + circle.calculateArea());

System.out.println("Circle Perimeter: " + circle.calculatePerimeter());

System.out.println("Triangle Area: " + triangle.calculateArea());

System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());

}

}

1. A) Write a java program to Move all zeroes to end of array

Input: arr[] = {1, 2, 0, 4, 3, 0, 5, 0};

Output: arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

B)Write a Java program to create an interface Sortable with a method sort() that sorts an array of integers in ascending order. Create two classes BubbleSort and SelectionSort that implement the Sortable interface and provide their own implementations of the sort() method.

interface Sortable {

void sort(int[] arr);

}

class BubbleSort implements Sortable {

public void sort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

}

class SelectionSort implements Sortable {

public void sort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

int minIdx = i;

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

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

int temp = arr[i];

arr[i] = arr[minIdx];

arr[minIdx] = temp;

}

}

}

public class Example41 {

public static void moveZeroesToEnd(int[] arr) {

int index = 0; // index for placing non-zero elements

for (int num : arr) {

if (num != 0) {

arr[index++] = num;

}

}

while (index < arr.length) {

arr[index++] = 0;

}

}

public static void printArray(int[] arr) {

for (int num : arr)

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

System.out.println();

}

public static void main(String[] args) {

System.out.println("Part A: Move all zeroes to the end");

int[] arrWithZeroes = {1, 2, 0, 4, 3, 0, 5, 0};

moveZeroesToEnd(arrWithZeroes);

printArray(arrWithZeroes);

System.out.println("\nPart B: Sort arrays using interface");

int[] arr1 = {5, 2, 8, 1, 3};

int[] arr2 = {9, 4, 6, 2, 7};

Sortable bubbleSort = new BubbleSort();

Sortable selectionSort = new SelectionSort();

System.out.print("Bubble Sort: ");

bubbleSort.sort(arr1);

printArray(arr1);

System.out.print("Selection Sort: ");

selectionSort.sort(arr2);

printArray(arr2);

}

}

1. A) Write a Java program to create a class called "Book" with instance variables title, author, and price. Implement a default constructor and two parameterized constructors:

One constructor takes the title and author as parameters.

The other constructor takes title, author, and price as parameters.

Print the values of the variables for each constructor.

public class Example42A {

public String title;

public int price;

public BookConstructors(){

System.out.println("Default constructor");

}

public BookConstructors(String s, int p){

title = s;

price = p;

System.out.println("Parameterized constructor");

}

public static void main(String[] args) {

BookConstructors b1 = new BookConstructors();

BookConstructors b2 = new BookConstructors("Book title", 1000);

System.out.println("Book1 "+b1.title + " price: " + b1.price);

System.out.println("Book2 "+b2.title + " price: " + b2.price);

}

}

B) Write a Java program to create a class called "TrafficLight" with attributes for color and duration, and methods to change the color and check for red or green.

class TrafficLight {

private String color;

private int duration;

public TrafficLight(String color, int duration) {

this.color = color;

this.duration = duration;

}

public void changeColor(String newColor, int newDuration) {

this.color = newColor;

this.duration = newDuration;

System.out.println("Traffic light changed to: " + color + " for " + duration + " seconds.");

}

public boolean isRed() {

return color.equalsIgnoreCase("Red");

}

public boolean isGreen() {

return color.equalsIgnoreCase("Green");

}

public void displayStatus() {

System.out.println("Current Color: " + color + ", Duration: " + duration + " seconds");

}

}

public class Example42B {

public static void main(String[] args) {

TrafficLight light = new TrafficLight("Red", 30);

light.displayStatus();

System.out.println("Is it red? " + light.isRed());

System.out.println("Is it green? " + light.isGreen());

light.changeColor("Green", 45);

System.out.println("Is it red? " + light.isRed());

System.out.println("Is it green? " + light.isGreen());

}

}