**Ex No: Interface and Collections**

**Date:**

**AIM**

The aim of the provided Java programs is to implement the Interface and Collections

**STEP 1: Contact Manager Program:**

* Define a Contact class to represent individual contacts.
* Create a ContactManager class with a Map to store contacts using Aadhar numbers.
* Implement methods to add, view, update, and remove contacts, as well as list unique Aadhar numbers.
* In the main method, create a loop to display a menu for users to interact with the contact manager.

**STEP 2: Payment System Program:**

* Define interfaces Payment, DebitCardPayment, and GPayPayment.
* Implement a class UserAccount that represents a user's account and implements both DebitCardPayment and GPayPayment interfaces.
* In the main method, interactively take user input to perform Debit Card and GPay transactions, handling exceptions appropriately.

**STEP 3: Sudoku Puzzle Solver:**

* Create a 4x4 Sudoku puzzle represented by a 2D array.
* Implement a backtracking algorithm (solveSudoku) to find a solution for the puzzle.
* Implement helper methods (isSafe, usedInRow, usedInColumn, usedInSubgrid) to check if a number can be placed in a given cell.
* In the main method, take user input for the initial state of the puzzle, and then attempt to solve and display the solution.

1. **Write a contact manager in Java that can store multiple contacts for a person by using a List and Map. Each contact should include the following details: Aadhar number, name, phone number, email.**

**CODE**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Scanner;

class Contact {

private String aadharNumber;

private String name;

private String phoneNumber;

private String email;

public Contact(String aadharNumber, String name, String phoneNumber, String email) {

this.aadharNumber = aadharNumber;

this.name = name;

this.phoneNumber = phoneNumber;

this.email = email;

}

public String getAadharNumber() {

return aadharNumber;

}

public String getName() {

return name;

}

public String getPhoneNumber() {

return phoneNumber;

}

public String getEmail() {

return email;

}

}

public class ContactManager {

private Map<String, List<Contact>> contacts = new HashMap<>();

public void addContact(Contact contact) {

String aadharNumber = contact.getAadharNumber();

if (!contacts.containsKey(aadharNumber)) {

contacts.put(aadharNumber, new ArrayList<>());

}

contacts.get(aadharNumber).add(contact);

}

public void viewContacts(String aadharNumber) {

if (contacts.containsKey(aadharNumber)) {

List<Contact> contactList = contacts.get(aadharNumber);

for (Contact c : contactList) {

System.out.println("Name: " + c.getName() + ", Email: " + c.getEmail() + ", Phone: " + c.getPhoneNumber());

}

} else {

System.out.println("No contacts found for Aadhar Number: " + aadharNumber);

}

}

public void updateContact(String aadharNumber, Contact updatedContact) {

if (contacts.containsKey(aadharNumber)) {

List<Contact> contactList = contacts.get(aadharNumber);

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

Contact existingContact = contactList.get(i);

if (existingContact.getPhoneNumber().equals(updatedContact.getPhoneNumber())) {

contactList.set(i, updatedContact);

System.out.println("Contact updated successfully.");

return;

}

}

System.out.println("Contact not found for the given phone number.");

} else {

System.out.println("No contacts found for Aadhar Number: " + aadharNumber);

}

}

public void removeContact(String aadharNumber, String phoneNumber) {

if (contacts.containsKey(aadharNumber)) {

List<Contact> contactList = contacts.get(aadharNumber);

for (Contact contact : contactList) {

if (contact.getPhoneNumber().equals(phoneNumber)) {

contactList.remove(contact);

System.out.println("Contact removed successfully.");

return;

}

}

System.out.println("Contact not found for the given phone number.");

} else {

System.out.println("No contacts found for Aadhar Number: " + aadharNumber);

}

}

public void listUniqueAadharNumbers() {

System.out.println("Unique Aadhar Numbers in the Contact Manager:");

for (String aadharNumber : contacts.keySet()) {

System.out.println(aadharNumber);

} }

public static void main(String[] args) {

ContactManager contactManager = new ContactManager();

Scanner scanner = new Scanner(System.in);

while (true) {

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

System.out.println("1. Add Contact");

System.out.println("2. View Contacts");

System.out.println("3. Update Contact");

System.out.println("4. Remove Contact");

System.out.println("5. List Unique Aadhar Numbers");

System.out.println("6. Exit");

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

int choice = scanner.nextInt();

scanner.nextLine(); // Consume the newline character

switch (choice) {

case 1:

System.out.print("Enter Aadhar Number: ");

String aadharNumber = scanner.nextLine();

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

String name = scanner.nextLine();

System.out.print("Enter Phone Number: ");

String phoneNumber = scanner.nextLine();

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

String email = scanner.nextLine();

Contact newContact = new Contact(aadharNumber, name, phoneNumber, email);

contactManager.addContact(newContact);

System.out.println("Contact added successfully.");

break;

case 2:

System.out.print("Enter Aadhar Number: ");

aadharNumber = scanner.nextLine();

contactManager.viewContacts(aadharNumber);

break;

case 3:

System.out.print("Enter Aadhar Number: ");

aadharNumber = scanner.nextLine();

System.out.print("Enter Phone Number of Contact to Update: ");

phoneNumber = scanner.nextLine();

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

String updatedName = scanner.nextLine();

System.out.print("Enter New Email: ");

String updatedEmail = scanner.nextLine();

Contact updatedContact = new Contact(aadharNumber, updatedName, phoneNumber, updatedEmail);

contactManager.updateContact(aadharNumber, updatedContact);

break;

case 4:

System.out.print("Enter Aadhar Number: ");

aadharNumber = scanner.nextLine();

System.out.print("Enter Phone Number of Contact to Remove: ");

phoneNumber = scanner.nextLine();

contactManager.removeContact(aadharNumber, phoneNumber);

break;

case 5:

contactManager.listUniqueAadharNumbers();

break;

case 6:

System.out.println("Exiting the Contact Manager.");

scanner.close();

System.exit(0);

break;

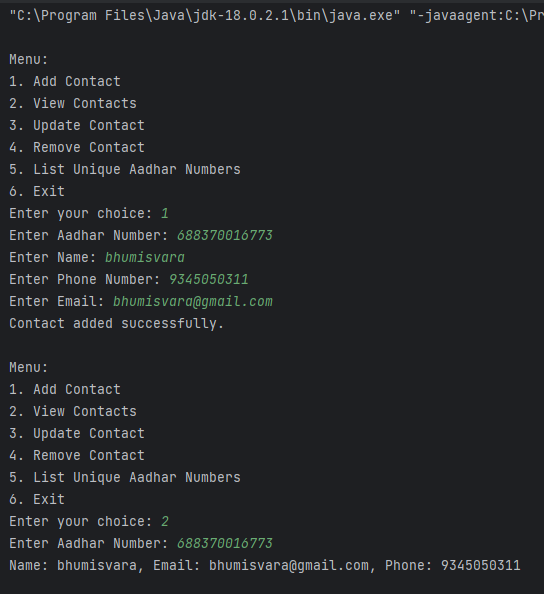
default:

System.out.println("Invalid choice. Please select a valid option.");

}} }

}

**OUTPUT**

****

**2. Write a Java code for payment system that demonstrates hybrid and multiple inheritance using specific interfaces, classes, methods, and custom exceptions. The payment system includes two payment methods: Debit Cards and Google Pay (GPay).**

**CODE**

import java.util.Scanner;

class InvalidPaymentOptionException extends Exception {

public InvalidPaymentOptionException(String message) {

super(message);

}

}

class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String message) {

super(message);

}

}

interface Payment {

void makePayment(double amount) throws InsufficientBalanceException;

double getAccountBalance();

void receivePayment(double amount);

}

interface DebitCardPayment extends Payment {

void withdrawCash(double amount) throws InsufficientBalanceException;

}

interface GPayPayment extends Payment {

void rewardPoints(double amount);

}

class UserAccount implements DebitCardPayment, GPayPayment {

private double accountBalance;

public UserAccount(double initialBalance) {

this.accountBalance = initialBalance;

}

@Override

public void makePayment(double amount) throws InsufficientBalanceException {

if (accountBalance >= amount) {

accountBalance -= amount;

} else {

throw new InsufficientBalanceException("Insufficient balance for payment");

}

}

@Override

public double getAccountBalance() {

return accountBalance;

}

@Override

public void receivePayment(double amount) {

accountBalance += amount;

}

@Override

public void withdrawCash(double amount) throws InsufficientBalanceException {

if (accountBalance >= amount) {

accountBalance -= amount;

} else {

throw new InsufficientBalanceException("Insufficient balance for cash withdrawal");

}

}

@Override

public void rewardPoints(double amount) {

accountBalance += (amount \* 0.02); // Reward 2% of the transaction amount

}

}

public class PaymentDemo3704 {

public static void main(String[] args) {

System.out.println("Bhumisvara");

System.out.println("2021503704");

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the initial account balance: ");

double initialBalance = scanner.nextDouble();

UserAccount user = new UserAccount(initialBalance);

try {

System.out.println("Enter the amount to pay using Debit Card: ");

double debitCardAmount = scanner.nextDouble();

user.makePayment(debitCardAmount);

System.out.println("Enter the amount to receive: ");

double receiveAmount = scanner.nextDouble();

user.receivePayment(receiveAmount);

if (user instanceof GPayPayment) {

System.out.println("Enter the amount to pay using GPay: ");

double gPayAmount = scanner.nextDouble();

user.makePayment(gPayAmount);

} else {

throw new InvalidPaymentOptionException("Invalid payment option for GPay");

}

if (user instanceof DebitCardPayment) {

System.out.println("Enter the amount to withdraw cash: ");

double withdrawAmount = scanner.nextDouble();

user.withdrawCash(withdrawAmount);

} else {

throw new InvalidPaymentOptionException("Invalid payment option for cash withdrawal");

}

} catch (InvalidPaymentOptionException | InsufficientBalanceException e) {

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

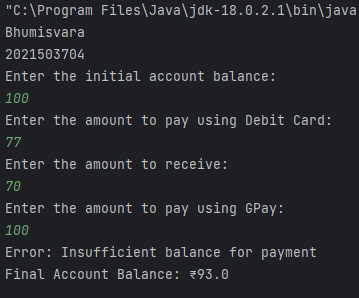
}

System.out.println("Final Account Balance: ₹" + user.getAccountBalance());

}

}

**OUTPUT**

****

1. **Write a Sudoku puzzle to fill a 4x4 grid with numbers so that each row, column, and 2x2 subgrid contains every number from 1 to 4 exactly once.**

**CODE**

import java.util.\*;

public class Sudoku3704 {

private static final int SIZE = 4;

private static final int SUBGRID\_SIZE = 2;

public static void main(String[] args) {

System.out.println("Bhumisvara");

System.out.println("2021503704");

int[][] puzzle = new int[SIZE][SIZE];

Scanner scanner = new Scanner(System.in);

// Get user input for the initial state of the Sudoku puzzle

System.out.println("Enter the Sudoku puzzle (use 0 to represent empty cells):");

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

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

puzzle[i][j] = scanner.nextInt();

}

}

if (solveSudoku(puzzle, 0, 0)) {

System.out.println("4x4 Sudoku Puzzle Solution:");

displayPuzzle(puzzle);

} else {

System.out.println("No solution exists for the given constraints.");

}

scanner.close();

}

private static boolean solveSudoku(int[][] puzzle, int row, int col) {

if (row == SIZE - 1 && col == SIZE)

return true;

if (col == SIZE) {

row++;

col = 0;

}

if (puzzle[row][col] != 0)

return solveSudoku(puzzle, row, col + 1);

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

if (isSafe(puzzle, row, col, num)) {

puzzle[row][col] = num;

if (solveSudoku(puzzle, row, col + 1))

return true;

puzzle[row][col] = 0;

}

}

return false;

}

private static boolean isSafe(int[][] puzzle, int row, int col, int num) {

return !usedInRow(puzzle, row, num) && !usedInColumn(puzzle, col, num) && !usedInSubgrid(puzzle, row - row % SUBGRID\_SIZE, col - col % SUBGRID\_SIZE, num);

}

private static boolean usedInRow(int[][] puzzle, int row, int num) {

for (int col = 0; col < SIZE; col++) {

if (puzzle[row][col] == num)

return true;

}

return false;

}

private static boolean usedInColumn(int[][] puzzle, int col, int num) {

for (int row = 0; row < SIZE; row++) {

if (puzzle[row][col] == num)

return true;

}

return false;

}

private static boolean usedInSubgrid(int[][] puzzle, int subgridStartRow, int subgridStartCol, int num) {

for (int i = 0; i < SUBGRID\_SIZE; i++) {

for (int j = 0; j < SUBGRID\_SIZE; j++) {

if (puzzle[i + subgridStartRow][j + subgridStartCol] == num)

return true;

}

}

return false;

}

private static void displayPuzzle(int[][] puzzle) {

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

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

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

}

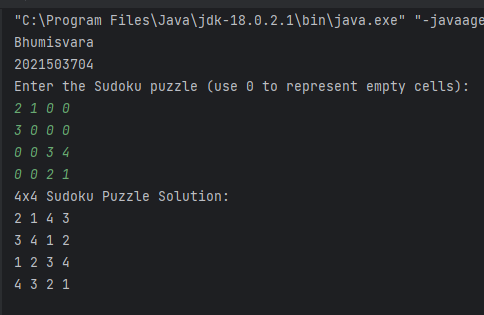
System.out.println();

}

}

}

**OUTPUT**

****

**RESULT**

Thus, The Interface and Collections program has been successfully implemented.