|  |
| --- |
| Exp: 12 Date: 25/10/2023  **JAVA PROGRAMMING**  **CS6308** |

Interface and Collections

Name: VIJAI SURIA M

Reg No.: 2021503568

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

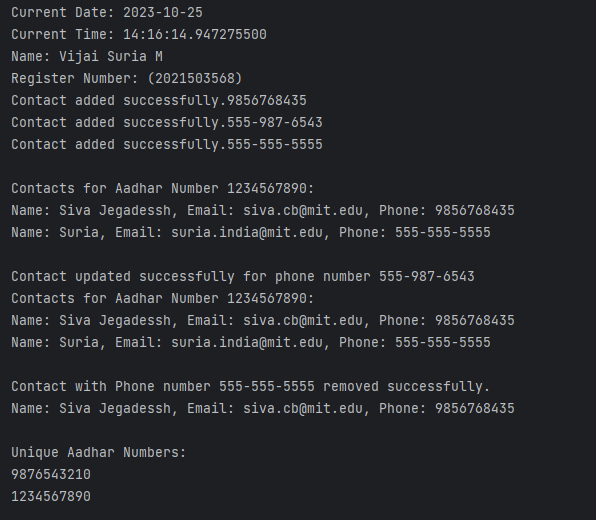
* Create a Contactclass that includes fields for Aadhar number, phone number, name, and email.
* Implement a ContactManagerclass that uses a map to associate a person'sAadhar number with a list of contacts (each contact should be an instanceofthe Contactclass). Use the Aadhar number asthe key and a list of contactsas the value.Provide methods in the ContactManagerclass for the following operations:

1. Adding a new contact for a person.Map<String, List<Contact>>contacts = new HashMap<>();...if (!contacts.containsKey(phoneNumber)) {contacts.put(phoneNumber, new ArrayList<>());}contacts.get(phoneNumber).add(contact);
2. Viewing all contacts for a person using their Aadhar number.for (Map.Entry<String, List<Contact>> entry: contacts.entrySet()) {String phoneNumber = entry.getKey();List<Contact> contactList = entry.getValue();for (Contact c : contactList) {System.out.println("Name: " + c.getName() + ", Email: " + c.getEmail() + ", Phone: " + phoneNumber);}}
3. Updating a specific contact for a person.
4. Removing a contact for a person.
5. Listing all unique Aadhar numbers in the contact manager.

**CODE**

|  |
| --- |
| import java.util.\*;  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 getName() {  return name;  }   public String getPhoneNumber() {  return phoneNumber;  }   public String getEmail() {  return email;  }   public void setName(String name) {  this.name = name;  }   public void setPhoneNumber(String phoneNumber) {  this.phoneNumber = phoneNumber;  }   public void setEmail(String email) {  this.email = email;  } }  class ContactManager {  private Map<String, List<Contact>> contactsMap;   public ContactManager() {  contactsMap = new HashMap<>();  }   public void addContact(String aadharNumber, String name, String phoneNumber, String email) {  Contact contact = new Contact(aadharNumber, name, phoneNumber, email);  if (!contactsMap.containsKey(aadharNumber)) {  contactsMap.put(aadharNumber, new ArrayList<>());  }  contactsMap.get(aadharNumber).add(contact);  System.*out*.println("Contact added successfully." + phoneNumber);  }   public void viewAllContactsByAadhar(String aadharNumber) {  if (contactsMap.containsKey(aadharNumber)) {  List<Contact> contactList = contactsMap.get(aadharNumber);  for (Contact contact : contactList) {  System.*out*.println("Name: " + contact.getName() + ", Email: " + contact.getEmail() + ", Phone: " + contact.getPhoneNumber());  }  } else {  System.*out*.println("No contacts found for Aadhar Number: " + aadharNumber);  }  }   public void updateContact(String aadharNumber, String oldPhoneNumber, String newName, String newPhoneNumber, String newEmail) {  if (contactsMap.containsKey(aadharNumber)) {  List<Contact> contactList = contactsMap.get(aadharNumber);  for (Contact contact : contactList) {  if (contact.getPhoneNumber().equals(oldPhoneNumber)) {  contact.setName(newName);  contact.setPhoneNumber(newPhoneNumber);  contact.setEmail(newEmail);  System.*out*.println("Contact updated successfully for phone number " + oldPhoneNumber);  return;  }  }  }  System.*out*.println("Contact not found for updating.");  }   public void removeContact(String aadharNumber, String phoneNumber) {  if (contactsMap.containsKey(aadharNumber)) {  List<Contact> contactList = contactsMap.get(aadharNumber);  for (Contact contact : contactList) {  if (contact.getPhoneNumber().equals(phoneNumber)) {  contactList.remove(contact);  System.*out*.println("Contact with Phone number " + phoneNumber +" removed successfully.");  return;  }  }  }  System.*out*.println("Contact not found for removal.");  }   public Set<String> listAllUniqueAadharNumbers() {  return contactsMap.keySet();  } }  public class ContactManager3568 {  public static void main(String[] args) {  ContactManager contactManager = new ContactManager();   // Adding contacts  contactManager.addContact("1234567890", "John Doe", "555-123-4567", "john.doe@example.com");  contactManager.addContact("9876543210", "Jane Smith", "555-987-6543", "jane.smith@example.com");  contactManager.addContact("1234567890", "Alice Johnson", "555-555-5555", "alice.johnson@example.com");  System.*out*.println();   // Viewing contacts by Aadhar number  System.*out*.println("Contacts for Aadhar Number 1234567890:");  contactManager.viewAllContactsByAadhar("1234567890");  System.*out*.println();   // Updating a contact  contactManager.updateContact("1234567890", "555-123-4567", "John Updated", "555-543-2100", "john.updated@example.com");  System.*out*.println("Contacts for Aadhar Number 1234567890:");  contactManager.viewAllContactsByAadhar("1234567890");  System.*out*.println();   // Removing a contact  contactManager.removeContact("1234567890", "555-555-5555");  contactManager.viewAllContactsByAadhar("1234567890");  System.*out*.println();   // Listing all unique Aadhar numbers  System.*out*.println("Unique Aadhar Numbers:");  Set<String> uniqueAadharNumbers = contactManager.listAllUniqueAadharNumbers();  for (String aadharNumber : uniqueAadharNumbers) {  System.*out*.println(aadharNumber);  }  } } |

**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).

Interfaces: Payment, DebitCardPayment, GpayPayment  
Classes: User, Account  
Methods: getAccountBalance, makePayment, withdrawCash, rewardPoints, receivePaymentCustom   
Exceptions:InvalidPaymentOptionException, InsufficientBalanceException

Note: Withdraw method is applicable only for DebitCardPayment, rewardPoints method is applicable only for GpayPayment

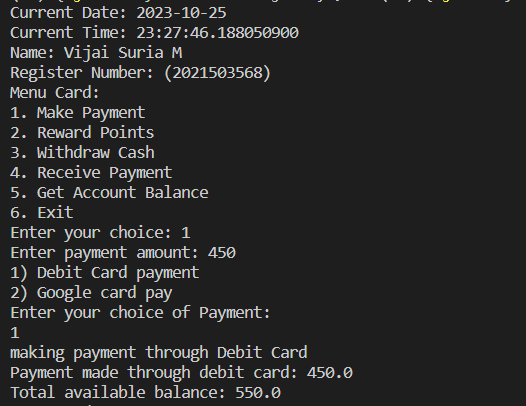
Create a PaymentDemo class and provide a scenario where a user with an initial account balance of ₹1,000 interacts with the payment system, making payments and withdrawals using both the DebitCard and GPay methods. Specify how custom exceptions, including InvalidPaymentOptionException and InsufficientBalanceException, are raised and handled during these transactions.

**CODE**

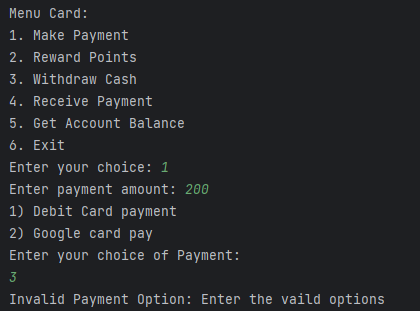
|  |
| --- |
| package LAB13\_2510;  import java.util.\*;  interface Payment {  void makePayment(double amount) throws InvalidPaymentOptionException, InsufficientBalanceException;  void recievePayment(double amount);  double getAccountBalance();  }  interface DebitCardPayment extends Payment{  public default void makePayment(double amount) throws InvalidPaymentOptionException, InsufficientBalanceException {  System.out.println("making payment through Debit Card");  }  void withdrawCash(double amount) throws InsufficientBalanceException;  }  interface GPayPayment extends Payment {  public default void makePayment(double amount) throws InvalidPaymentOptionException, InsufficientBalanceException {  System.out.println("making payment through Google Pay");  }  void rewardPoints();  }  class InsufficientBalanceException extends Exception {  InsufficientBalanceException(String s){  super(s);  }  }  class InvalidPaymentOptionException extends Exception {  InvalidPaymentOptionException(String s){  super(s);  }  }  class User implements DebitCardPayment, GPayPayment{  double accountBalance=1000;  User(double accountBalance){  this.accountBalance = accountBalance;  }  @Override  public void makePayment(double amount) throws InvalidPaymentOptionException, InsufficientBalanceException{  System.out.println("1) Debit Card payment ");  System.out.println("2) Google card pay ");  Scanner in = new Scanner(System.in);  System.out.println("Enter your choice of Payment: ");  int choice = in.nextInt();  switch (choice) {  case 1 -> {  if (accountBalance <= amount) {  throw new InsufficientBalanceException("Not sufficient balance");  } else {  DebitCardPayment.super.makePayment(amount);  accountBalance -= amount;  System.out.println("Payment made through debit card: " + amount);  System.out.println("Total available balance: " + accountBalance);  }  }  case 2 -> {  if (accountBalance <= amount) {  throw new InsufficientBalanceException("Not sufficient balance");  } else {  GPayPayment.super.makePayment(amount);  accountBalance -= amount;  System.out.println("Payment made through Google Pay: " + amount);  System.out.println("Total available balance: " + accountBalance);  }  }  default -> throw new InvalidPaymentOptionException("Enter the vaild options");  }  }  @Override  public void rewardPoints(){  System.out.println("Recieved reward points through Gpay payment");  }  @Override  public void withdrawCash(double amount) throws InsufficientBalanceException{  if (accountBalance <= amount) {  throw new InsufficientBalanceException("Not sufficient balance");  } else {  accountBalance -= amount;  System.out.println("Amount withdrawn through debit card: " + amount);  System.out.println("Total available balance: " + accountBalance);  }  }  public void recievePayment(double amount){  accountBalance += amount;  System.out.println("Amount recieved Rs." + amount);  System.out.println("Total available balance: " + accountBalance);  }  public double getAccountBalance(){  return accountBalance;  }  }  public class PaymentDemo3568 {  public static void main(String args[]){  User user = new User(1000);  Scanner scanner = new Scanner(System.in);  double amount;  while(true){  try {  System.out.println("Menu Card:");  System.out.println("1. Make Payment");  System.out.println("2. Reward Points");  System.out.println("3. Withdraw Cash");  System.out.println("4. Receive Payment");  System.out.println("5. Get Account Balance");  System.out.println("6. Exit");  System.out.print("Enter your choice: ");  int choice = scanner.nextInt();  switch (choice) {  case 1:  System.out.print("Enter payment amount: ");  amount = scanner.nextDouble();  user.makePayment(amount);  break;  case 2:  System.out.println("Earned reward points for the GPay transaction.");  user.rewardPoints();  break;  case 3:  System.out.print("Enter cash withdrawal amount: ");  amount = scanner.nextDouble();  user.withdrawCash(amount);  break;  case 4:  System.out.print("Enter received payment amount: ");  amount = scanner.nextDouble();  user.recievePayment(amount);  break;  case 5:  System.out.println("Available balance: " + user.getAccountBalance());  break;  case 6:  System.out.println("Exiting the menu.");  scanner.close();  System.exit(0);  default:  System.out.println("Invalid choice. Please select a valid option.");  }  } catch (InsufficientBalanceException e) {  System.out.println("Insufficient Balance: " + e.getMessage());  } catch (InvalidPaymentOptionException e) {  System.out.println("Invalid Payment Option: " + e.getMessage());  }  }  }  } |

**OUTPUT**

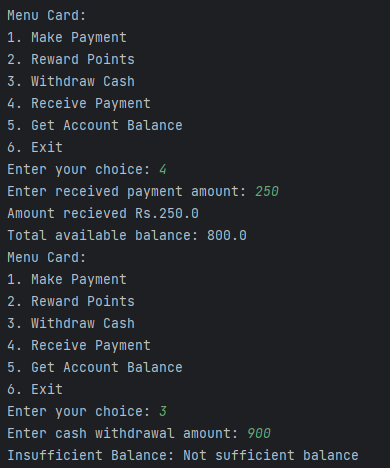
***MAKE PAYMENT THROGH DEBIT CARD***

******

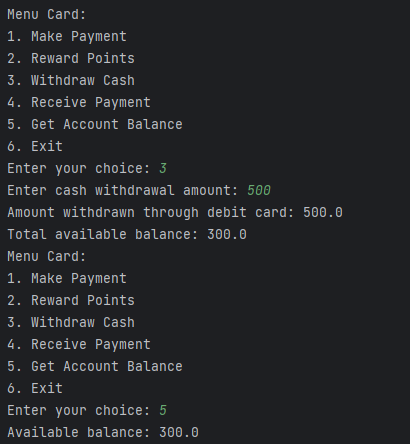
***INVALID OPTION EXCEPTION:***



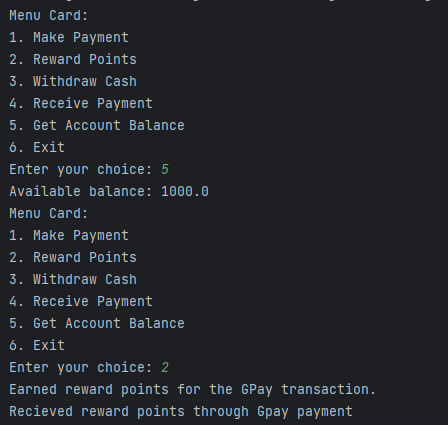
***RECEIVE PAYMENT & INSUFFICIENT ACCOUNT BALANCE EXCEPTION:***



***WITHDRAW CASH & GET ACCOUNT BALANCE:***



***REWARD POINTS:***



3. 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.

Example: input  
2 1 0 0  
3 0 0 0  
0 0 3 4  
0 0 2 1

Example: output  
2 1 4 3  
3 4 1 2  
1 2 3 4  
4 3 2 1

**CODE**

|  |
| --- |
| import java.time.LocalDate;  import java.time.LocalTime;  import java.util.\*;  public class Sudoku3568  {      public static boolean isSafe(int[][] board,                                   int row, int col,                                   int num)      {          // Row has the unique (row-clash)          for (int d = 0; d < board.length; d++)          {              // Check if the number we are trying to              // place is already present in              // that row, return false;              if (board[row][d] == num) {                  return false;              }          }          // Column has the unique numbers (column-clash)          for (int r = 0; r < board.length; r++)          {              // Check if the number              // we are trying to              // place is already present in              // that column, return false;              if (board[r][col] == num)              {                  return false;              }          }          // Corresponding square has          // unique number (box-clash)          int sqrt = (int)Math.sqrt(board.length);          int boxRowStart = row - row % sqrt;          int boxColStart = col - col % sqrt;          for (int r = boxRowStart;               r < boxRowStart + sqrt; r++)          {              for (int d = boxColStart;                   d < boxColStart + sqrt; d++)              {                  if (board[r][d] == num)                  {                      return false;                  }              }          }          // if there is no clash, it's safe          return true;      }      public static boolean solvePuzzle(int[][]board, int n)      {          int row = -1;          int col = -1;          boolean isEmpty = true;          for (int i = 0; i < n; i++)          {              for (int j = 0; j < n; j++)              {                  if (board[i][j] == 0)                  {                      row = i;                      col = j;                      isEmpty = false; // there are missing values in the board                      break;                  }              }              if (!isEmpty)              {                  break;              }          }          // No empty space left          if (isEmpty)          {              return true;          }          // Else for each-row backtrack          for (int num = 1; num <= n; num++)          {              if (isSafe(board, row, col, num))              {                  board[row][col] = num;                  if (solvePuzzle(board, n))                  {                      // print(board, n);                      return true;                  }                  else                  {                      // replace it                      board[row][col] = 0;                  }              }          }          return false;      }      public static void printBoard(int[][]board, int n)      {          for(int r=0; r<n; r++)          {              for(int c = 0; c<n; c++)              {                  System.out.print(board[r][c]+" ");              }              System.out.println();          }      }      public static void main(String[] args)      {          System.out.println("Current Date: " + LocalDate.now());          System.out.println("Current Time: " + LocalTime.now());          System.out.println("Name: Vijai Suria M \nRegister Number: (2021503568)");          int [][] board;          board = new int[][]{                  {2, 1, 0, 0},                  {3, 0, 0, 0},                  {0, 0, 3, 4},                  {0, 0, 2, 1}          };          int len = board.length;          solvePuzzle(board, len);          printBoard(board, len);      }  } |

**OUTPUT**

