Name: Nguyen Phuong Le

ID: 104178943

# **Assignment 1**

## Question A:

import java.util.List;

import java.util.ArrayList;

import java.util.Scanner;

//Create ticket class

class Ticket {

int id;

String allocatedSeat;

int seatNumber;

// Constructor

public Ticket(int id, String allocatedSeat, int seatNumber) {

this.id = id;

this.allocatedSeat = allocatedSeat;

this.seatNumber = seatNumber;

}

//Get methods

public int getId() {return id;}

public void setId(int id) {this.id = id;}

public String getAllocatedSeat() {return allocatedSeat;}

public void setAllocatedSeat(String allocatedSeat) {

this.allocatedSeat = allocatedSeat;

}

//Set methods

public int getSeatNumber() {

return seatNumber;

}

public void setSeatNumber(int seatNumber) {

this.seatNumber = seatNumber;

}

@Override

public String toString() {

return "Ticket: Id: " + id + ", allocatedSeat: " + allocatedSeat + ", seatNumber: " + seatNumber;

}

}

//Create TicketService class to create, edit, get Tickets

class TicketService {

//Create an array of ticket

private List<Ticket> tickets;

// Contructor

public TicketService(List<Ticket> tickets) {

this.tickets = tickets;

}

// Getter and setter

public List<Ticket> getTickets() {

return tickets;

}

public void setTickets(List<Ticket> tickets) {

this.tickets = tickets;

}

//Create new ticket

// Methor to create new ticket, parameter is name of seat allocated(allocatedSeat), and number of seat(seatNumber)

public void createNewTicket(String allocatedSeat, int seatNumber) {

int ticketId;

// If array ticket empty, the first ticket have id = 1

if (this.tickets.isEmpty()) {

ticketId = 1;

} else {

// If array ticket not empty, the ticket have id = last ticket of array ticket + 1

Ticket lastTicket = this.tickets.get(tickets.size() - 1);

ticketId = lastTicket.getId() + 1;

}

// Create new ticket and push it in tickets array

Ticket newTicket = new Ticket(ticketId, allocatedSeat, seatNumber);

this.tickets.add(newTicket);

return;

}

//Search for tickets by ticket ID number

public Ticket findTicketById(int id) {

for (Ticket ticket : this.tickets) {

if (ticket.getId() == id) {

return ticket;

}

}

return null;

}

//Print total number of seats booked

public void printAllTickets() {

for (Ticket ticket : this.tickets) {

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

}

}

}

/\*\*

\* Main

\*/

public class Question\_A {

// Function to initialize 2D array

public static void initializeCinema(Boolean[][] cinema) {

//All seats are empty

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 10; j++) {

cinema[i][j] = false;

}

}

}

//case 2: In the state of the sitting - Function Print all availability seat

static void printCurrentAvailability(Boolean[][] cinema) {

int row = 8;

int colum = 10;

char [][]array = new char [row][colum];

System.out.println("Current Availability: ");

//create from 1 to 10

System.out.print(" ");

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

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

}

System.out.println();

for (int i = 0; i < 8; i++) {

//create from char A to H

int a = i + 65;

char letter = (char)a;

System.out.print(letter);

System.out.print(" ");

//Number of seats available and booked

for (int j = 0; j < 10; j++) {

if (!cinema[i][j]) {

// If seat is available, show 'o'

System.out.print("o "); //empty seat

} else {

// If seat is not available, show 'x'

System.out.print("x "); //seat booked

}

}

System.out.print("\n");

}

System.out.print("\_\_\_\_\_\_\_SCREEN\_\_\_\_\_\_ \n");

}

//case 3: count and print the number of empty seats in the cinema

static int printCountAvailability(Boolean[][] cinema) {

int count = 0;

for (int i = 0; i < 8; i++) {

for (int j = 0; j < 10; j++) {

// if seat is available, count = count + 1

if (!cinema[i][j]) {

count++;

}

}

}

System.out.println("" + count + " seats are available for reservation.");

return count;

}

/\*Convert position description string to row and column index

Get seat indicaes

Example: if user input seat A2 => so we change it to posion cinema[0][1] \*/

public static int[] getSeatIndices(String seat) {

// Get the first chare, in the example rowChar is A

char rowChar = seat.charAt(0);

// Get the second number, in the example colNUm is 2

int colNum = Integer.parseInt(seat.substring(1));

// RowIndex = A - 'A' = 0

int rowIndex = rowChar - 'A';

// ColIndex = 2 - 1 = 0

int colIndex = colNum - 1;

// Return [0, 1] => cinema[0][1]

return new int[] { rowIndex, colIndex };

}

//Arrange the seats that the user order

// Get seat name mean allocatedSeat

public static String getSeatName(TicketService ticketService, int column, int row, int numberSeat) {

char rowChar = (char) (row + 'A');

int columnChar = column + 1;

String beginSeat = rowChar + "" + columnChar;

String endSeat = rowChar + "" + (columnChar + numberSeat - 1); //last seat number in row when booking

ticketService.createNewTicket(beginSeat + " to " + endSeat, numberSeat);

return beginSeat + " to " + endSeat;

}

// If seat is avaible, book seat to change it from false to be true

static void bookSeat(Boolean[][] cinema, TicketService ticketService, int column, int row, int numberSeat) {

for (int k = column; k < column + numberSeat; k++) {

cinema[row][k] = true;

}

System.out.println("Seats reserved. " + getSeatName(ticketService, column, row, numberSeat));

return;

}

//case 1: Ticket booking and seat creation process

static void reverseTicket(Boolean[][] cinema, TicketService ticketService, Scanner input) {

System.out.print("Enter how many tickets you wish to reserve? ");

int numberSeat = input.nextInt();

// Number seat must greater than 0 and less or equal than 10

if (numberSeat <= 0) {

System.out.println("Sorry – Minimum 1 tickets can be reserved at a time.");

return;

} else if (numberSeat > 10) {

System.out.println("Sorry – Maximum 10 tickets can be reserved at a time.");

return;

}

System.out.print("Do you wish the system to allocate the seats for you Y/N? ");

String isAutoSystem = input.next();

int column = -1;

int row = -1;

//System ticket allocation system

if (isAutoSystem.equals("Y") || isAutoSystem.equals("y")) {

outerLoop: for (int i = 0; i < 8; i++) {

int countAvailability = 0; //Number of empty seats in an initial row

for (int j = 0; j < 10; j++) {

if (!cinema[i][j]) //The seat has been booked.

countAvailability++;

else

countAvailability = 0;

//Check if there are enough seats for the number of tickets in a row that the user booked

if (countAvailability >= numberSeat) {

row = i;

column = j - numberSeat + 1;

break outerLoop;

}

}

}

//Seat availability or insufficient seating announcement

if (column != -1 && row != -1) {

bookSeat(cinema, ticketService, column, row, numberSeat);

} else {

System.out.println("Sorry, no allocation can be made. Insufficient seats in the row");

}

return;

}

//Get seat by user choose

else if (isAutoSystem.equals("N") || isAutoSystem.equals("n")) {

System.out.print("Please select the row & seat number that you wish to reserve your seats from: ");

String beginSeat = input.next();

int[] seat = getSeatIndices(beginSeat);

//Check if the seat is available

int countAvailability = 0; //Number of empty seats in an initial row

for (int j = seat[1]; j < 10; j++) {

// seat [1]: index of columns -> Column index starting from the entered seat number to check

// seat [0]: index of rows

if (!cinema[seat[0]][j])

countAvailability++;

else {

System.out.println("Sorry, no allocation can be made. Seats already taken.");

return;

}

if (countAvailability >= numberSeat) {

row = seat[0];

column = seat[1];

break;

}

}

//Check if there is enough space in the row to allocate seats.

if (countAvailability >= numberSeat) {

row = seat[0];

column = seat[1];

} else {

System.out.println("Sorry, no allocation can be made. Insufficient seats in the row");

return;

}

if (column != -1 && row != -1) {

bookSeat(cinema, ticketService, column, row, numberSeat);

}

return;

}

else {

System.out.println("No option " + isAutoSystem + ", Just have Y or N");

return;

}

}

//case 4: Enter ticket to find equivalent tickets

//Function search ticket by ticket id

public static void searchTicket(Scanner input, TicketService ticketService) {

System.out.print("Input ticket id: ");

int ticketId = input.nextInt();

input.nextLine();

Ticket ticket = ticketService.findTicketById(ticketId);

if (ticket == null) {

System.out.println("No ticket with id " + ticketId);

return;

}

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

}

//case 5: Print total number of seats booked

public static void printAllTickets(Scanner input, TicketService ticketService) {

System.out.println("Tickets: ");

ticketService.printAllTickets();

}

//menu

public static int menu(Scanner input) {

System.out.println("/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/");

System.out.println("1 – Reserve Tickets");

System.out.println("2 – Show Current Availability");

System.out.println("3 – Show Count of Availability");

System.out.println("4 – Search Ticket");

System.out.println("5 – Print All Tickets");

System.out.println("6 – Exit");

System.out.print("Please enter your choice: ");

int option = input.nextInt();

return option;

}

public static void main(String[] args) {

// Create a 2D array with boolean

Boolean[][] cinema = new Boolean[8][10];

// Update 2D array to be false of all position

initializeCinema(cinema);

boolean isOpenMenu = true;

// Create new array of Ticket

List<Ticket> tickets = new ArrayList<Ticket>();

// Create new Ticket serivice instance

TicketService ticketService = new TicketService(tickets);

Scanner input = new Scanner(System.in);

while (isOpenMenu) {

int clientOption = menu(input);

switch (clientOption) {

case 1:

reverseTicket(cinema, ticketService, input);

break;

case 2:

printCurrentAvailability(cinema);

break;

case 3:

printCountAvailability(cinema);

break;

case 4:

searchTicket(input, ticketService);

break;

case 5:

printAllTickets(input, ticketService);

break;

case 6:

// Exit

isOpenMenu = false;

break;

default:

System.out.println("No option " + clientOption);

break;

}

}

}

}

### UML Diagrams:

|  |
| --- |
| Ticket |
| ------------------------------------------------------------------------------------ |
| - id: int |
| - allocatedSeat: String |
| - seatNumber: int |
| ------------------------------------------------------------------------------------ |
| + Ticket(id: int, allocatedSeat: String, seatNumber: int) |
| + getId(): int |
| + setId(id: int): void |
| + getAllocatedSeat(): String |
| + setAllocatedSeat(seat: String): void |
| + setSeatNumber(seatNumber: int): void |
| + toString(): String |

1

|  |
| --- |
| TicketService |
| ------------------------------------------------------------------------------------  1 |
| - tickets: List<Ticket> |
| ------------------------------------------------------------------------------------ |
| + TicketService(tickets: List<Ticket>) |
| + getTickets(): List<Ticket> |
| + setTickets(tickets: List<Ticket>): void |
| + createNewTicket(allocatedSeat: String, seatNumber: int): void |
| + findTicketById(id: int): Ticket |
| + printAllTickets(): void |

2

2

|  |
| --- |
| Question\_A |
| ----------------------------------------------------------------------------------------------------- |
| + initializeCinema(cinema: Boolean[][]): void |
| + printCurrentAvailability(cinema: Boolean[][]): void |
| + printCountAvailability(cinema: Boolean[][]): int |
| + getSeatIndices(seat: String): int[] |
| + getSeatName(ticketService: TicketService, column: int, row: int, numberSeat: int): String |
| + bookSeat(cinema: Boolean[][], ticketService: TicketService, column: int, row: int, numberSeat: int): void |
| + reverseTicket(cinema: Boolean[][], ticketService: TicketService, input: Scanner): void |
| + searchTicket(input: Scanner, ticketService: TicketService): void |
| + printAllTickets(input: Scanner, ticketService: TicketService): void |
| + menu(input: Scanner): int |
| + main(args: String[]): void |

**Note:**

"Question A" coordinates with "TicketService".

"TicketService" coordinates with "Ticket".

"Question\_A" can indirectly access "Ticket" information through "TicketService".

**Question\_A → TicketService → Ticket.**

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

## Question B:

import java.util.ArrayList;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

//Creat enum for cover type, other is not valid

enum CoverType {

Vehicle,

Health,

Travel,

Property,

Pet,

}

//Class customer

class Customer {

// Max customer's policy is 5

private static final int MAX\_POLICY\_NUMBER = 5;

private long customerId;

private String name;

private String address;

// new array list of policies

private ArrayList<Policy> policies = new ArrayList<Policy>(MAX\_POLICY\_NUMBER);

// Constructor

public Customer(long customerId, String name, String address) {

this.customerId = customerId;

this.name = name;

this.address = address;

}

//Get methods and Set methods

public long getCustomerId() {

return customerId;

}

public void setCustomerId(long customerId) {

this.customerId = customerId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public static int getMaxPolicyNumber() {

return MAX\_POLICY\_NUMBER;

}

public ArrayList<Policy> getPolicies() {

return policies;

}

public void setPolicies(ArrayList<Policy> policies) {

this.policies = policies;

}

// To string

@Override

public String toString() {

return "Customer\_Id: C" + customerId + ", Name: " + name + ", Address: " + address;

}

}

//Class policy

class Policy {

private long policyId;

private long customerId;

private CoverType coverType;

private long coverValue;

private String duration;

private long installmentAmount;

private String paymentPlan;

private Date startDate;

private boolean isActive;

//Constructor

public Policy(long policyId, long customerId, CoverType coverType, long coverValue, String duration,

long installmentAmount, String paymentPlan, Date startDate) {

this.policyId = policyId;

this.customerId = customerId;

this.coverType = coverType;

this.coverValue = coverValue;

this.duration = duration;

this.installmentAmount = installmentAmount;

this.paymentPlan = paymentPlan;

this.startDate = startDate;

this.isActive = true;

}

//Get methods and Set methods

public long getPolicyId() {return policyId;}

public void setPolicyId(long policyId) {this.policyId = policyId;}

public long getCustomerId() {return customerId;}

public void setCustomerId(long customerId) {this.customerId = customerId;}

public CoverType getCoverType() {return coverType;}

public void setCoverType(CoverType coverType) {this.coverType = coverType;}

public long getCoverValue() {return coverValue;}

public void setCoverValue(long coverValue) {this.coverValue = coverValue;}

public String getDuration() {return duration;}

public void setDuration(String duration) {this.duration = duration;}

public long getInstallmentAmount() {return installmentAmount;}

public void setInstallmentAmount(long installmentAmount) {

this.installmentAmount = installmentAmount;

}

public String getPaymentPlan() {return paymentPlan;}

public void setPaymentPlan(String paymentPlan) {

this.paymentPlan = paymentPlan;

}

public Date getStartDate() {return startDate;}

public void setStartDate(Date startDate) {

this.startDate = startDate;

}

//format the date to "YYYY/MM/DD" format

public String formatDate(Date date) {

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");

// Date format and in console out

String formattedDate = dateFormat.format(date);

return formattedDate;

}

// To string

@Override

public String toString() {

return "Policy\_ID: P" + policyId + ", Customer\_ID: " + customerId + ", Cover Type: " + coverType

+ ", Cover Value: $" + coverValue + ", Duration: " + duration + ", Installment Amount: $" + installmentAmount

+ ", Payment Plan: " + paymentPlan + ", Start Date: " + formatDate(startDate) + ", Is Active: " + isActive;

}

public boolean isActive() {

return isActive;

}

public void setActive(boolean isActive) {

this.isActive = isActive;

}

}

//Create class customer service to create new customer, update customer, delete customer and get customers

class CustomerService {

// An Array list of customers

private List<Customer> customers;

//Constructor

public CustomerService(List<Customer> customers) {

this.customers = customers;

}

// Getter and Setter

public List<Customer> getCustomers() {

return customers;

}

public void setCustomers(List<Customer> customers) {

this.customers = customers;

}

// Creation of new Customers [case 1]

public void createNewCustomer(String customerName, String address) {

long customerId;

// If array customer is empty, new customer id is 1001

if (this.customers.isEmpty()) {

customerId = 1001;

} else {

// If array customer have some customer, new customer id is last customer id in array + 1

Customer lastCustomer = this.customers.get(this.customers.size() - 1);

customerId = lastCustomer.getCustomerId() + 1;

}

Customer newCustomer = new Customer(customerId, customerName, address);

this.customers.add(newCustomer);

}

// Print all customer in arrays

//Check if there are any customers in the class Customer [case 2]

public void printAllCustomer(boolean isPrintPolicy) {

System.out.println("List customers:");

if (customers.isEmpty()) {

//Empty list

System.out.println("Empty customer!!!");

return;

}

for (Customer customer : customers) {

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

// If user want to print policy of customer, print array policies of each customer

//Check if there are any customers in the class Policy

if (isPrintPolicy) {

if (customer.getPolicies().isEmpty()) {

//Empty list

System.out.println("\tPolicies: None");//false

} else {

for (Policy policy : customer.getPolicies()) {

System.out.println("\t\t" + policy.toString());//true

}

}

}

}

}

//Search for a customer in the customer list by customer ID [case 5]

public Customer findCustomerById(String customerId) {

//Iterate over all Customer objects in the customers list.

for (Customer customer : customers) {

if (customerId.equals("C" + customer.getCustomerId())) {

return customer;

}

}

return null;

}

//Search for a customer in the customer list by customer Name

public List<Customer> findCustomerByName(String customerName) {

List<Customer> listCustomer = new ArrayList<Customer>();

for (Customer customer : customers) {

if (customer.getName().equals(customerName)) {

listCustomer.add(customer);

}

}

return listCustomer;

}

//Search for a customer based on the policy ID

public Customer findByPolicyId(String policyId) {

for (Customer customer : customers) {

List<Policy> policies = customer.getPolicies();

for (Policy policy : policies) {

if (policyId.equals("P" + policy.getPolicyId())) {

return customer;

}

}

}

return null;

}

}

//Create class policy service to create new policy, update policy, delete policy and get policies

class PolicyService {

// An Array list of policies

private List<Policy> policies;

// Constructor

public PolicyService(List<Policy> policies) {

this.policies = policies;

}

//create a new policy for a customer [case 3]

public void createNewPolicy(String customerId, CustomerService customerService, CoverType coverType,

Long coverValue, String duration, long installmentAmount, String paymentPlan, Date startDate) {

//It helps to find and connect between Policy and Customer.

//Search for customers based on customer ID help for

Customer customer = customerService.findCustomerById(customerId);

// If cannot find customer by customer id, return fail

if (customer == null) {

System.out.println("No customer with id " + customerId);

return;

}

//Store the ID of the new policy

long policyId;

if (this.policies.isEmpty()) { //Check if policies list is empty

policyId = 101; //If array policy is empty, new policy id is 101

} else {

// If array policy have some policies, new policy id is last policy id in array + 1

Policy lastPolicy = policies.get(policies.size() - 1); //see last value in list

policyId = lastPolicy.getPolicyId() + 1; //Create a new ID number for the policy

}

//All the information that the new policy needs to have

Policy newPolicy = new Policy(policyId, customer.getCustomerId(), coverType, coverValue, duration,

installmentAmount, paymentPlan, startDate);

//Push it into policies array list

this.policies.add(newPolicy);

//Push it into customer policies array list

customer.getPolicies().add(newPolicy);

return;

}

// Check if there are any policies in the class Policy [case 4]

public void printAllPolicies() {

System.out.println("List policies:");

if (this.policies.isEmpty()) {

//Empty list

System.out.println("Empty policies!!!");

return;

}

for (Policy policy : policies) {

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

}

}

//Search for a policy in the policies list by policy ID

public Policy findPolicyById(String policyId) {

for (Policy policy : policies) {

if (policyId.equals("P" + policy.getPolicyId())) {

return policy;

}

}

return null;

}

// Method calculate total cover of all policy [case 8]

public long totalCoverValue() {

long total = 0;

for (Policy policy : policies) {

total += policy.getCoverValue();

}

return total;

}

//Method calculate total installment amount of all policy [case 9]

public long totalInstallmentAmount() {

long total = 0;

for (Policy policy : policies) {

total += policy.getInstallmentAmount();

}

return total;

}

}

/\*\*

\* Main

\*/

public class Question\_B {

// Max customer is 7

private static final int MAX\_CUSTOMER = 7;

//case 1: Creation of new Customers

public static void createNewCustomer(Scanner input, CustomerService customerService) {

System.out.print("Input name: ");

String customerName = input.nextLine();

System.out.print("Input address: ");

String address = input.nextLine();

customerService.createNewCustomer(customerName, address);

return;

}

//case 2: print All Customer

public static void printAllCustomer(Scanner input, CustomerService customerService) {

System.out.print("Are you want to print policy too? ");

String yonChoice = input.nextLine();

if (yonChoice.equals("Y") || yonChoice.equals("y")) {

customerService.printAllCustomer(true);

} else if (yonChoice.equals("N") || yonChoice.equals("n")) {

customerService.printAllCustomer(false);

} else {

System.out.println("No have choice " + yonChoice);

}

return;

}

//case 3:Creation of new Policy

public static void createNewPolicy(Scanner input, CustomerService customerService, PolicyService policyService) {

System.out.print("Input customer Id: ");

String customerId = input.nextLine();

System.out.print("Input cover type (Vehicle / Health / Travel / Property / Pet): ");

String coverTypeStr = input.nextLine();

CoverType coverType;

//Helps handle exceptions, keeping the application running smoothly

//Check if the 5 values ​​are entered correctly in CoverType

try {

coverType = CoverType.valueOf(coverTypeStr);

} catch (Exception e) {

System.out.println("Invalid cover type!!!");

return;

}

System.out.print("Input cover value: ");

long coverValue = input.nextLong();

input.nextLine();

System.out.print("Input duration: ");

String duration = input.nextLine();

System.out.print("Input installment amount: ");

long installmentAmount = input.nextLong();

input.nextLine();

System.out.print("Input payment plan: ");

String paymentPlan = input.nextLine();

policyService.createNewPolicy(customerId, customerService, coverType, coverValue, duration, installmentAmount,

paymentPlan, new Date());

//new Date(): Get the date from the laptop system in use

return;

}

//case 4:print All Policies

public static void printAllPolicies(PolicyService policyService) {

policyService.printAllPolicies();

return;

}

//case 5: Search for customer information

public static void searchCustomer(Scanner input, CustomerService customerService) {

System.out.println("Search by: ");

System.out.println("a. Customer id.");

System.out.println("b. Customer name.");

System.out.println("c. Policy id.");

System.out.print("Input your option: ");

char option = input.nextLine().charAt(0);

if (option == 'a') {

System.out.print("Input customer id: ");

String customerId = input.nextLine();

Customer customer = customerService.findCustomerById(customerId);

if (customer == null) {

System.out.println("No customer with id " + customerId);

return;

}

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

return;

} else if (option == 'b') {

System.out.print("Input customer name: ");

String customerName = input.nextLine();

List<Customer> listCustomer = customerService.findCustomerByName(customerName);

if (listCustomer.isEmpty()) {

System.out.println("No customer with name " + customerName);

return;

}

for (Customer customer : listCustomer) {

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

}

return;

} else if (option == 'c') {

System.out.print("Input policy id: ");

String policyId = input.nextLine();

Customer customer = customerService.findByPolicyId(policyId);

if (customer == null) {

System.out.println("No customer with policy id " + policyId);

return;

}

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

return;

} else {

System.out.println("No option " + option);

return;

}

}

//case 6: update Policy

public static void updatePolicy(Scanner input, PolicyService policyService) {

System.out.print("Input policy id to update: ");

String policyId = input.nextLine();

//search for Policy by Id and then store it in policy variable

Policy policy = policyService.findPolicyById(policyId);

if (policy == null) {

System.out.println("No policy with id " + policyId);

return;

}

//menu for update

System.out.println("What do you want to update ?");

System.out.println("a. Update cover type.");

System.out.println("b. Update cover value.");

System.out.println("c. Update cover duration.");

System.out.println("d. Update cover installment amount.");

System.out.println("f. Update cover payment plan.");

System.out.print("Input your option: ");

char option = input.nextLine().charAt(0);

switch (option) {

case 'a':

System.out.print("Input cover type: ");

System.out.print("Input cover type (Vehicle / Health / Travel / Property / Pet): ");

String coverTypeStr = input.nextLine();

CoverType coverType;

try {

coverType = CoverType.valueOf(coverTypeStr);

} catch (Exception e) {

System.out.println("Invalid cover type!!!");

break;

}

policy.setCoverType(coverType);

break;

case 'b':

System.out.print("Input cover value: ");

long coverValue = input.nextLong();

input.nextLine();

policy.setCoverValue(coverValue);

break;

case 'c':

System.out.print("Input duration: ");

String duration = input.nextLine();

policy.setDuration(duration);

break;

case 'd':

System.out.print("Input installment amount: ");

long installmentAmount = input.nextLong();

input.nextLine();

policy.setInstallmentAmount(installmentAmount);

break;

case 'f':

System.out.print("Input payment plan: ");

String paymentPlan = input.nextLine();

policy.setPaymentPlan(paymentPlan);

break;

default:

break;

}

return;

}

//case 7: Function delete policy to change is active to false

public static void deletePolicy(Scanner input, PolicyService policyService) {

System.out.print("Input policy id to delete: ");

String policyId = input.nextLine();

//search for Policy by Id and then store it in policy variable

Policy policy = policyService.findPolicyById(policyId);

if (policy == null) {

System.out.println("No policy with id " + policyId);

return;

}

//disable policy

policy.setActive(false);

return;

}

//case 8: Function get totle cover value

public static void totalCoverValue(PolicyService policyService) {

System.out.println("Total cover value: " + policyService.totalCoverValue());

}

//case 9: Function get total installment amount

public static void totalInstallmentAmount(PolicyService policyService) {

System.out.println("Total amount value: " + policyService.totalInstallmentAmount());

}

public static void menu() {

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

System.out.println("1. Create new Customer.");

System.out.println("2. Print All Customer.");

System.out.println("3. Create new Policy.");

System.out.println("4. Print all Policy.");

System.out.println("5. Search customer.");

System.out.println("6. Update Policy.");

System.out.println("7. Delete Policy.");

System.out.println("8. Total Cover Value.");

System.out.println("9. Total Installment Amount.");

System.out.println("10. Exit.");

System.out.print("Input your option: ");

}

public static void main(String[] args) {

boolean isOpenMenu = true;

// Create Customer Array List

List<Customer> customers = new ArrayList<Customer>(MAX\_CUSTOMER);

CustomerService customerService = new CustomerService(customers);

// Create Policy Array List

List<Policy> policies = new ArrayList<Policy>();

PolicyService policyService = new PolicyService(policies);

Scanner input = new Scanner(System.in);

while (isOpenMenu) {

menu();

int option = input.nextInt();

input.nextLine();

switch (option) {

case 1:

createNewCustomer(input, customerService);

break;

case 2:

printAllCustomer(input, customerService);

break;

case 3:

createNewPolicy(input, customerService, policyService);

break;

case 4:

printAllPolicies(policyService);

break;

case 5:

searchCustomer(input, customerService);

break;

case 6:

updatePolicy(input, policyService);

break;

case 7:

deletePolicy(input, policyService);

break;

case 8:

totalCoverValue(policyService);

break;

case 9:

totalInstallmentAmount(policyService);

break;

case 10:

isOpenMenu = false;

break;

default:

break;

}

}

input.close();

}

}

### UML Diagrams

|  |
| --- |
| CoverType |
| ------------------------------------------------------------------------------------ |
| + Vehicle |
| + Health |
| + Travel |
| + Property |
| + Pet |

|  |
| --- |
| Customer |
| ------------------------------------------------------------------------------------ |
| - MAX\_POLICY\_NUMBER: int |
| - name: String |
| - address: String |
| - policies: ArrayList<Policy |
| ------------------------------------------------------------------------------------ |
| + Customer(long, String, String) |
| + getCustomerId(): long |
| + setCustomerId(long): void |
| + setName(String): void |
| + getAddress(): String |
| + setAddress(String): void |
| + getMaxPolicyNumber(): int |
| + getPolicies(): ArrayList<Policy> |
| + setPolicies(ArrayList<Policy>): void |
| + toString(): String |

1

|  |
| --- |
| CustomerService |
| ------------------------------------------------------------------------------------  1 |
| - customers: List<Customer> |
| ------------------------------------------------------------------------------------ |
| + CustomerService(List<Customer>) |
| + getCustomers(): List<Customer> |
| + setCustomers(List<Customer>): void |
| + createNewCustomer(String, String): void |
| + printAllCustomer(boolean): void |
| + findCustomerById(String): Customer |
| + findCustomerByName(String): List<Customer> |
| + findByPolicyId(String): Customer |

|  |
| --- |
| Policy |
| ------------------------------------------------------------------------------------ |
| - policyId: long |
| - customerId: long |
| - coverType: CoverType |
| - coverValue: long |
| - duration: String |
| - installmentAmount: long |
| - paymentPlan: String |
| - startDate: Date |
| - isActive: boolean |
| ------------------------------------------------------------------------------------ |
| + Policy(long, long, CoverType, long, String, long, String, Date) |
| + getPolicyId(): long |
| + setPolicyId(long): void |
| + getCustomerId(): long |
| + setCustomerId(long): void |
| + getCoverType(): CoverType |
| + setCoverType(CoverType): void |
| + getCoverValue(): long |
| + setCoverValue(long): void |
| + getDuration(): String |
| + setDuration(String): void |
| + getInstallmentAmount(): long |
| + setInstallmentAmount(long): void |
| + getPaymentPlan(): String |
| + setPaymentPlan(String): void |
| + getStartDate(): Date |
| + setStartDate(Date): void |
| + formatDate(Date): String |
| + isActive(): boolean |
| + setActive(boolean): void |
| + toString(): String |

2

2

|  |
| --- |
| PolicyService |
| ------------------------------------------------------------------------------------ |
| - policies: List<Policy> |
| ------------------------------------------------------------------------------------ |
| + PolicyService(List<Policy>) |
| + createNewPolicy(String, CustomerService, CoverType, long, String, long, String, Date): void |
| + printAllPolicies(): void |
| + findPolicyById(String): Policy |
| + totalCoverValue(): long |
| + totalInstallmentAmount(): long |
| + findCustomerByName(String): List<Customer> |
| + findByPolicyId(String): Customer |

|  |
| --- |
| Question\_B |
| ----------------------------------------------------------------------------------------------------- |
| - MAX\_CUSTOMER: int |
| ----------------------------------------------------------------------------------------------------- |
| + createNewCustomer(Scanner, CustomerService): void |
| + printAllCustomer(Scanner, CustomerService): void |
| + createNewPolicy(Scanner, CustomerService, PolicyService): void |
| + printAllPolicies(PolicyService): void |
| + searchCustomer(Scanner, CustomerService): void |
| + updatePolicy(Scanner, PolicyService): void |
| + deletePolicy(Scanner, PolicyService): void |
| + totalCoverValue(PolicyService): void |
| + totalInstallmentAmount(PolicyService): void |
| + menu(): void |
| + main(args: String[]): void |

**Note:**

1. **Create New Customer:**

Question\_B → CustomerService → Customer.

1. **Create New Policy:**

Question\_B → PolicyService → CustomerService (to find customers) → Policy → update “Policy” list of “Customer”

1. **Search for customers by code:**

Question\_B → CustomerService → PolicyService (find policy) → return "Customer"

1. **Print Customer and Policy:**

Question\_B → CustomerService → Customer → Policy

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A black background with white lines

Description automatically generated

A screenshot of a computer program

Description automatically generated

A black background with white and blue lines

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A black rectangular frame with small colored lines

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated