Java Programming

**Name:** Colm Kearney

**Assignments:** Question 1: Develop a match ticket delivery system.

Contents

1.0 Question 1 3

A). 4

B). 10

C). 12

References: 12

Appendix 1 51

Question 1 51

Source Code: TicketDelivery.Java 51

Source Code: StandardTicketDelivery.Java 54

Source Code: NextDayTicketDelivery.Java 55

Source Code: TicketDeliveryController.Java 56

Source Code: TicketDeliveryControllerTest.Java 61

Source Code: Drone.Java 65

Source Code: DroneFlight.Java 69

Source Code: DroneFlightPaths.Java 78

Source Code: FlightPath.Java 80

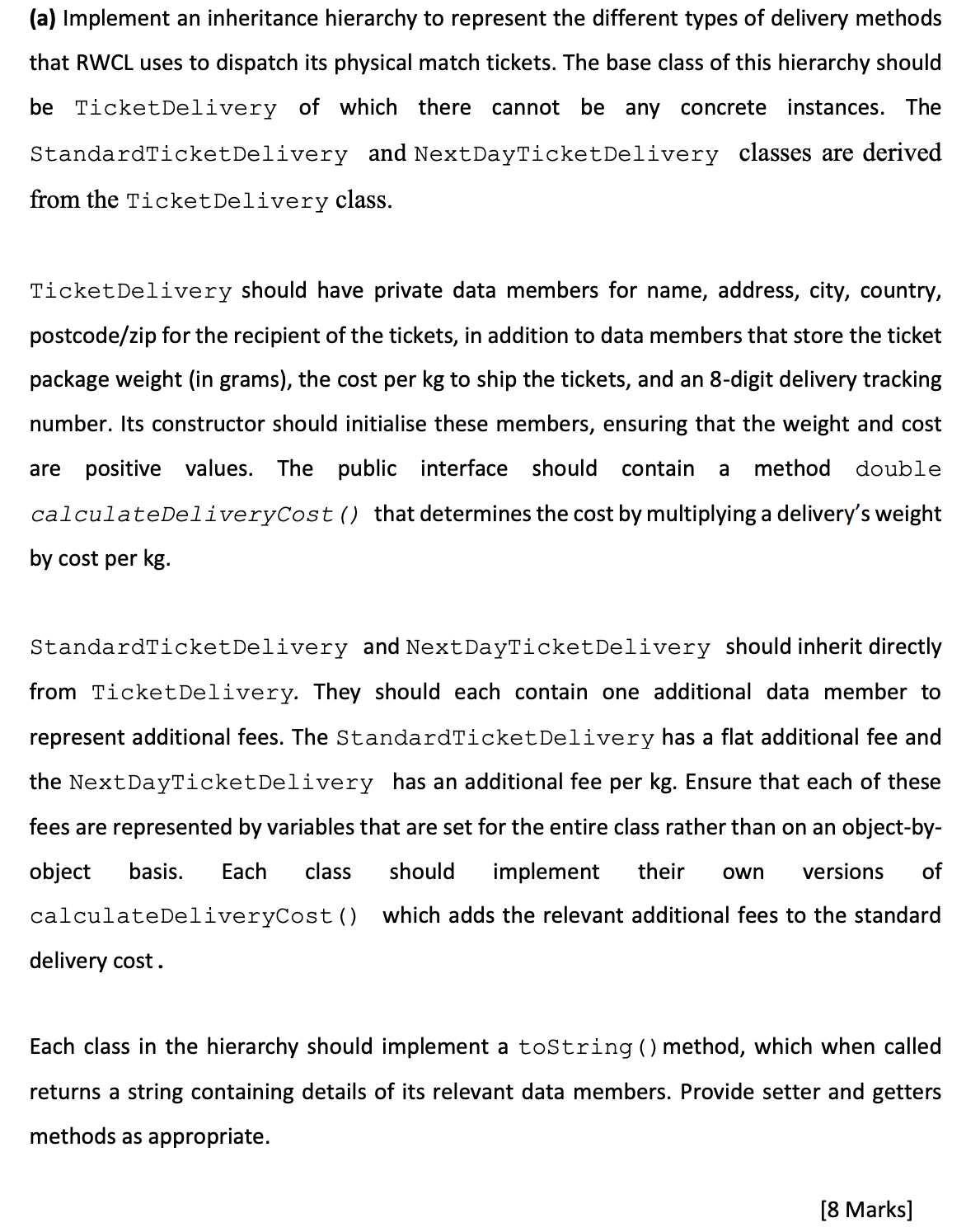
Source Code: Waypoint.Java 91

Source Code: ClosestWaypoint.Java 93

Source Code: DroneWaypoint.Java 94

# 1.0 Question 1

****



## A).

I began this question by designing the TicketDelivery class. The question specifies that there cannot be concrete instances so it will be a public abstract class. The variables listed in the question were then created. All the address details are private Strings, the ticketPackageWeight is a double and costPerKgToShip is a float as it is financial calculation. The ticketPackageTrackingNumber began as a double but was later changed to a String as the double was unable to record the correct eight digit number during testing. Within the constructor I have two if statements to ensure the ticketPackageWeight and costPerKgToShip values are both a number above zero. The constructor and variables can be seen below in Fig. 1.1:



Fig. 1.1 TicketDelivery variables and constructor

Next, I created the interface calculateDeliveryCost(). This calculates the cost by multiplying the delivery’s weight by cost per kg. This can be seen in Fig. 1.2 below:

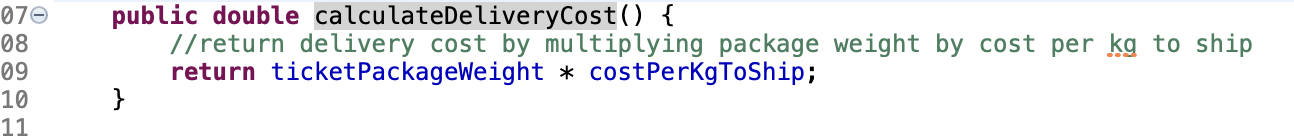


Fig. 1.2 calculateDeliveryCost()

The StandardTicketDelivery inherits directly from TicketDelivery and adds a variable for a standard additional fee. It also has a version of calculateDeliveryCost with its additional fee present. This can be seen below in Fig. 1.3:

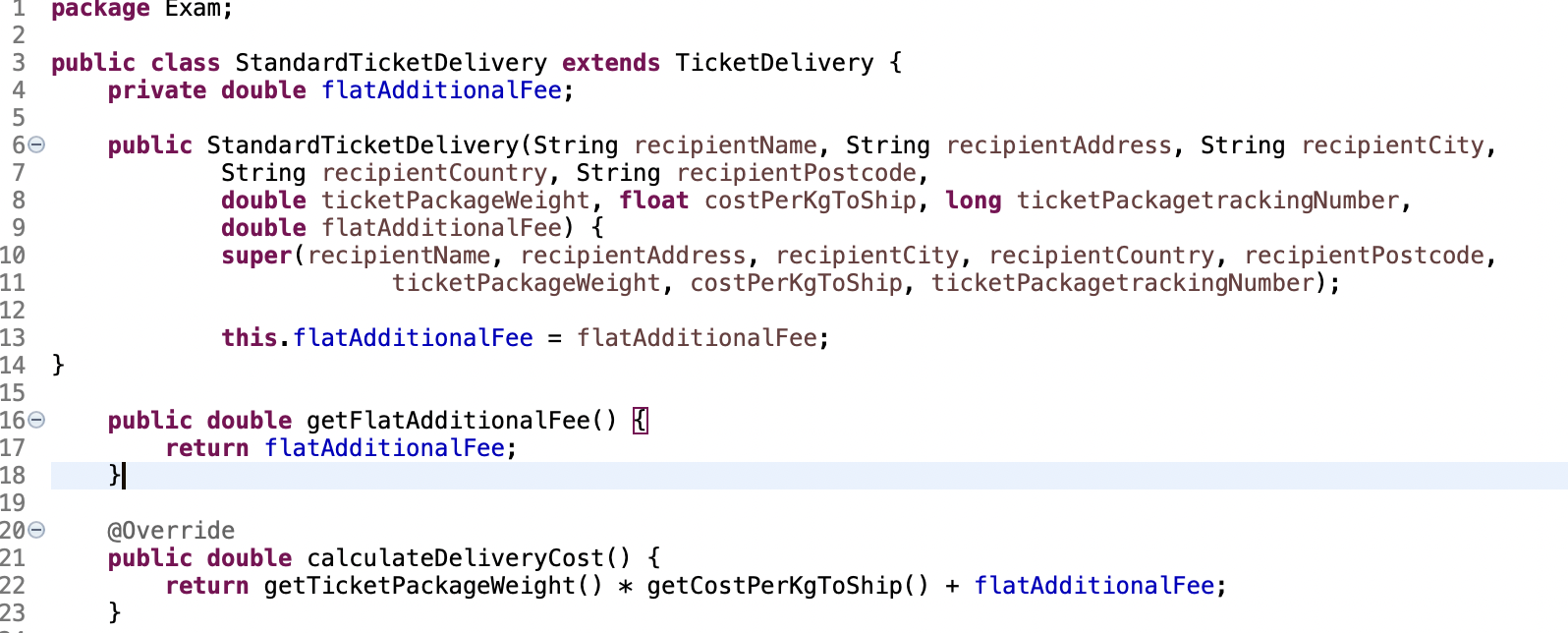


Fig. 1.3: StandardTicketDelivery constructor and calculateDeliveryCost() method

A similar fee is present in the NextDayTicketDelivery class and it is called perKgAdditionalFee. This class also has a version of calculateDeliveryCost with its additional fees added on. This can be seen in Fig.1.4 below:



Fig. 1.4 NextDayTicketDelivery constructor and calculateDeliveryCost() method

Each class in the hierarchy also contains a toString method. The first toString method created was for TicketDelivery and the other classes were based of this. The TicketDelivery toString method can be seen below in Fig. 1.5:

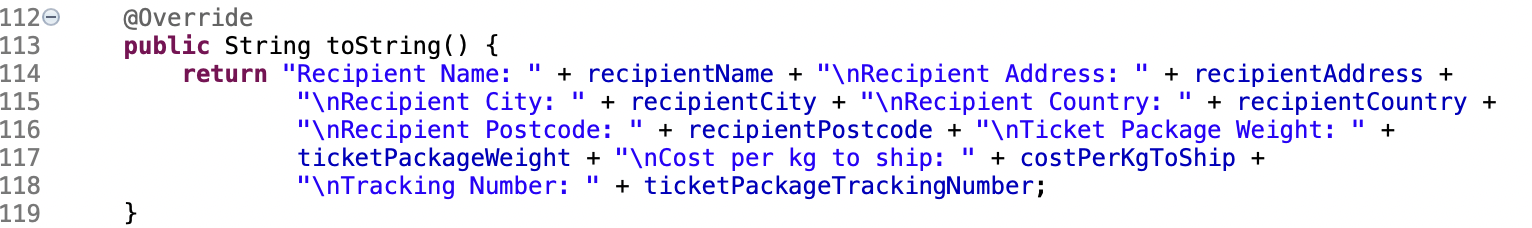


Fig. 1.5: TicketDelivery toString()

Once toString was created for the TicketDelivery class I expanded upon it for the next two classes and added their additional requirements. The toString method for StandardTicketDelivery is in Fig. 1.6 below:

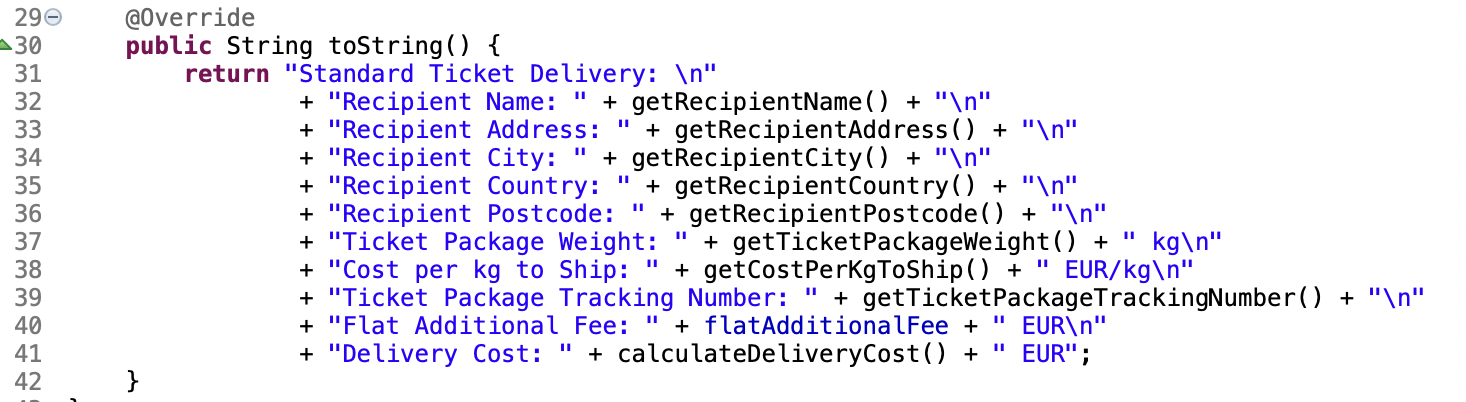


Fig. 1.6 toString method in StandardTicketDelivery class

The toString method for the NextDayTicketDelivery class is below in Fig. 1.7 below:



Fig.1.7 toString method in NextDayTicketDelivery class.

Additionally, getter and setter methods were required predominately in the TicketDelivery class but also some in StandardTicketDelivery and NextDayTicketDelivery especially to calculate their additional fees. The getter and setter for StandardTicketDelivery can be seen below in Fig. 1.8:

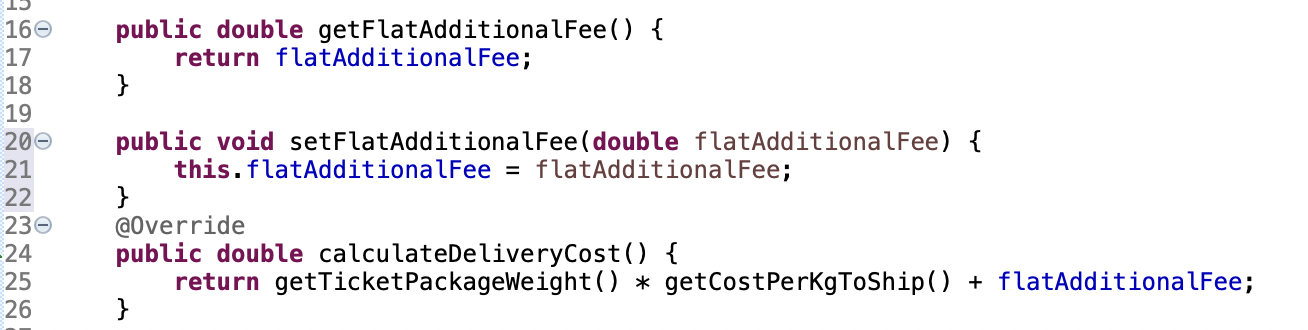


Fig.1.8 Getter and Setter for Standard Ticket Delivery

The getter and setter for NextDayTicketDelivery can be seen below in Fig. 1.8:

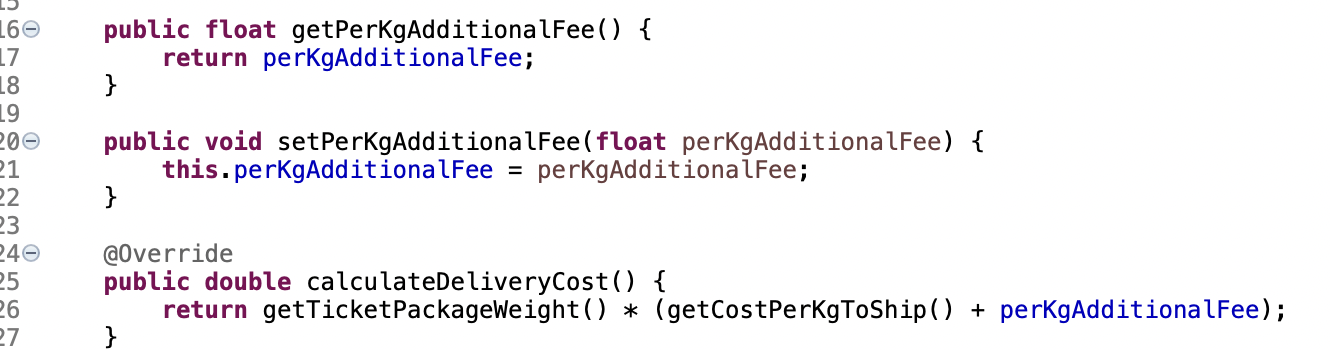


Fig.1.8 Getter and Setter for NextDayTicketDelivery

The TicketDelivery class required a getter and setter for each variable it held. These an be seen in Fig. 1.9 & 1.10 below:



Fig. 1.9 TicketDelivery Getters and Setters 1 of 2

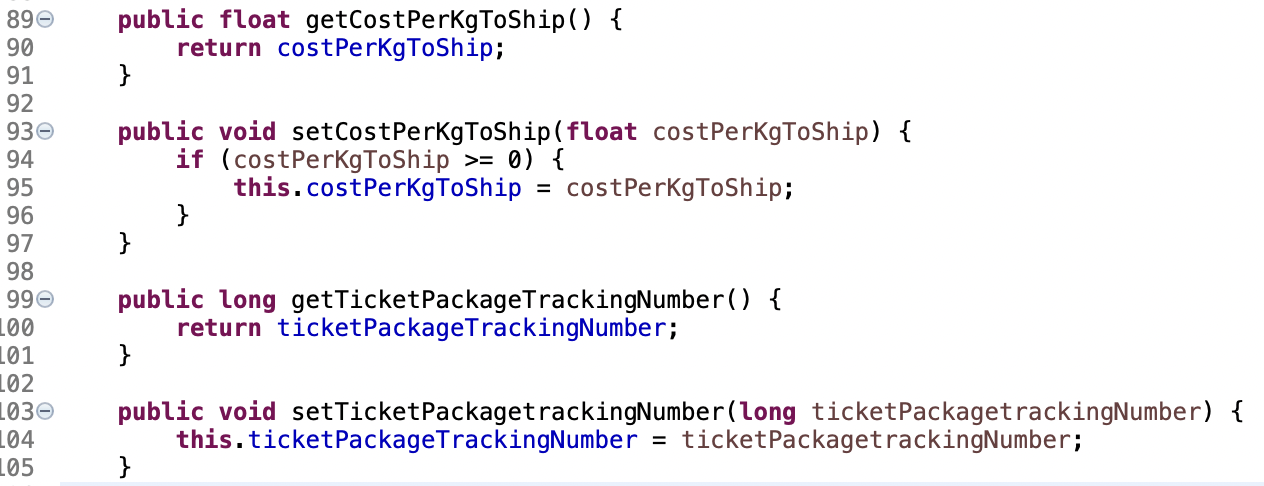
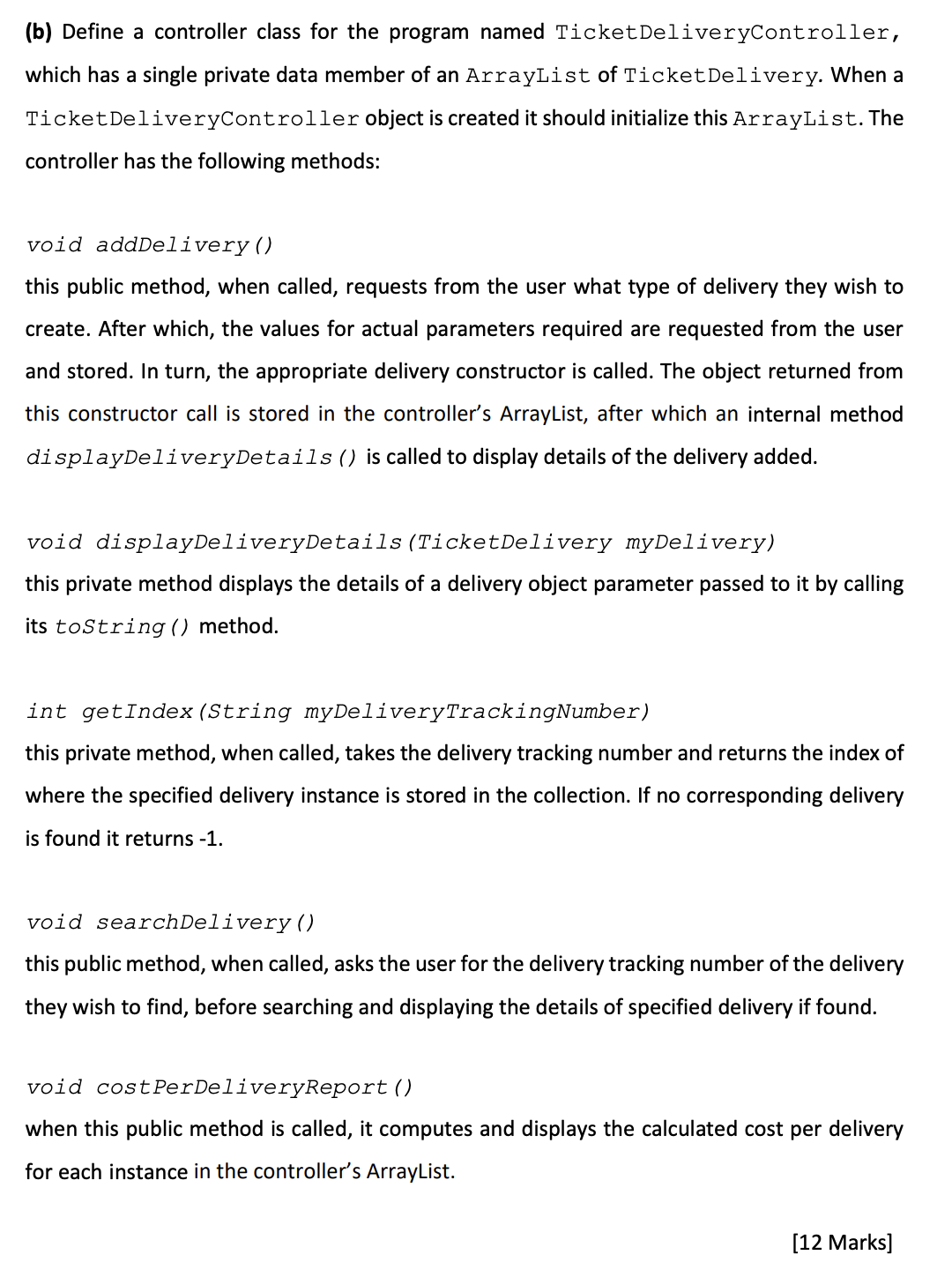


Fig. 1.10 Fig. 1.9 TicketDelivery Getters and Setters 2 of 2



## B).

The TicketDeliveryController class was created with the required ArrayList of TicketDelivery. When the controller object is created it initialises the ArrayList. This can be seen in the code in Fig. 1.11 below:

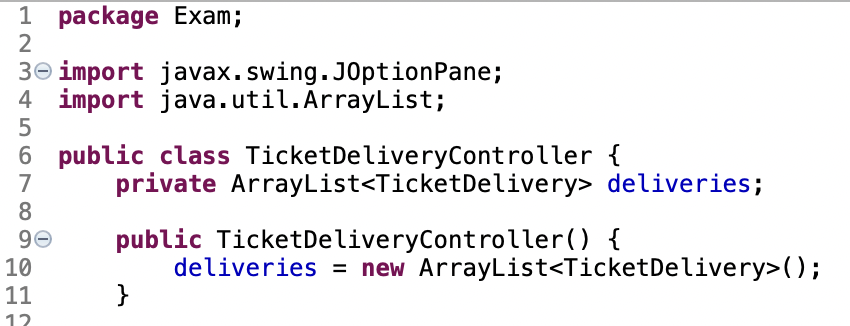


Fig. 1.11 TicketDeliveryController class with ArrayList.

When creating the addDelivery() method I opted to utilise JOptionPane for user input. In this method , the delivery details will be gathered from the user and stored in the ArrayList. The delivery details will also be presented to the user after entering them. This is performed by the last line in the method: displayDeliveryDetails(delivery). The addDelivery() method can be seen in Fig.1.12 below:

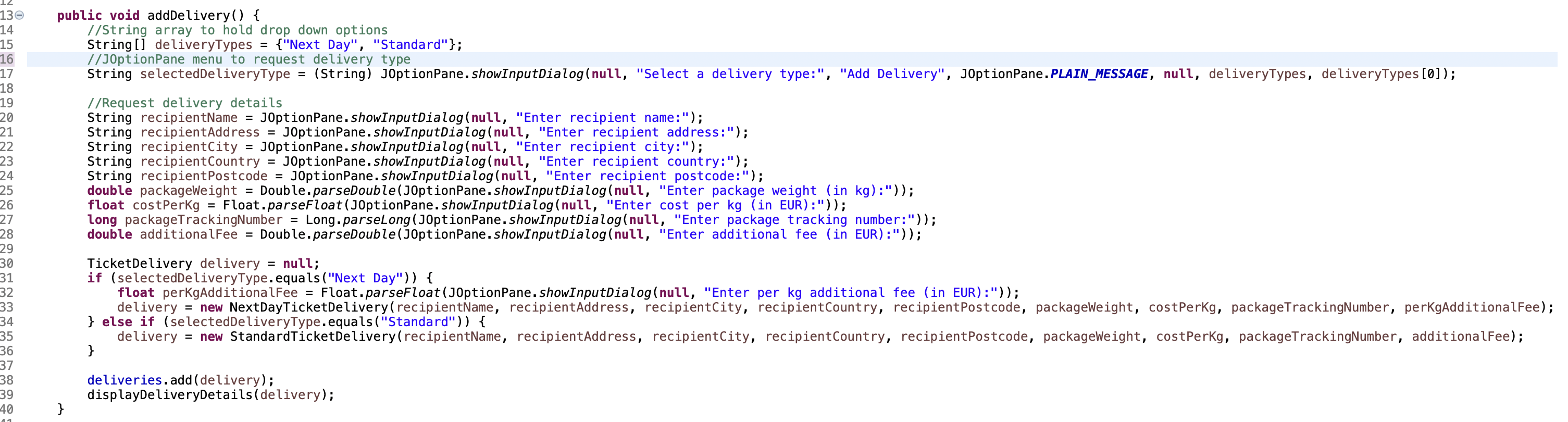


Fig.1.12 addDelivery().

The associated method displayDeliveryDetails() is responsible for showing the inputted details back to the end user. This is a very simple method that utilise the toString method to relay the information back to the user. This method can be seen below in Fig. 1.13:

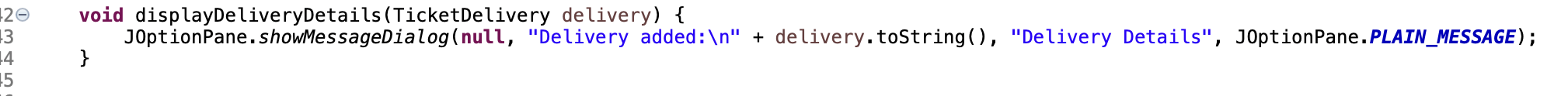


Fig. 1.13 displayDeliveryDetails() method.

The next required method is a getIndex method which searches for a tracking number and returns the location where it is stored in the collection. This method can be seen below in Fig. 1.14 below:

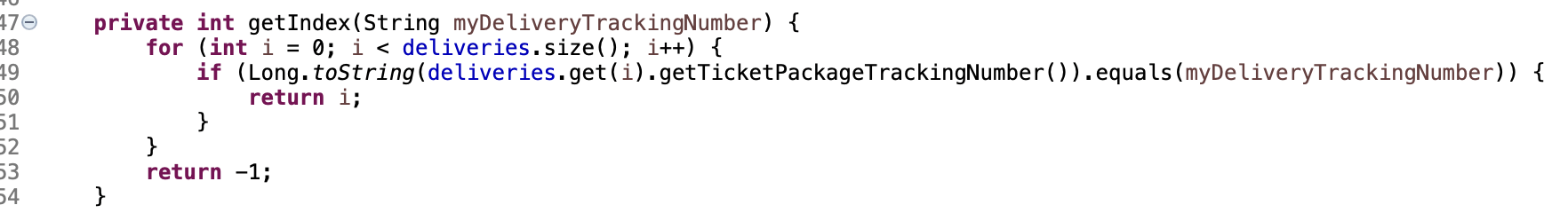


Fig. 1.14 getIndex().

The searchDelivery() method is the method the end user can utilise to search for a delivery by its tracking number and display it on the screen for the user. If the tracking number exists in the index then the .toString method is called to display the information of the delivery on the GUI. If it is not found then an error message is sent to the GUI instead. This code can be seen in Fig. 1.15 below:

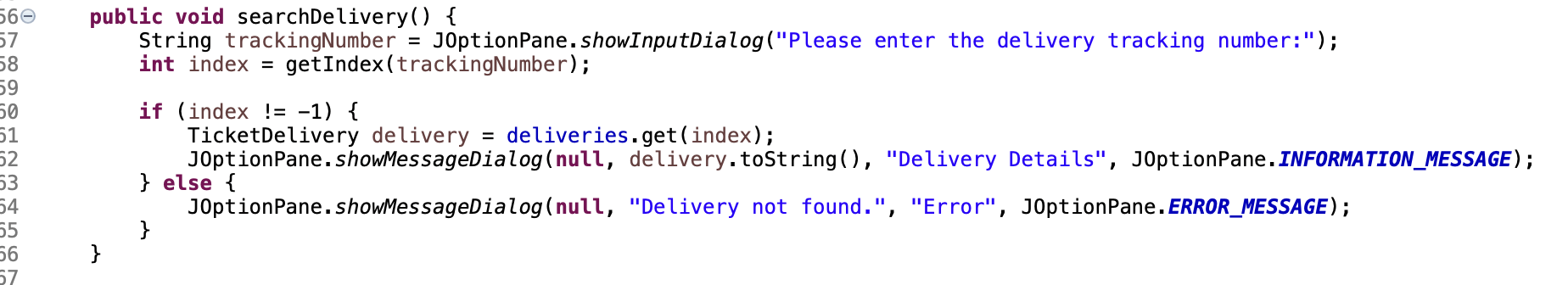


Fig. 1.15 searchDelivery().

The method costPerDeliveryReport() is used to calculate the final cost of the delivery. In this method, the for loop will loop through the deliveries and a call is made to the method calculateDeliveryCost() to calculate the delivery. This method exists in StandardTicketDelivery and NextDayTicketDelivery classes and can calculate the final delivery costs including their specific fees. The code for costPerDeliveryReport() is below in Fig. 1.16:

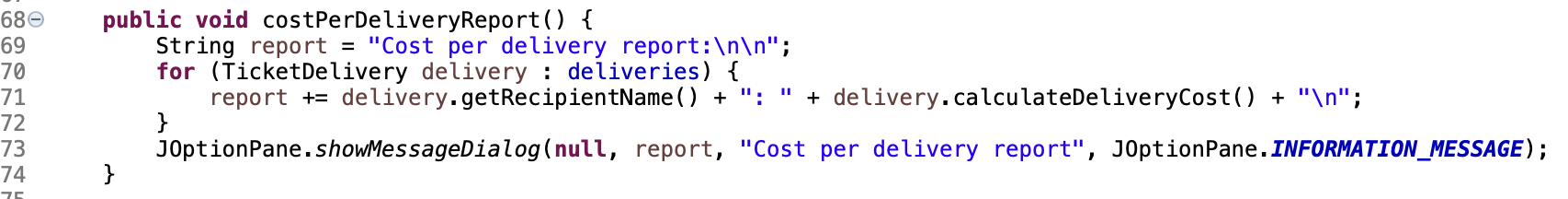
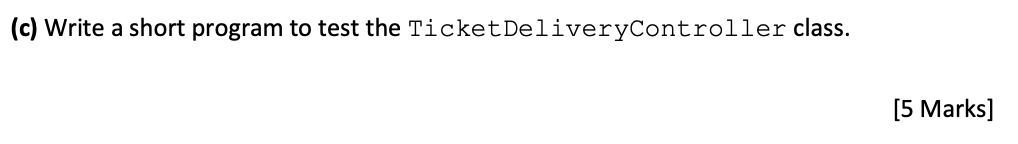


Fig. 1.16 costPerDeliveryReport().

## C).

  
In order to test the TicketDeliveryController class, I will need to test all the available functionality contained within it. To do this I will create a delivery, search for a delivery by the tracking number and produce a cost per delivery report. I created the TicketDeliveryControllerTest class to for the test. I have also added a menu for the end user to access these features. The code for this can be seen below in Fig. 1.17:

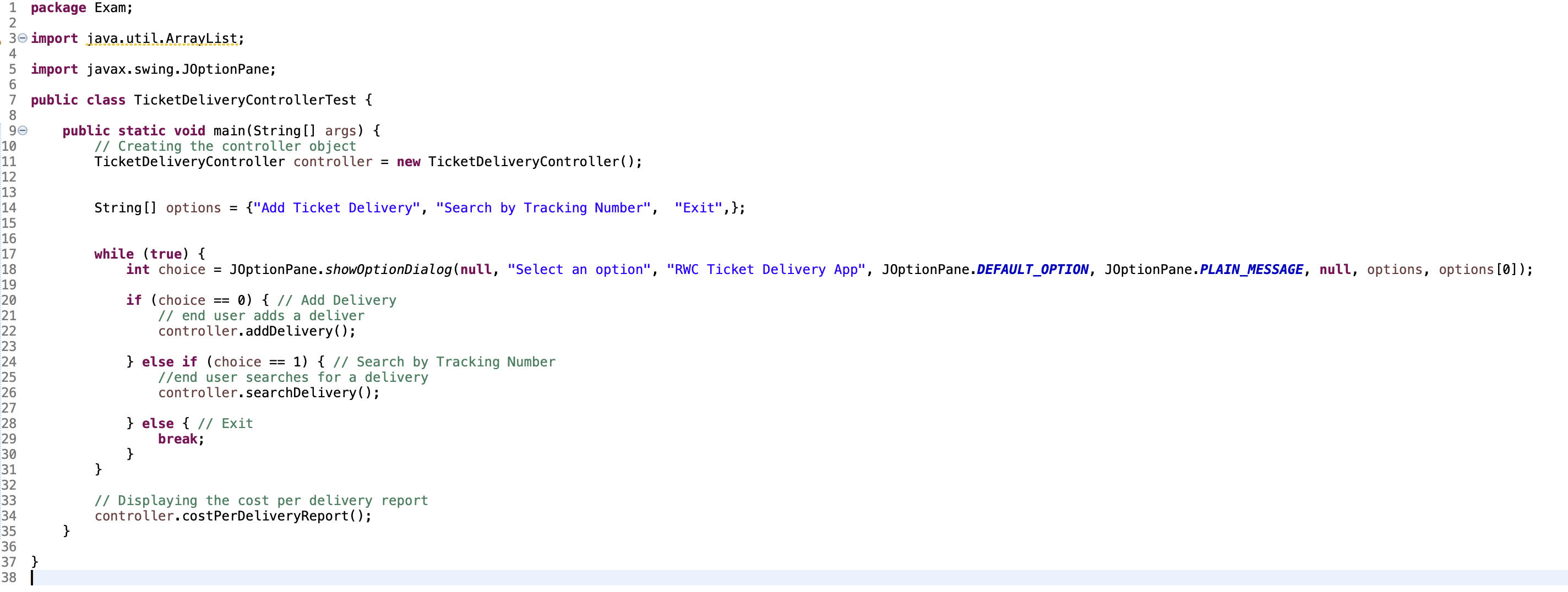


Fig. 1.17 TicketDeliveryControllerTest class

# Appendix 1

# Question 1

# Source Code: **TicketDelivery.Java**

**package** Exam;

**public** **abstract** **class** TicketDelivery {

**private** String recipientName;

**private** String recipientAddress;

**private** String recipientCity;

**private** String recipientCountry;

**private** String recipientPostcode;

**private** **double** ticketPackageWeight;

**private** **float** costPerKgToShip;

**private** **long** ticketPackageTrackingNumber;

**public** TicketDelivery(String recipientName, String recipientAddress, String recipientCity,

String recipientCountry, String recipientPostcode, **double** ticketPackageWeight,

**float** costPerKgToShip, **long** ticketPackageTrackingNumber) {

**this**.recipientName = recipientName;

**this**.recipientAddress = recipientAddress;

**this**.recipientCity = recipientCity;

**this**.recipientCountry = recipientCountry;

**this**.recipientPostcode = recipientPostcode;

**if** (ticketPackageWeight > 0) {

**this**.ticketPackageWeight = ticketPackageWeight;

} **else** {

**throw** **new** IllegalArgumentException("Ticket package weight must be positive");

}

**if** (costPerKgToShip > 0) {

**this**.costPerKgToShip = costPerKgToShip;

} **else** {

**throw** **new** IllegalArgumentException("Cost per kg to ship must be positive");

}

**this**.ticketPackageTrackingNumber = ticketPackageTrackingNumber;

}

**public** String getRecipientName() {

**return** recipientName;

}

**public** **void** setRecipientName(String recipientName) {

**this**.recipientName = recipientName;

}

**public** String getRecipientAddress() {

**return** recipientAddress;

}

**public** **void** setRecipientAddress(String recipientAddress) {

**this**.recipientAddress = recipientAddress;

}

**public** String getRecipientCity() {

**return** recipientCity;

}

**public** **void** setRecipientCity(String recipientCity) {

**this**.recipientCity = recipientCity;

}

**public** String getRecipientCountry() {

**return** recipientCountry;

}

**public** **void** setRecipientCountry(String recipientCountry) {

**this**.recipientCountry = recipientCountry;

}

**public** String getRecipientPostcode() {

**return** recipientPostcode;

}

**public** **void** setRecipientPostcode(String recipientPostcode) {

**this**.recipientPostcode = recipientPostcode;

}

**public** **double** getTicketPackageWeight() {

**return** ticketPackageWeight;

}

**public** **void** setTicketPackageWeight(**double** ticketPackageWeight) {

**if** (ticketPackageWeight >= 0) {

**this**.ticketPackageWeight = ticketPackageWeight;

}

}

**public** **float** getCostPerKgToShip() {

**return** costPerKgToShip;

}

**public** **void** setCostPerKgToShip(**float** costPerKgToShip) {

**if** (costPerKgToShip >= 0) {

**this**.costPerKgToShip = costPerKgToShip;

}

}

**public** **long** getTicketPackageTrackingNumber() {

**return** ticketPackageTrackingNumber;

}

**public** **void** setTicketPackagetrackingNumber(**long** ticketPackagetrackingNumber) {

**this**.ticketPackageTrackingNumber = ticketPackagetrackingNumber;

}

**public** **double** calculateDeliveryCost() {

//return delivery cost by multiplying package weight by cost per kg to ship

**return** ticketPackageWeight \* costPerKgToShip;

}

@Override

**public** String toString() {

**return** "Recipient Name: " + recipientName + "\nRecipient Address: " + recipientAddress +

"\nRecipient City: " + recipientCity + "\nRecipient Country: " + recipientCountry +

"\nRecipient Postcode: " + recipientPostcode + "\nTicket Package Weight: " +

ticketPackageWeight + "\nCost per kg to ship: " + costPerKgToShip +

"\nTracking Number: " + ticketPackageTrackingNumber;

}

}

# Source Code: **StandardTicketDelivery.Java**

**package** Exam;

**public** **class** StandardTicketDelivery **extends** TicketDelivery {

**private** **double** flatAdditionalFee;

**public** StandardTicketDelivery(String recipientName, String recipientAddress, String recipientCity,

String recipientCountry, String recipientPostcode,

**double** ticketPackageWeight, **float** costPerKgToShip, **long** ticketPackagetrackingNumber,

**double** flatAdditionalFee) {

**super**(recipientName, recipientAddress, recipientCity, recipientCountry, recipientPostcode,

ticketPackageWeight, costPerKgToShip, ticketPackagetrackingNumber);

**this**.flatAdditionalFee = flatAdditionalFee;

}

**public** **double** getFlatAdditionalFee() {

**return** flatAdditionalFee;

}

**public** **void** setFlatAdditionalFee(**double** flatAdditionalFee) {

**this**.flatAdditionalFee = flatAdditionalFee;

}

@Override

**public** **double** calculateDeliveryCost() {

**return** getTicketPackageWeight() \* getCostPerKgToShip() + flatAdditionalFee;

}

@Override

**public** String toString() {

**return** "Standard Ticket Delivery: \n"

+ "Recipient Name: " + getRecipientName() + "\n"

+ "Recipient Address: " + getRecipientAddress() + "\n"

+ "Recipient City: " + getRecipientCity() + "\n"

+ "Recipient Country: " + getRecipientCountry() + "\n"

+ "Recipient Postcode: " + getRecipientPostcode() + "\n"

+ "Ticket Package Weight: " + getTicketPackageWeight() + " kg\n"

+ "Cost per kg to Ship: " + getCostPerKgToShip() + " EUR/kg\n"

+ "Ticket Package Tracking Number: " + getTicketPackageTrackingNumber() + "\n"

+ "Flat Additional Fee: " + flatAdditionalFee + " EUR\n"

+ "Delivery Cost: " + calculateDeliveryCost() + " EUR";

}

}

# Source Code: **NextDayTicketDelivery.Java**

**package** Exam;

**public** **class** NextDayTicketDelivery **extends** TicketDelivery {

**private** **float** perKgAdditionalFee;

**public** NextDayTicketDelivery(String recipientName, String recipientAddress, String recipientCity,

String recipientCountry, String recipientPostcode,

**double** ticketPackageWeight, **float** costPerKgToShip, **long** ticketPackagetrackingNumber,

**float** perKgAdditionalFee) {

**super**(recipientName, recipientAddress, recipientCity, recipientCountry, recipientPostcode,

ticketPackageWeight, costPerKgToShip, ticketPackagetrackingNumber);

**this**.perKgAdditionalFee = perKgAdditionalFee;

}

**public** **float** getPerKgAdditionalFee() {

**return** perKgAdditionalFee;

}

**public** **void** setPerKgAdditionalFee(**float** perKgAdditionalFee) {

**this**.perKgAdditionalFee = perKgAdditionalFee;

}

@Override

**public** **double** calculateDeliveryCost() {

**return** getTicketPackageWeight() \* (getCostPerKgToShip() + perKgAdditionalFee);

}

@Override

**public** String toString() {

**return** "NextDayTicketDelivery" +

"recipientName='" + getRecipientName() + "\n" +

"recipientAddress='" + getRecipientAddress() + "\n" +

"recipientCity='" + getRecipientCity() + "\n" +

"recipientCountry='" + getRecipientCountry() + "\n" +

"recipientPostcode='" + getRecipientPostcode() + "\n" +

"ticketPackageWeight=" + getTicketPackageWeight() + "\n" +

"costPerKgToShip=" + getCostPerKgToShip() + "EUR/kg\n" +

"ticketPackageTrackingNumber=" + getTicketPackageTrackingNumber() + "\n" +

"perKgAdditionalFee=" + perKgAdditionalFee + "EUR\n" +

"Delivery Cost: " + calculateDeliveryCost()+ " EUR";

}

}

# Source Code: **TicketDeliveryController.Java**

package Exam;

import javax.swing.JOptionPane;

import java.util.ArrayList;

public class TicketDeliveryController {

private ArrayList<TicketDelivery> deliveries;

public TicketDeliveryController() {

deliveries = new ArrayList<TicketDelivery>();

}

public void addDelivery() {

//String array to hold drop down options

String[] deliveryTypes = {"Next Day", "Standard"};

//JOptionPane menu to request delivery type

String selectedDeliveryType = (String) JOptionPane.showInputDialog(null, "Select a delivery type:", "Add Delivery", JOptionPane.PLAIN\_MESSAGE, null, deliveryTypes, deliveryTypes[0]);

//Request delivery details

String recipientName = JOptionPane.showInputDialog(null, "Enter recipient name:");

String recipientAddress = JOptionPane.showInputDialog(null, "Enter recipient address:");

String recipientCity = JOptionPane.showInputDialog(null, "Enter recipient city:");

String recipientCountry = JOptionPane.showInputDialog(null, "Enter recipient country:");

String recipientPostcode = JOptionPane.showInputDialog(null, "Enter recipient postcode:");

double packageWeight = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter package weight (in kg):"));

float costPerKg = Float.parseFloat(JOptionPane.showInputDialog(null, "Enter cost per kg (in EUR):"));

long packageTrackingNumber = Long.parseLong(JOptionPane.showInputDialog(null, "Enter package tracking number:"));

double additionalFee = Double.parseDouble(JOptionPane.showInputDialog(null, "Enter additional fee (in EUR):"));

TicketDelivery delivery = null;

if (selectedDeliveryType.equals("Next Day")) {

float perKgAdditionalFee = Float.parseFloat(JOptionPane.showInputDialog(null, "Enter per kg additional fee (in EUR):"));

delivery = new NextDayTicketDelivery(recipientName, recipientAddress, recipientCity, recipientCountry, recipientPostcode, packageWeight, costPerKg, packageTrackingNumber, perKgAdditionalFee);

} else if (selectedDeliveryType.equals("Standard")) {

delivery = new StandardTicketDelivery(recipientName, recipientAddress, recipientCity, recipientCountry, recipientPostcode, packageWeight, costPerKg, packageTrackingNumber, additionalFee);

}

deliveries.add(delivery);

displayDeliveryDetails(delivery);

}

void displayDeliveryDetails(TicketDelivery delivery) {

JOptionPane.showMessageDialog(null, "Delivery added:\n" + delivery.toString(), "Delivery Details", JOptionPane.PLAIN\_MESSAGE);

}

private int getIndex(String myDeliveryTrackingNumber) {

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

if (Long.toString(deliveries.get(i).getTicketPackageTrackingNumber()).equals(myDeliveryTrackingNumber)) {

return i;

}

}

return -1;

}

public void searchDelivery() {

String trackingNumber = JOptionPane.showInputDialog("Please enter the delivery tracking number:");

int index = getIndex(trackingNumber);

if (index != -1) {

TicketDelivery delivery = deliveries.get(index);

JOptionPane.showMessageDialog(null, delivery.toString(), "Delivery Details", JOptionPane.INFORMATION\_MESSAGE);

} else {

JOptionPane.showMessageDialog(null, "Delivery not found.", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

public void costPerDeliveryReport() {

String report = "Cost per delivery report:\n\n";

for (TicketDelivery delivery : deliveries) {

report += delivery.getRecipientName() + ": " + delivery.calculateDeliveryCost() + "\n";

}

JOptionPane.showMessageDialog(null, report, "Cost per delivery report", JOptionPane.INFORMATION\_MESSAGE);

}

}

# Source Code: **TicketDeliveryControllerTest.Java**

package Exam;

import java.util.ArrayList;

import javax.swing.JOptionPane;

public class TicketDeliveryControllerTest {

public static void main(String[] args) {

// Creating the controller object

TicketDeliveryController controller = new TicketDeliveryController();

String[] options = {"Add Ticket Delivery", "Search by Tracking Number", "Exit",};

while (true) {

int choice = JOptionPane.showOptionDialog(null, "Select an option", "RWC Ticket Delivery App", JOptionPane.DEFAULT\_OPTION, JOptionPane.PLAIN\_MESSAGE, null, options, options[0]);

if (choice == 0) { // Add Delivery

// end user adds a deliver

controller.addDelivery();

} else if (choice == 1) { // Search by Tracking Number

//end user searches for a delivery

controller.searchDelivery();

} else { // Exit

break;

}

}

// Displaying the cost per delivery report

controller.costPerDeliveryReport();

}

}