UNIVERSITI TUNKU ABDUL RAHMAN

JUNE 2025 Trimester

UECS2354 SOFTWARE TESTING

GROUP ASSIGNMENT

GROUP NUMBER: *43*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Student Name** | **Student ID** | **Practical Group** | **Part A**  **(CO2)** | **Part B**  **(CO2)** | **Part C**  **(CO3)** | **Part D**  **(CO2)** | **Total Marks** |
|  | **Leon Siow Yi Hong** | **2204403** | **P4** |  |  |  |  |  |
| **2.** | **Quak Jing** | **2205378** | **P4** |  |  |  |  |  |
| **3.** | **Edmund Chan Chee An** | **2202241** | **P3** |  |  |  |  |  |
| **4.** | **Narvin A/L Chandrasegaran** | **2202920** | **P1** |  |  |  |  |  |

**Part A: Test Plan (10 marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Introduction (objectives, scope and test basis), assumptions, entry, exit criteria and test items | 5 |  |  |
| Test plans (features to be tested, test conditions) | 5 |  |  |
| Total | 10 |  |  |

**Part B: Test Design (30 marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Appropriate usage of test design approach: Decision table, Partition (BVA/EP) | 15 |  |  |
| Correctly created test cases | 15 |  |  |
| Total | 30 |  |  |

**Part C: Java Program (application code and test code) (50 marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Correctly implemented test method for valid cases:   * Appropriate use of assertsXXX methods. * Using parameterised tests correctly | 15 |  |  |
| Correctly implemented test method for invalid cases:   * Appropriate use of assertsXXX methods. * Using parameterised tests correctly | 15 |  |  |
| Correctly implemented test method for test doubles:   * Appropriate creation of test doubles (Mock and Stubs) * Appropriate use of assertsXXX methods. * Using parameterised tests correctly | 10 |  |  |
| Setting up some tests so that test values are read from a text file instead of hardcoding into test code | 5 |  |  |
| Perform integration testing after unit tests have been completed | 5 |  |  |
| Total | 50 |  |  |

**Part D: Presentation (10 marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Individual Presentation** | **Max Mark** | **Marks Obtained** | **Remark/Comment** |
| Student 1: | 10 |  |  |
| Student 2: | 10 |  |  |
| Student 3: | 10 |  |  |
| Student 4: | 10 |  |  |
| Total | 10 |  |  |

Comments:

|  |
| --- |
|  |

# **Assumptions (if any)**

* 1. User/Guest registrations are handled in the front-end of the software and appended to the existing list of Users/Guest, the new list is then written in to the specified file to record the new User/Guest.
  2. The front-end of the software directly invokes the methods in “Booking.java” separately to get and display data relevant to the fare, such as total fare, discounted fare, discount details and payment amount separately.

# **Changes made (if any)**

* 1. “Booking.java” invokes each method from “CalculateFare.java” separately, which include “calculateTotalFare”, “calculateDiscountedFare”, and “calculatePayment” to calculate the total fare, discounted fare, payment amount separately as well as return the discount details.
  2. The total fare, discounted fare, discount details, and payment amount are all stored in the “CalculateFare.java” class.
  3. The discount details of the fare are saved and returned as a list of String based on the amount of passenger types provided by “CalculateFare.java”.
  4. “ApplyDiscountSurcharge.java” and “PaymentMethodAdjustment.java” are combined into a singular class called “FareAdjustment.java” which handles the calculation of the fare adjustment rate using parameters provided by “CalculateFare.java”.
  5. The fare discount and surcharge are not applied to the fare directly. Instead, the fare adjustment rate is retrieved from the methods in “FareAdjustment.java” and are applied to the fare in “CalculateFare.java”.
  6. A new class “Station.java” is created to store data relevant to each station, such as station name, linking stations and distance between stations.
  7. A new interface “IUser.java” is created to be implemented by both “User.java” and “Guest.java” such that both objects can be stored in “Booking.java” under the same declaration.
  8. A new class, “Guest.java” is created to store all relevant Guest data, such as name, email and phone number.
  9. “AddNewUser.java” and “ReadUser.java” are combined into a singular class called “UserFile.java” which handles both read and write functions for Users.
  10. “AddNewGuest.java” is combined with a new method called “readGuestFromFile” into a singular class called “GuestFile.java” which handles both read and write functions for Guests.
  11. New Users/Guests are appended to the existing list of Users/Guests and both lists are written to the file instead appending it.

# **Class diagram (to reflect the application code design)**

A diagram of a computer

AI-generated content may be incorrect.

# **The copy of application source code and test codes**

# **Source Code**

Booking.java

package utar.edu.my;

import java.util.List;

public class Booking {

// Booking Data

private IUser booker;

private String travelDay;

private String travelTime;

private String startStation;

private String endStation;

private List<String> passengerType;

private List<Integer> passengerQuantity;

private String paymentMethod;

// Status

private String bookingStatus = "Not Paid";

private String paymentStatus = "Pending";

// Classes

private CalculateFare cf;

private Payment p;

// Constructors

public Booking(CalculateFare cf, Payment p) {

this.cf = cf;

this.p = p;

}

public Booking(IUser booker, String travelDay, String travelTime, String startStation, String endStation, List<String> passengerType,

List<Integer> passengerQuantity, String paymentMethod) {

// Classes

cf = new CalculateFare();

p = new Payment();

// Data

this.booker = booker;

this.travelDay = travelDay;

this.travelTime = travelTime;

this.startStation = startStation;

this.endStation = endStation;

this.passengerType = passengerType;

this.passengerQuantity = passengerQuantity;

this.paymentMethod = paymentMethod;

}

// Get Methods

public String getBookingStatus() {

return bookingStatus;

}

public String getPaymentStatus() {

return paymentStatus;

}

// Set Methods

public void setTravelDay(String travelDay) {

this.travelDay = travelDay;

}

public void setTravelTime(String travelTime) {

this.travelTime = travelTime;

}

public void setStartStation(String startStation) {

this.startStation = startStation;

}

public void setEndStation(String endStation) {

this.endStation = endStation;

}

public void setPassengerType(List<String> passengerType) {

this.passengerType = passengerType;

}

public void setPassengerQuantity(List<Integer> passengerQuantity) {

this.passengerQuantity = passengerQuantity;

}

public void setPaymentMethod(String paymentMethod) {

this.paymentMethod = paymentMethod;

}

// Calculate Total Fare For One Person

public double getTotalFare() {

cf.calculateTotalFare(startStation, endStation);

return cf.getTotalFare();

}

// Calculate Discounted Fare Amount For All Person

public double getDiscountedFare() {

cf.calculateDiscountedFare(travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity);

return cf.getDiscountedFare();

}

// Get Details of Discount

public List<String> getDiscountDetails() {

List<String> details = cf.getAdjustmentDetails();

if (details == null || details.size() == 0)

throw new IllegalArgumentException("No Details Found");

else

return details;

}

// Make Payment

public void makePayment() {

// Calculate Payment Amount

cf.calculatePayment(paymentMethod);

// Make Payment

p.makePayment(cf.getPaymentAmount());

// Print Receipt On Completion

paymentStatus = p.getStatus();

if (paymentStatus.equals("Paid"))

p.emailReceipt();

bookingStatus = "Confirmed Booking";

}

}

CalculateFare.java

package utar.edu.my;

import java.util.ArrayList;

import java.util.List;

public class CalculateFare {

// Integrated Classes

private RouteInfo ri;

private FareAdjustment fa;

// Data Fields

private double distance;

private double totalFare;

private double discountedFare;

private double paymentAmount;

private List<String> adjustmentDetails;

// Constructors

public CalculateFare() {

ri = new RouteInfo();

fa = new FareAdjustment();

}

public CalculateFare(RouteInfo ri, FareAdjustment fa) {

this.ri = ri;

this.fa = fa;

}

// Get Methods

public double getDistance() {

return distance;

}

public double getTotalFare() {

return totalFare;

}

public double getDiscountedFare() {

return discountedFare;

}

public double getPaymentAmount() {

return paymentAmount;

}

public List<String> getAdjustmentDetails() {

return adjustmentDetails;

}

// Set Methods

public void setDiscountedFare(double discountedFare) {

this.discountedFare = discountedFare;

}

// Other Methods

// Calculate Total Fare Based on Distance Traveled

public void calculateTotalFare(String startStation, String endStation) {

Double fare = 0.0;

// Get Distance Traveled

distance = ri.getRouteDistance(startStation, endStation);

fa.setTravelDistance(distance);

// Invalid Distance Range

if (distance < 1)

throw new IllegalArgumentException("No Routes Found");

else if (distance > 30)

throw new IllegalArgumentException("Distance Exceeds Limit");

// Valid Distance

else {

// Calculate Fare for Distance

if (distance > 20)

fare += 20;

else if (distance > 15)

fare += 15;

else if (distance > 10)

fare += 10;

else if (distance > 5)

fare += 5;

else

fare += 2;

}

totalFare = fare;

}

// Calculate Fare After Discount + Passenger

public void calculateDiscountedFare(String travelDay, String travelTime, String startStation, String endStation, List<String> passengerType, List<Integer> passengerQuantity) {

double sumOfFare = 0;

List<String> detailList = new ArrayList<> ();

// Calculate Distance Fare for Distance Traveled

calculateTotalFare(startStation, endStation);

// Null or Empty Passenger Type

if (passengerType == null || passengerType.size() == 0)

throw new IllegalArgumentException("Passenger Type Cannot Be Null or Empty");

// Null or Empty Passenger Quantity

else if (passengerQuantity == null || passengerQuantity.size() == 0) {

throw new IllegalArgumentException("Passenger Quantity Cannot Be Null or Empty");

}

// Passenger Type & Passenger Quantity Are of Different Lengths

else if (passengerType.size() != passengerQuantity.size())

throw new IllegalArgumentException("Passenger Quantity Does Not Match Passenger Type");

// Loop through Passengers

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

double fare = totalFare;

String detail = new String();

// Calculate Fare By Passenger Type

double passengerDiscount = fa.passengerAdjustment(passengerType.get(i));

fare \*= passengerDiscount;

detail += "Passenger Adjustment : " + Math.*ceil*((passengerDiscount - 1) \* 100) + " %\n";

// Calculate Fare By Day & Time

double dayTimeDiscount = fa.dayTimeAdjustment(travelDay, travelTime);

if (dayTimeDiscount == 2) {

fare += dayTimeDiscount;

detail += "Day Time Adjustment : + RM 2.00\n";

}

else {

fare \*= dayTimeDiscount/100;

detail += "Day Time Adjustment : " + (double) (dayTimeDiscount - 100) + " %\n";

}

// Multiply Fare By Passenger Amount

fare \*= passengerQuantity.get(i);

detail += "Passenger Amount : " + passengerQuantity.get(i) + "\n";

// Add to Total Fare

sumOfFare += fare;

detailList.add(detail);

}

discountedFare = sumOfFare;

adjustmentDetails = detailList;

}

// Calculate Payment For Fare Based On Payment Method

public void calculatePayment(String paymentMethod) {

// Null Fare Amount

if (discountedFare <= 0)

throw new IllegalArgumentException("Fare Has Not Been Calculated");

// Calculate Payment By Payment Method

double paymentAdjustment = fa.paymentMethodAdjustment(paymentMethod);

paymentAmount = discountedFare \* paymentAdjustment;

}

}

FareAdjustment.java

package utar.edu.my;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class FareAdjustment {

private double travelDistance = 0;

// Set Methods

public void setTravelDistance(double travelDistance) {

this.travelDistance = travelDistance;

}

// Passenger Type

// Validate Passenger Type & Return Formated Passenger Type

public String validatePassengerType(String passengerType) {

// Valid Passenger Types

List<String> validPassengers = new ArrayList<> (Arrays.*asList*("adult", "senior citizen"

, "student", "child"));

// Null Passenger Type

if (passengerType == null) {

throw new IllegalArgumentException("Passenger Type Cannot Be Null");

}

// Format Passenger Type

passengerType = passengerType.trim().toLowerCase();

// Invalid Passenger Type

if (!validPassengers.contains(passengerType))

throw new IllegalArgumentException("Invalid Passenger Type");

// Valid Passenger Type

else

return passengerType;

}

// Fare Adjustment Based on Passenger Type

public double passengerAdjustment(String passengerType) {

double adjustment = 1.0;

// Invalid Travel Distance Range

if (travelDistance < 1 || travelDistance > 30)

throw new IllegalArgumentException("Invalid Travel Distance");

// Providing discount Rate Based on Passenger Type

switch (validatePassengerType(passengerType)) {

case "adult" -> adjustment = 1.0;

case "senior citizen" -> adjustment = 0.5;

case "student" -> adjustment = 0.7;

case "child" -> {

if (travelDistance >= 5)

adjustment = 0.5;

else

adjustment = 0.0;

}

}

return adjustment;

}

// Travel Day

// Validate Travel Day & Return Boolean isWeekend

public boolean isWeekend(String travelDay) {

// Valid Days of Week

List<String> daysOfWeek = new ArrayList<> (Arrays.*asList*("monday", "tuesday", "wednesday",

"thursday", "friday", "saturday", "sunday"));

// Null Travel Day

if (travelDay == null)

throw new IllegalArgumentException("Travel Day Cannot Be Null");

// Format Travel Day

travelDay = travelDay.trim().toLowerCase();

// Invalid Travel Day

if (!daysOfWeek.contains(travelDay))

throw new IllegalArgumentException("Invalid Travel Day");

// Valid Travel Day

else

// Weekends

if (travelDay.equals("saturday") || travelDay.equals("sunday"))

return true;

// Weekdays

else

return false;

}

// Validate Travel Time & Return Time as Integer

public int validateTravelTime(String travelTime) {

int time;

// Null Travel Time

if (travelTime == null)

throw new IllegalArgumentException("Travel Time Cannot Be Null");

try {

// Convert Time to Integer

time = Integer.*parseInt*(travelTime);

// Invalid Time Range

if (time < 0 || time > 2359)

throw new NumberFormatException();

// Invalid Time Format

else if (time%100 >= 60)

throw new NumberFormatException();

}

catch (NumberFormatException e) {

throw new IllegalArgumentException("Invalid Time Format");

}

return time;

}

// Payment Adjustment Based on Day & Time (24 hour Format)

public int dayTimeAdjustment(String travelDay, String travelTime) {

// Normal Discount Rate

int adjustment = 100;

// Validate Inputs

boolean weekend = isWeekend(travelDay);

int time = validateTravelTime(travelTime);

// Night Sur-charge (+RM2)

if (time >= 2200)

adjustment = 2;

// Weekend Discount (-10%)

else if (weekend)

adjustment = 90;

// Weekday Rush Hour Sur-charge (+20%)

else

if ((time >= 630 && time <= 930) || (time >= 1700 && time <= 2000))

adjustment = 120;

// Return Discounted Rate

return adjustment;

}

// Payment Method

// Validate Payment Method

public String validatePaymentMethod(String paymentMethod) {

// Valid Payment Methods

List<String> validPaymentMethods = new ArrayList<> (Arrays.*asList*("e-wallet", "credit card",

"online banking"));

// Null Payment Method

if (paymentMethod == null) {

throw new IllegalArgumentException("Payment Method Cannot Be Null");

}

// Format Payment Method

paymentMethod = paymentMethod.trim().toLowerCase();

// Invalid Payment Method

if (!validPaymentMethods.contains(paymentMethod))

throw new IllegalArgumentException("Invalid Payment Method");

// Valid Payment Method

else

return paymentMethod;

}

// Fare Adjustment Based on Payment Method

public double paymentMethodAdjustment(String paymentMethod) {

double adjustment = 100;

// Payment Method Discount/Sur-charge

switch (validatePaymentMethod(paymentMethod)) {

case "e-wallet" -> adjustment = 1.0;

case "credit card" -> adjustment = 1.05;

case "online banking" -> adjustment = 0.95;

}

// Final Fare Amount

return adjustment;

}

}

FileFunctionality.java

package utar.edu.my;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.List;

public class FileFunctionality {

// Write Array of String into File

public void writeToFile(String[] inputArray, String fileName) {

// Null File Name

if (fileName == null)

throw new IllegalArgumentException("File Name Cannot Be Null/Empty");

// Null Input Array

else if (inputArray == null)

throw new IllegalArgumentException("Input Array Cannot Be Null");

// Set File Name to File Class

File targetFile = new File("Database\\" + fileName);

// Create Buffered Reader

try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(targetFile))) {

// Write To File Line by Line

for (String inputLine : inputArray) {

fileWriter.write(inputLine);

fileWriter.newLine();

}

}

// Target File Not Found

catch (FileNotFoundException fnf) {

throw new IllegalArgumentException("File \"" + fileName + "\" Does Not Exist In Database");

}

// Error Writing to File

catch (IOException io) {

throw new IllegalArgumentException("Error Writing To :" + fileName);

}

}

// Read Data from Specified File Path and Parse to Array of String

public String[] readFromFile(String fileName) {

List<String> outputList = new ArrayList<>();

//Invalid File Name

if (fileName == null)

throw new IllegalArgumentException("File Name Cannot Be Null/Empty");

// Set File Name to File Class

File targetFile = new File("Database\\" + fileName);

// Create Buffered Reader

try (BufferedReader fileReader = new BufferedReader(new FileReader(targetFile))) {

// Read Data Into String

String fileLine;

while ((fileLine = fileReader.readLine()) != null) {

outputList.add(fileLine);

}

}

// Target File Not Found

catch (FileNotFoundException fnf) {

throw new IllegalArgumentException("File \"" + fileName + "\" Does Not Exist In Database");

}

// Error Reading from File

catch (IOException io) {

throw new IllegalArgumentException("Error Reading From : " + fileName);

}

// Return ArrayList as Array

return outputList.toArray(new String[outputList.size()]);

}

}

Guest.java

package utar.edu.my;

public class Guest implements IUser {

private String name;

private String email;

private String phoneNo;

public Guest(String name, String email, String phoneNo) {

this.name = name;

this.email = email;

this.phoneNo = phoneNo;

}

// Get Methods

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public String getPhoneNo() {

return phoneNo;

}

// Set Methods

public void setName(String name) {

this.name = name;

}

public void setEmail(String email) {

this.email = email;

}

public void setPhoneNo(String phoneNo) {

this.phoneNo = phoneNo;

}

// Base Functions

*@Override*

public String toString() {

return name + ";" + email + ";" + phoneNo;

}

*@Override*

public boolean equals(Object o) {

if (this == o)

return true;

if (o == null || getClass() != o.getClass())

return false;

Guest otherGuest = (Guest) o;

return toString().equals(otherGuest.toString());

}

*@Override*

public int hashCode() {

return name != null ? name.hashCode() : 0;

}

}

GuestFile.java

package utar.edu.my;

import java.util.ArrayList;

import java.util.List;

public class GuestFile {

private FileFunctionality ff;

// Constructors

public GuestFile() {

ff = new FileFunctionality();

}

public GuestFile(FileFunctionality ff) {

this.ff = ff;

}

// Read Guest Data from File

public List<Guest> readGuestFromFile(String fileName) {

List<Guest> guestList = new ArrayList<> ();

// Retrieve Guest Details from File

String[] guestArray = ff.readFromFile(fileName);

// Construct Guest ArrayList

for (String guestLine : guestArray) {

String[] guestDetail = guestLine.split(";");

// Invalid Guest File Format

if (guestDetail.length != 3)

throw new IllegalArgumentException("Invalid Guest Detail Format");

// Convert String to Guest Object

else

guestList.add(new Guest(guestDetail[0], guestDetail[1], guestDetail[2]));

}

return guestList;

}

// Write New Guest to File

public void writeGuestToFile(Guest[] guestArray, String fileName) {

List<String> guestStringList = new ArrayList<>();

// Null Guest Array

if (guestArray == null)

throw new IllegalArgumentException("Guest List Cannot Be Null");

// Convert Guest Class Array to String List

for (Guest guest : guestArray) {

String guestString = guest.toString();

guestStringList.add(guestString);

}

// Convert Array List to Array

String[] guestStringArray = guestStringList.toArray(new String[guestStringList.size()]);

// Write to File

ff.writeToFile(guestStringArray, fileName);

}

}

IUser.java

package utar.edu.my;

public interface IUser {

// Get Functions

public String getName();

public String getEmail();

public String getPhoneNo();

// Set Functions

public void setName(String name);

public void setEmail(String email);

public void setPhoneNo(String phoneNo);

// Base Functions

public String toString();

public boolean equals(Object o);

public int hashCode();

}

Payment.java

package utar.edu.my;

public class Payment {

// Make Payment

public void makePayment(double Payment) {

// Not to be Implemented

}

// Get Payment Status

public String getStatus() {

return "Paid";

}

public void emailReceipt() {

// Not to be Implemented

}

}

RouteInfo.java

package utar.edu.my;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class RouteInfo {

private List<Station> stationList = new ArrayList<>();

// Station Names

private final String[] stationNameList = {"kl sentral", "mid valley", "subang jaya", "shah alam", "bangsar", "kepong sentral", "sentul timur", "titiwangsa",

"ampang park", "klcc", "masjid jamek", "bandaraya", "batu kentonmen", "rawang", "sungai buloh", "serdang", "kajang", "semenyih sentral",

"gombak", "taman melati", "wangsa maju", "setiawangsa"};

// Station Path From

private final String[] station1List = {"kl sentral", "kl sentral", "subang jaya", "bangsar", "kl sentral", "sentul timur", "titiwangsa", "ampang park",

"klcc", "masjid jamek", "bandaraya", "batu kentonmen", "rawang", "sungai buloh", "serdang", "kajang",

"gombak", "taman melati", "wangsa maju", "setiawangsa"};

// Station Path To

private final String[] station2List = {"mid valley", "subang jaya", "shah alam", "kl sentral", "kepong sentral", "titiwangsa", "ampang park", "klcc",

"masjid jamek", "bandaraya", "batu kentonmen", "rawang", "sungai buloh", "kepong sentral", "kajang", "semenyih sentral",

"taman melati", "wangsa maju", "setiawangsa", "kl sentral"};

// Station Path Distance

private final double[] stationDistance = {5.0, 18.0, 7.0, 2.0, 12.5, 3.5, 4.0, 1.0, 3.0, 1.5, 10.0, 26.0, 12.0, 9.5, 10.0, 6.0, 4.0, 2.0, 4.5, 16.0};

// Setup all Stations, Paths & Distance

public RouteInfo() {

for (String stationName : stationNameList) {

stationList.add(new Station(stationName));

}

// Add All Station Paths

for (int i = 0; i < 20; i++) {

Station stationFrom = null;

Station stationTo = null;

for (Station station : stationList) {

// Get Starting Station

if (station.getStationName().equals(station1List[i])) {

stationFrom = station;

}

// Get Ending Station

else if (station.getStationName().equals(station2List[i])) {

stationTo = station;

}

// Set Path for Stations to Run Both Ways

if (stationFrom != null && stationTo != null) {

stationFrom.addStation(stationTo, stationDistance[i]);

stationTo.addStation(stationFrom, stationDistance[i]);

break;

}

}

}

}

// Validate Station Name & Return Station Object

public Station validateStation(String stationName) {

// Null Station Name

if (stationName == null)

throw new IllegalArgumentException("Station Name Cannot Be Null");

// Format Station Name

stationName = stationName.trim().toLowerCase();

// Search For Station

for (Station station : stationList) {

// Found Station

if (station.getStationName().equals(stationName))

return station;

}

// Station Not Found

throw new IllegalArgumentException("No Stations Found");

}

// Find Start & End Stations

// Helper Method for Routing Algorithm

public Double getRouteDistance(String startStation, String endStation) {

double distance;

// Find Stations

Station originStation = validateStation(startStation);

Station destinationStation = validateStation(endStation);

// Same Station

if (originStation.equals(destinationStation))

throw new IllegalArgumentException("Same Start & End Station");

// Start Routing

distance = findRoute(null, originStation, destinationStation);

// No Routes Found

if (distance <= 0)

throw new IllegalArgumentException("No Routes Found");

// Found Valid Route

else

return distance;

}

// Calculate Shortest Distance Between Stations

private Double findRoute(Station previousStation, Station currentStation, Station destination) {

Double currentDistance = Double.***MAX\_VALUE***;

List<Double> distances = new ArrayList<>();

// Check If Arrived at End Station

if (currentStation.equals(destination)) {

return 0.0;

}

// Map Route

else {

// List of Next Stations

HashMap<Station, Double> nextStations = currentStation.getLinkedStations();

// Map Through All Possible Routes

boolean hasRoute = false;

for (Map.Entry<Station, Double> nextStation : nextStations.entrySet()) {

// Ensure Path Cannot Go Backwards

if (!nextStation.getKey().equals(previousStation)) {

// Get Next Distance (-1) if Dead End

Double nextDistance = findRoute(currentStation, nextStation.getKey(), destination);

// Valid Route

if (nextDistance >= 0.0) {

distances.add(nextStation.getValue() + nextDistance);

hasRoute = true;

}

// Invalid Route

else {

distances.add(nextDistance);

}

}

}

// Get Shortest Distance

for (Double distance : distances) {

if (distance > 0 && distance < currentDistance) {

currentDistance = distance;

}

}

// Return Shortest Distance

if (hasRoute) {

return currentDistance;

}

// No Route

else {

return -1.0;

}

}

}

}

Station.java

package utar.edu.my;

import java.util.HashMap;

public class Station {

private final String stationName;

private final HashMap<Station, Double> linkedStations = new HashMap<>();

public Station(String stationName) {

this.stationName = stationName;

}

// Get Functions

public String getStationName() {

return stationName;

}

public HashMap<Station, Double> getLinkedStations() {

return linkedStations;

}

// Add Next Linking Stations to Current Station

public void addStation(Station nextStation, double distance) {

linkedStations.put(nextStation, distance);

}

// Base Functions

*@Override*

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

Station station = (Station) obj;

return stationName.equals(station.getStationName());

}

*@Override*

public int hashCode() {

return stationName != null ? stationName.hashCode() : 0;

}

}

User.java

package utar.edu.my;

public class User implements IUser {

private String ID;

private String name;

private String email;

private String phoneNo;

// Constructor

public User() {

}

public User(String ID, String name, String email, String phoneNo) {

this.ID = ID;

this.name = name;

this.email = email;

this.phoneNo = phoneNo;

}

// Get Functions

public String getID() {

return ID;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public String getPhoneNo() {

return phoneNo;

}

// Set Functions

public void setName(String name) {

this.name = name;

}

public void setEmail(String email) {

this.email = email;

}

public void setPhoneNo(String phoneNo) {

this.phoneNo = phoneNo;

}

// Base Functions

*@Override*

public String toString() {

return ID + ";" + name + ";" + email + ";" + phoneNo;

}

*@Override*

public boolean equals(Object o) {

if (this == o)

return true;

if (o == null || getClass() != o.getClass())

return false;

User otherUser = (User) o;

return toString().equals(otherUser.toString());

}

*@Override*

public int hashCode() {

return ID != null ? ID.hashCode() : 0;

}

}

UserFile.java

package utar.edu.my;

import java.util.ArrayList;

import java.util.List;

public class UserFile {

private FileFunctionality ff;

// Constructors

public UserFile() {

ff = new FileFunctionality();

}

public UserFile(FileFunctionality ff) {

this.ff = ff;

}

// Read User Data from File

public List<User> readUserFromFile(String fileName) {

List<User> userList = new ArrayList<> ();

// Retrieve User Details from File

String[] userArray = ff.readFromFile(fileName);

// Construct User ArrayList

for (String userLine : userArray) {

String[] userDetail = userLine.split(";");

// Invalid User File Format

if (userDetail.length != 4)

throw new IllegalArgumentException("Invalid User Detail Format");

// Convert String to User Object

else

userList.add(new User(userDetail[0], userDetail[1], userDetail[2], userDetail[3]));

}

return userList;

}

// Write New User to File

public void writeUserToFile(User[] userArray, String fileName) {

List<String> userStringList = new ArrayList<>();

// Null User Array

if (userArray == null)

throw new IllegalArgumentException("User List Cannot Be Null");

// Convert User Class Array to String List

for (User user : userArray) {

String userString = user.toString();

userStringList.add(userString);

}

// Convert Array List to Array

String[] userStringArray = userStringList.toArray(new String[userStringList.size()]);

// Write to File

ff.writeToFile(userStringArray, fileName);

}

}

# **Test Code**

BookingIntegrationTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import static org.junit.Assert.*assertEquals*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class BookingIntegrationTest {

// Get Total Fare

// Valid Parameters

private Object getValidGetTotalFareParams() {

return new Object[] {

new Object[] {"kl sentral" , "mid valley" , 2.0},

new Object[] {"kl sentral" , "subang jaya" , 15.0},

new Object[] {"subang jaya" , "shah alam" , 5.0},

new Object[] {"bangsar" , "kl sentral" , 2.0},

new Object[] {"kl sentral" , "kepong sentral" , 10.0}

};

}

*@Test*

*@Parameters* (method = "getValidGetTotalFareParams")

public void testGetTotalFareValid(String startStation, String endStation, double expectedResult) {

// Setup Class

Booking b = new Booking(null, null, null, startStation, endStation, null, null,null);

// Run Method

double result = b.getTotalFare();

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidGetTotalFareParams() {

return new Object[] {

// Start Station

new Object[] {null , "klcc"}, // Null Station

new Object[] {"INVALID" , "klcc"}, // Invalid Station

// End Station

new Object[] {"klcc" , null}, // Null Station

new Object[] {"klcc" , "INVALID"}, // Invalid Station

// Same Station

new Object[] {"klcc" , "klcc"},

// Invalid Route

new Object[] {"kajang" , "taman melati"},

// Exceed Distance Limit

new Object[] {"klcc" , "sungai buloh"}

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidGetTotalFareParams")

public void testGetTotalFareInvalid(String startStation, String endStation) {

// Setup Class

Booking b = new Booking(null, null, null, startStation, endStation, null, null,null);

// Run Method

b.getTotalFare();

}

// Get Discounted Fare + Get Discount Details

// Valid Parameters

private Object getValidGetDiscountedFareParams() {

// Sample Array List

List<String> passengerType1 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "child", "senior citizen"}));

List<String> passengerType2 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "adult", "child"}));

List<String> passengerType3 = new ArrayList<> (Arrays.*asList*(new String[] {"student", "adult", "child"}));

List<String> passengerType4 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "student", "child"}));

List<String> passengerType5 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "senior citizen", "child"}));

List<Integer> passengerQuantity1 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 1, 1}));

List<Integer> passengerQuantity2 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 2, 2}));

List<Integer> passengerQuantity3 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 2, 3}));

List<Integer> passengerQuantity4 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 3, 1}));

List<Integer> passengerQuantity5 = new ArrayList<> (Arrays.*asList*(new Integer[] {3, 1, 2}));

String[] detail1 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n"

};

String[] detail2 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n"

};

String[] detail3 = {

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 3\n"

};

String[] detail4 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 3\n",

"Passenger Adjustment : -100.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 1\n"

};

String[] detail5 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 3\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 2\n"

};

return new Object[] {

new Object[] {"monday" , "0300", "kl sentral" , "mid valley" , passengerType1, passengerQuantity1, 4.00, detail1},

new Object[] {"wednesday" , "0800", "kl sentral" , "subang jaya" , passengerType2, passengerQuantity2, 72.00, detail2},

new Object[] {"friday" , "1300", "subang jaya" , "shah alam" , passengerType3, passengerQuantity3, 21.00, detail3},

new Object[] {"saturday" , "1850", "bangsar" , "kl sentral" , passengerType4, passengerQuantity4, 5.58, detail4},

new Object[] {"sunday" , "2200", "kl sentral" , "kepong sentral" , passengerType5, passengerQuantity5, 57.00, detail5}

};

}

*@Test*

*@Parameters* (method = "getValidGetDiscountedFareParams")

public void testGetDiscountedFareValid(String travelDay, String travelTime, String startStation, String endStation,

List<String> passengerType, List<Integer> passengerQuantity, double expectedFare, String[] expectedDetail) {

// Setup Class

Booking b = new Booking(null, travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity, null);

// Run Method

double resultFare = b.getDiscountedFare();

List<String> adjDetail = b.getDiscountDetails();

String[] resultDetail = adjDetail.toArray(new String[adjDetail.size()]);

// Compare Result

*assertEquals*(resultFare, expectedFare, 0.01);

*assertArrayEquals*(resultDetail, expectedDetail);

}

// Get Discounted Fare

// Invalid Parameters

private Object getInvalidGetDiscountedFareParams() {

// Sample Array

List<String> passengerType = new ArrayList<> (Arrays.*asList*("adult", "child", "senior citizen"));

List<Integer> passengerQuantity = new ArrayList<> (Arrays.*asList*(2, 1, 3));

// Invalid Array

List<Integer> invalidQuantity = new ArrayList<> (Arrays.*asList*(1, 3));

// Empty Array

List<String> emptyType = new ArrayList<> ();

List<Integer> emptyQuantity = new ArrayList<> ();

return new Object[] {

// Travel Day

new Object[] {null , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"INVALID" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

// Travel Time

new Object[] {"monday" , null , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"monday" , "INVALID" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

// Start Station

new Object[] {"monday" , "0200" , null , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"monday" , "0200" , "INVALID" , "batu kentonmen" , passengerType , passengerQuantity },

// End Station

new Object[] {"monday" , "0200" , "titiwangsa" , null , passengerType , passengerQuantity },

new Object[] {"monday" , "0200" , "titiwangsa" , "INVALID" , passengerType , passengerQuantity },

// Passenger Type

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , null , passengerQuantity },

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , emptyType , passengerQuantity },

// Passenger Quantity

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , null },

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , emptyQuantity },

// Incorrect Type & Quantity Length

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , invalidQuantity }

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidGetDiscountedFareParams")

public void testGetDiscountedFareParams(String travelDay, String travelTime, String startStation, String endStation,

List<String> passengerType, List<Integer> passengerQuantity) {

// Setup Class

Booking b = new Booking(null, travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity, null);

// Run Method

b.getDiscountedFare();

}

// Get Discount Details

// Invalid Parameters

*@Test* (expected = IllegalArgumentException.class)

public void testGetDiscountDetailParams() {

// Setup Class

Booking b = new Booking(null, null, null, null, null, null, null, null); // Null Details

// Run Method

b.getDiscountedFare();

}

// Make Payment

// Valid Parameters

private Object getMakePaymentValidParams() {

return new Object[] {

new Object[] {"e-wallet"},

new Object[] {"credit card"},

new Object[] {"online banking"}

};

}

*@Test*

*@Parameters* (method = "getMakePaymentValidParams")

public void testMakePaymentValid(String paymentMethod) {

// Sample Array

List<String> passengerType = new ArrayList<> (Arrays.*asList*("adult", "child", "senior citizen"));

List<Integer> passengerQuantity = new ArrayList<> (Arrays.*asList*(2, 1, 3));

// Setup Class

Booking b = new Booking(null, "monday", "0200", "titiwangsa", "batu kentonmen", passengerType, passengerQuantity, paymentMethod);

// Run Method

b.getDiscountedFare();

b.makePayment();

String result = b.getBookingStatus();

// Compare Result

*assertEquals*(result, "Confirmed Booking");

}

// Invalid Parameters

private Object getInvalidMakePaymentParams() {

return new Object[] {

new Object[] {null}, // Null Method Method

new Object[] {"INVALID"}, // Invalid Payment Method

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidMakePaymentParams")

public void testMakePaymentInvalid1(String paymentMethod) {

// Sample Array

List<String> passengerType = new ArrayList<> (Arrays.*asList*("adult", "child", "senior citizen"));

List<Integer> passengerQuantity = new ArrayList<> (Arrays.*asList*(2, 1, 3));

// Setup Class

Booking b = new Booking(null, "monday", "0200", "titiwangsa", "batu kentonmen", passengerType, passengerQuantity, paymentMethod);

// Run Method

b.makePayment();

}

*@Test* (expected = IllegalArgumentException.class)

public void testMakePaymentInvalid2() {

// Setup Class

Booking b = new Booking(null, null, null, null, null, null, null, "e-wallet");

// Run Method

b.makePayment(); // Null Fare

}

}

BookingUnitTest.java

package utar.edu.my;

import static org.mockito.ArgumentMatchers.*anyDouble*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*times*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.Parameters;

import junitparams.JUnitParamsRunner;

*@RunWith*(JUnitParamsRunner.class)

public class BookingUnitTest {

// Target Class

private CalculateFare cfMock;

private Payment pMock;

private Booking b;

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Class

cfMock= *mock*(CalculateFare.class);

pMock = *mock*(Payment.class);

b = new Booking(cfMock, pMock);

}

// Get Total Fare

public Object getTotalFareValidParams() {

return new Object[] {

new Object[] {"kl sentral" , "mid valley"},

new Object[] {"kl sentral" , "subang jaya"},

new Object[] {"subang jaya" , "shah alam"},

new Object[] {"bangsar" , "kl sentral"},

new Object[] {"kl sentral" , "kepong sentral"}

};

}

*@Test*

*@Parameters* (method = "getTotalFareValidParams")

public void testGetTotalFare(String startStation, String endStation) {

// Set Parameters

b.setStartStation(startStation);

b.setEndStation(endStation);

// Run Method

b.getTotalFare();

// Verify Method Execution

*verify*(cfMock, *times*(1)).calculateTotalFare(startStation, endStation);

*verify*(cfMock, *times*(1)).getTotalFare();

}

// Get Discounted Fare

private Object getDiscountedFareValidParams() {

// Sample Array List

List<String> passengerType1 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "child", "senior citizen"}));

List<String> passengerType2 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "adult", "child"}));

List<String> passengerType3 = new ArrayList<> (Arrays.*asList*(new String[] {"student", "adult", "child"}));

List<String> passengerType4 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "student", "child"}));

List<String> passengerType5 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "senior citizen", "child"}));

List<Integer> passengerQuantity1 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 1, 1}));

List<Integer> passengerQuantity2 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 2, 2}));

List<Integer> passengerQuantity3 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 2, 3}));

List<Integer> passengerQuantity4 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 3, 1}));

List<Integer> passengerQuantity5 = new ArrayList<> (Arrays.*asList*(new Integer[] {3, 1, 2}));

return new Object[] {

new Object[] {"monday" , "0300", "kl sentral" , "mid valley" , passengerType1, passengerQuantity1},

new Object[] {"wednesday" , "0800", "kl sentral" , "subang jaya" , passengerType2, passengerQuantity2},

new Object[] {"friday" , "1300", "subang jaya" , "shah alam" , passengerType3, passengerQuantity3},

new Object[] {"saturday" , "1850", "bangsar" , "kl sentral" , passengerType4, passengerQuantity4},

new Object[] {"sunday" , "2200", "kl sentral" , "kepong sentral" , passengerType5, passengerQuantity5}

};

}

*@Test*

*@Parameters* (method = "getDiscountedFareValidParams")

public void testGetDiscountedFare(String travelDay, String travelTime, String startStation, String endStation, List<String> passengerType, List<Integer> passengerQuantity) {

// Set Parameters

b.setTravelDay(travelDay);

b.setTravelTime(travelTime);

b.setStartStation(startStation);

b.setEndStation(endStation);

b.setPassengerType(passengerType);

b.setPassengerQuantity(passengerQuantity);

// Run Method

b.getDiscountedFare();

// Verify Method Execution

*verify*(cfMock, *times*(1)).calculateDiscountedFare(travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity);

*verify*(cfMock, *times*(1)).getDiscountedFare();

}

// Get Discounted Detail

// Valid Parameters

*@Test*

public void testGetDiscountDetailValid() {

// Mock Output

*when*(cfMock.getAdjustmentDetails()).thenReturn(new ArrayList<String> (Arrays.*asList*(new String[]{""})));

// Run Method

b.getDiscountDetails();

// Verify Method Execution

*verify*(cfMock, *times*(1)).getAdjustmentDetails();

}

// Invalid Parameters

*@Test* (expected = IllegalArgumentException.class)

public void testGetDiscountDetailInvalid() {

// Run Method

b.getDiscountDetails(); // Null Details

}

// Make Payment

private Object getMakePaymentValidParams() {

return new Object[] {

new Object[] {"e-wallet"},

new Object[] {"credit card"},

new Object[] {"online banking"}

};

}

*@Test*

*@Parameters* (method = "getMakePaymentValidParams")

public void testMakePaymentValid(String paymentMethod) {

// Set Parameters

b.setPaymentMethod(paymentMethod);

// Mock Status Output

*when*(pMock.getStatus()).thenReturn("Paid");

// Run Method

b.makePayment();

// Verify Method Execution

*verify*(cfMock, *times*(1)).calculatePayment(paymentMethod);

*verify*(cfMock, *times*(1)).getPaymentAmount();

*verify*(pMock, *times*(1)).makePayment(*anyDouble*());

*verify*(pMock, *times*(1)).getStatus();

*verify*(pMock, *times*(1)).emailReceipt();

}

}

CalculateFareIntegrationTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import static org.junit.Assert.*assertEquals*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class CalculateFareIntegrationTest {

// Target Class

private CalculateFare cf;

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Class

cf = new CalculateFare();

}

// Calculate Total Fare

// Valid Parameters

private Object getValidCalculateTotalFareParams() {

return new Object[] {

new Object[] {"kl sentral" , "mid valley" , 2.0},

new Object[] {"kl sentral" , "subang jaya" , 15.0},

new Object[] {"subang jaya" , "shah alam" , 5.0},

new Object[] {"bangsar" , "kl sentral" , 2.0},

new Object[] {"kl sentral" , "kepong sentral" , 10.0}

};

}

*@Test*

*@Parameters* (method = "getValidCalculateTotalFareParams")

public void testCalculateTotalFareValid(String startStation, String endStation, double expectedResult) {

// Run Method

cf.calculateTotalFare(startStation, endStation);

double result = cf.getTotalFare();

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidCalculateTotalFareParams() {

return new Object[] {

// Start Station

new Object[] {null , "klcc"}, // Null Station

new Object[] {"INVALID" , "klcc"}, // Invalid Station

// End Station

new Object[] {"klcc" , null}, // Null Station

new Object[] {"klcc" , "INVALID"}, // Invalid Station

// Same Station

new Object[] {"klcc" , "klcc"},

// Invalid Route

new Object[] {"kajang" , "taman melati"},

// Exceed Distance Limit

new Object[] {"klcc" , "sungai buloh"}

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculateTotalFareParams")

public void testCalculateTotalFareInvalid(String startStation, String endStation) {

// Run Method

cf.calculateTotalFare(startStation, endStation);

}

// Calculate Discounted Fare

// Valid Parameters

private Object getValidCalculateDiscountedFareParams() {

// Sample Array List

List<String> passengerType1 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "child", "senior citizen"}));

List<String> passengerType2 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "adult", "child"}));

List<String> passengerType3 = new ArrayList<> (Arrays.*asList*(new String[] {"student", "adult", "child"}));

List<String> passengerType4 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "student", "child"}));

List<String> passengerType5 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "senior citizen", "child"}));

List<Integer> passengerQuantity1 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 1, 1}));

List<Integer> passengerQuantity2 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 2, 2}));

List<Integer> passengerQuantity3 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 2, 3}));

List<Integer> passengerQuantity4 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 3, 1}));

List<Integer> passengerQuantity5 = new ArrayList<> (Arrays.*asList*(new Integer[] {3, 1, 2}));

String[] detail1 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n"

};

String[] detail2 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n"

};

String[] detail3 = {

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 3\n"

};

String[] detail4 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 3\n",

"Passenger Adjustment : -100.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 1\n"

};

String[] detail5 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 3\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 2\n"

};

return new Object[] {

new Object[] {"monday" , "0300", "kl sentral" , "mid valley" , passengerType1, passengerQuantity1, 4.00, detail1},

new Object[] {"wednesday" , "0800", "kl sentral" , "subang jaya" , passengerType2, passengerQuantity2, 72.00, detail2},

new Object[] {"friday" , "1300", "subang jaya" , "shah alam" , passengerType3, passengerQuantity3, 21.00, detail3},

new Object[] {"saturday" , "1850", "bangsar" , "kl sentral" , passengerType4, passengerQuantity4, 5.58, detail4},

new Object[] {"sunday" , "2200", "kl sentral" , "kepong sentral" , passengerType5, passengerQuantity5, 57.00, detail5}

};

}

*@Test*

*@Parameters* (method = "getValidCalculateDiscountedFareParams")

public void testCalculateDiscountedFareValid(String travelDay, String travelTime, String startStation, String endStation,

List<String> passengerType, List<Integer> passengerQuantity, double expectedFare, String[] expectedDetail) {

// Run Method

cf.calculateDiscountedFare(travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity);

double resultFare = cf.getDiscountedFare();

List<String> adjDetail = cf.getAdjustmentDetails();

String[] resultDetail = adjDetail.toArray(new String[adjDetail.size()]);

// Compare Result

*assertEquals*(resultFare, expectedFare, 0.01);

*assertArrayEquals*(resultDetail, expectedDetail);

}

// Invalid Parameters

private Object getInvalidCalculateDiscountedFareParams() {

// Sample Array

List<String> passengerType = new ArrayList<> (Arrays.*asList*("adult", "child", "senior citizen"));

List<Integer> passengerQuantity = new ArrayList<> (Arrays.*asList*(2, 1, 3));

// Invalid Array

List<Integer> invalidQuantity = new ArrayList<> (Arrays.*asList*(1, 3));

// Empty Array

List<String> emptyType = new ArrayList<> ();

List<Integer> emptyQuantity = new ArrayList<> ();

return new Object[] {

// Travel Day

new Object[] {null , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"INVALID" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

// Travel Time

new Object[] {"monday" , null , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"monday" , "INVALID" , "titiwangsa" , "batu kentonmen" , passengerType , passengerQuantity },

// Start Station

new Object[] {"monday" , "0200" , null , "batu kentonmen" , passengerType , passengerQuantity },

new Object[] {"monday" , "0200" , "INVALID" , "batu kentonmen" , passengerType , passengerQuantity },

// End Station

new Object[] {"monday" , "0200" , "titiwangsa" , null , passengerType , passengerQuantity },

new Object[] {"monday" , "0200" , "titiwangsa" , "INVALID" , passengerType , passengerQuantity },

// Passenger Type

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , null , passengerQuantity },

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , emptyType , passengerQuantity },

// Passenger Quantity

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , null },

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , emptyQuantity },

// Incorrect Type & Quantity Length

new Object[] {"monday" , "0200" , "titiwangsa" , "batu kentonmen" , passengerType , invalidQuantity }

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculateDiscountedFareParams")

public void testCalculateDiscountedFareParams(String travelDay, String travelTime, String startStation, String endStation,

List<String> passengerType, List<Integer> passengerQuantity) {

// Run Method

cf.calculateDiscountedFare(travelDay, travelTime, startStation, endStation, passengerType, passengerQuantity);

}

// Calculate Payment

// Valid Parameters

private Object getValidCalculatePaymentParams() {

return new Object[] {

// Mock E-Wallet

new Object[] {10.0, "E-WALLET", 10.0},

new Object[] {20.0, "e-wallet", 20.0},

// Mock Credit Card

new Object[] {10.0, "CREDIT CARD", 10.5},

new Object[] {20.0, "credit card", 21.0},

// Mock Online Banking

new Object[] {10.0, "ONLINE BANKING", 9.5},

new Object[] {20.0, "online banking", 19.0}

};

}

*@Test*

*@Parameters* (method = "getValidCalculatePaymentParams")

public void testCalculatePaymentValid(double totalFare, String paymentMethod, double expectedResult) {

// Run Method

cf.setDiscountedFare(totalFare);

cf.calculatePayment(paymentMethod);

double result = cf.getPaymentAmount();

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidCalculatePaymentParams() {

return new Object[] {

// Total Fare

new Object[] { 0.0, "e-wallet"},

// Payment Method

new Object[] {10.0, null},

new Object[] {10.0, "INVALID"}

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculatePaymentParams")

public void testCalculatePaymentParams(double totalFare, String paymentMethod) {

// Run Method

cf.setDiscountedFare(totalFare);

cf.calculatePayment(paymentMethod);

}

}

CalculateFareUnitTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import static org.junit.Assert.*assertEquals*;

import static org.mockito.ArgumentMatchers.*anyString*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*when*;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class CalculateFareUnitTest {

// Target Class

private RouteInfo riMock;

private FareAdjustment faMock;

private CalculateFare cf;

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Class

riMock = *mock*(RouteInfo.class);

faMock = *mock*(FareAdjustment.class);

cf = new CalculateFare(riMock, faMock);

}

// Calculate Total Fare

// Valid Parameters - Equivalent Partitioning (EP)

private Object getValidCalculateTotalFareParamsEP() {

return new Object[] {

new Object[] { 3.0, 2.0}, // 0 < x <= 5

new Object[] { 8.0, 5.0}, // 5 < x <= 10

new Object[] {13.0, 10.0}, // 10 < x <= 15

new Object[] {18.0, 15.0}, // 15 < x <= 20

new Object[] {25.0, 20.0}, // 20 < x <= 30

};

}

*@Test*

*@Parameters* (method = "getValidCalculateTotalFareParamsEP")

public void testCalculateTotalFareValidEP(double distance, double expectedResult) {

// Mock Route Info Output

*when*(riMock.getRouteDistance(*anyString*(), *anyString*())).thenReturn(distance);

// Run Method

cf.calculateTotalFare("VALID", "VALID");

double result = cf.getTotalFare();

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Values - Equivalent Partitioning (EP)

private Object getInvalidCalculateTotalFareParamsEP() {

return new Object[] {

new Object[] {-5.0}, // Invalid Distance Range

new Object[] {35.0},

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculateTotalFareParamsEP")

public void testCalculateTotalFareInvalidEP(double distance) {

// Mock Route Info Output

*when*(riMock.getRouteDistance(*anyString*(), *anyString*())).thenReturn(distance);

// Run Method

cf.calculateTotalFare("INVALID", "INVALID");

}

// Valid Parameters - Boundary Value Analysis (BVA)

private Object getValidCalculateTotalFareParamsBVA() {

return new Object[] {

new Object[] { 1.0, 2.0}, // 0 < x <= 5

new Object[] { 5.0, 2.0},

new Object[] { 6.0, 5.0}, // 5 < x <= 10

new Object[] {10.0, 5.0},

new Object[] {11.0, 10.0}, // 10 < x <= 15

new Object[] {15.0, 10.0},

new Object[] {16.0, 15.0}, // 15 < x <= 20

new Object[] {20.0, 15.0},

new Object[] {21.0, 20.0}, // 20 < x <= 30

new Object[] {30.0, 20.0},

};

}

*@Test*

*@Parameters* (method = "getValidCalculateTotalFareParamsBVA")

public void testCalculateTotalFareValidBVA(double distance, double expectedResult) {

// Mock Route Info Output

*when*(riMock.getRouteDistance(*anyString*(), *anyString*())).thenReturn(distance);

// Run Method

cf.calculateTotalFare("VALID", "VALID");

double result = cf.getTotalFare();

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters - Boundary Value Analysis (BVA)

private Object getInvalidCalculateTotalFareParamsBVA() {

return new Object[] {

new Object[] {0.5}, // 0 < x <= 30 (Valid)

new Object[] {30.5}

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculateTotalFareParamsBVA")

public void testCalculateTotalFareInvalidBVA(double distance) {

// Mock Route Info Output

*when*(riMock.getRouteDistance(*anyString*(), *anyString*())).thenReturn(distance);

// Run Method

cf.calculateTotalFare("INVALID", "INVALID");

}

// Calculate Discounted Fare

// Valid Parameters

private Object getValidCalculateDiscountedFareParams() {

// Sample Array List

List<String> passengerType1 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "child", "senior citizen"}));

List<String> passengerType2 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "adult", "child"}));

List<String> passengerType3 = new ArrayList<> (Arrays.*asList*(new String[] {"student", "adult", "child"}));

List<String> passengerType4 = new ArrayList<> (Arrays.*asList*(new String[] {"senior citizen", "student", "child"}));

List<String> passengerType5 = new ArrayList<> (Arrays.*asList*(new String[] {"adult", "senior citizen", "child"}));

List<Integer> passengerQuantity1 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 1, 1}));

List<Integer> passengerQuantity2 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 2, 2}));

List<Integer> passengerQuantity3 = new ArrayList<> (Arrays.*asList*(new Integer[] {1, 2, 3}));

List<Integer> passengerQuantity4 = new ArrayList<> (Arrays.*asList*(new Integer[] {2, 3, 1}));

List<Integer> passengerQuantity5 = new ArrayList<> (Arrays.*asList*(new Integer[] {3, 1, 2}));

String[] detail1 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n"

};

String[] detail2 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 20.0 %\nPassenger Amount : 2\n"

};

String[] detail3 = {

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 1\n",

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : 0.0 %\nPassenger Amount : 3\n"

};

String[] detail4 = {

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 2\n",

"Passenger Adjustment : -30.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 3\n",

"Passenger Adjustment : -100.0 %\nDay Time Adjustment : -10.0 %\nPassenger Amount : 1\n"

};

String[] detail5 = {

"Passenger Adjustment : 0.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 3\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 1\n",

"Passenger Adjustment : -50.0 %\nDay Time Adjustment : + RM 2.00\nPassenger Amount : 2\n"

};

return new Object[] {

new Object[] {passengerType1, passengerQuantity1, 5.0, 1.00, 0.50, 0.50, 100, 4.00, detail1},

new Object[] {passengerType2, passengerQuantity2, 10.0, 0.50, 1.00, 0.50, 120, 24.00, detail2},

new Object[] {passengerType3, passengerQuantity3, 15.0, 0.70, 1.00, 0.50, 100, 42.00, detail3},

new Object[] {passengerType4, passengerQuantity4, 20.0, 0.50, 0.70, 0.00, 90, 41.85, detail4},

new Object[] {passengerType5, passengerQuantity5, 25.0, 1.00, 0.50, 0.50, 2, 102.00, detail5},

};

}

*@Test*

*@Parameters* (method = "getValidCalculateDiscountedFareParams")

public void testCalculateDiscountedFareValidFare(List<String> passengerType, List<Integer> passengerQuantity, double distance,

double pDiscount1, double pDiscount2, double pDiscount3, int dayTimeDiscount, double expectedFare, String[] expectedDetail) {

// Mock Route Distance Output

*when*(riMock.getRouteDistance(*anyString*(), *anyString*())).thenReturn(distance);

*when*(faMock.passengerAdjustment(*anyString*())).thenReturn(pDiscount1).thenReturn(pDiscount2).thenReturn(pDiscount3);

*when*(faMock.dayTimeAdjustment(*anyString*(), *anyString*())).thenReturn(dayTimeDiscount);

// Run Method

cf.calculateDiscountedFare("day", "time", "startStation", "endStation", passengerType, passengerQuantity);

double resultFare = cf.getDiscountedFare();

List<String> adjDetail = cf.getAdjustmentDetails();

String[] resultDetail = adjDetail.toArray(new String[adjDetail.size()]);

// Compare Result

*assertEquals*(resultFare, expectedFare, 0.01);

*assertArrayEquals*(resultDetail, expectedDetail);

}

// Invalid Parameters

private Object getInvalidCalculateDiscountedFareParams() {

// Sample Array List

List<String> passengerType = new ArrayList<> (Arrays.*asList*(new String[] {"", "", ""}));

List<Integer> passengerQuantity = new ArrayList<> (Arrays.*asList*(new Integer[] {0, 0, 0, 0}));

// Empty Array List

List<String> emptyType = new ArrayList<> ();

List<Integer> emptyQuantity = new ArrayList<> ();

return new Object[] {

new Object[] {null , passengerQuantity}, // Null Passenger Type List

new Object[] {emptyType , passengerQuantity}, // Empty Passenger Type List

new Object[] {passengerType , null}, // Null Passenger Quantity

new Object[] {passengerType , emptyQuantity}, // Empty Passenger Quantity

new Object[] {passengerType , passengerQuantity} // Different Passenger Type & Quantity Length

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculateDiscountedFareParams")

public void testCalculateDiscountedFareInvalid(List<String> passengerType, List<Integer> passengerQuantity) {

// Run Method

cf.calculateDiscountedFare("INVALID", "INVALID", "INVALID", "INVALID", passengerType, passengerQuantity);

}

// Calculate Payment

// Valid Parameters

private Object getValidCalculatePaymentParams() {

return new Object[] {

new Object[] {10.0, "e-wallet", 1.00, 10.0},

new Object[] {10.0, "credit card", 1.05, 10.5},

new Object[] {10.0, "online banking", 0.95, 9.5},

};

}

*@Test*

*@Parameters* (method = "getValidCalculatePaymentParams")

public void testCalculatePaymentValid(double totalFare, String paymentMethod, double paymentAdjustment, double expectedResult) {

// Mock Payment Adjustment Output

*when*(faMock.paymentMethodAdjustment(*anyString*())).thenReturn(paymentAdjustment);

// Run Method

cf.setDiscountedFare(totalFare);

cf.calculatePayment(paymentMethod);

double result = cf.getPaymentAmount();

// Compare Results

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidCalculatePaymentParams() {

return new Object[] {

new Object[] {0, "E-Wallet"} // Invalid Total Fare

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidCalculatePaymentParams")

public void testCalculatePaymentInvalid(double totalFare, String paymentMethod) {

// Run Method

cf.setDiscountedFare(totalFare);

cf.calculatePayment(paymentMethod);

}

}

FareAdjustmentUnitTest.java

package utar.edu.my;

import static org.junit.Assert.*assertEquals*;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class FareAdjustmentUnitTest {

// Target Class

FareAdjustment fa;

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Class

fa = new FareAdjustment();

}

// Validate Passenger Type

// Valid Parameters

private Object getValidPassengerTypeParams() {

return new Object[] {

// Adult

new Object[] {"ADULT", "adult"},

new Object[] {"adult", "adult"},

// Senior Citizen

new Object[] {"SENIOR CITIZEN", "senior citizen"},

new Object[] {"senior citizen", "senior citizen"},

// Student

new Object[] {"STUDENT", "student"},

new Object[] {"student", "student"},

// Child

new Object[] {"CHILD", "child"},

new Object[] {"child", "child"}

};

}

*@Test*

*@Parameters* (method = "getValidPassengerTypeParams")

public void testValidatePassengerTypeValid(String passengerType, String expectedResult) {

// Run Method

String result = fa.validatePassengerType(passengerType);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters

private Object getInvalidPassengerTypeParams() {

return new Object[] {

new Object[] {null}, // Null Passenger Type

new Object[] {"INVALID"} // Invalid Passenger Type

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidPassengerTypeParams")

public void testValidatePassengerTypeInvalid(String passengerType) {

// Run Method

fa.validatePassengerType(passengerType);

}

// Passenger Adjustment

// Valid Parameters - Equivalent Partitioning (EP)

private Object getValidPassengerAdjustmentParamsEP() {

return new Object[] {

// ADULT

new Object[] {3.0, "ADULT", 1.0},

new Object[] {8.0, "adult", 1.0},

// SENIOR CITIZEN

new Object[] {3.0, "SENIOR CITIZEN", 0.5},

new Object[] {8.0, "senior citizen", 0.5},

// Student

new Object[] {3.0, "STUDENT", 0.7},

new Object[] {8.0, "student", 0.7},

// Child

new Object[] {3.0, "CHILD", 0.0},

new Object[] {8.0, "child", 0.5}

};

}

*@Test*

*@Parameters* (method = "getValidPassengerAdjustmentParamsEP")

public void testPassengerAdjustmentValidEP(double travelDistance, String passengerType, double expectedResult) {

// Run Method

fa.setTravelDistance(travelDistance);

double result = fa.passengerAdjustment(passengerType);

// Compare Results

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters - Equivalent Partitioning (EP)

private Object getInvalidPassengerAdjustmentParamsEP() {

return new Object[] {

// Travel Distance

new Object[] {-5.0, "adult"}, // Invalid Distance Range

new Object[] {35.0, "adult"},

// Passenger Type

new Object[] {3.0, null}, // Null Passenger Type

new Object[] {3.0, "INVALID"} // Invalid Passenger Type

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters*(method = "getInvalidPassengerAdjustmentParamsEP")

public void testPassengerAdjustmentInvalidEP(double travelDistance, String passengerType) {

// Run Method

fa.setTravelDistance(travelDistance);

fa.passengerAdjustment(passengerType);

}

// Valid Parameters - Boundary Value Analysis (BVA)

private Object getValidPassengerAdjustmentParamsBVA() {

return new Object[] {

// ADULT

new Object[] { 1.0, "adult", 1.0},

new Object[] { 4.0, "adult", 1.0},

new Object[] { 5.0, "adult", 1.0},

new Object[] {30.0, "adult", 1.0},

// SENIOR CITIZEN

new Object[] { 1.0, "senior citizen", 0.5},

new Object[] { 4.0, "senior citizen", 0.5},

new Object[] { 5.0, "senior citizen", 0.5},

new Object[] {30.0, "senior citizen", 0.5},

// Student

new Object[] { 1.0, "student", 0.7},

new Object[] { 4.0, "student", 0.7},

new Object[] { 5.0, "student", 0.7},

new Object[] {30.0, "student", 0.7},

// Child

new Object[] { 1.0, "child", 0.0},

new Object[] { 4.0, "child", 0.0},

new Object[] { 5.0, "child", 0.5},

new Object[] {30.0, "child", 0.5}

};

}

*@Test*

*@Parameters* (method = "getValidPassengerAdjustmentParamsBVA")

public void testPassengerAdjustmentValidBVA(double travelDistance, String passengerType, double expectedResult) {

// Run Method

fa.setTravelDistance(travelDistance);

double result = fa.passengerAdjustment(passengerType);

// Compare Results

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters - Boundary Value Analysis (BVA)

private Object getInvalidPassengerAdjustmentParamsBVA() {

return new Object[] {

// Travel Distance

new Object[] { 0.0 , "adult"}, // Invalid Distance Range

new Object[] {31.0 , "adult"},

// Passenger Type

new Object[] { 1.0 , null}, // Null Passenger Type

new Object[] { 1.0 , "INVALID"}, // Invalid Passenger Type

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters*(method = "getInvalidPassengerAdjustmentParamsBVA")

public void testPassengerAdjustmentInvalidBVA(double travelDistance, String passengerType) {

// Run Method

fa.setTravelDistance(travelDistance);

fa.passengerAdjustment(passengerType);

}

// Is Weekend

// Valid Parameters

private Object getValidIsWeekendParams() {

return new Object[] {

// Monday

new Object[] {"MONDAY", false},

new Object[] {"monday", false},

// Tuesday

new Object[] {"TUESDAY", false},

new Object[] {"tuesday", false},

// Wednesday

new Object[] {"WEDNESDAY", false},

new Object[] {"wednesday", false},

// Thursday

new Object[] {"THURSDAY", false},

new Object[] {"thursday", false},

// Friday

new Object[] {"FRIDAY", false},

new Object[] {"friday", false},

// Saturday

new Object[] {"SATURDAY", true},

new Object[] {"saturday", true},

// Sunday

new Object[] {"SUNDAY", true},

new Object[] {"sunday", true}

};

}

*@Test*

*@Parameters* (method = "getValidIsWeekendParams")

public void testIsWeekendValid(String travelDay, boolean expectedResult) {

// Run Method

boolean result = fa.isWeekend(travelDay);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters

private Object getInvalidIsWeekendParams() {

return new Object[] {

new Object[] {null}, // Null Day

new Object[] {"INVALID"} // Invalid Day

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters*(method = "getInvalidIsWeekendParams")

public void testIsWeekendInvalid(String travelDay) {

// Run Method

fa.isWeekend(travelDay);

}

// Validate Travel Time

// Valid Parameters

private Object getValidTravelTimeParams() {

return new Object[] {

new Object[] {"0000", 0},

new Object[] {"0600", 600},

new Object[] {"1200", 1200},

new Object[] {"1800", 1800},

new Object[] {"2359", 2359}

};

}

*@Test*

*@Parameters* (method = "getValidTravelTimeParams")

public void testValidateTravelTimeValid(String travelTime, int expectedResult) {

// Run Method

int result = fa.validateTravelTime(travelTime);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters

private Object getInvalidTravelTimeParams() {

return new Object[] {

new Object[] {null}, // Null Travel Time

new Object[] {"INVALID"}, // Invalid Travel Time

new Object[] {"-0001",}, // Invalid Time Range

new Object[] {"2400"},

new Object[] {"0060"}, // Invalid Time Format

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidTravelTimeParams")

public void testValidateTravelTimeInvalid(String travelTime) {

// Run Method

fa.validatePassengerType(travelTime);

}

// Day Time Adjustment

// Valid Parameters - Equivalent Partitioning (EP)

private Object getValidDayTimeAdjustmentParamsEP() {

return new Object[] {

// Weekday - Monday

new Object[] {"monday", "0300", 100},

new Object[] {"monday", "0800", 120},

new Object[] {"monday", "1300", 100},

new Object[] {"MONDAY", "1850", 120},

new Object[] {"MONDAY", "2100", 100},

new Object[] {"MONDAY", "2300", 2},

// Weekend - Saturday

new Object[] {"saturday", "0300", 90},

new Object[] {"saturday", "0800", 90},

new Object[] {"saturday", "1300", 90},

new Object[] {"SATURDAY", "1850", 90},

new Object[] {"SATURDAY", "2100", 90},

new Object[] {"SATURDAY", "2300", 2}

};

}

*@Test*

*@Parameters* (method = "getValidDayTimeAdjustmentParamsEP")

public void testDayTimeAdjustmentValidEP(String travelDay, String travelTime, int expectedResult) {

// Run Method

int result = fa.dayTimeAdjustment(travelDay, travelTime);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters - Equivalent Partitioning (EP)

private Object getInvalidDayTimeAdjustmentParamsEP() {

return new Object[] {

new Object[] {null , "0000"}, // Null Travel Day

new Object[] {"INVALID" , "0000"}, // Invalid Travel Day

new Object[] {"monday" , null}, // Null Travel Time

new Object[] {"monday" , "INVALID"}, // Invalid Travel Time

new Object[] {"monday" , "-0500"}, // Invalid Time Range

new Object[] {"monday" , "2500"},

new Object[] {"monday" , "0070"} // Invalid Time Format

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidDayTimeAdjustmentParamsEP")

public void testDayTimeAdjustmentInvalidEP(String travelDay, String travelTime) {

// Run Method

fa.dayTimeAdjustment(travelDay, travelTime);

}

// Day Time Adjustment

// Valid Parameters - Boundary Value Analysis (BVA)

private Object getValidDayTimeAdjustmentParamsBVA() {

return new Object[] {

// Weekday - Monday

new Object[] {"monday", "0000", 100},

new Object[] {"monday", "0629", 100},

new Object[] {"monday", "0630", 120},

new Object[] {"monday", "0930", 120},

new Object[] {"monday", "0931", 100},

new Object[] {"monday", "1659", 100},

new Object[] {"MONDAY", "1700", 120},

new Object[] {"MONDAY", "2000", 120},

new Object[] {"MONDAY", "2001", 100},

new Object[] {"MONDAY", "2159", 100},

new Object[] {"MONDAY", "2200", 2},

new Object[] {"MONDAY", "2359", 2},

// Weekend - Saturday

new Object[] {"saturday", "0000", 90},

new Object[] {"saturday", "0629", 90},

new Object[] {"saturday", "0630", 90},

new Object[] {"saturday", "0930", 90},

new Object[] {"saturday", "0931", 90},

new Object[] {"saturday", "1659", 90},

new Object[] {"SATURDAY", "1700", 90},

new Object[] {"SATURDAY", "2000", 90},

new Object[] {"SATURDAY", "2001", 90},

new Object[] {"SATURDAY", "2159", 90},

new Object[] {"SATURDAY", "2200", 2},

new Object[] {"SATURDAY", "2359", 2},

};

}

*@Test*

*@Parameters* (method = "getValidDayTimeAdjustmentParamsBVA")

public void testDayTimeAdjustmentValidBVA(String travelDay, String travelTime, int expectedResult) {

// Run Method

int result = fa.dayTimeAdjustment(travelDay, travelTime);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters - Boundary Value Analysis (BVA)

private Object getInvalidDayTimeAdjustmentParamsBVA() {

return new Object[] {

new Object[] {null , "0000"}, // Null Travel Day

new Object[] {"INVALID" , "0000"}, // Invalid Travel Day

new Object[] {"monday" , null}, // Null Travel Time

new Object[] {"monday" , "INVALID"}, // Invalid Travel Time

new Object[] {"monday" , "-0001"}, // Invalid Time Range

new Object[] {"monday" , "2400"},

new Object[] {"monday" , "0060"}, // Invalid Time Format

new Object[] {"monday" , "0099"}

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidDayTimeAdjustmentParamsBVA")

public void testDayTimeAdjustmentInvalidBVA(String travelDay, String travelTime) {

// Run Method

fa.dayTimeAdjustment(travelDay, travelTime);

}

// Validate Payment Method

// Valid Parameters

private Object getValidPaymentMethodParams() {

return new Object[] {

// e-Wallet

new Object[] {"E-WALLET", "e-wallet"},

new Object[] {"e-wallet", "e-wallet"},

// Credit Card

new Object[] {"CREDIT CARD", "credit card"},

new Object[] {"credit card", "credit card"},

// Online Banking

new Object[] {"ONLINE BANKING", "online banking"},

new Object[] {"online banking", "online banking"}

};

}

*@Test*

*@Parameters* (method = "getValidPaymentMethodParams")

public void testValidatePaymentMethodValid(String paymentMethod, String expectedResult) {

// Run Method

String result = fa.validatePaymentMethod(paymentMethod);

// Compare Results

*assertEquals*(result, expectedResult);

}

// Invalid Parameters

private Object getInvalidPaymentMethodParams() {

return new Object[] {

new Object[] {null}, // Null Payment Method

new Object[] {"INVALID"} // Invalid Payment Method

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidPaymentMethodParams")

public void testValidatePaymentMethodInvalid(String paymentMethod) {

// Run Method

fa.validatePaymentMethod(paymentMethod);

}

// Payment Method Adjustment

// Valid Parameters

private Object getValidPaymentMethodAdjustmentParams() {

return new Object[] {

new Object[] {"e-wallet", 1.00},

new Object[] {"credit card", 1.05},

new Object[] {"online banking", 0.95},

};

}

*@Test*

*@Parameters* (method = "getValidPaymentMethodAdjustmentParams")

public void testPaymentMethodAdjustment(String paymentMethod, double expectedResult) {

// Run Method

double result = fa.paymentMethodAdjustment(paymentMethod);

// Compare Results

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidPaymentMethodAdjustmentParams() {

return new Object[] {

new Object[] {null}, // Null Payment Method

new Object[] {"INVALID"} // Invalid Payment Method

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidPaymentMethodAdjustmentParams")

public void testPaymentMethodAdjustmentInvalid(String paymentMethod) {

// Run Method

fa.paymentMethodAdjustment(paymentMethod);

}

}

FileFunctionality.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.Parameters;

import junitparams.JUnitParamsRunner;

*@RunWith* (JUnitParamsRunner.class)

public class FileFunctionalityUnitTest {

// Target Class

private FileFunctionality ff;

// Sample File Read/Write

private String[] sampleArray = {"USER001;MIKU;miku@gmail.com;0123456789", "USER002;TETO;teto@gmail.com;0123456789"};

// File Paths

private String validFile = "TestData\\dummyFile1.txt";

// Setup For all Test Classes

*@Before*

public void setupClasses() {

ff = new FileFunctionality();

}

// Test for Both Read and Write Functionality

*@Test*

public void testReadnWriteFunction() {

// Run Read n Write Method

ff.writeToFile(sampleArray, validFile);

String[] resultArray = ff.readFromFile(validFile);

// Assert Results

*assertArrayEquals*(resultArray, sampleArray);

}

// Write To File

// Invalid Parameters

private Object getWriteToFileInvalidParams() {

return new Object[] {

new Object[] {null, validFile}, // Null Input Array

new Object[] {sampleArray, null}, // Null File Name

new Object[] {sampleArray, ""} // Invalid File Path

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getWriteToFileInvalidParams")

public void testWriteToFileInvalid(String[] inputArray, String fileName) {

// Run Method

ff.writeToFile(inputArray, fileName);

}

// Read From File

// Invalid Parameters

private Object getReadFromFileInvalidParams() {

return new Object[] {

new Object[] {null}, // Null File Name

new Object[] {""} // Invalid File Name

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getReadFromFileInvalidParams")

public void testReadFromFileInvalid(String fileName) {

// Run Method

ff.readFromFile(fileName);

}

}

GuestFileIntegrationTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.Parameters;

import junitparams.JUnitParamsRunner;

*@RunWith*(JUnitParamsRunner.class)

public class GuestFileIntegrationTest {

// Mock and Target Class

private GuestFile gf;

// Sample Guest Objects

private Guest validGuest1 = new Guest("MIKU", "miku@gmail.com", "0123456789");

private Guest validGuest2 = new Guest("TETO", "teto@gmail.com", "0123456789");

private Guest validGuest3 = new Guest("GUMI", "gumi@gmail.com" , "0121892398");

private Guest validGuest4 = new Guest("LEN" , "len@gmail.com" , "0129850802");

private Guest validGuest5 = new Guest("RIN" , "rin@gmail.com" , "0123435887");

private Guest invalidGuest = new Guest("MIKU", "miku@gmail.com", "");

// File Paths

private String validFile = "TestData\\guestDummy.txt";

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Mock and Class

gf = new GuestFile();

}

// Read & Write Guest

// Valid Parameters

private Object getReadWriteIntegrationValidParams() {

// Sample Array

Guest[] array1 = {validGuest1, validGuest2};

Guest[] array2 = {validGuest2, validGuest3};

Guest[] array3 = {validGuest3, validGuest4};

Guest[] array4 = {validGuest4, validGuest5};

Guest[] array5 = {validGuest5, validGuest1};

return new Object[] {

new Object[] {array1},

new Object[] {array2},

new Object[] {array3},

new Object[] {array4},

new Object[] {array5}

};

}

*@Test*

*@Parameters* (method = "getReadWriteIntegrationValidParams")

public void testReadWriteIntegrationValid(Guest[] readWriteData) {

// Run Method

gf.writeGuestToFile(readWriteData, validFile);

List<Guest> guestList = gf.readGuestFromFile(validFile);

Guest[] result = guestList.toArray(new Guest[guestList.size()]);

// Compare Results

*assertArrayEquals*(result, readWriteData);

}

// Read Guest From File

// Invalid File Data Format

*@Test* (expected = IllegalArgumentException.class)

public void testReadGuestFromFileInvalid() {

// Sample Data

Guest[] readWriteData = {invalidGuest};

// Run Method

gf.writeGuestToFile(readWriteData, validFile);

gf.readGuestFromFile(validFile);

}

// Invalid Parameters

private Object getInvalidReadGuestFromFileParams() {

return new Object[] {

new Object[] {null}, // Null File Name

new Object[] {""}, // Invalid File Path

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidReadGuestFromFileParams")

public void testReadGuestFromFileInvalid2(String fileName) {

// Run Method

gf.readGuestFromFile(fileName);

}

// Write Guest To File

// Invalid Parameters

private Object getInvalidWriteGuestToFileParams() {

Guest[] sampleArray = {validGuest1, validGuest2};

return new Object[] {

new Object[] {null, validFile}, // Null Guest List

new Object[] {sampleArray, null}, // Null File Name

new Object[] {sampleArray, ""}, // Invalid File Path

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidWriteGuestToFileParams")

public void testWriteUserToFileInvalid(Guest[] guestArray, String fileName) {

// Run Method

gf.writeGuestToFile(guestArray, fileName);

}

}

GuestFileUnitTest.java

package utar.edu.my;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*times*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class GuestFileUnitTest {

// Mock and Target Class

private FileFunctionality ffMock;

private GuestFile gf;

// Sample Guest Objects

private Guest validGuest1 = new Guest("MIKU", "miku@gmail.com", "0123456789");

private Guest validGuest2 = new Guest("TETO", "teto@gmail.com", "0123456789");

private Guest validGuest3 = new Guest("GUMI", "gumi@gmail.com" , "0121892398");

private Guest validGuest4 = new Guest("LEN" , "len@gmail.com" , "0129850802");

private Guest validGuest5 = new Guest("RIN" , "rin@gmail.com" , "0123435887");

// File Paths

private String validFile = "TestData\\guestDummy.txt";

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Mock and Class

ffMock = *mock*(FileFunctionality.class);

gf = new GuestFile(ffMock);

}

// Read Guest From File

// Valid Output

private Object getReadGuestFromFileValidParams() {

// Sample Arrays

String[] result1 = {validGuest1.toString(), validGuest2.toString()};

String[] result2 = {validGuest2.toString(), validGuest3.toString()};

String[] result3 = {validGuest3.toString(), validGuest4.toString()};

String[] result4 = {validGuest4.toString(), validGuest5.toString()};

String[] result5 = {validGuest5.toString(), validGuest1.toString()};

Guest[] array1 = {validGuest1, validGuest2};

Guest[] array2 = {validGuest2, validGuest3};

Guest[] array3 = {validGuest3, validGuest4};

Guest[] array4 = {validGuest4, validGuest5};

Guest[] array5 = {validGuest5, validGuest1};

return new Object[] {

new Object[] {result1, array1},

new Object[] {result2, array2},

new Object[] {result3, array3},

new Object[] {result4, array4},

new Object[] {result5, array5}

};

}

*@Test*

*@Parameters* (method = "getReadGuestFromFileValidParams")

public void testReadGuestFromFileValid(String[] readOutput, Guest[] expectedArray) {

// Setup Mock Return Value

*when*(ffMock.readFromFile(validFile)).thenReturn(readOutput);

// Run Method

List<Guest> guestList = gf.readGuestFromFile(validFile);

Guest[] guestArray = guestList.toArray(new Guest[guestList.size()]);

// Compare Results

*assertArrayEquals*(guestArray, expectedArray);

}

// Invalid Output

private Object[] getInvalidReadGuestFromFileOutput() {

return new Object[] {

// Invalid Guest Detail Format

new Object[] {"MIKU;miku@gmail.com"}, // Less Data

new Object[] {"MIKU;miku@gmail.com;0123456789;INVALID"} // Additional Data

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidReadGuestFromFileOutput")

public void testReadGuestFromFileInvalid(String fileReadLine) {

// Sample Data

String[] fileReadOutput = {fileReadLine};

// Setup Mock Return Value

*when*(ffMock.readFromFile(validFile)).thenReturn(fileReadOutput);

// Run Method

gf.readGuestFromFile(validFile);

}

// Write Guest To File

// Valid Parameters

private Object getWriteGuestToFileValidParams() {

// Sample Arrays

Guest[] array1 = {validGuest1, validGuest2};

Guest[] array2 = {validGuest2, validGuest3};

Guest[] array3 = {validGuest3, validGuest4};

Guest[] array4 = {validGuest4, validGuest5};

Guest[] array5 = {validGuest5, validGuest1};

String[] result1 = {validGuest1.toString(), validGuest2.toString()};

String[] result2 = {validGuest2.toString(), validGuest3.toString()};

String[] result3 = {validGuest3.toString(), validGuest4.toString()};

String[] result4 = {validGuest4.toString(), validGuest5.toString()};

String[] result5 = {validGuest5.toString(), validGuest1.toString()};

return new Object[] {

new Object[] {array1, result1},

new Object[] {array2, result2},

new Object[] {array3, result3},

new Object[] {array4, result4},

new Object[] {array5, result5}

};

}

*@Test*

*@Parameters* (method = "getWriteGuestToFileValidParams")

public void testWriteGuestToFileValid(Guest[] writeArray, String[] expectedArray) {

// Run Method

gf.writeGuestToFile(writeArray, validFile);

// Verify Method Execution

*verify*(ffMock, *times*(1)).writeToFile(expectedArray, validFile);

}

// Invalid Parameters

*@Test* (expected = IllegalArgumentException.class)

public void testWriteGuestToFileInvalid() {

// Run Method

gf.writeGuestToFile(null, validFile); // Null Guest Array

}

}

RouteInfoUnitTest.java

package utar.edu.my;

import static org.junit.Assert.*assertEquals*;

import org.junit.Before;

import org.junit.runner.RunWith;

import org.junit.Test;

import junitparams.JUnitParamsRunner;

import junitparams.Parameters;

*@RunWith*(JUnitParamsRunner.class)

public class RouteInfoUnitTest {

// Target Class

private RouteInfo ri;

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Class

ri = new RouteInfo();

}

// Validate Station

// Valid Parameters

private Object getValidStationNameParams() {

return new Object[] {

new Object[] {"kl sentral" , "kl sentral" },

new Object[] {"mid valley" , "mid valley" },

new Object[] {"subang jaya" , "subang jaya" },

new Object[] {"shah alam" , "shah alam" },

new Object[] {"bangsar" , "bangsar" },

new Object[] {"kepong sentral" , "kepong sentral" },

new Object[] {"sentul timur" , "sentul timur" },

new Object[] {"titiwangsa" , "titiwangsa" },

new Object[] {"ampang park" , "ampang park" },

new Object[] {"klcc" , "klcc" },

new Object[] {"masjid jamek" , "masjid jamek" },

new Object[] {"bandaraya" , "bandaraya" },

new Object[] {"batu kentonmen" , "batu kentonmen" },

new Object[] {"rawang" , "rawang" },

new Object[] {"sungai buloh" , "sungai buloh" },

new Object[] {"serdang" , "serdang" },

new Object[] {"kajang" , "kajang" },

new Object[] {"semenyih sentral", "semenyih sentral"},

new Object[] {"gombak" , "gombak" },

new Object[] {"taman melati" , "taman melati" },

new Object[] {"wangsa maju" , "wangsa maju" },

new Object[] {"setiawangsa" , "setiawangsa" }

};

}

*@Test*

*@Parameters* (method = "getValidStationNameParams")

public void testValidateStationValid(String stationName, String expectedName) {

// Run Method

Station resultStation = ri.validateStation(stationName);

// Compare Results

*assertEquals*(resultStation.getStationName(), expectedName);

}

// Invalid Parameters

private Object getInvalidStationNameParams() {

return new Object[] {

new Object[] {null}, // Null Station Name

new Object[] {"INVALID"} // Invalid Station Name

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidStationNameParams")

public void testValidateStationInvalid(String stationName) {

// Run Method

ri.validateStation(stationName);

}

// Route Info

// Valid Parameters

private Object getValidGetRouteDistanceParams() {

return new Object[] {

// Random Stations

new Object[] {"kl sentral" , "mid valley" , 5.0},

new Object[] {"kl sentral" , "subang jaya" , 18.0},

new Object[] {"subang jaya" , "shah alam" , 7.0},

new Object[] {"bangsar" , "kl sentral" , 2.0},

new Object[] {"kl sentral" , "kepong sentral" , 12.5},

new Object[] {"sentul timur" , "titiwangsa" , 3.5},

new Object[] {"titiwangsa" , "ampang park" , 4.0},

new Object[] {"ampang park" , "klcc" , 1.0},

new Object[] {"klcc" , "masjid jamek" , 3.0},

new Object[] {"masjid jamek" , "bandaraya" , 1.5},

new Object[] {"bandaraya" , "batu kentonmen" , 10.0},

new Object[] {"batu kentonmen" , "rawang" , 26.0},

new Object[] {"rawang" , "sungai buloh" , 12.0},

new Object[] {"sungai buloh" , "kepong sentral" , 9.5},

new Object[] {"serdang" , "kajang" , 10.0},

new Object[] {"kajang" , "semenyih sentral", 6.0},

new Object[] {"gombak" , "taman melati" , 4.0},

new Object[] {"taman melati" , "wangsa maju" , 2.0},

new Object[] {"wangsa maju" , "setiawangsa" , 4.5},

new Object[] {"setiawangsa" , "kl sentral" , 16.0},

};

}

*@Test*

*@Parameters* (method = "getValidGetRouteDistanceParams")

public void testGetRouteDistanceValid(String startStation, String endStation, double expectedResult) {

// Run Method

double result = ri.getRouteDistance(startStation, endStation);

// Compare Result

*assertEquals*(result, expectedResult, 0.01);

}

// Invalid Parameters

private Object getInvalidGetRouteDistanceParams() {

return new Object[] {

// Origin Station

new Object[] {null , "klcc"}, // Null Station

new Object[] {"INVALID" , "klcc"}, // Invalid Station

// End Station

new Object[] {"klcc" , null}, // Null Station

new Object[] {"klcc" , "INVALID"}, // Invalid Station

// Same Station

new Object[] {"klcc" , "klcc"}, // Same Station

// Invalid Route

new Object[] {"kajang" , "taman melati"} // Invalid Route

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidGetRouteDistanceParams")

public void testGetRouteDistanceInvalid(String startStation, String endStation) {

// Run Method

ri.getRouteDistance(startStation, endStation);

}

}

UserFileIntegrationTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.Parameters;

import junitparams.JUnitParamsRunner;

*@RunWith*(JUnitParamsRunner.class)

public class UserFileIntegrationTest {

// Mock and Target Class

private UserFile uf;

// Sample User Objects

private User validUser1 = new User("USER001", "MIKU", "miku@gmail.com" , "0123456789");

private User validUser2 = new User("USER002", "TETO", "teto@gmail.com" , "0198765432");

private User validUser3 = new User("USER003", "GUMI", "gumi@gmail.com" , "0121892398");

private User validUser4 = new User("USER004", "LEN" , "len@gmail.com" , "0129850802");

private User validUser5 = new User("USER005", "RIN" , "rin@gmail.com" , "0123435887");

private User invalidUser = new User("USER001", "MIKU", "miku@gmail.com", "");

// File Paths

private String validFile = "TestData\\user.txt";

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Mock and Class

uf = new UserFile();

}

// Read & Write User

// Valid Parameters

private Object getReadWriteIntegrationValidParams() {

// Sample Array

User[] array1 = {validUser1, validUser2};

User[] array2 = {validUser2, validUser3};

User[] array3 = {validUser3, validUser4};

User[] array4 = {validUser4, validUser5};

User[] array5 = {validUser5, validUser1};

return new Object[] {

new Object[] {array1},

new Object[] {array2},

new Object[] {array3},

new Object[] {array4},

new Object[] {array5}

};

}

*@Test*

*@Parameters* (method = "getReadWriteIntegrationValidParams")

public void testReadWriteIntegrationValid(User[] readWriteData) {

// Run Method

uf.writeUserToFile(readWriteData, validFile);

List<User> userList = uf.readUserFromFile(validFile);

User[] result = userList.toArray(new User[userList.size()]);

// Compare Results

*assertArrayEquals*(result, readWriteData);

}

// Read User From File

// Invalid File Data Format

*@Test* (expected = IllegalArgumentException.class)

public void testReadUserFromFileInvalid() {

// Sample Data

User[] readWriteData = {invalidUser};

// Run Method

uf.writeUserToFile(readWriteData, validFile);

uf.readUserFromFile(validFile);

}

// Invalid Parameters

private Object getInvalidReadUserFromFileParams() {

return new Object[] {

new Object[] {null}, // Null File Name

new Object[] {""} // Invalid File Path

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidReadUserFromFileParams")

public void testReadUserFromFileInvalid2(String fileName) {

// Run Method

uf.readUserFromFile(fileName);

}

// Write User To File

// Invalid Parameters

private Object getInvalidWriteUserToFileParams() {

User[] sampleArray = {validUser1, validUser2};

return new Object[] {

new Object[] {null, validFile}, // Null User List

new Object[] {sampleArray, null}, // Null File Name

new Object[] {sampleArray, ""} // Invalid File Path

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidWriteUserToFileParams")

public void testWriteUserToFileInvalid(User[] userArray, String fileName) {

// Run Method

uf.writeUserToFile(userArray, fileName);

}

}

UserFileUnitTest.java

package utar.edu.my;

import static org.junit.Assert.*assertArrayEquals*;

import static org.mockito.Mockito.*mock*;

import static org.mockito.Mockito.*times*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import java.util.List;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import junitparams.Parameters;

import junitparams.JUnitParamsRunner;

*@RunWith*(JUnitParamsRunner.class)

public class UserFileUnitTest {

// Mock and Target Class

private FileFunctionality ffMock;

private UserFile uf;

// Sample User Objects

private User validUser1 = new User("USER001", "MIKU", "miku@gmail.com" , "0123456789");

private User validUser2 = new User("USER002", "TETO", "teto@gmail.com" , "0198765432");

private User validUser3 = new User("USER003", "GUMI", "gumi@gmail.com" , "0121892398");

private User validUser4 = new User("USER004", "LEN" , "len@gmail.com" , "0129850802");

private User validUser5 = new User("USER005", "RIN" , "rin@gmail.com" , "0123435887");

// File Paths

private String validFile = "TestData\\userDummy.txt";

// Setup For all Test Classes

*@Before*

public void setupClass() {

// Setup Mock and Class

ffMock = *mock*(FileFunctionality.class);

uf = new UserFile(ffMock);

}

// Read User From File

// Valid Output

private Object getReadUserFromFileValidParams() {

// Sample Arrays

String[] result1 = {validUser1.toString(), validUser2.toString()};

String[] result2 = {validUser2.toString(), validUser3.toString()};

String[] result3 = {validUser3.toString(), validUser4.toString()};

String[] result4 = {validUser4.toString(), validUser5.toString()};

String[] result5 = {validUser5.toString(), validUser1.toString()};

User[] array1 = {validUser1, validUser2};

User[] array2 = {validUser2, validUser3};

User[] array3 = {validUser3, validUser4};

User[] array4 = {validUser4, validUser5};

User[] array5 = {validUser5, validUser1};

return new Object[] {

new Object[] {result1, array1},

new Object[] {result2, array2},

new Object[] {result3, array3},

new Object[] {result4, array4},

new Object[] {result5, array5}

};

}

*@Test*

*@Parameters* (method = "getReadUserFromFileValidParams")

public void testReadUserFromFileValid(String[] readOutput, User[] expectedArray) {

// Setup Mock Return Value

*when*(ffMock.readFromFile(validFile)).thenReturn(readOutput);

// Run Method

List<User> userList = uf.readUserFromFile(validFile);

User[] userArray = userList.toArray(new User[userList.size()]);

// Compare Results

*assertArrayEquals*(userArray, expectedArray);

}

// Invalid Output

private Object[] getInvalidReadUserFromFileOutput() {

return new Object[] {

// Invalid User Detail Format

new Object[] {"MIKU;miku@gmail.com;0123456789"}, // Less Data

new Object[] {"USER001;MIKU;miku@gmail.com;0123456789;INVALID"} // Additional Data

};

}

*@Test* (expected = IllegalArgumentException.class)

*@Parameters* (method = "getInvalidReadUserFromFileOutput")

public void testReadUserFromFileInvalid(String fileReadLine) {

// Sample Data

String[] fileReadOutput = {fileReadLine};

// Setup Mock Return Value

*when*(ffMock.readFromFile(validFile)).thenReturn(fileReadOutput);

// Run Method

uf.readUserFromFile(validFile);

}

// Write User To File

// Valid Parameters

private Object getWriteUserToFileValidParams() {

// Sample Arrays

User[] array1 = {validUser1, validUser2};

User[] array2 = {validUser2, validUser3};

User[] array3 = {validUser3, validUser4};

User[] array4 = {validUser4, validUser5};

User[] array5 = {validUser5, validUser1};

String[] result1 = {validUser1.toString(), validUser2.toString()};

String[] result2 = {validUser2.toString(), validUser3.toString()};

String[] result3 = {validUser3.toString(), validUser4.toString()};

String[] result4 = {validUser4.toString(), validUser5.toString()};

String[] result5 = {validUser5.toString(), validUser1.toString()};

return new Object[] {

new Object[] {array1, result1},

new Object[] {array2, result2},

new Object[] {array3, result3},

new Object[] {array4, result4},

new Object[] {array5, result5}

};

}

*@Test*

*@Parameters* (method = "getWriteUserToFileValidParams")

public void testWriteUserToFileValid(User[] writeArray, String[] expectedArray) {

// Run Method

uf.writeUserToFile(writeArray, validFile);

// Verify Method Execution

*verify*(ffMock, *times*(1)).writeToFile(expectedArray, validFile);

}

// Invalid Parameters

*@Test* (expected = IllegalArgumentException.class)

public void testWriteUserToFileInvalid() {

// Run Method

uf.writeUserToFile(null, validFile); // Null User List

}

}