ALireza Karamoozian 4043624017

Assignment 1 Date: December 30, 2025

Preface:

- ➢ **Videos are available at:**
  https://github.com/cynidn2x/AdSE-Assignment-1/tree/execution-video/Videos
- ➢ **Tests' execution are also accessible through *github actions:***
  https://github.com/cynidn2x/AdSE-Assignment-1/actions/runs/20578937191/job/59102105994
- ➢ **The Implementation is accessible at:**
  https://github.com/cynidn2x/AdSE-Assignment-1/tree/main

Section 1: Black-box Analysis

Firstly, interface methods are extracted from their specific path in the project.("*train-reservation-master/src/main/java/fr/univnantes/trainreservatio*"). However, writing black-box tests for a proportion of these functions was either meaningless or impossible. After selecting particular interface methods, the Classification Tree Method was performed on each interface method[0,1,2].

Below, Interface methods are classified based on their class interface:

- **City:**
  - ❖ No appropriate method was found.
- **Train Interface:**
  - ❖ No appropriate method was found.
- **Trip Interface:**

❖ `Duration getPlannedDepartureTime();`

Input:

```
this ∈ Trip
    this.getRealDepartureTime() ∈ Time
    this.getRealArrivalTime() ∈ Time
```

Output:

```
this ∈ Trip
    this.findRealDuration() ∈ Time
```

ALireza Karamoozian 4043624017

Assignment 1

Date: December 30, 2025

	this.getPlannedDepartureTime()	this.getPlannedArrivalTime()	findRealDuration()
C1	"2025-12-30 13:00"	"2025-12-31 15:00"	"0000-00-01 02:00"

❖ Duration **findRealDuration**()

Input:

- this ∈ Trip
  - this.getPlannedDepartureTime() ∈ Time
  - this.getPlannedArrivalTime() ∈ Time
  - this.getDepartureDelay() ∈ Time
  - this.getArrivalDelay() ∈ Time

Output:

- Result ∈ Time

	this.getPlannedDepartureTime()	this.getPlannedArrivalTime()	this.getDepartureDelay()	this.getArrivalDelay()	Result
C1	"2025-12-30 13:00"	"2025-12-30 15:00"	5m	null	"0000-00-00 1:55"
C2	"2025-12-30 13:00"	"2025-12-30 15:00"	null	5m	"0000-00-0 2:05"
C3	"2025-12-30 13:00"	"2025-12-30 15:00"	null	null	"0000-00-00 2:00"
C4	"2025-12-30 13:00"	"2025-12-30 15:00"	5m	10m	"0000-00-0 2:05"

❖boolean isDelayed()

Input:

- this ∈ Trip
  - this.getDepartureDelay() ∈ Time
  - this.getArrivalDelay() ∈ Time

Output:

- Result ∈ bool

	this.getDepartureDelay()	this.getArrivalDelay()	Result
C1	null	null	False
C2	null	0s	False
C3	0s	null	False
C4	0s	0s	False
C5	1s	null	True
C6	null	2s	True
C7	2s	1h	True

❖Duration getArrivalDelay();

Input:

- this ∈ Trip
  - this.addArrivalDelay(delay)∈Time

Output:

- Result ∈ Time
  - Result.getArrivalDelay() ∈ Time

	this.addArrivalDelay(delay)	Result.getArrivalDelay()
C1	5s	5s
C2	null	0

```
Duration getDepartureDelay();
```

Same as above !

❖ `Ticket bookTicket(String passengerName) throws ReservationException;`

Input:

- passengerName ∈ String
- this ∈ Trip
  - this.getTrain().getMaxPassengers() ∈ int
  - this.getBookedTickets().size() ∈ int

Output:

- Result ∈ Ticket
  - Result.getPassengerName() ∈ String
- ReservationException

	passernName	this.getTrain().getMaxPassengers()	this.getBookedTickets().size()	Result.getPassengerName()	ReservationException
C1	"Ali"	2	0	"Ali"	
C2	"Reza"	2	2		✖

❖`void addDepartureDelay(Duration delay)`

Input:

`delay ∈ Time`
`this ∈ Trip`
`this.getPlannedDepartureTime() ∈ Time`
`this.getPlannedArrivalTime() ∈ Time`

Output:

`this ∈ Trip`
`this.getDepartureDelay() ∈ Time`
`this.findRealDepartureTime() ∈ Time`

	delay	this.getPlannedDepartureTime()	this.getPlannedArrivalTime()	this.getDepartureDelay()	thisfindRealDepartureTime()
C1	5s	"2025-12-31 14:00"	"2025-12-31 15:00"	"2025-12-31 14:05"	5s

❖`void addArrivalDelay(Duration delay);`

Same as above !

- **Ticket Interface:**
  - ❖ `Ticket exchangeTicket(Trip trip) throws ReservationException;`

    Input:
    - `trip ∈ Trip`
      - `trip.getOrigin() ∈ City`
      - `trip.getDestination() ∈ City`
      - `trip.getPlannedDepartureTime()∈ Time`
      - `trip.iscancelled()∈ boolean`
    - `this ∈ Trip`
      - `this.getOrigin() ∈ City`
      - `this.getDestination() ∈ City`
      - `this.getPlannedDepartureTime()∈ Time`

    Output:
    - `Result ∈ Ticket`
    - `ReservationException`

	trip	this.getOrigin()	this.getDestination()	this.getPlannedDepartureTime()	trip.getOrigin()	trip.getDestination()	trip.getPlannedDepartureTime()	trip.iscancelled()	Result	ReservationException
C1	null	"Tehran"	"isfahan"	"2022-10-19 13:00"	null	null	null	null		✖
C2	t1	"Tehran"	"Isfahan"	"2022-10-19 13:00"	"Tehran"	"Isfahan"	"2022-10-20 13:00"	False	✖	
C3	t1	"Tehran"	"Yazd"	"2022-10-19 13:00"	"Tehran"	"Kerman"	"2022-10-20 13:00"	False		✖
C4	t1	"Yazd"	"Isfahan"	"2022-10-19 13:00"	"Tehran"	"Isfahan"	"2022-10-20 13:00"	False		✖
C5	t1	"Tehran"	"Isfahan"	"2022-10-19 13:00"	"Tehran"	"Isfahan"	"2022-10-20 12:00"	False		✖
C6	t1	"Tehran"	"Isfahan"	"2022-10-19 13:00"	"Tehran"	"Isfahan"	"2022-10-20 12:00"	True		✖

- **TicketReservationSystem Interface:**

```
❖Trip createTrip(City origin, City
  destination, Train train, Instant departure,
  Instant arrival) throws TripException;
```

Input:

- origin ∈ City
- destination ∈ City
- train ∈ Train
- Departure ∈ Time
- arrival ∈ Time

Output:

- Result ∈ Trip
- TripException

	origin	destination	train	departure	arrival	Result	TripException
C1	"Isfahan"	"Tehran"	t1	"2025-12-30 14:05"	"2025-12-31 14:05"	✖	
C2	"Tehran"	"Tehran"	t1	"2025-12-30 14:05"	"2025-12-31 14:05"		✖
C3	"Tehran"	"isfahan"	t1	"2025-12-30 14:05"	"2025-12-30 14:05"		✖
C4	"Tehran"	"isfahan"	t1	"2025-12-31 14:05"	"2025-12-30 14:05"		✖
C5	"Tehran"	"Isfahan"	null	"2025-12-30 14:05"	"2025-12-31 14:05"		✖
C6	null	"Isfahan"	t1	"2025-12-30 14:05"	"2025-12-31 14:05"		✖

C7	"Tehran"	null	t1	"2025-12-30 14:05"	"2025-12-31 14:05"		✖
C8	"tehran	"Isfahan"	t1	null	"2025-12-31 14:05"		✖
C9	"tehran	"Isfahan"	t1	"2025-12-30 14:05"	null		✖

Overall, testing these methods from a black-box perspective requires at **36** Unit tests at least.

Section 2: Test Case Design

Initially, Five functions were selected randomly.Next, a scenario is constructed for each; accordingly, their input, expected output and fixture are then defined.

❖ Trip createtrip():

```
   /**
    * Creates and registers a new trip in the system.
    * This new trip must take place *after* the last trip of the chosen train (based on the real arrival time).
    * In addition, a trip can only be created if the following constraints are satisfied:
    * - The last destination of the train must be in the same city as the origin of the new trip.
    * - The last arrival of the train must be at least 10 minutes from the departure of the new trip (using real
departure and arrival times).
    * - The arrival must come after the departure.
    * - The origin must be different from the destination.
    * @return The created registered trip.
    * @throws TripException If one of the above constraints was not satisfied.
    */
   Trip createTrip(City origin, City destination, Train train, Instant departure, Instant arrival) throws
TripException;
```

- **Scenario:**

  An admin attempts to create a trip from certain cities, and in a scheduled time; However, if the admin set the departure time a date later than the arrival time. The system is expected to throw an exception.Moreover, the system should check that the origin and departures are not the same.

- **Prefix:**
  1. An object of the *TicketReservationSystem* must have been created with a specific time zone.
  2. Two distinct objects of the *City* must have been created.
  3. Two distinct Instant of the *Timemangement* must have been created.
  4. An instance of Train must be constructed
- **Input:**
  1. The first input is an object of the *City* such as “Tehran”
  2. The second input is an object of the *City* such as “Isfahan”
  3. The departure instance is given as “"2022-05-12 12:00"”
  4. The arrival instance is given as “2021-05-13 13:00”
  5. A train object such as “Fadak”

- **Expected output:**
  1. An exception indicating that the arrival time can merely be later than the departure time
  2. An exception if the same city are given for origin and the departure
- **Postfix:**
  1. It is advisable to set the objects of time and city to null.

❖ **Ticket bookTicket():**

```
/**
 * Creates a new Ticket for the trip, and records the ticket in the trip.
 * It is only possible to book a ticket if the trip had not already reached the maximum amount of passengers.
 * @param passengerName The name of the passenger.
 * @return The ticket that has been booked.
 * @throws ReservationException If the trip has already reached the maximum amount of passengers.
 */
Ticket bookTicket(String passengerName) throws ReservationException;
```

- **Scenario:**

  A passenger attempts to book a specific ticket. The system is expected to accept this request and book a ticket with the passenger name if and only if there is a vacancy on that trip.

- **Prefix:**
  1. A trip object must have been created.
- **Input:**
  1. The passengers name as a *String*
  2. The trip object
- **Expected output:**
  1. A message indicating the request was successful
  2. In case of the trip being fully-booked, throw an exception notifying the user that the trip is not available.
  3. If a list of tickets is retrieved, the list should contain the ticket
- **Postfix:**
  1. No action is needed.

❖ **Ticket cancel():**

```
/**
 * Cancels this ticket. Cannot be undone.
 */
void cancel();
```

- **Scenario:**

  When a passenger decides to cancel the ticket, the system is expected to mark that ticket as cancelled and show it in the list of cancelled tickets. If the ticket has been already canceled, the system is obligated to notify it.

- **Prefix:**
  1. A trip must have been created.
  2. A ticket must have been booked
- **Input:**
  1. The object of ticket
  2. The object of the Trip
- **Expected output:**
  1. If the ticket has been already canceled, display message indicating that the operation is not possible
  2. Otherwise, a message indicating that the cancellation was successful
- **Postfix:**
  1. No action is needed.

❖ **TicketReservationSystem delayTripArrival():**

```
/**
 * Adds an arrival delay to a trip.
 * Also automatically also adds the delay to the departure of the next trip of the train.
 * @param delay The amount of delay to add.
 */
void delayTripArrival(Trip trip, Duration delay);
```

- **Scenario:**

  An admin intends to add delay to the arrival time of a single trip. The system is expected to modify the arrival time of that trip, and add the delay to it.Moreover, the system is expected to add the same delay to the departure time of the train's next trip.

- **Prefix:**
  1. A trip must have been created.
- **Input:**
  1. An object of trip similar to the *Trip createtrip()*'s input.
  2. A delay in the *java.time.duration* format; the delay might be in second, minute or hour.
- **Expected output:**
  1. The trip's arrival time must be altered; The subtract of *findRealArrivalTime() from getPlannedArrivalTime() should be equivalent to the value of delay.*

- **Postfix:**
  1. No action is needed

Section 3: Gherkin Scenarios

Ten scenarios are written, while half expect a successful execution, and the rest expect exceptions[3].

```
Feature: BDD Feature for Train Reservation System
```

```
    @happypaths @implementation
    Scenario: Admin can create a new trip
        Given Arrival time has been set to "2025-12-26T12:00:00Z"
        And Departure time has been set to "2025-12-26T10:00:00Z"
        And Destination city has been set to "Isfahan"
        And Origin city has been set to "Tehran"
        When the Admin attempts to create a new trip from "Tehran" to "Isfahan" departing at "2025-12-26T10:00:00Z" and arriving at
"2025-12-26T12:00:00Z"
        Then the system should have created exactly 1 trip
        And the trip must have "Tehran" as origin and "Isfahan" as destination

    @happypaths  @implementation
    Scenario: Admin can delay a trip's arrival time
        Given a trip exists from "Kerman" to "Yazd" departing at "2025-12-26T10:00:00Z" and arriving at "2025-12-26T12:00:00Z"
        When the Admin delays the trip arrival time by 4 minutes
        Then the trip should be marked as delayed
        And  the arrival delay should be 4 minutes

    @happypaths
    Scenario: Admin can delay a trip's departure time
        Given a trip has been created by the Admin
        When the user requests to delay the trip departure time
        Then the system adds a delay to the trip departure time

    @happypaths
    Scenario: User can book a train ticket
        Given the user has searched for available trains
        When the user selects a train and provides passenger details
        And makes a payment
        Then the system confirms the booking and provides a ticket

    @happypaths
    Scenario: User can search for available trips between two cities
        Given a trip has been created successfully
        When the user enters departure and destination cities along with travel dates
        Then the system displays a list of available trains matching the criteria

    @exceptionpaths @implementation
    Scenario: User cannot book a ticket on a fully booked train
        Given a trip has been created with capacity of 2
        And the trip has reached maximum passenger capacity of 2
        When the user attempts to book a ticket
        Then the system throws a ReservationException indicating no available seats

    @exceptionpaths
    Scenario: User cannot book a ticket on a cancelled trip
        Given a trip has been created
        And the trip has been cancelled
        When the user attempts to book a ticket
        Then the system throws a ReservationException indicating the trip has been cancelled

    @exceptionpaths
    Scenario: User cannot book tickets with departure time in the past
        Given a trip has been created
        And the trip departure time has already passed
        When the user attempts to book a ticket
        Then the system throws a ReservationException indicating invalid departure time

    @exceptionpaths
    Scenario: Admin cannot create a trip with the same departure and destination cities
        Given a city has been created
        When the Admin enters the same city for both departure and destination
        Then the system throws a ValidationException indicating invalid trip details

    @exceptionpaths
    Scenario: Admin cannot create a trip with arrival time before departure time
        Given a departure time has been created
        And an arrival time has been created
        When the Admin enters an arrival time that is before the departure time
        Then the system throws a ValidationException indicating invalid trip details
```

Section 4: Implementation using Junit

https://github.com/cynidn2x/AdSE-Assignment-1/blob/main/train-reservation-master/src/test/java/fr/univnantes/trainreservation/SimpleJUnitTest.java

Section 5: Implementation Using Cucumber

https://github.com/cynidn2x/AdSE-Assignment-1/blob/main/train-reservation-master/src/test/java/fr/univnantes/trainreservation/stepdefinitions/TicketReservationSteps.java

Section 6: Implementation Using Mockito

https://github.com/cynidn2x/AdSE-Assignment-1/blob/main/train-reservation-master/src/test/java/fr/univnantes/trainreservation/MockitoTests.java

Acknowledgement:

I acknowledge the use of Claude 4.5 and ChatGPT 5 to generate code suggestions ,inspiration and to clarify some concepts aiding my understanding.

Sources:

[0] Ammann, Paul, and Jeff Offutt. Introduction to Software Testing. 2nd ed. Cambridge: Cambridge University Press, 2016
[1] Peter M. Kruse and Magdalena Luniak: Automated Test Case Generation Using Classification Trees in Proceedings of StarEast 2010,Orlando,Florida,USA,2010
[2]https://gl.univ-nantes.io/testing/course-m1alma/cours/2025-2026/_exports/cours-2025-2026.pdf#introduction
[3]https://cucumber.io/docs/gherkin/reference
[4]https://docs.junit.org/5.14.1/overview.html
[5]https://github.com/mockito/mockito/wiki