Railway Booking System

Software Engineering

Assignment 4

Test Report

Ashutosh Kumar Singh

19CS30008

Unit Test Plan

1 Testing the `Stations` class

Positive Test Cases

1.1 Testing the constructor `Station(const string&)`

1. Create an object by passing a string as name.
   PASSED

1.2 Testing the `CreateStation(const string&)` function

1. Create an object with a valid Station name
   PASSED

1.3 Testing the `GetName()` function

1. Call the function for an already constructed object
   PASSED

1.4 Testing the `GetDistance(const Station&)` function

1. Call the function for 2 Stations
   PASSED

1.5 Testing the output streaming operator

1. Print a Station object to the screen
   PASSED

Negative Test Cases

1.6 Testing the `CreateStation(const string&)` function

1. Try to create a Station object with name as an empty string
   FAILED

2 Testing the `Railways` class

Positive Test Cases

2.1 Testing the constructor `Railways()` and `IndianRailways()` function

1. Check for the singleton behaviour
   PASSED

2.2 Testing the `GetDistance(.,.)` function

1. Check for a A-B pair
   PASSED
2. Check for the corresponding B-A pair
   PASSED

2.3 Testing the `GetStation(const string& name)` function

1. Call the function for a valid Station as a string
   PASSED

2.4 Testing the output streaming operator

1. Print a Railways object to the screen
   PASSED

Negative Test Cases

2.5 Testing the `GetStation(const string& name)` function

1. Call the function with an invalid Station name as a string
   FAILED

3 Testing the `Date` and `Duration` classes

Positive Test Cases

3.1 Testing the constructor `Date(int, int, int)`

1. Create a new valid Date
   PASSED

3.2 Testing the `day()` function

1. Retrieve the day for a Date object
   PASSED

3.3 Testing the `CreateDate(...)` function

1. Create a Date object with valid values
   PASSED

2. For 29 Feb, 2020
   PASSED

3. For 29 Feb, 2000
   PASSED

3.4 Testing the `friend Duration operator-(const Date&, const Date&)` operator

1. Get the difference between two dates
   PASSED

3.5 Testing the `bool operator>(const Date&)` operator

1. Check for a true condition
   PASSED

2. Check for a false condition
   PASSED

3.6 Testing the `bool operator==(const Date&)` function

1. Check for a true condition
   PASSED

2. Check for a false condition
   PASSED

3.7 Testing the `Date` output streaming operator

1. Print a Date object to the screen
   PASSED

Negative Test Cases

3.8 Testing the `CreateDate(...)` function

1. 29 Feb, 2021 - a non-leap year
   FAILED
2. 29 Feb, 2021 - a century year
   FAILED
3. Try for a date with 31st in a month of 30 days
   FAILED
4. A string with the wrong format
   FAILED
5. A string with position of date and month interchanged
   FAILED
6. A date with year below the lower limit of 1900
   FAILED
7. A date with year above the upper limit of 2099
   FAILED

4 Testing the `Name` class

Positive Test Cases

4.1 Testing the `CreateName(...)` `function`

1. When first name, middle name and last name are present
   PASSED

2. When only first name and last name are present
   PASSED

3. When only first name is present
   PASSED

4. When only last name is present
   PASSED

5. When first name and middle name are present
   PASSED

6. When middle name and last name are present
   PASSED

Negative Test Cases

7. When only middle name is present
   FAILED

8. When all are left empty
   FAILED

5 Testing the `BookingClass` class and its hierarchy

Here, for all the 8 leaf classes we test the following items :

1. Singleton behaviour by checking the addresses as described before
2. The output streaming operator
3. The getter functions :
   - `GetName()`
   - `GetLoadFactor()`
   - `IsSitting()`
   - `IsAC()`
   - `GetNumberOfTiers()`
   - `IsLuxury()`
   - `GetReservationCharge()`
   - `GetTatkalLoadFactor()`
   - `GetMinTatkalCharge()`
   - `GetMaxTatkalCharge()`
   - `GetMinTatkalDistance()`

Golden Outputs

5.1 `ACFirstClass`

5.1.1 Testing the output streaming operator

PASSED

5.1.2 Testing the getter functions

PASSED

5.2 `ExecutiveChairCar`

PASSED

5.2.1 Testing the getter functions

PASSED

5.3 `AC2Tier`

5.3.1 Testing the output streaming operator

PASSED

5.3.2 Testing the getter functions

PASSED

5.4 `FirstClass`

5.4.1 Testing the output streaming operator

PASSED

5.4.2 Testing the getter functions

PASSED

5.5 `AC3Tier`

5.5.1 Testing the output streaming operator

PASSED

5.5.2 Testing the getter functions

PASSED

5.6 `ACChairCar`

5.6.1 Testing the output streaming operator

PASSED

5.6.2 Testing the getter functions

PASSED

5.7 `Sleeper`

5.7.1 Testing the output streaming operator

PASSED

5.7.2 Testing the getter functions

PASSED

5.8 `SecondSitting`

5.8.1 Testing the output streaming operator

textbfPASSED

5.8.2 Testing the getter functions

PASSED

6 Testing the `BookingCategory` and `Divyaang` classes and their hierarchies

6.1 Testing `BookingCategory::General`

6.1.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.1.2 Testing the `IsEligible(...) function`

1. Call the function for any passenger, it always returns true
   PASSED

6.2 Testing `BookingCategory::Ladies`

6.2.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.2.2 Testing the `IsEligible(...) function`

1. Call the function for a female passenger
   PASSED
2. Call the function for a male passenger
   PASSED

6.3 Testing `BookingCategory::SeniorCitizen`

6.3.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.3.2 Testing the `IsEligible(...) function`

1. Call the function for a senior citizen passenger
   PASSED
2. Call the function for a non senior citizen passenger
   PASSED

6.4 Testing `Divaang::Blind`

6.4.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.4.2 Testing the `IsEligible(...)` function

1. Call the function for a blind passenger
   PASSED

2. Call the function for a passenger with no disablity
   PASSED

6.5 Testing `Divaang::OrthoHandicapped`

6.5.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.5.2 Testing the `IsEligible(...)` function

1. Call the function for an orthopaedically handicapped passenger
   PASSED

2. Call the function for a passenger with no disablity
   PASSED

6.6 Testing `Divaang::Cancer`

6.6.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.6.2 Testing the `IsEligible(...)` function

1. Call the function for a passenger having cancer
   PASSED

2. Call the function for a passenger with no disablity
   PASSED

6.7 Testing `Divaang::TB`

6.7.1 Testing the `GetName()` function

1. Call the function and verify the name
   PASSED

6.7.2 Testing the `IsEligible(...)` function

1. Call the function for a passenger having TB
   PASSED

2. Call the function for a passenger with no disablity
   PASSED

7 Testing the `Concessions` class and its hierarchy

7.1 Testing the `GeneralConcession` derived class

7.1.1 Testing the constructor `GeneralConcession(string&)` and `Type()` function

1. Check for singleton behaviour
   PASSED

7.1.2 Testing the `GetFactor()` function

1. Check the value returned by the function
   PASSED

7.2 Testing the `LadiesConcession` derived class

7.2.1 Testing the constructor `LadiesConcession(string&)` and `Type()` function

1. Check for singleton behaviour
   PASSED

7.2.2 Testing the `GetFactor()` function

1. Check the value returned by the function
   PASSED

7.3 Testing the `SeniorCitizenConcession` derived class

7.3.1 Testing the constructor `SeniorCitizenConcession(string&)` and `Type()` function

1. Check for singleton behaviour
   PASSED

7.3.2 Testing the `GetFactor(Passenger&)` function

1. Get the concession factor for a male senior citizen
   PASSED

2. Get the concession factor for a female senior citizen
   PASSED

7.4 Testing the `DivyaangConcessions` derived class

7.4.1 Testing the constructor `DivyaangConcessions(string&)` and `Type()` function

1. Check for singleton behaviour
   PASSED

7.4.2 Testing the `GetFactor(Passenger&, BookingClass&)` function

For `Divyaang::Blind`, `Divyaang::OrthoHandicapped`, `Divyaang::Cancer` and `Divyaang::TB`, chek the various concession factors for each of the 8 Booking Classes.
PASSED

8 Testing the `Passenger` class

Positive Test Cases

8.1 Testing the constructor `Passenger(...)`

1. Create an object by passing all the arguments.
   PASSED

2. Create an object by leaving out the defaulted arguments - `mobile_`, `disabilityType_`, `disabilityID_`
   PASSED

8.2 Testing the `CreatePassenger(...)` function

1. Create an object by passing all the arguments and all the arguments should be valid.
   PASSED

Negative Test Cases

8.3 Testing the `CreatePassenger(...)` function

1. Pass an invalid name - keep first and last name as empty
   FAILED

2. Pass an invaild DOB - in the future
   FAILED

3. Pass an invaild aadhaar number
   FAILED

4. Pass an invalid mobile number
   FAILED

9 Testing the `Booking` class and its hierarchy

9.1 Testing the `ComputeFare()` function

9.1.1 Test for each `BookingCategory` and each pair of stations

Since there are 10 Booking Categories(General, Ladies, Male Senior Citizen, Female Senior Citizen, Tatkal, Premium Tatkal + 4 Divyaang categories) and also 10pairs of to and from stations, we can combine these test cases together.
ALL 10 PASSED

9.1.2 Test for each `BookingClass`

Keep the folowing attributes constant, and vary the `bookingClass_` :

```
fromStation_ = "Delhi"
toStation_ = "Mumbai"
gender_ = Male
bookingCategory_ = General
dateOfBooking_ = "15/04/2021"
```

1. **Input :** `bookingClass_ = ACFirstClass`
   Output : `fare_ = 4763`
   PASSED
2. **Input :** `bookingClass_ = ExecutiveChairCar`
   Output : `fare_ = 3678`
   PASSED
3. **Input :** `bookingClass_ = AC2Tier`
   Output : `fare_ = 2944`
   PASSED
4. **Input :** `bookingClass_ = AC3Tier`
   Output : `fare_ = 1849`
   PASSED
5. **Input :** `bookingClass_ = ACChairCar`
   Output : `fare_ = 1487`
   PASSED
6. **Input :** `bookingClass_ = FirstClass`
   Output : `fare_ = 2221`
   PASSED
7. **Input :** `bookingClass_ = Sleeper`
   Output : `fare_ = 744`
   PASSED
8. **Input :** `bookingClass_ = SecondSitting`
   Output : `fare_ = 449`
   PASSED

Application Test Plan

Positive Test Cases

Here, create 10 bookings one corresponding to each Booking Category.

1. Booking Category - General
   PASSED
2. Booking Category - Ladies
   PASSED
3. Booking Category - Senior Citizen Male
   PASSED
4. Booking Category - Senior Citizen Female
   PASSED
5. Booking Category - Tatkal
   PASSED
6. Booking Category - Premium Tatkal
   PASSED
7. Booking Category - Divyaang - Blind
   PASSED
8. Booking Category - Divyaang - Orthopaedically Handicapped
   PASSED
9. Booking Category - Divyaang - Cancer
   PASSED
10. Booking Category - Divyaang - TB
    PASSED

Negative Test Cases

1. Invalid `fromStation_`
   FAILED
2. Invalid `toStation_`
   FAILED
3. `fromStation_` and `toStation_` are the same
   FAILED
4. `dateOfBooking_` is invalid
   FAILED
5. Same day booking, i.e., `dateOfBooking_` and `dateOfReservation_` are same
   FAILED

6. `dateOfBooking_` is more than one year later than the `dateOfReservation_`
   FAILED

7. Trying for `Tatkal` but date difference is more than one day
   FAILED

8. Not eligible for the `Divyaang` category
   FAILED

9. Not eligible for the `SeniorCitizen` Category
   FAILED

10. Invalid `Passenger` information
    FAILED