Railway Booking System

Software Engineering

Assignment 4

Test Plan

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.
   Input : `Station s1("Kolkata")`
   Output : Object should be created with `name_ = Kolkata`

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

1. Create an object with a valid Station name
   Input : `Station::CreateStation("Mumbai")`
   Output : Object should be created with `name_ = Mumbai`

1.3 Testing the `GetName()` function

1. Call the function for an already constructed object
   Input : `s1.GetName()`
   Output : `Kolkata`

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

1. Call the function for 2 Stations
   Input : `Station s1("Kolkata"), s3("Delhi)`
   `s1.GetDistance(s3)`
   Output : `1472`

1.5 Testing the output streaming operator

1. Print a Station object to the screen
   Input : `cout << s1`
   Output : `Station : Kolkata`

Negative Test Cases

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

1. Try to create a Station object with name as an empty string
   Input : `Station::CreateStation("")`
   Output : `Station name cannot be empty`

2 Testing the `Railways` class

Positive Test Cases

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

1. Check for the singleton behaviour
   Input : `firstPointer = &Railways::IndianRailways()`
   `secondPointer = &Railways::IndianRailways()`
   Output : On asserting, both pointers should be equal

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

1. Check for a A-B pair
   Input : `GetDistance(Station("Bangalore"), Station("Kolkata")))`
   Output : `1871`

2. Check for the corresponding B-A pair
   Input : `GetDistance(Station("Kolkata"), Station("Bangalore")))`
   Output : `1871`

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

1. Call the function for a valid Station as a string
   Input : `Station* sp = Railways::IndianRailways().GetStation("Mumbai")`
   Output : `sp -> GetName() = Mumbai`

2.4 Testing the output streaming operator

1. Print a Railways object to the screen **Input :** `cout << Railways::IndianRailways()`
   Output :

```
Indian Railways

List of Stations
Station :  Mumbai
Station :  Delhi
Station :  Bangalore
Station :  Kolkata
Station :  Chennai

Pairwise Distances
Bangalore - Chennai = 350 km
Bangalore - Kolkata = 1871 km
Delhi - Bangalore = 2150 km
Delhi - Chennai = 2180 km
Delhi - Kolkata = 1472 km
Kolkata - Chennai = 1659 km
Mumbai - Bangalore = 981 km
Mumbai - Chennai = 1338 km
```

```
Mumbai - Delhi = 1447 km
Mumbai - Kolkata = 2014 km
```

Negative Test Cases

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

1. Call the function with an invalid Station name as a string
   Input : `GetStation("Bombay")`
   Output : `Station name is invalid :  Bombay`

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
   Input : `Date d1(8, 4, 2021)`
   Output : `d1.date_ = 8, d1.month_ = Apr, d1.year_ = 2021`

3.2 Testing the `day()` function

1. Retrieve the day for a Date object
   Input : `d1.day()`
   Output : `Thu`

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

1. Create a Date object with valid values
   Input : `Date* d3p = Date::CreateDate("9/4/2021")`
   Output : `d3p -> date_ = 9, d3p -> month_ = 4, d3p -> year_ = 2021`

2. For 29 Feb, 2020
   Input : `Date* d3p = Date::CreateDate("29/2/2020")`
   Output : `d3p -> date_ = 29, d3p -> month_ = 2, d3p -> year_ = 2020`

3. For 29 Feb, 2000
   Input : `Date* d3p = Date::CreateDate("29/2/2000")`
   Output : `d3p -> date_ = 29, d3p -> month_ = 2, d3p -> year_ = 2000`

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

1. Get the difference between two dates
   Input : `d3 = 9/4/2021`, `d4 = 15/5/2022`
   `Duration dur = d4 - d3`
   Output : `1 years, 1 months, 6 days`

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

1. Check for a true condition
   Input : `d4 > d3`
   Output : `true`

2. Check for a false condition
   Input : `d3 > d4`
   Output : `false`

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

1. Check for a true condition
   Input : `d1 == d2`
   Output : `true`

2. Check for a false condition
   Input : `d3 == d4`
   Output : `false`

3.7 Testing the `Date` output streaming operator

1. Print a Date object to the screen
   Input : `cout << d1`
   Output : `Thu, 8/Apr/2021`

Negative Test Cases

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

1. 29 Feb, 2021 - a non-leap year
   Input : `Date::CreateDate("29/02/2021")`
   Output : `Date is invalid for : 29/02/2021`

2. 29 Feb, 2021 - a century year
   Input : `Date::CreateDate("29/02/1900")`
   Output : `Date is invalid for : 29/02/1900`

3. Try for a date with 31st in a month of 30 days
   Input : `Date::CreateDate("31/04/2020")`
   Output : `Date is invalid for : 31/04/2020`

4. A string with the wrong format
   Input : `Date::CreateDate("3104/2020")`
   Output : `Date is invalid for : 3104/2020`

5. A string with position of date and month interchanged
   Input : `Date::CreateDate(04/31/2020")`
   Output : `Date is invalid for : 04/31/2020`

6. A date with year below the lower limit of 1900
   Input : `Date::CreateDate("11/03/1899")`
   Output : `Year 1899 is not in the valid range`

7. A date with year above the upper limit of 2099
   Input : `Date::CreateDate("31/04/2100")`
   Output : `Year 2100 is not in the valid range`

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
   Input : `Name::CreateName("Ashutosh", "Kumar", "Singh")`
   Output : `firstName_ = "Ashutosh", middleName_ = "Kumar", lastName_ = "Singh"`

2. When only first name and last name are present
   Input : `Name::CreateName("Ashutosh", "", "Singh")`
   Output : `firstName_ = "Ashutosh", middleName_ = "", lastName_ = "Singh"`

3. When only first name is present
   Input : `Name::CreateName("Ashutosh", "", "")`
   Output : `firstName_ = "Ashutosh", middleName_ = "", lastName_ = ""`

4. When only last name is present
   Input : `Name::CreateName("", "", "Singh")`
   Output : `firstName_ = "", middleName_ = "", lastName_ = "Singh"`

5. When first name and middle name are present
   Input : `Name::CreateName("Ashutosh", "Kumar", "")`
   Output : `firstName_ = "Ashutosh", middleName_ = "Kumar", lastName_ = ""`

6. When middle name and last name are present
   Input : `Name::CreateName("", "Kumar", "Singh")`
   Output : `firstName_ = "", middleName_ = "Kumar", lastName_ = "Singh"`

Negative Test Cases

7. When only middle name is present
   Input : `Name::CreateName("", "Kumar", "")`
   Output : `At least one of first name or last name should be present`

8. When all are left empty
   Input : `Name::CreateName("", "", "")`
   Output : `Name cannot be completely empty`

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

```
Travel Class = AC First Class
   : Mode: Sleeping
   : Comfort: AC
   : Bunks: 2
   : Luxury: Yes
```

5.1.2 Testing the getter functions

```
GetName() - AC First Class
GetLoadFactor() - 6.50
IsSitting() - false
IsAC() - true
GetNumberOfTiers() - 2
IsLuxury() - true
GetReservationCharge() - 60.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 400.00
GetMaxTatkalCharge() - 500.00
GetMinTatkalDistance() - 500
```

5.2 ExecutiveChairCar

5.2.1 Testing the output streaming operator

```
Travel Class = Executive Chair Car
   : Mode: Sitting
   : Comfort: AC
   : Bunks: 0
   : Luxury: Yes
```

5.2.2 Testing the getter functions

```
GetName() - AC First Class
GetLoadFactor() - 5.00
IsSitting() - true
IsAC() - true
GetNumberOfTiers() - 0
IsLuxury() - true
GetReservationCharge() - 60.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 400.00
GetMaxTatkalCharge() - 500.00
GetMinTatkalDistance() - 250
```

5.3 AC2Tier

5.3.1 Testing the output streaming operator

```
Travel Class = AC 2 Tier
   : Mode: Sleeping
   : Comfort: AC
   : Bunks: 2
   : Luxury: No
```

5.3.2 Testing the getter functions

```
GetName() - AC 2 Tier
GetLoadFactor() - 4.00
IsSitting() - false
IsAC() - true
GetNumberOfTiers() - 2
IsLuxury() - false
GetReservationCharge() - 50.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 400.00
GetMaxTatkalCharge() - 500.00
GetMinTatkalDistance() - 500
```

5.4 FirstClass

5.4.1 Testing the output streaming operator

```
Travel Class = First Class
   : Mode: Sleeping
   : Comfort: Non-AC
   : Bunks: 2
   : Luxury: Yes
```

5.4.2 Testing the getter functions

```
GetName() - First Class
GetLoadFactor() - 3.00
IsSitting() - false
IsAC() - false
GetNumberOfTiers() - 2
IsLuxury() - true
GetReservationCharge() - 50.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 400.00
GetMaxTatkalCharge() - 500.00
GetMinTatkalDistance() - 500
```

5.5 AC3Tier

5.5.1 Testing the output streaming operator

```
Travel Class = First Class
   : Mode: Sleeping
   : Comfort: AC
   : Bunks: 3
   : Luxury: No
```

5.5.2 Testing the getter functions

```
GetName() - AC 3 Tier
GetLoadFactor() - 2.50
IsSitting() - false
IsAC() - true
GetNumberOfTiers() - 3
IsLuxury() - false
GetReservationCharge() - 40.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 300.00
GetMaxTatkalCharge() - 400.00
GetMinTatkalDistance() - 500
```

5.6 ACChairCar

5.6.1 Testing the output streaming operator

```
Travel Class = AC Chair Car
   : Mode: Sitting
   : Comfort: AC
   : Bunks: 0
   : Luxury: No
```

5.6.2 Testing the getter functions

```
GetName() - AC Chair Car
GetLoadFactor() - 2.00
IsSitting() - true
IsAC() - true
GetNumberOfTiers() - 0
IsLuxury() - false
GetReservationCharge() - 40.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 125.00
GetMaxTatkalCharge() - 225.00
GetMinTatkalDistance() - 250
```

5.7 Sleeper

5.7.1 Testing the output streaming operator

```
Travel Class = Sleeper
   : Mode: Sleeping
   : Comfort: Non-AC
   : Bunks: 3
   : Luxury: No
```

5.7.2 Testing the getter functions

```
GetName() - Sleeper
GetLoadFactor() - 1.00
IsSitting() - false
IsAC() - false
GetNumberOfTiers() - 3
IsLuxury() - false
GetReservationCharge() - 20.00
GetTatkalLoadFactor() - 0.30
GetMinTatkalCharge() - 100.00
GetMaxTatkalCharge() - 200.00
GetMinTatkalDistance() - 500
```

5.8 SecondSitting

5.8.1 Testing the output streaming operator

```
Travel Class = Second Sitting
   : Mode: Sitting
   : Comfort: Non-AC
   : Bunks: 0
   : Luxury: No
```

5.8.2 Testing the getter functions

```
GetName() - Second Sitting
GetLoadFactor() - 0.60
IsSitting() - true
IsAC() - false
GetNumberOfTiers() - 0
IsLuxury() - false
GetReservationCharge() - 15.00
GetTatkalLoadFactor() - 0.10
GetMinTatkalCharge() - 10.00
GetMaxTatkalCharge() - 15.00
GetMinTatkalDistance() - 100
```

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

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

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

6.2 Testing `BookingCategory::Ladies`

6.2.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for a female passenger
   Input : Passenger with gender female
   Output : `true`

2. Call the function for a male passenger
   Input : Passenger with gender male
   Output : `false`

6.3 Testing `BookingCategory::SeniorCitizen`

6.3.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for a senior citizen passenger
   Input : Passenger with age = 70
   Output : `true`

2. Call the function for a non senior citizen passenger
   Input : Passenger with age = 19
   Output : `false`

6.4 Testing `Divaang::Blind`

6.4.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for a blind passenger
   Input : Passenger with disability type = blind
   Output : `true`

2. Call the function for a passenger with no disablity
   Input : Passenger with disability type = `NULL`
   Output : `false`

6.5 Testing `Divaang::OrthoHandicapped`

6.5.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for an orthopaedically handicapped passenger
   Input : Passenger with disability type = orthopaedically handicapped
   Output : `true`

2. Call the function for a passenger with no disablity
   Input : Passenger with disability type = `NULL`
   Output : `false`

6.6 Testing `Divaang::Cancer`

6.6.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for a passenger having cancer
   Input : Passenger with disability type = Cancer
   Output : `true`

2. Call the function for a passenger with no disablity
   Input : Passenger with disability type = `NULL`
   Output : `false`

6.7 Testing `Divaang::TB`

6.7.1 Testing the `GetName()` function

1. Call the function and verify the name

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

1. Call the function for a passenger having TB
   Input : Passenger with disability type = TB
   Output : `true`

2. Call the function for a passenger with no disablity
   Input : Passenger with disability type = `NULL`
   Output : `false`

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
   Input : `firstPointer = &GeneralConcession::Type()`
   `secondPointer = &GeneralConcession::Type()`
   Output : On asserting both the pointers should be equal

7.1.2 Testing the `GetFactor()` function

1. Check the value returned by the function
   Input : `GeneralConcession::Type().GetFactor()`
   Output : `0.00`

7.2 Testing the `LadiesConcession` derived class

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

1. Check for singleton behaviour
   Input : `firstPointer = &LadiesConcession::Type()`
   `secondPointer = &LadiesConcession::Type()`
   Output : On asserting both the pointers should be equal

7.2.2 Testing the `GetFactor()` function

1. Check the value returned by the function
   Input : `LadiesConcession::Type().GetFactor()`
   Output : `0.00`

7.3 Testing the `SeniorCitizenConcession` derived class

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

1. Check for singleton behaviour
   Input : `firstPointer = &SeniorCitizenConcession::Type()`
   `secondPointer = &SeniorCitizenConcession::Type()`
   Output : On asserting both the pointers should be equal

7.3.2 Testing the `GetFactor(Passenger&)` function

1. Get the concession factor for a male senior citizen
   Input : Call the function with a male senior citizen as the parameter
   Output : `0.40`

2. Get the concession factor for a female senior citizen
   Input : Call the function with a female senior citizen as the parameter
   Output : `0.50`

7.4 Testing the `DivyaangConcessions` derived class

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

1. Check for singleton behaviour
   Input : `firstPointer = &DivyaangConcessions::Type()`
   `secondPointer = &DivyaangConcessions::Type()`
   Output : On asserting both the pointers should be equal

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.

8 Testing the `Passenger` class

Positive Test Cases

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

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

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

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

Negative Test Cases

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

1. Pass an invalid name - keep first and last name as empty
   Input : `firstName_ = ""`
   `middleName_ = "Kumar"`
   `lastName_ = ""`
   `dateOfBirth_ = "11/03/2002"`
   `gender_ = Male`
   `aadhaar_ = "012345678901"`
   `mobile_ = "9988774567"`
   `disabilityType_ = Blind`
   `disabilityID_ = "012"`
   Output : `At least one of first name or last name should be present`
2. Pass an invaild DOB - in the future
   Input : `firstName_ = "Ashutosh"`
   `middleName_ = "Kumar"`
   `lastName_ = "Singh"`
   `dateOfBirth_ = "11/03/2030"`
   `gender_ = Male`
   `aadhaar_ = "012345678901"`
   `mobile_ = "9988774567"`
   `disabilityType_ = Blind`
   `disabilityID_ = "012"`
   Output : `Date of Birth cannot be in the future`
3. Pass an invaild aadhaar number
   Input : `firstName_ = "Ashutosh"`
   `middleName_ = "Kumar"`
   `lastName_ = "Singh"`
   `dateOfBirth_ = "11/03/2002"`
   `gender_ = Male`

aadhaar_ = "007"
mobile_ = "9988774567"
disabilityType_ = Blind
disabilityID_ = "012"

Output : `Aadhaar No. is not of length 12 :  007`

4. Pass an invalid mobile number

Input : `firstName_ = "Ashutosh"`
middleName_ = "Kumar"
lastName_ = "Singh"
dateOfBirth_ = "11/03/2002"
gender_ = Male
aadhaar_ = "012345678901"
mobile_ = "007"
disabilityType_ = Blind
disabilityID_ = "012"

Output : `Mobile No. is not of length 10 :  007`

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.

1. **Input :** `fromStation_ = "Delhi"`
   `toStation_ = "Mumbai"`
   `gender_ = Male`
   `bookingCategory_ = General`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 1849`

2. **Input :** `fromStation_ = "Kolkata"`
   `toStation_ = "Delhi"`
   `gender_ = Female`
   `bookingCategory_ = Ladies`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 1880`

3. **Input :** `fromStation_ = "Delhi"`
   `toStation_ = "Chennai"`
   `gender_ = Male`
   `bookingCategory_ = Senior Citizen`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 1675`

4. **Input :** `fromStation_ = "Delhi"`
   `toStation_ = "Bangalore"`
   `gender_ = Female`
   `bookingCategory_ = Senior Citizen`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 1384`

5. **Input :** `fromStation_ = "Bangalore"`
   `toStation_ = "Mumbai"`
   `gender_ = Male`
   `bookingCategory_ = Tatkal`
   `dateOfBooking_ = "10/04/2021"`
   Output : `fare_ = 1634`

6. **Input :** `fromStation_ = "Chennai"`
   `toStation_ = "Bangalore"`
   `gender_ = Female`
   `bookingCategory_ = Premium Tatkal`
   `dateOfBooking_ = "10/04/2021"`
   Output : `fare_ = 478`

7. **Input :** `fromStation_ = "Bangalore"`
   `toStation_ = "Kolkata"`
   `gender_ = Male`
   `bookingCategory_ = Blind`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 625`

8. **Input :** `fromStation_ = "Kolkata"`
   `toStation_ = "Mumbai"`
   `gender_ = Male`
   `bookingCategory_ = Cancer`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 40`

9. **Input :** `fromStation_ = "Mumbai"`
   `toStation_ = "Chennai"`
   `gender_ = Male`
   `bookingCategory_ = Orthopaedically Handicapped`
   `dateOfBooking_ = "15/04/2021"`
   Output : `fare_ = 458`

10. **Input :** `fromStation_ = "Chennai"`
    `toStation_ = "Kolkata"`
    `gender_ = Male`
    `bookingCategory_ = TB`
    `dateOfBooking_ = "15/04/2021"`
    Output : `fare_ = 2114`

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`

2. **Input :** `bookingClass_ = ExecutiveChairCar`
   Output : `fare_ = 3678`

3. **Input :** `bookingClass_ = AC2Tier`
   Output : `fare_ = 2944`

4. **Input :** `bookingClass_ = AC3Tier`
   Output : `fare_ = 1849`

5. **Input :** `bookingClass_ = ACChairCar`
   Output : `fare_ = 1487`

6. **Input :** `bookingClass_ = FirstClass`
   Output : `fare_ = 2221`

7. **Input :** `bookingClass_ = Sleeper`
   Output : `fare_ = 744`

8. **Input :** `bookingClass_ = SecondSitting`
   Output : `fare_ = 449`

Application Test Plan

Positive Test Cases

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

1. Booking Category - General

   Output :
   BOOKING SUCCEEDED
   PNR Number = 1
   From Station = Delhi
   To Station = Mumbai
   Travel Date = Thu, 15/Apr/2021
   Travel Class = AC 3 Tier
   : Mode: Sleeping
   : Comfort: AC
   : Bunks: 3
   : Luxury: No
   Booking Category = General
   Passenger Information :
   Name: Ashutosh Kumar Singh
   Date Of Birth: Mon, 11/Mar/2002
   Gender: Male
   Aadhaar: 845626586698
   Mobile: 9999888877

   Reservation Date = Fri, 9/Apr/2021
   Fare = 1849

2. Booking Category - Ladies

   Output :
   BOOKING SUCCEEDED
   PNR Number = 2
   From Station = Kolkata
   To Station = Mumbai
   Travel Date = Thu, 15/Apr/2021
   Travel Class = Executive Chair Car
   : Mode: Sitting
   : Comfort: AC
   : Bunks: 0
   : Luxury: Yes
   Booking Category = Ladies
   Passenger Information :
   Name: Arya Kumari Singh
   Date Of Birth: Sat, 11/Jul/1992

Gender: Female
Aadhaar: 745634695243
Mobile: 9763425843

Reservation Date = Fri, 9/Apr/2021
Fare = 5095

3. Booking Category - Senior Citizen Male

Output :
BOOKING SUCCEEDED
PNR Number = 3
From Station = Bangalore
To Station = Kolkata
Travel Date = Thu, 15/Apr/2021
Travel Class = AC 2 Tier
: Mode: Sleeping
: Comfort: AC
: Bunks: 2
: Luxury: No
Booking Category = Senior Citizen
Passenger Information :
Name: Manku Barnawal
Date Of Birth: Sat, 11/Mar/1950
Gender: Male
Aadhaar: 652147896325
Mobile: 9874562130

Reservation Date = Fri, 9/Apr/2021
Fare = 2295

4. Booking Category - Senior Citizen Female

Output :
BOOKING SUCCEEDED
PNR Number = 4
From Station = Chennai
To Station = Bangalore
Travel Date = Tue, 15/Feb/2022
Travel Class = First Class
: Mode: Sleeping
: Comfort: Non-AC
: Bunks: 2
: Luxury: Yes
Booking Category = Senior Citizen
Passenger Information :

Name: Vashisth Garg
Date Of Birth: Sat, 11/Mar/1961
Gender: Female
Aadhaar: 100011111111
Mobile: 9876543210

Reservation Date = Fri, 9/Apr/2021
Fare = 313

5. Booking Category - Tatkal

Output :
BOOKING SUCCEEDED
PNR Number = 5
From Station = Delhi
To Station = Mumbai
Travel Date = Sat, 10/Apr/2021
Travel Class = Executive Chair Car
: Mode: Sitting
: Comfort: AC
: Bunks: 0
: Luxury: Yes
Booking Category = Tatkal
Passenger Information :
Name: Varun Phogat
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 632547896259
Mobile: 8521479632

Reservation Date = Fri, 9/Apr/2021
Fare = 4178

6. Booking Category - Premium Tatkal

Output :
BOOKING SUCCEEDED
PNR Number = 6
From Station = Delhi
To Station = Kolkata
Travel Date = Sat, 10/Apr/2021
Travel Class = Executive Chair Car
: Mode: Sitting
: Comfort: AC
: Bunks: 0
: Luxury: Yes

Booking Category = Premium Tatkal
Passenger Information :
Name: Harish Vardhan Mundhra
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 100011111111
Mobile: 8761669365
Disability Type: Blind
Disability ID: 0221

Reservation Date = Fri, 9/Apr/2021
Fare = 4740

7. Booking Category - Divyaang - Blind

Output :
BOOKING SUCCEEDED
PNR Number = 7
From Station = Delhi
To Station = Mumbai
Travel Date = Thu, 15/Apr/2021
Travel Class = AC 3 Tier
: Mode: Sleeping
: Comfort: AC
: Bunks: 3
: Luxury: No
Booking Category = Divyaang - Blind
Passenger Information :
Name: Manish Pandey
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 100011111111
Disability Type: Blind
Disability ID: 012

Reservation Date = Fri, 9/Apr/2021
Fare = 492

8. Booking Category - Divyaang - Orthopaedically Handicapped

Output :
BOOKING SUCCEEDED
PNR Number = 8
From Station = Delhi
To Station = Mumbai
Travel Date = Thu, 15/Apr/2021

Travel Class = AC 3 Tier
: Mode: Sleeping
: Comfort: AC
: Bunks: 3
: Luxury: No
Booking Category = Divyaang - Orthopaedically Handicapped
Passenger Information :
Name: Rakesh Chandra
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 100011111111
Mobile: 7456325896
Disability Type: Orthopaedically Handicapped
Disability ID: 007

Reservation Date = Fri, 9/Apr/2021
Fare = 492

9. Booking Category - Divyaang - Cancer

Output :
BOOKING SUCCEEDED
PNR Number = 9
From Station = Delhi
To Station = Mumbai
Travel Date = Thu, 15/Apr/2021
Travel Class = AC 3 Tier
: Mode: Sleeping
: Comfort: AC
: Bunks: 3
: Luxury: No
Booking Category = Divyaang - Cancer
Passenger Information :
Name: Laxman Mittal
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 100011111111
Disability Type: Cancer
Disability ID: 125

Reservation Date = Fri, 9/Apr/2021
Fare = 40

10. Booking Category - Divyaang - TB

Output :

BOOKING SUCCEEDED
PNR Number = 10
From Station = Delhi
To Station = Mumbai
Travel Date = Thu, 15/Apr/2021
Travel Class = AC 3 Tier
: Mode: Sleeping
: Comfort: AC
: Bunks: 3
: Luxury: No
Booking Category = Divyaang - TB
Passenger Information :
Name: Ishan Chandra Pandey
Date Of Birth: Mon, 11/Mar/2002
Gender: Male
Aadhaar: 100011111111
Mobile: 6823457896
Disability Type: TB
Disability ID: 215

Reservation Date = Fri, 9/Apr/2021
Fare = 1849

Negative Test Cases

1. Invalid `fromStation_`
   Input : `fromStation_ = "Dilli"`
   Output : `Station name is invalid : Dilli`
   `Could not create Booking`

2. Invalid `toStation_`
   Input : `toStation_ = "Bombay"`
   Output : `Station name is invalid : Bombay`
   `Could not create Booking`

3. `fromStation_` and `toStation_` are the same
   Input : `fromStation_ = "Delhi"`
   `toStation_ = "Delhi"`
   Output : `From Station and To Station are same, which is not possible`
   `Could not create Booking`

4. `dateOfBooking_` is invalid
   Input : `dateOfBooking_ = 15/04/2500`
   Output : `Year 2500 is not in the valid range`
   `Could not create Booking`

5. Same day booking, i.e., `dateOfBooking_` and `dateOfReservation_` are same
   Input : `dateOfReservation_ = 09/04/2500`
   `dateOfBooking_ = 09/04/2500`

Output : `Same day booking is not allowed`
`Date Of Booking should be later than Date of Reservation`
`Could not create Booking`

6. `dateOfBooking_` is more than one year later than the `dateOfReservation_`
Input : `dateOfReservation_ = 09/04/2500`
`dateOfBooking_ = 15/04/2022`
Output : `Date Of Booking should be within one year of Date of Reservation`
`Could not create Booking`

7. Trying for `Tatkal` but date difference is more than one day
Input : `dateOfReservation_ = 09/04/2500`
`dateOfBooking_ = 30/04/2022`
`bookingCategory_ = BookingCategory::Tatkal`
Output : `For Tatkal or Premium Tatkal, booking should be done 1 day before travel`
`Could not create Booking`

8. Not eligible for the `Divyaang` category
Input : `disabilityType_ = NULL`
`bookingCategory_ = Divyaang::Blind`
Output : `Not eligible for the given Booking Category : Divyaang - Blind`
`Could not create Booking`

9. Not eligible for the `SeniorCitizen` Category
Input : `dateOfBirth_ = 11/03/2002`
`bookingCategory_ = BookingCategory::SeniorCitizen`
Output : `Not eligible for the given Booking Category : Senior Citizen`
`Could not create Booking`

10. Invalid `Passenger` information
Input : `name_` is left totally empty
Output : `Name cannot be completely empty`
`Could not create Passenger`

`Could not create Booking`