TEST PLAN

for

Railway Booking Software

Prepared by -

Suhas Jain (19CS30048)

Assignment - 5

Software Engineering (CS20006)

Indian Institute of Technology, Kharagpur

April 9, 2021

Contents

Unit Test Cases

1 Testing the `Stations` class

Positive test cases

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

1. **Input:** Create an object `Station s1("Kolkata")`
   Golden Output: On checking the name "Kolkata" should be printed.

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

1. **Input:** Passing a valid station name `Station::CreateStation("Kolkata")`
   Golden Output: On checking the name "Kolkata" should be printed

1.3 Testing the `string GetName() const` function

1. **Input:** Calling `GetName` for already constructed object `s1.GetName()`
   Golden Output: On checking the output "Kolkata" should be printed

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

1. **Input:** Calling `GetDistance` for already constructed object `s1.GetDistance(s3)` (where s3 is Bangalore station)
   Golden Output: On checking the output 1871 should be printed

1.5 Testing the `friend ostream& operator<<(ostream&, const Station&)` function

1. **Input:** Calling output streaming operator for already constructed object `cout << s1`
   Golden Output: On checking the name "Station : Kolkata" should be printed

Negative test cases

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

1. **Input:** Passing an empty string to the function : `Station::CreateStation("")`
   Golden Output: An exception should be printed saying "Station name cannot be empty"

2 Testing the `Railways` class

Positive test cases

2.1 Testing the constructor `Railways()` and `static const Railways& IndianRailways()` function

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &Railways::IndianRailways()` and `secondPointer = &Railways::IndianRailways()`
   Golden Output: On asserting both the pointers should be equal

2.2 Testing the `int GetDistance(const Station&, const Station&)` function

1. **Input:** Checking the diatance between any 2 station from both ways: `GetDistance(Station("Bangalore") , Station("Delhi")))` and `GetDistance(Station("Delhi"), Station("Bangalore")))`
Golden Output: Both the distances should be equal to 1871

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

1. **Input:** Store a station in a local pointer by calling : `GetStation("Chennai")`
Golden Output: When printing the name of this station ”Chennai” should be printed

2.4 Testing the `friend ostream& operator<<(ostream&, const Railways&)` function

1. **Input:** Printing `Railways::IndianRailways()` via output streaming operator
Golden Output: A string containing list of all stations and the corrent distances between them should be printed.

Negative test cases

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

1. **Input:** Calling GetStation using string that does not match any of the station names : `GetStation("Bombay")`
Golden Output: An exception should be printed saying ”Station name is invalid : Bombay”

3 Testing the `Date` class

Positive test cases

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

1. **Input:** Creating a new date using the constructor `(04, 12, 2021)`
Golden Output: All the fields of the date d1 should be correct `date = 04, month = Dec, year = 2021`

3.2 Testing the copy constructor `Date(const Date&)`

1. **Input:** Copying a previously constructed date into a new date `d2(d1)`
Golden Output: All the fields of the date d2 should be same as d1 `date = 04, month = Dec, year = 2021`

3.3 Testing the `Day day() const` function

1. **Input:** Calling the function for a already constructed date : `d1.day()`
Golden Output: The output should be equal to the corrent day on that date : ”Sat”

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

1. **Input:** Storing the duration between 2 dates in a local duration object `dur = d4 - d3` (where d3 is 10/4/2021 and d4 is 16/4/2023)
Golden Output: When printing the duration between the objects correct duration should be printed `days = 6, months = 1, years = 2`

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

1. **Input:** Case when it should return true : `d4 > d3`
   Golden Output: The boolean output should be equal to `true`
2. **Input:** Case when it should return false : `d3 > d4`
   Golden Output: The boolean output should be equal to `false`

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

1. **Input:** Case when it should return true : `d1 == d2`
   Golden Output: The boolean output should be equal to `true`
2. **Input:** Case when it should return false : `d3 == d4`
   Golden Output: The boolean output should be equal to `false`

3.7 Testing the `friend ostream& operator<<(ostream&, const Date&)` function

1. **Input:** Printing a date using output streaming operator `cout << d1`
   Golden Output: It should print date in the exact format ”Sat, 4/Dec/2021”

Negative test cases

1. **Input:** Creation of bad date by calling 29th day in a non-leap year : `CreateDate("29/02/2019")`
   Golden Output: An exception should be thrown that should say ”Date is invalid for : 29/02/2019”
2. **Input:** Creation of bad date by calling 29th day in a non-leap year : `CreateDate("29/02/1900")`
   Golden Output: An exception should be thrown that should say ”Date is invalid for : 29/02/1900”
3. **Input:** Creation of bad date by calling 31st day in a June : `CreateDate("31/06/2019")`
   Golden Output: An exception should be thrown that should say ”Date is invalid for : 31/06/2019”
4. **Input:** Creation of bad date by calling the function using wrong syntax : `CreateDate("2902/2020")`
   Golden Output: An exception should be thrown that should say ”Date is invalid for : 2902/2020”
5. **Input:** Creation of bad date by calling the function using wrong syntax : `CreateDate(02/29/2019")`
   Golden Output: An exception should be thrown that should say ”Date is invalid for : 02/29/2019”
6. **Input:** Creation of bad date by creating a date before the year 1900 : `CreateDate("11/03/1899")`
   Golden Output: An exception should be thrown that should say ”Year 1899 is not in the valid range”
7. **Input:** Creation of bad date by creating a date after 2050 : `CreateDate("31/04/2100")`
   Golden Output: An exception should be thrown that should say ”Year 2100 is not in the valid range”

4 Testing the `Name` class

Positive test cases

4.1 When First, Middle and Last name are present

1. **Input:** Call CreateName with appropriate inputs : CreateName(”Daaku”, ”Mangal”, ”Singh”)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "Daaku", Middle Name = "Mangal", Last Name = "Singh"`

4.2 When First and Last name are present

1. **Input:** Call CreateName with appropriate inputs : CreateName(”Daaku”, ””, ”Singh”)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "Daaku", Middle Name = "", Last Name = "Singh"`

4.3 When Last name is present

1. **Input:** Call CreateName with appropriate inputs : CreateName(””, ””, ”Singh”)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "", Middle Name = "", Last Name = "Singh"`

4.4 When First name is present

1. **Input:** Call CreateName with appropriate inputs : CreateName(”Daaku”, ””, ””)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "Daaku", Middle Name = "", Last Name = ""`

4.5 When First and Middle are present

1. **Input:** Call CreateName with appropriate inputs : CreateName(”Daaku”, ”Mangal”, ””)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "Daaku", Middle Name = "Mangal", Last Name = ""`

4.6 When Middle and Last name are present

1. **Input:** Call CreateName with appropriate inputs : CreateName(””, ”Mangal”, ”Singh”)
   Golden Output: Object should be constructed and all 3 strings should get stored in appropriate location : `First name = "", Middle Name = "Mangal", Last Name = "Singh"`

Negative test cases

4.7 When only middle name is present

1. **Input:** Call CreateName with appropriate inputs : CreateName(””, ”Mangal”, ””)
   Golden Output: Object does not get constructed and throws an error that says ”At least one of first name or last name should be present”

4.8 When none of the names are present

1. **Input:** Call CreateName with appropriate inputs : CreateName(””, ””, ””)
   Golden Output: Object does not get constructed and throws an error that says ”Name cannot be completely empty”

5 Testing the `Gender` class and hierarchy

5.1 Testing `Male` derived class made using template GenderTypes¡T¿

Positive test cases

5.1.1 Testing the constructor `GenderTypes(const string& name = GenderTypes<T>::sName)`

1. **Input:**Two new gender pointer should be created : `firstPointer = Gender::Male::Type() and SecondPointer = Gender::Male::Type()`
   Golden Output: On asserting both the pointers should be equal because the class is singleton.

5.1.2 Testing the `const string GetName() const` function

1. **Input:** Calling `GetName()` for the singleton gender object of type `Male`
   Golden Output: It should return the string ”Male”

5.1.3 Testing the `const string GetTitle() const` function

1. **Input:** Calling `GetTitle()` for the singleton gender object of type `Male`
   Golden Output: It should return the string ”Mr.”

5.1.4 Testing the `friend ostream& operator<<(ostream&, const Gender&)` function

1. **Input:** Singleton gender object of type `Male` should be printed using output streaming operator
   Golden Output: ”Male” should be printed

5.2 Testing `Female` derived class made using template GenderTypes¡T¿

Positive test cases

5.2.1 Testing the constructor `GenderTypes(const string& name = GenderTypes<T>::sName)`

1. **Input:**Two new gender pointer should be created : `firstPointer = Gender::Female::Type() and SecondPointer = Gender::Female::Type()`
   Golden Output: On asserting both the pointers should be equal because the class is singleton.

5.2.2 Testing the `const string GetName() const` function

1. **Input:** Calling `GetName()` for the singleton gender object of type `Female`
   Golden Output: It should return the string ”Female”

5.2.3 Testing the `const string GetTitle() const` function

1. **Input:** Calling `GetTitle()` for the singleton gender object of type `Female`
   Golden Output: It should return the string ”Ms.”

5.2.4 Testing the `friend ostream& operator<<(ostream&, const Gender&)` function

1. **Input:** Singleton gender object of type `Female` should be printed using output streaming operator
   Golden Output: ”Female” should be printed

6 Testing the `Passenger` class

Positive test cases

6.1 Testing the `static Passenger& CreatePassenger(const Name, const Date, const Gender&, const string&, const string&, const Divyaang&, const string&)` function

1. **Input:** Create a sample object using the function : `p1 = Passenger::CreatePassenger("Daaku", "Mangal", "Singh", "11/03/2002", Gender::Male::Type(), "012345678901", Divyaang::Blind::Type(), "012", "9988774567")`
   Golden Output: The object should be constructed without errors and on asseting the values of all the attributes of the passenger object `p1` all the attributes must match :

   ```
   assert(Date(11, 03, 2002) == p1 -> dateOfBirth)
   assert(Gender::IsMale(p1 -> gender))
   assert("012345678901" == p1 -> aadhaar)
   assert("9988774567" == p1 -> mobile)
   assert(Divyaang::Blind::Type() == p1 -> disabilityType)
   assert("012" == p1 -> disabilityID)
   ```

6.2 Testing the `const Date GetDateOfBirth() const` function

1. **Input:** Calling `GetDateofBirth()` for p1
   Golden Output: Should return `Date(11, 03, 2002)`

6.3 Testing the `const Gender& GetGender() const` function

1. **Input:** Calling `GetGender()` for p1
   Golden Output: Should return `Gender::Male` singleton

6.4 Testing the `const Divyaang& GetDisability() const` function

1. **Input:** Calling `GetDisability()` for p1
   Golden Output: Should return `Divyaang::Blind::Type()` singleton

Negative test cases

1. **Input:** Constructing a passenger without first or last name
   Golden Output: Exception should be printed with message ”Atleast first or last name should be present”

2. **Input:** Constructing a passenger with dob greater than current date
   Golden Output: Exception should be printed with message ”DOB is invalid”

3. **Input:** Constructing a passenger with non-10 digit mobile number
   Golden Output: Exception should be printed with message ”Mobile No. is not of length 10”

4. **Input:** Constructing passenger with non-12 digit aadhaar number
   Golden Output: Exception should be printed with message ”Aadhaar No. is not of length 12 ”

7 Testing the `BookingClass` class and hierarchy

7.1 Testing `ACFirstClass` the derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.1.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &BookingClass::ACFirstClass ::Type() and secondPointer = &BookingClass::ACFirstClass::Type()`
   Golden Output: On asserting both the pointers should be equal

7.1.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.1.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.1.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.1.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `6.50`

7.1.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `2`

7.1.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `60.00`

7.1.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.1.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `400.00`

7.1.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500.00`

7.1.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500`

7.2 Testing the `ExecutiveChairCar` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.2.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::ExecutiveChairCar::Type() and secondPointer = &BookingClass::ExecutiveChairCar::Type()`
   Golden Output: On asserting both the pointers should be equal

7.2.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.2.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.2.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.2.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `5.00`

7.2.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0`

7.2.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `60.00`

7.2.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.2.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `400.00`

7.2.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500.00`

7.2.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `250`

7.3 Testing the `AC2Tier` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.3.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &BookingClass::AC2Tier ::Type() and secondPointer = &BookingClass::AC2Tier::Type()`
   Golden Output: On asserting both the pointers should be equal

7.3.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.3.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.3.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.3.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `4.00`

7.3.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `2`

7.3.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `50.00`

7.3.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.3.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `400.00`

7.3.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500.00`

7.3.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500`

7.4 Testing the `FirstClass` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.4.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::FirstClass::Type(` and `secondPointer = &BookingClass::FirstClass::Type()`
   Golden Output: On asserting both the pointers should be equal

7.4.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.4.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.4.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.4.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `3.00`

7.4.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `2`

7.4.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `50.00`

7.4.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.4.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `400.00`

7.4.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500.00`

7.4.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500`

7.5 Testing the `AC3Tier` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.5.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::AC3Tier::Type()` and `secondPointer = &BookingClass::AC3Tier::Type()`
   Golden Output: On asserting both the pointers should be equal

7.5.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.5.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.5.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.5.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `6.50`

7.5.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `3`

7.5.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `40.00`

7.5.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.5.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `300.00`

7.5.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `400.00`

7.5.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500`

7.6 Testing the `ACChairCar` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.6.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::ACChairCar::Type() and secondPointer = &BookingClass::ACChairCar::Type()`
   Golden Output: On asserting both the pointers should be equal

7.6.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.6.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.6.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.6.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `2.00`

7.6.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0`

7.6.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `40.00`

7.6.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.6.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `125.00`

7.6.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `225.00`

7.6.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `250`

7.7 Testing the `Sleeper` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.7.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::Sleeper::Type()` and `secondPointer = &BookingClass::Sleeper::Type()`
   Golden Output: On asserting both the pointers should be equal

7.7.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.7.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.7.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.7.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `1.00`

7.7.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `3`

7.7.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `20.00`

7.7.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.30`

7.7.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `100.00`

7.7.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `100.00`

7.7.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `500`

7.8 Testing the `SecondSitting` derived class modelled using the template `BookingClassType<T>`

Positive test cases

7.8.1 Testing the constructor `BookingClassType(const string& name = BookingClassType<T>::sName)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = BookingClass::SecondSitting::Type() and secondPointer = &BookingClass::SecondSitting::Type()`
   Golden Output: On asserting both the pointers should be equal

7.8.2 Testing the `bool IsAC() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.8.3 Testing the `bool IsLuxury() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `false`

7.8.4 Testing the `bool IsSitting() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `true`

7.8.5 Testing the `double GetLoadFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.60`

7.8.6 Testing the `int GetNumberOfTiers() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0`

7.8.7 Testing the `double GetReservationCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `15.00`

7.8.8 Testing the `double GetTatkalFactor() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0.10`

7.8.9 Testing the `double GetTatkalMinCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `10.00`

7.8.10 Testing the `double GetTatkalMaxCharge() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `15.00`

7.8.11 Testing the `int GetMinTatkalDistance() const` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `100`

8 Testing the `BookingCategory` class and hierarchy

First we check GetName for each derived class and then make an passenger object of each category type and call isEligible for that object

8.1 Testing the `General` derived class modelled by the template `BookingCategoryType<T>`

Positive test cases

8.1.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "General"

8.1.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

8.2 Testing the `Ladies` derived class modelled by the template `BookingCategoryType<T>`

Positive test cases

8.2.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "Ladies"

8.2.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and gender `Female`, and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

Negative test cases

8.2.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and gender `Male` and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `false`.

8.3 Testing the `SeniorCitizen` derived class modelled by the template `BookingCategoryType<T>`

Positive test cases

8.3.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "Senior Citizen"

8.3.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and age 70 (more than threshold), and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

Negative test cases

8.3.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and gender age 20 (less than threshold) and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `false`.

8.4 Testing the `Tatkal` derived class modelled by the template `BookingCategoryType<T>`

Positive test cases

8.4.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "Tatkal"

8.4.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

8.5 Testing the `PremiumTatkal` derived class modelled by the template `BookingCategoryType<T>`

Positive test cases

8.5.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "Premium Tatkal"

8.5.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

9 Testing the `Divyaang` class and hierarchy

9.1 Testing the Blind derived class modelled by the template `DivyaangType<T>`

Positive test cases

9.1.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be "Divyaang - Blind"

9.1.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and divyaang type as `Blind` with a non-empty disability id, and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

Negative test cases

9.1.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and NULL in place of divyaang type with a empty disability id, and calling IsEligible for pointer of that object
Golden Output: The boolean value returned should be `false`.

9.2 Testing the OrthoHandicapped derived class modelled by the template `DivyaangType<T>`

Positive test cases

9.2.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
Golden Output: The returned string should be ”Divyaang - Orthopedically Handicapped”

9.2.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and divyaang type as `OrthoHandicapped` with a non-empty disability id, and calling IsEligible for pointer of that object
Golden Output: The boolean value returned should be `true`.

Negative test cases

9.2.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and NULL in place of divyaang type with a empty disability id, and calling IsEligible for pointer of that object
Golden Output: The boolean value returned should be `false`.

9.3 Testing the Cancer derived class modelled by the template `DivyaangType<T>`

Positive test cases

9.3.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
Golden Output: The returned string should be ”Divyaang - Cancer”

9.3.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and divyaang type as `Cancer` with a non-empty disability id, and calling IsEligible for pointer of that object
Golden Output: The boolean value returned should be `true`.

Negative test cases

9.3.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and NULL in place of divyaang type with a empty disability id, and calling IsEligible for pointer of that object
Golden Output: The boolean value returned should be `false`.

9.4 Testing the TB derived class modelled by the template `DivyaangType<T>`

Positive test cases

9.4.1 Testing the `GetName()` function

1. **Input:** Calling the `GetName()` function for the static object
   Golden Output: The returned string should be ”Divyaang - TB”

9.4.2 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and divyaang type as `TB` with a non-empty disability id, and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `true`.

Negative test cases

9.4.3 Testing the `bool IsEligible(Passenger&) const` function

1. **Input:** Creating an passenger object with all the valid parameters and NULL in place of divyaang type with a empty disability id, and calling IsEligible for pointer of that object
   Golden Output: The boolean value returned should be `false`.

10 Testing the `Concessions` class and hierarchy

10.1 Testing the `GeneralConcession` derived class

Positive test cases

10.1.1 Testing the constructor `GeneralConcession(string&)`

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &GeneralConcession::Type()` and `secondPointer = &GeneralConcession::Type()`
   Golden Output: On asserting both the pointers should be equal

10.1.2 Testing the `double GetFactor()` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0`

10.2 Testing the `LadiesConcession` derived class

Positive test cases

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &LadiesConcession::Type()` and `secondPointer = &LadiesConcession::Type()`
   Golden Output: On asserting both the pointers should be equal

10.2.1 Testing the `double GetFactor()` function

1. **Input:** On calling for the singleton object should return the correct value
   Golden Output: The value should be `0`

10.3 Testing the `SeniorCitizenConcession` derived class

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &SeniorCitizenConcession::Type()` and `secondPointer = &SeniorCitizenConcession::Type()`
   Golden Output: On asserting both the pointers should be equal

10.3.1 Testing the `double GetFactor()` function

1. **Input:** On calling for the singleton object for male passenger should return the correct value
   Golden Output: The value should be `0.40`

2. **Input:** On calling for the singleton object for female passenger should return the correct value
   Golden Output: The value should be `0.50`

10.4 Testing the `DivyaangConcessions` derived class

Positive test cases

10.5 Testing the constructor and singleton behavior

1. **Input:** Checking the singleton creation via constructor: `firstPointer = &DivyaangConcession::Type()` and `secondPointer = &DivyaangConcession::Type()`
   Golden Output: On asserting both the pointers should be equal

10.5.1 Testing the `double GetFactor()` function

For Diyaang of type Blind

1. **Input:** On calling for the singleton object for booking class AC First Class should return the correct value
   Golden Output: The value should be `0.50`

2. **Input:** On calling for the singleton object for booking class Executive Chair Car should return the correct value
   Golden Output: The value should be `0.75`

3. **Input:** On calling for the singleton object for booking class AC 2 Tier should return the correct value
   Golden Output: The value should be `0.50`

4. **Input:** On calling for the singleton object for booking class First Class should return the correct value
   Golden Output: The value should be `0.75`

5. **Input:** On calling for the singleton object for booking class AC 3 Tier should return the correct value
   Golden Output: The value should be `0.75`

6. **Input:** On calling for the singleton object for booking class AC Chair Car should return the correct value
   Golden Output: The value should be `0.75`

7. **Input:** On calling for the singleton object for booking class Sleeper should return the correct value
   Golden Output: The value should be `0.75`

8. **Input:** On calling for the singleton object for booking class Second Sitting should return the correct value
   Golden Output: The value should be `0.75`

For Diyaang of type Orthopedically Handicapped

1. **Input:** On calling for the singleton object for booking class AC First Class should return the correct value
   Golden Output: The value should be `0.50`
2. **Input:** On calling for the singleton object for booking class Executive Chair Car should return the correct value
   Golden Output: The value should be `0.75`
3. **Input:** On calling for the singleton object for booking class AC 2 Tier should return the correct value
   Golden Output: The value should be `0.50`
4. **Input:** On calling for the singleton object for booking class First Class should return the correct value
   Golden Output: The value should be `0.75`
5. **Input:** On calling for the singleton object for booking class AC 3 Tier should return the correct value
   Golden Output: The value should be `0.75`
6. **Input:** On calling for the singleton object for booking class AC Chair Car should return the correct value
   Golden Output: The value should be `0.75`
7. **Input:** On calling for the singleton object for booking class Sleeper should return the correct value
   Golden Output: The value should be `0.75`
8. **Input:** On calling for the singleton object for booking class Second Sitting should return the correct value
   Golden Output: The value should be `0.75`

For Diyaang of type Cancer Patient

1. **Input:** On calling for the singleton object for booking class AC First Class should return the correct value
   Golden Output: The value should be `0.50`
2. **Input:** On calling for the singleton object for booking class Executive Chair Car should return the correct value
   Golden Output: The value should be `0.75`
3. **Input:** On calling for the singleton object for booking class AC 2 Tier should return the correct value
   Golden Output: The value should be `0.50`
4. **Input:** On calling for the singleton object for booking class First Class should return the correct value
   Golden Output: The value should be `0.75`

5. **Input:** On calling for the singleton object for booking class AC 3 Tier should return the correct value
   Golden Output: The value should be `1.00`
6. **Input:** On calling for the singleton object for booking class AC Chair Car should return the correct value
   Golden Output: The value should be `1.00`
7. **Input:** On calling for the singleton object for booking class Sleeper should return the correct value
   Golden Output: The value should be `1.00`
8. **Input:** On calling for the singleton object for booking class Second Sitting should return the correct value
   Golden Output: The value should be `1.00`

For Diyaang of type TB Patient

1. **Input:** On calling for the singleton object for booking class AC First Class should return the correct value
   Golden Output: The value should be `0.00`
2. **Input:** On calling for the singleton object for booking class Executive Chair Car should return the correct value
   Golden Output: The value should be `0.00`
3. **Input:** On calling for the singleton object for booking class AC 2 Tier should return the correct value
   Golden Output: The value should be `0.00`
4. **Input:** On calling for the singleton object for booking class First Class should return the correct value
   Golden Output: The value should be `0.75`
5. **Input:** On calling for the singleton object for booking class AC 3 Tier should return the correct value
   Golden Output: The value should be `0.00`
6. **Input:** On calling for the singleton object for booking class AC Chair Car should return the correct value
   Golden Output: The value should be `0.00`
7. **Input:** On calling for the singleton object for booking class Sleeper should return the correct value
   Golden Output: The value should be `0.75`
8. **Input:** On calling for the singleton object for booking class Second Sitting should return the correct value
   Golden Output: The value should be `0.75`

11 Testing the `Booking` class and hierarchy

Positive test cases

Here instead of testing individual functions we book for various circumstances and check if the fare and all the other details of the booking are printed correctly

1. **Input:** On calling ReserveBooking between Delhi and Mumbai for AC 3 Tier and General category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1849`
2. **Input:** On calling ReserveBooking between Kolkata and Delhi for AC 3 Tier and Ladies category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1880`
3. **Input:** On calling ReserveBooking between Delhi and Chennai for AC 3 Tier and Senior Citizen Male category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1675`
4. **Input:** On calling ReserveBooking between Delhi and Bangalore for AC 3 Tier and Senior Citizen Female category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1384`
5. **Input:** On calling ReserveBooking between Bangalore and Mumbai for AC 3 Tier and Tatkal category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1634`
6. **Input:** On calling ReserveBooking between Chennai and Bangalore for AC 3 Tier and Premium Tatkal category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 478`
7. **Input:** On calling ReserveBooking between Bangalore and Chennai for AC 3 Tier and Divyaang Blind category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 149`
8. **Input:** On calling ReserveBooking between Kolkata and Mumbai for AC 3 Tier and Divyaang Ortho Handicapped category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 40`
9. **Input:** On calling ReserveBooking between Mumbai and Chennai for AC 3 Tier and Divyaang Cancer Patient category
   Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 458`
10. **Input:** On calling ReserveBooking between Chennai and Kolkata for AC 3 Tier and Divyaang TB Patient category
    Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 2114`
11. **Input:** On calling ReserveBooking between Delhi and Mumbai for AC First Class and General category
    Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 4763`

12. **Input:** On calling ReserveBooking between Kolkata and Mumbai for AC 2 Tier and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 4078`

13. **Input:** On calling ReserveBooking between Kolkata and Chennai for AC 3 Tier and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 2114`

14. **Input:** On calling ReserveBooking between Delhi and Bangalore for AC Chair Car and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 2190`

15. **Input:** On calling ReserveBooking between Bangalore and Mumbai for First Class and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 1522`

16. **Input:** On calling ReserveBooking between Bangalore and Kolkata for Sleeper and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 956`

17. **Input:** On calling ReserveBooking between Chennai and Kolkata for Second Sitting and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 513`

18. **Input:** On calling ReserveBooking between Kolkata and Mumbai for Executive Chair Car and General category
Golden Output: Correct value of fare and other booking attributes should be printed fare = `Rs. 5095`

Application Test Cases

12 When booking is done in General

1. **Input:** Doing booking for a General passenger in General Booking Category
   Golden Output: Object should be constructed with the following output when printed

```
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:  Daaku Mangal Singh
Date Of Birth:  Mon, 11/Mar/2002
Gender:  Male
Aadhaar:  100011111111

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

13 When booking is done in Ladies

1. **Input:** Doing booking for a Ladies passenger in Ladies Booking Category
   Golden Output: Object should be constructed with the following output when printed

```
BOOKING SUCCEEDED
PNR Number = 2
From Station = Delhi
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:  Daaki Mangali Singh
Date Of Birth:  Sat, 11/Jul/1992
Gender:  Female
Aadhaar:  100011111111

Reservation Date = Fri, 9/Apr/2021
Fare = 3678
```

14 When booking is done in Senior Citizen (Male)

1. **Input:** Doing booking for a Male passenger with age 60+ in Senior Citizen Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 3
   From Station = Delhi
   To Station = Mumbai
   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:  Daaku Mangal Singh
   Date Of Birth:  Sat, 11/Mar/1950
   Gender:  Male
   Aadhaar:  100011111111

   Reservation Date = Fri, 9/Apr/202
   1 Fare = 1786
   ```

15 When booking is done in Senior Citizen (Female)

1. **Input:** Doing booking for a Female passenger with age 58+ in Senior Citizen Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 4
   From Station = Mumbai
   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:  Daaki Mangali Singh
   Date Of Birth:  Sat, 11/Mar/1961
   Gender:  Female
   Aadhaar:  100011111111

   Reservation Date = Fri, 9/Apr/2021
   Fare = 786
   ```

16 When booking is done in Tatkal (for General person)

1. **Input:** Doing booking for a General passenger with Tatkal Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   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:  Daaku Mangal Singh
   Date Of Birth:  Mon, 11/Mar/2002
   Gender:  Male
   Aadhaar:  100011111111

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

17 When booking is done in Premium Tatkal (for General person)

1. **Input:** Doing booking for a General passenger with Tatkal Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 6
   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 = Premium Tatkal
   Passenger Information =
   Name:  Daaku Mangal Singh
   Date Of Birth:  Mon, 11/Mar/2002
   Gender:  Male
   Aadhaar:  100011111111

   Reservation Date = Fri, 9/Apr/2021
   Fare = 4678
   ```

18 When booking is done in Tatkal (for person who can avail concession)

1. This test case is to show that with the Tatkal class other types of concessions cannot be made
   Input: Doing booking for a General passenger with Tatkal Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 7
   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:  Daaku Mangal Singh
   Date Of Birth:  Mon, 11/Mar/2002
   Gender:  Male
   Aadhaar:  100011111111
   Disability Type:  Blind
   Disability ID: 0221

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

19 When booking is done in Premium Tatkal (for person who can avail concession)

1. This test case is to show that with the Premium Tatkal class other types of concessions cannot be made
   Input: Doing booking for a General passenger with Tatkal Booking Category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 8
   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 = Premium Tatkal
   Passenger Information =
   Name:  Daaku Mangal Singh
   Date Of Birth:  Mon, 11/Mar/2002
   Gender:  Male
   Aadhaar:  100011111111
   Disability Type:  Blind
   ```

```
Disability ID: 0221

Reservation Date = Fri, 9/Apr/2021
Fare = 4678
```

20 When booking is done in Divyaang of type Blind

1. **Input:** Doing booking for a Divyaang - Blind for Divyaang Blind booking category
   Golden Output: Object should be constructed with the following output when printed
```
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 - Blind
Passenger Information =
Name:  Daaku Mangal Singh
Date Of Birth:  Mon, 11/Mar/2002
Gender:  Male
Aadhaar:  100011111111
Disability Type:  Blind
Disability ID: 0221

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

21 When booking is done in Divyaang of type Orthopedically Handicapped

1. **Input:** Doing booking for a Divyaang - Orthopedically Handicapped for Divyaang OrthoHandicapped booking category
   Golden Output: Object should be constructed with the following output when printed
```
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 - Orthopaedically Handicapped
Passenger Information =
Name:  Daaku Mangal Singh
```

```
Date Of Birth:  Mon, 11/Mar/2002
Gender:  Male
Aadhaar:  100011111111
Disability Type:  Orthopaedically Handicapped
Disability ID: 0221

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

22 When booking is done in Divyaang of type Cancer

1. **Input:** Doing booking for a Divyaang - Cancer for Divyaang Cancer booking category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 11
   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:  Daaku Mangal Singh
   Date Of Birth:  Mon, 11/Mar/2002
   Gender:  Male
   Aadhaar:  100011111111
   Disability Type:  Cancer
   Disability ID: 0221
   Reservation Date = Fri, 9/Apr/2021
   Fare = 40
   ```

23 When booking is done in Divyaang of type TB

1. **Input:** Doing booking for a Divyaang - TB for Divyaang TB booking category
   Golden Output: Object should be constructed with the following output when printed
   ```
   BOOKING SUCCEEDED
   PNR Number = 12
   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:  Daaku Mangal Singh
Date Of Birth:  Mon, 11/Mar/2002
Gender:  Male
Aadhaar:  100011111111
Disability Type:  TB
Disability ID: 0221

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

Negative Application Test Cases

24 When the source station is misspelled

1. **Input:** When the source station is spelled as "Dilli" which does not exist
   Golden Output: Station name is invalid : Dilli
   Could not create Booking

25 When the destination station is misspelled

1. **Input:** When the destination station is spelled as "Bombay" which does not exist
   Golden Output: Station name is invalid : Bombay
   Could not create Booking

26 When the source station and destination are the same

1. **Input:** When the source station and the destination station both are "Delhi"
   Golden Output: Source and destination stations cannot be same
   Could not create Booking

27 When the date of booking is out of range of guidelines

1. **Input:** When the year of reservation date is 2500
   Golden Output: Year 2500 is not in the valid range
   Could not create Booking

28 When date of booking and reservation are the same

1. **Input:** When the booking is made for "09/04/2021" (ie. the day the code is run)
   Golden Output: Booking on the same day is not allowed
   Could not create Booking

29 Date of booking cannot be beyond 1 year from date of reservation

1. **Input:** When the booking is made for "15/04/2022" (ie. the day more than 1 year from when the code was run)

Golden Output: Date Of Booking cannot be beyond 1 year from Date of Reservation
Could not create Booking

30 When the person is not eligible for the booking category applied (Divyaang)

1. **Input:** A non-blind person applies for booking category - Blind (Divyaang)
Golden Output: Passenger is not eligible for the booking category : Divyaang - Blind
Could not create Booking

31 When the person is not eligible for the booking category applied (Senior Citizen)

1. **Input:** When a male with age less than 60 applies for booking category - SeniorCitizen
Golden Output: Passenger is not eligible for the booking category : Senior Citizen
Could not create Booking