TEST PLAN AND REPORT

Course: SOFTWARE ENGINEERING (CS20006)

Indian Institute of Technology Kharagpur

West Bengal, India, Pin-721302

Name: Parth Tusham

Roll No.: 19CS30034

Contents

Unit Testing,Application testing,and exception handling of Classes

Date

We have overloaded many operators and there are also many getter functions we have used.These are tested in unit testing. The following functions are being tested

1. Greater than operator (overloaded)
2. Overloaded difference operator
3. Overloaded assignment and equate operator
4. Also name and no. of day month is also being tested etc

The following function is being used for testing for example
Date date = Date::createDate("11/02/2001");

```
void DateUnitTest() {

Date date = Date::createDate("11/02/2001");

assert(date.GiveDate() == 11);
assert(date.GiveMonth() == "Feb");
assert(date.GiveMonthNo() == 2);
assert(date.GiveYear() == 2001);

}
```

Here we check the different invalid dates and date formats

- Input String that is sent to create date must be of correct format otherwise error is detected
- Validity: Various invalid dates examples can be
    - 1/01/2030 gives error
    - 11/01/1830 gives error
    - 35/01/2010 gives error
    - 29/02/2001 gives error
    - 11/01/2010 gives correct value

BookingCategories

The booking categories class is a simple hierarchy to provide concession factors according to the category of the passenger. It contains the following child classes

1. General
2. SeniorCitizen
3. Ladies
4. Tatkal
5. PremiumTatkal
6. Divyaang

And other disabilities classes are provided through divyaang abstract class which contains the following classes

1. TB
2. Orthopedic
3. Cancer
4. Blind

Unit tests of this are done by creating every type of booking category and then is checked and asserted for correctness using getter functions.
The following is used for testing

1. Get Names of classes some examples
    a. Blind::Type().GetName() -> Output: Blind
    b. Male::Type().GetName() -> Output: Male
2. Check DivyaangConcessions class
    a. Blind, ACFirstClass ->Output: 0.50
    b. Cancer Patient, ACChairCar -> Output: 1
    c. TB Patient, Sleeper -> Output: 0.75

Few of the exception are we have handled are:
We have assumed that this class is error free according to the problem statement

Passenger

This is the class representing the traveller in which we are testing many cases like valid aadhaar, no surname or first name ,invlaid dob etc

Input:

Name->"Rahul" “” "Sharma"
aadhaar->1234522344"
DOB->"04/04/2002

Gender->Male
Mobile number ->6261171632
Categoryid->1
Disability ID->3
Disability ID->Divyaang

Golden output:
Name->"Rahul" “” "Sharma"
aadhaar->1234522344"
DOB->"04/04/2002
Gender->Male
Mobile number ->6261171632
Categoryid->1
Disability ID->3
Disability ID->Divyaang

The following function is used for testing

```
const Passenger p4 = Passenger::createPassenger("Rahul", "", "Sharma", "1234522344",
"04/04/2002", Gender::Male::Type(), "6261171632", 1, 3, "Divyaang") ;
    assert(p4.GetName() == "Rahul");
    assert(p4.GetAadharNo() == "1234522344" );
    assert(d1 == d2);
    assert(p4.GetGender() == "Male");
    assert(p4.GetMobileNumber() == "6261171632");
    assert(p4.GetCategory() == "Senior Citizen");
```

Few of the exception are we have handled are:

1. More or less digits are provided for aadhaar or mobile
2. Dob is of wrong format
3. First or last name is not provided
4. Disability id is not matching with disability type

Various exceptions are to be handled in this class.

a) Exception because of Aadhaar number.

i) Aadhaar number(given as string) = “12231456892”(11 digits)
Output: “Bad_Passenger” Exception is thrown

ii) Aadhaar Number = “100213564a562”
Output: “Bad_Passenger” Exception is thrown

iii) Aadhaar Number = “112456890012”
Output: No error because of aadhaar number

b) Exceptions because of Mobile number.

i) Mobile number(given as string) = “125588794”(9 digits)

Output: “Bad_Passenger” Exception is thrown

ii) Mobile number = “456415a598”
Output: “Bad_Passenger” Exception is thrown

iii) Mobile number = “”
Output: No error because of mobile number

iv) Mobile Number = “8688664667”
Output: No error because of mobile number

c) Exceptions because of date of birth.

i) Date of birth = 29/2/2021
Output: “Bad_Date” exception is thrown

ii) Date of birth = “28/3/2025”
Output: “Bad_Passenger” exception is thrown

iii) Date of birth = “30/8/2002”
Output: No error because of date of birth

d) Exceptions because of diyaangID
i) Disability = Blind, divyaangID = “”(left empty)
Output = “Bad_Passenger” exception is thrown

BookingClasses

The booking class now contains a simple hierarchy with put any intermediate abstract classes and every child class is a concrete leaf class overriding the functions of the parent base class.for unit testing these classes we are simply creating an instance of the child class and asserting it with getter functions to Verify various factors of each booking class Like Name, Reservation charge, IsAC() etc

● Testing of AcFirstClass class:
We make a function to check the equality of the following:
1. IsLuxury() should return true.
2. IsAc() should return true.
3. GetLoadFactor() should return 6.50.
4. GetName() should return “Ac 2Tier”.
5. IsSitting() should return false.
6. GetMinTatkalCharge() should returns 400.
7. GetMaxTatkalCharge() should returns 500.
8. GetReservationCharge() should returns 60.00.

9. GetMinTatkalDistance() should returns 500.
10. GetTatkalLoadFactor() should returns 0.30.
11. GetNumberOfTiers() should return 2.

The following function was used for Unit Testing:

```
const BookingClasses& ACFirstClass_ =
BookingClasses::ACFirstClass::Type() ;
assert(ACFirstClass_.GetName() == "AC First Class") ;
assert(ACFirstClass_.GetLoadFactor() == 6.50) ;
assert(ACFirstClass_.IsAC() == true) ;
assert(ACFirstClass_.IsLuxury() == true) ;
assert(ACFirstClass_.IsSitting() == false) ;
assert(ACFirstClass_.GetNumberOfTiers() == 2) ;
assert(ACFirstClass_.GetMinTatkalCharge() == 400) ;
assert(ACFirstClass_.GetMaxTatkalCharge() == 500) ;
assert(ACFirstClass_.GetReservationCharge() == 60.00) ;
assert(ACFirstClass_.GetMinTatkalDistance() == 500) ;
assert(ACFirstClass_.GetTatkalLoadFactor() == 0.3)  ;
```

- Testing of Ac2Tier class:

We make a function to check the equality of the following:

1. IsLuxury() should return true.
2. IsAc() should return false.
3. GetLoadFactor() should return 4.00.
4. GetName() should return “Ac 2Tier”.
5. IsSitting() should return false.
6. GetMinTatkalCharge() should returns 400.
7. GetMaxTatkalCharge() should returns 500.
8. GetReservationCharge() should returns 50.00.
9. GetMinTatkalDistance() should returns 500.
10. GetTatkalLoadFactor() should returns 0.30.
11. GetNumberOfTiers() should return 2.

The following function was used for Unit Testing:

```
const BookingClasses& AC2Tier_ = BookingClasses::AC2Tier::Type() ;
assert(AC2Tier_.GetName() == "AC 2 Tier") ;
assert(AC2Tier_.GetLoadFactor() == 4.00) ;
assert(AC2Tier_.IsAC() == true) ;
assert(AC2Tier_.IsLuxury() == true) ;
assert(AC2Tier_.IsSitting() == false) ;
assert(AC2Tier_.GetNumberOfTiers() == 2) ;
assert(AC2Tier_.GetMinTatkalCharge() == 400) ;
assert(AC2Tier_.GetMaxTatkalCharge() == 500) ;
assert(AC2Tier_.GetReservationCharge() == 40.00) ;
assert(AC2Tier_.GetMinTatkalDistance() == 500) ;
```

```
assert(AC2Tier_.GetTatkalLoadFactor() == 0.3)  ;
```

- Testing of FirstClass class:

  We make a function to check the equality of the following:

  1. IsLuxury() should return true.
  2. IsAc() should return false.
  3. GetLoadFactor() should return 3.00.
  4. GetName() should return “First Class”.
  5. IsSitting() should return false.
  6. GetMinTatkalCharge() should returns 400.
  7. GetMaxTatkalCharge() should returns 500.
  8. GetReservationCharge() should returns 50.00.
  9. GetMinTatkalDistance() should returns 500.
  10. GetTatkalLoadFactor() should returns 0.30.
  11. GetNumberOfTiers() should return 3.

  The following function was used for Unit Testing:

```
const BookingClasses& FirstClass_ = BookingClasses::FirstClass::Type() ;
assert(FirstClass_.GetName() == "First Class") ;
assert(FirstClass_.GetLoadFactor() == 3.50) ;
assert(FirstClass_.IsAC() == false) ;
assert(FirstClass_.IsLuxury() == true) ;
assert(FirstClass_.IsSitting() == false) ;
assert(FirstClass_.GetNumberOfTiers() == 2) ;
assert(FirstClass_.GetMinTatkalCharge() == 400) ;
assert(FirstClass_.GetMaxTatkalCharge() == 500) ;
assert(FirstClass_.GetReservationCharge() == 50.00) ;
assert(FirstClass_.GetMinTatkalDistance() == 500) ;
assert(FirstClass_.GetTatkalLoadFactor() == 0.3)  ;
```

- Testing of Ac3Tier class:

  We make a function to check the equality of the following:

  1. IsLuxury() should return false.
  2. IsAc() should return true.
  3. GetLoadFactor() should return 1.00.
  4. GetName() should return “ Ac3Tier ”.
  5. IsSitting() should return false.
  6. GetMinTatkalCharge() should returns 300.
  7. GetMaxTatkalCharge() should returns 400.
  8. GetReservationCharge() should returns 40.00.
  9. GetMinTatkalDistance() should returns 500.
  10. GetTatkalLoadFactor() should returns 0.30.
  11. GetNumberOfTiers() should return 3.

  The following function was used for Unit Testing:

```
const BookingClasses& AC3Tier_ = BookingClasses::AC3Tier::Type() ;
assert(AC3Tier_.GetName() == "AC 3 Tier") ;
assert(AC3Tier_.GetLoadFactor() == 2.50) ;
assert(AC3Tier_.IsAC() == true) ;
assert(AC3Tier_.IsLuxury() == false) ;
assert(AC3Tier_.IsSitting() == false) ;
assert(AC3Tier_.GetNumberOfTiers() == 3) ;
assert(AC3Tier_.GetMinTatkalCharge() == 300) ;
assert(AC3Tier_.GetMaxTatkalCharge() == 400) ;
assert(AC3Tier_.GetReservationCharge() == 40.00) ;
assert(AC3Tier_.GetMinTatkalDistance() == 500) ;
assert(AC3Tier_.GetTatkalLoadFactor() == 0.3)  ;
```

- Testing of ACChairCar class:

  We make a function to check the equality of the following:

  1. IsLuxury() should return false.
  2. IsAc() should return true.
  3. GetLoadFactor() should return 2.00.
  4. GetName() should return “AC Chair Car ”.
  5. GetNumberOfTiers() should return 0.
  6. IsSitting() should return true.
  7. GetMinTatkalCharge() should returns 125.
  8. GetMaxTatkalCharge() should returns 225.
  9. GetReservationCharge() should returns 40.00.
  10. GetMinTatkalDistance() should returns 250.
  11. GetTatkalLoadFactor() should returns 0.30.

```
const BookingClasses& ACChairCar_ = BookingClasses::ACChairCar::Type() ;
    assert(ACChairCar_.GetName() == "AC Chair Car") ;
    assert(ACChairCar_.GetLoadFactor() == 2.00) ;
    assert(ACChairCar_.IsAC() == true) ;
    assert(ACChairCar_.IsLuxury() == false) ;
    assert(ACChairCar_.IsSitting() == true) ;
    assert(ACChairCar_.GetNumberOfTiers() == 0) ;
    assert(ACChairCar_.GetMinTatkalCharge() == 125) ;
    assert(ACChairCar_.GetMaxTatkalCharge() == 225) ;
    assert(ACChairCar_.GetReservationCharge() == 40.00) ;
    assert(ACChairCar_.GetMinTatkalDistance() == 250) ;
    assert(ACChairCar_.GetTatkalLoadFactor() == 0.3)  ;
```

- Testing of Sleeper class:

  We make a function to check the equality of the following:

1. IsLuxury() should return false.
2. IsAc() should return false.
3. GetLoadFactor() should return 1.00.
4. GetName() should return “ Sleeper ”.
5. IsSitting() should return false.
6. GetMinTatkalCharge() should returns 100.
7. GetMaxTatkalCharge() should returns 200.
8. GetReservationCharge() should returns 20.00.
9. GetMinTatkalDistance() should returns 500.
10. GetTatkalLoadFactor() should returns 0.30.
11. GetNumberOfTiers() should return 3.

The following function was used for Unit Testing:

```
const BookingClasses& Sleeper_ = BookingClasses::Sleeper::Type() ;
assert(Sleeper_.GetName() == "Sleeper") ;
assert(Sleeper_.GetLoadFactor() == 1.00) ;
assert(Sleeper_.IsAC() == false) ;
assert(Sleeper_.IsLuxury() == false) ;
assert(Sleeper_.IsSitting() == false) ;
assert(Sleeper_.GetNumberOfTiers() == 3) ;
assert(Sleeper_.GetMinTatkalCharge() == 100) ;
assert(Sleeper_.GetMaxTatkalCharge() == 200) ;
assert(Sleeper_.GetReservationCharge() == 20.00) ;
assert(Sleeper_.GetMinTatkalDistance() == 500) ;
assert(Sleeper_.GetTatkalLoadFactor() == 0.3)  ;
```

● Testing of SecondSitting class:

We make a function to check the equality of the following:

1. IsLuxury() should return false.
2. IsAc() should return false.
3. GetLoadFactor() should return 0.60.
4. GetName() should return “ Second Sitting”.
5. IsSitting() should return true.
6. GetMinTatkalCharge() should returns 10.
7. GetMaxTatkalCharge() should returns 15.
8. GetReservationCharge() should returns 15.00.
9. GetMinTatkalDistance() should returns 100.
10. GetTatkalLoadFactor() should returns 0.10.
11. GetNumberOfTiers() should return 0.

The following function was used for Unit Testing:

```
const BookingClasses& SecondSitting_ =
BookingClasses::SecondSitting::Type() ;
```

```
assert(SecondSitting_.GetName() == "Second Sitting") ;
assert(SecondSitting_.GetLoadFactor() == 0.60) ;
assert(SecondSitting_.IsAC() == false) ;
assert(SecondSitting_.IsLuxury() == false) ;
assert(SecondSitting_.IsSitting() == true) ;
assert(SecondSitting_.GetNumberOfTiers() == 0) ;
assert(SecondSitting_.GetMinTatkalCharge() == 10) ;
assert(SecondSitting_.GetMaxTatkalCharge() == 15) ;
assert(SecondSitting_.GetReservationCharge() == 15.00) ;
assert(SecondSitting_.GetMinTatkalDistance() == 100) ;
assert(SecondSitting_.GetTatkalLoadFactor() == 0.1)  ;
```

- Testing of ExecutiveChairCar class:

  We make a function to check the equality of the following:

  1. IsLuxury() should return true.
  2. IsAc() should return true.
  3. GetLoadFactor() should return 1.00.
  4. GetName() should return “Executive Chair Car ”.
  5. GetNumberOfTiers() should return 0.
  6. IsSitting() should return true.
  7. GetMinTatkalCharge() should returns 400.
  8. GetMaxTatkalCharge() should returns 500.
  9. GetReservationCharge() should returns 60.00.
  10. GetMinTatkalDistance() should returns 250.
  11. GetTatkalLoadFactor() should returns 0.30.

  The following function was used for Unit Testing:

```
const BookingClasses& ExecutiveChairCar_ =
BookingClasses::ExecutiveChairCar::Type() ;
assert(ExecutiveChairCar_.GetName() == "Executive Chair Car") ;
assert(ExecutiveChairCar_.GetLoadFactor() == 5.00) ;
assert(ExecutiveChairCar_.IsAC() == true) ;
assert(ExecutiveChairCar_.IsLuxury() == true) ;
assert(ExecutiveChairCar_.IsSitting() == true) ;
assert(ExecutiveChairCar_.GetNumberOfTiers() == 0) ;
assert(ExecutiveChairCar_.GetMinTatkalCharge() == 400) ;
assert(ExecutiveChairCar_.GetMaxTatkalCharge() == 500) ;
assert(ExecutiveChairCar_.GetReservationCharge() == 60.00) ;
assert(ExecutiveChairCar_.GetMinTatkalDistance() == 250) ;
assert(ExecutiveChairCar_.GetTatkalLoadFactor() == 0.3)  ;
```

Station

We are testing by doing the following

Checking the constructor with various names
Check distances between stations

Station s1 = Station("Mumbai"), s2 = Station("Delhi"), s3 = Station("Mumbai");
create objects s1, s2, and s3 and check for equality.

Make a function to check the equality of the following:
Get name should return the string initially passed during object construction ie
s1.GetName() should return “Mumbai”

The following Function is used :

```
void Station::UnitTest()
{
Station s1 = Station("Mumbai"), s2 = Station("Delhi"), s3 = Station("Mumbai");
assert(s1.GetName() == "Mumbai");
assert(s2.GetName() == "Delhi");
assert(s3.GetName() == "Mumbai");

assert(s1 == Station("Mumbai"));
assert(s2 == Station("Delhi"));
assert(s3 == Station("Mumbai"));

//assert(s1.operator==(Station("Mumbai")));

cout << "Success in Station Unit Testing" << endl;

}
```

Few examples of how Testing has been done :

1. Checking the constructor with various names
    a. Station("")
    b. Station(“Kharagpur”)->Output: Bad_Station Exception is thrown
2. Check distances between station
    a. Station(“Mumbai”).GetDistance(Station(“Delhi”)) ->Output: 1447
    b. Station(“Delhi”).GetDistance(Station(“Mumbai”)) ->Output: 144

Railways

Dummy railways is created to test by adding 3 stations and then using the Get Distance function which returns the distance between given two station according to the hard coded distance matrix. Then it's tested by the following function

```
void Railways::UnitTest()
{
	Station s1 = Station("Mumbai");
	Station s2 = Station("Delhi");
	Station s3 = Station("Kolkata");
	assert(GetDistance(s1, s2) == 1447);

	cout << "Success in Railways Unit Testing" << endl;
}
```

We are basically matching the golden which is the to station, from station and distance between them

Booking

For each of the booking categories, we will verify its properties using golden outputs. Even though most of them will be validated in the final booking tests. The is used for testing function is used for testing

In test we are creating an instance of passenger than creating an instance of booking then checking the get fare method like this i have written many cases
For each booking category class and concession etc

For example we can check the following functions

1. GetFromStation
2. GetFromStation etc

Few examples of how exceptional handling has been done has been done :

1. Checking the constructor with various names

2. Invalid date of reservation for tatkal etc

We can check attributes and methods like

1. Check generalConcessions class.
2. Check LadiesConcessions class.
3. Check DivyaangConcessions class

a) Executive Chair Car, no disability

Station(“Delhi”)
Station(“Chennai”)
Date(30,4,2021)
Passenger: Person with No disability

Output: Bad_Booking exception

a) Sleeper Class

Station(“Delhi”)
Station(“Bangalore”)
Date(25,4,2021)
Sleeper::Type()
Passenger: Female Passenger with Date of birth = 4/4/1964

Application Testing

In this basically we are testing the whole application from start to finish

Sample input and Golden output

Passenger input: p1

- Name->Prashant pawar
- DOB->4/Apr/2002
- Gender->Male
- Adhaar->842233101212

- Mobile No.->8447763108
- category->General

Booking input:

- From station Delhi
- To station Mumbai
- Reservation date 5/Jul/2021
- Booking class AC First Class
- Booking category General
- Passenger:p1

Output

BOOKING COMPLETED:
PNR Number = 1
From Station = Delhi
To Station = Mumbai
Travel Date = 5/Jul/2021
Travel Class = AC First Class
: Mode: Berth
: Comfort: AC
: Bunks: 2
: Luxury: Yes
Fare = 4762
Passenger is:
Passenger's Name : Prashant pawar
Passenger's Aadhar Number : 842233101212
Passenger's Date of Birth : 4/Apr/2002
Passenger's Gender : Male
Passenger's Mobile Number : 8447763108
Passenger's Category : General