Rajat Bachhawat 19CS10073

Test Plan for Railways Booking System

Unit Test

Booking

- Test the working of Reserve
    1. Booking correctly constructed for Ladies Booking
    2. Booking correctly constructed for Tatkal Booking
    3. Exception thrown when fromStation = toStation
- Test the working of polymorphic hierarchy
- Test the working of output streaming operator

BookingTypes<T>

- Test the working of BookingCategory::General::TestEligibility() (One valid and one invalid case)
- Test the working of BookingCategory::Ladies::TestEligibility() (One valid and one invalid case)
- Test the working of BookingCategory::SeniorCitizen::TestEligibility()(One valid and one invalid case)
- Test the working of BookingCategory::DivyaangCat::TestEligibility()(One valid and one invalid case)
- Test the working of BookingCategory::Tatkal::TestEligibility()(One valid and one invalid case)
- Test the working of BookingCategory::PremiumTatkal:TestEligibility()(One valid and one invalid case)

- Test the working of BookingCategory::Ladies::ComputeFare()
- Test the working of BookingCategory::General::ComputeFare()
- Test the working of BookingCategory::General::ComputeFare()
- Test the working of BookingCategory::SeniorCitizen::ComputeFare()
- Test the working of BookingCategory::SeniorCitizen::ComputeFare()

- Test the working of BookingCategory::DivyaangCat::ComputeFare()
- Test the working of BookingCategory::DivyaangCat::ComputeFare()
- Test the working of BookingCategory::Tatkal::ComputeFare()
- Test the working of BookingCategory::Tatkal::ComputeFare()
- Test the working of BookingCategory::PremiumTatkal:ComputeFare()
- Test the working of BookingCategory::PremiumTatkal:ComputeFare()

Gender

-
- Test working of polymorphic hierarchy
- Test the working of output streaming operator

GenderTypes<T>

- Test GetName() for Male
- Test GetName() for Female
- Test GetTitle() for Male
- Test GetTitle() for Female
- Test the working of output stream operator

Name

- Test the working of output streaming operator

Station

- Test the working of output streaming operator
- Test GetDistance() for Stations
    1. One direction
    2. Symmetrical opposite direction
- Test if GetName works correctly
- Test validity of Stations by IsValid()
    1. Empty Station Name
    2. Station not present in DataBase
- Test if GetDistance() throws Exception when asked for distances between same station
- Test if GetStation() works correctly
    1. Exception thrown for invalid Station
    2. Returns correct station for valid station

Railways

- Test if all correct stations are stored in list of Stations
- Test if sDistStations has correct distance between stations (matching with Golden Output)
- Test if testObj.GetDistance() returns correct distance between stations (matching with sDistStations)
- Test for symmetric ordering of Stations
- Test working of IsValid()
    1. Duplicate Station should not be in stations database
    2. Same pair of stations with a given ordering should be in distances database EXACTLY once
    3. Both directions of same pair of stations should not be in distances database
    4. Pair with same stations should not be present in distances database
- Test working for GetDistance
    1. Throws Exception when queried with the same stations
    2. Throws Exception when queried with station not in database

Passenger

- Testing is a passenger is valid
    1. Exception when both first and last names missing
    2. Valid Naming + aadhar + birthday + mobile no - Middle Name missing
    3. Valid Naming + aadhar + birthday + mobile no - No Name missing
    4. Exception when Bad Aadhaar - Not 12 digits
    5. Exception when Bad Aadhar - Non numeric
    6. Exception when Bad Mobile no - non empty with length not 10
    7. Exception when Bad Mobile no - non empty with non numeric
    8. Mobile Number is valid
    9. Exception when Bad Age - Not born yet
- Testing the overloaded== operator
- Testing GetPassenger - Valid Case
- Testing GetPassenger - InValid Case

- Test Output streaming operator for Passenger

BookingClasses

- Test the working of polymorphic hierarchy
- Test the working of output streaming operator

BookingClassTypes<T>

Where T -> ACFirstClassType, ExecutiveChairCarType, AC2TierType, FirstClassType, AC3TierType, ACChairCarType, SleeperType, SecondSittingType

- Test the working of all simple member functions
    - GetLoadFactor()
    - GetName()
    - IsAC()
    - IsLuxury()
    - IsSitting()
    - GetNumberOfTiers()
    - GetReservationCharge()
    - GetTatkalLoadFactor()
    - GetMinTatkalCharge()
    - GetMaxTatkalCharge()
    - GetMinTatkalDist()
    - Test the working of output streaming operator

BookingCategory

- Test the working of polymorphic hierarchy
- Test the working of output streaming operator
- Test the working of ReserveInCategory(), i.e., whether it returns NULL/non-NULL appropriately

BookingCategoryTypes<T>

- Test the working of BookingCategory::General::Eligibility()
    1. Exception when Date of Reservation after Date of Booking
    2. Exception when Date of Reservation more than an year before Booking
    3. No Exception when all cases above are dissatisfied
- Test the working of BookingCategory::Ladies::Eligibility()

1. Exception when Date of Reservation after Date of Booking
2. Exception when Date of Reservation more than an year before Booking
3. Exception when passenger is Male more than 12 years of age
4. No Exception when all cases above are dissatisfied

- Test the working of BookingCategory::SeniorCitizen::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
  2. Exception when Date of Reservation more than an year before Booking
  3. Exception when passenger is Male less than 60 years of age
  4. Exception when passenger is Female less than 58 years of age
  5. No Exception when all cases above are dissatisfied
- Test the working of BookingCategory::DivyaangCat::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
  2. Exception when Date of Reservation more than an year before Booking
  3. Passenger with Divyaang Id and/or Divyaang id absent
  4. No Exception when all cases above are dissatisfied
- Test the working of BookingCategory::Tatkal::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
  2. Exception when Date of Reservation more than an year before Booking
  3. Reservation done more than 1 day before actual booking timings
  4. No Exception when all cases above are dissatisfied
- Test the working of BookingCategory::PremiumTatkal:Eligibility()
  1. Exception when Date of Reservation after Date of Booking
  2. Exception when Date of Reservation more than an year before Booking
  3. Reservation done more than 1 day before actual booking timings
  4. No Exception when all cases above are dissatisfied
- Test the working of output streaming operator

Divyaang

- Test the working of polymorphic hierarchy
- Test the working of output streaming operator

DisabilityTypes<T>

Test GetDivyaangConcessionFactor iin a way which includes all Disability Types and all Booking Classes at least once

Where T -> BlindType, OrthopaedicallyHandicappedType, CancerPatientType, TBPatientType

- Test GetDivyaangConncessionFactor called by Blind Type for ACFirstClass
- Test GetDivyaangConncessionFactor called by Blind Type for ExecutiveChairCar
- Test GetDivyaangConncessionFactor called by Blind Type for FirstClass
- Test GetDivyaangConncessionFactor called by Blind Type for AC2Tier
- Test GetDivyaangConncessionFactor called by Blind Type for ExecutiveChairCar
- Test GetDivyaangConncessionFactor called by Blind Type for AC3Tier
- Test GetDivyaangConncessionFactor called by OrthopaedicallyHandicapped Type for AC Chair Car
- Test GetDivyaangConncessionFactor called by Cancer PatientType for Sleeper
- Test GetDivyaangConncessionFactor called by TBType for Second Sitting
- Test the working of output streaming operator

Date

- Test the working of output streaming operator
- Test Date construction with numbers
- Test Date construction with string
- Test copy constructor for Date
- Test if GetDay() returns correct Day of the Month
- Test if GetMonth() returns correct Month
- Test if GetYear() returns correct Year
- Test for working of IsLeapYear()
    1. Non-Leap year not divisible by 100
    2. Non-leap year divisible by 100 but not by 400

3. Leap year divisible by 400
4. Leap year not divisible by 400

- Test if CalculateAge() returns correct Age based on this year (input is taken as first of January to ensure the golden does not change within 1 year)
- CalculateSpan() working correctly
  1. when leap years are present in the middle
  2. when leap years are not present in the middle
- Test Date::Today()
- Test operator ==
  1. When matching
  2. When not matching
- Test the validation by IsValid() for integer inputs
  1. Invalid year (not in 1900-2099)
  2. Inavlid month (>12)
  3. Invalid month (<12)
  4. Invalid Day (<=0)
  5. Invalid Day (29 Days in February in a non-leap year)
  6. Valid Day (29 Days in February in a leap year)
  7. Invalid Day (>30 Days in a month with 30 days)
  8. Invalid Day (>31 Days in a month with 31 days)
  9. Valid Day
- Test the validation by IsValid() for integer inputs
  1. Invalid year (not in 1900-2099)
  2. Inavlid month (>12)
  3. Invalid month (<12)
  4. Invalid Day (<=0)
  5. Invalid Day (29 Days in February in a non-leap year)
  6. Valid Day (29 Days in February in a leap year)
  7. Invalid Day (>30 Days in a month with 30 days)
  8. Invalid Day (>31 Days in a month with 31 days)
  9. Valid Day
  10. Invalid Format (Not DD/MM/YYYY format with more characters)
  11. Invalid Format (Not DD/MM/YYYY format with less characters)
  12. Invalid Format (Non numeric characters present)
  13. Invalid Format (‘/’ not present/replaced)
- Correct working of GetDate()
  1. Valid Date - string
  2. Invalid Date - string
  3. Valid Date - Numbers
  4. Invalid Date - Numbers

Concessions

We do not Test GetConcessions() for every pair, we make sure all BookingClasses and Booking Types including subtypes of Divyaang are covered.

- Test Get Concessions for Blind Type and ACFirstClass
- Test Get Concessions for Blind Type and ExecutiveChairCar
- Test Get Concessions for Blind Type and FirstClass
- Test Get Concessions for Blind Type and AC2Tier
- Test Get Concessions for Blind Type and ExecutiveChairCar
- Test Get Concessions for Blind Type and AC3Tier
- Test Get Concessions for OrthopaedicallyHandicapped Type and AC Chair Car
- Test Get Concessions for Cancer PatientType and Sleeper
- Test Get Concessions for TBType and Second Sitting
- Test Get Concessions for Ladies Booking
- Test Get Concessions for female Senior Citizen
- Test Get concessions for male senior citizen

GeneralConcession

Testing is a subset of Concessions Testing (included there)

LadiesConcession

Testing is a subset of Concessions Testing (included there)

DivyaangConcession

Testing is a subset of Concessions Testing (included there)

SeniorCitizenConcession

Testing is a subset of Concessions Testing (included there)

Application Test

To be done on _DEBUG mode

- Test CONSTRUCTOR for all valid Classes
- Test DESTRUCTOR for all valid Classes
- Test Singleton Nature for all Singletons
- Test copy constructor wherever valid
- Test if all Bookings are executed correctly
- Test if List of Bookings is printed correctly
- Test that program throws expected Exceptions when needed

Rajat Bachhawat 19CS10073

Test Suite for Railways Booking System

Wherever Output is written, it actually means Golden Output

Unit Tests

Booking

- Test proper working of Reserve
    1. Booking correctly constructed for Ladies Booking
       Input Provided
       ```
       Passenger p1 =
       Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1988),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e");

       Booking::Reserve(Station::GetStation("Mumbai"),
       Station::GetStation("Delhi"), Date::GetDate("02/05/2021"),
       Date::Today(), BookingClasses::AC2Tier::Type(),
       BookingCategory::Ladies::Type(),p1);
       ```
       Output
       non-NULL Booking pointer pointing to a fully constructed Booking object
    2. Booking correctly constructed for Tatkal Booking
       Input Provided
       ```
       Passenger p2 =
       Passenger::GetPassenger(Name("Nick","Jonas"),Date::GetDate(5,1,1996),Gender::Male::Type(),"123456789123","0123456789");

       Booking::Reserve(Station::GetStation("Bangalore"),
       Station::GetStation("Chennai"), Date::Today(), Date::Today(),
       BookingClasses::ExecutiveChairCar::Type(),
       BookingCategory::Tatkal::Type(),p2);
       ```
       Output
       non-NULL Booking pointer pointing to a fully constructed Booking object
    3. Exception thrown when fromStation = toStation
       Input Provided
       ```
       Passenger p1 =
       Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1988),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e");
       ```

```
Booking::Reserve(Station::GetStation("Delhi"),
Station::GetStation("Delhi"), Date::GetDate("02/05/2021"),
Date::Today(), BookingClasses::AC2Tier::Type(),
BookingCategory::Ladies::Type(),p1);
```
Output
```
Exception thrown: Bad_Booking
```

- Check whether output streaming operator works correctly
  Input Provided
  ```
  Passenger p11 =
  Passenger::GetPassenger(Name("Bob","Voodoo","Dylan"),Date::GetDate(5,1,1900),
  Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"
  e");
  const Booking* bTest =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),D
  ate::Today(),Date::Today(), BookingClasses::AC3Tier::Type(),
  BookingCategory::General::Type(),p11);
  ```
  Output
  ```
  "BOOKING SUCCEEDED:\n-- Passenger Details --\nName = Bob Dylan Voodoo\nAge =
  121\nGender = Female\nAadhar Number = 123456789123\nMobile Number =
  0123456789\nDisability Type = Blind\nDisabilityID = e\n\n-- Booking Details
  -- \nPNR Number = 4\nFrom Station = Mumbai\nTo Station = Delhi\nTravel Date =
  2/Apr/2021\nReservation Date = 2/Apr/2021\nBooking Category = General\nTravel
  Class = AC 3 Tier\n : Mode: Sleeping\n : Comfort: AC\n : Bunks: 3\n : Luxury:
  No\nFare = 1849\n\n"
  ```

BookingTypes<T>

- Test proper working of BookingCategory::General::CheckEligibility() (One valid and one invalid case)
  Common Input Provided
  ```
  Passenger p1 =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1988),G
  ender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e
  ");
  Passenger p2 =
  Passenger::GetPassenger(Name("Nick","Jonas"),Date::GetDate(5,1,1996),Gender::
  Male::Type(),"123456789123","0123456789");
  ```

  1. Valid Booking
     Input Provided
     ```
     Booking::GeneralBooking::CheckEligibility(p1,
     BookingCategory::General::Type(), Date::Today(),
     Date::GetDate(2,5,2021))
     ```
     Output

   true
2. Invalid Booking: Date of Booking is before Date of reservation
   Input Provided
   `Booking::GeneralBooking::CheckEligibility(p1, BookingCategory::General::Type(), Date::Today(), Date::GetDate(3,5,1900))`
   Output
   Exception thrown: Bad_Chronology

- Test proper working of BookingCategory::Ladies::CheckEligibility() (One valid and one invalid case)
  1. Valid Booking
     Input Provided
     `Booking::LadiesBooking::CheckEligibility(p1, BookingCategory::Ladies::Type(), Date::Today(), Date::GetDate(2,5,2021))`
     Output
     true
  2. Invalid Booking: Male of age > 12
     Input Provided
     `Booking::LadiesBooking::CheckEligibility(p2, BookingCategory::Ladies::Type(), Date::Today(), Date::GetDate(2,5,2021))`
     Output
     Exception thrown: Ineligible_Ladies_Category
- Test proper working of BookingCategory::SeniorCitizen::CheckEligibility()(One valid and one invalid case)
  1. Valid Booking
     Input Provided

     `Booking::SeniorCitizenBooking::CheckEligibility(Passenger::GetPassenger(Name("Jai","Shah"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), BookingCategory::SeniorCitizen::Type(), Date::Today(), Date::GetDate(2,5,2021))`
     Output
     true
  2. Invalid Booking: Male of age < 60
     Input Provided
     `Booking::SeniorCitizenBooking::CheckEligibility(p2, BookingCategory::SeniorCitizen::Type(), Date::Today(), Date::GetDate(2,5,2021))`
     Output
     Exception thrown: Ineligible_SeniorCitizen_Category

- Test proper working of BookingCategory::DivyaangCat::CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Input Provided
       Booking::DivyaangBooking::CheckEligibility(p1, BookingCategory::DivyaangCat::Type(), Date::Today(), Date::GetDate(2,5,2021))
       Output
       true
    2. Invalid Booking: No disability in passenger
       Input Provided
       Booking::DivyaangBooking::CheckEligibility(p2, BookingCategory::DivyaangCat::Type(), Date::Today(), Date::GetDate(2,5,2021))
       Output
       Exception thrown: Ineligible_Divyaang_Category
- Test proper working of BookingCategory::Tatkal::CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Input Provided
       Booking::TatkalBooking::CheckEligibility(p1, BookingCategory::Tatkal::Type(), Date::Today(), Date::Today())
       Output
       true
    2. Invalid Booking: Date of booking is not within 1 day of date of reservation
       Input Provided
       Booking::TatkalBooking::CheckEligibility(p1, BookingCategory::Tatkal::Type(), Date::Today(), Date::GetDate(2,7,2021))
       Output
       Exception thrown: Ineligible_Tatkal_Category
- Test proper working of BookingCategory::PremiumTatkal:CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Input Provided
       Booking::PremiumTatkalBooking::CheckEligibility(p1, BookingCategory::Tatkal::Type(), Date::Today(), Date::Today())
       Output
       true
    2. Invalid Booking
       Input Provided

```
    Booking::TatkalBooking::CheckEligibility(p1,
    BookingCategory::PremiumTatkal::Type(), Date::Today(),
    Date::GetDate(2,7,2021))
    Output
    Exception thrown: Ineligible_Tatkal_Category
```

- Test proper working of BookingCategory::Ladies::ComputeFare()
  Input Provided

```
b4 =
Booking::Reserve(Station::GetStation("Kolkata"),Station::GetStation("Delhi"),
book,reser, BookingClasses::AC2Tier::Type(),
BookingCategory::Ladies::Type(),p11);

b4->ComputeFare();
```

  Output

```
2994
```

- Test proper working of BookingCategory::General::ComputeFare()
  Input Provided

```
b5 =
Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
ook,reser, BookingClasses::AC3Tier::Type(),
BookingCategory::General::Type(),p11);

b5->ComputeFare();
```

  Output

```
1849
```

- Test proper working of BookingCategory::General::ComputeFare()
  Input Provided

```
b6 =
Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
ook,reser, BookingClasses::ACFirstClass::Type(),
BookingCategory::General::Type(),p11);

b6->ComputeFare();
```

  Output

```
4763
```

- Test proper working of BookingCategory::SeniorCitizen::ComputeFare()
  Input Provided

```
b7 =
Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
ook,reser, BookingClasses::AC3Tier::Type(),
BookingCategory::SeniorCitizen::Type(),p21);

b7->ComputeFare();
```

Output
1125

- Test proper working of BookingCategory::SeniorCitizen::ComputeFare()
  Input Provided
  ```
  b8 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::ACFirstClass::Type(),
  BookingCategory::SeniorCitizen::Type(),p11);

  b8->ComputeFare();
  ```
  Output
  2411
- Test proper working of BookingCategory::DivyaangCat::ComputeFare()
  Input Provided
  ```
  b9 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::AC3Tier::Type(),
  BookingCategory::DivyaangCat::Type(),p11);

  b9->ComputeFare();
  ```
  Output
  492
- Test proper working of BookingCategory::DivyaangCat::ComputeFare()
  Input Provided
  ```
  b10 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::ACFirstClass::Type(),
  BookingCategory::DivyaangCat::Type(),p21);

  b10->ComputeFare();
  ```
  Output
  2411
- Test proper working of BookingCategory::Tatkal::ComputeFare()
  Input Provided
  ```
  b11 =
  Booking::Reserve(Station::GetStation("Delhi"),Station::GetStation("Mumbai"),b
  ook,reser, BookingClasses::AC3Tier::Type(),
  BookingCategory::Tatkal::Type(),p11);

  b11->ComputeFare();
  ```
  Output
  2249
- Test proper working of BookingCategory::Tatkal::ComputeFare()

Input Provided
```
b12 =
Booking::Reserve(Station::GetStation("Chennai"),Station::GetStation("Bangalor
e"),book,reser, BookingClasses::ACFirstClass::Type(),
BookingCategory::Tatkal::Type(),p11);

b12->ComputeFare();
```
Output
```
1198
```
- Test proper working of BookingCategory::PremiumTatkal:ComputeFare()

  Input Provided
  ```
  b13 =
  Booking::Reserve(Station::GetStation("Chennai"),Station::GetStation("Bangalor
  e"),book,reser, BookingClasses::ACFirstClass::Type(),
  BookingCategory::PremiumTatkal::Type(),p11);

  b13->ComputeFare();
  ```
  Output
  ```
  1198
  ```
- Test proper working of BookingCategory::PremiumTatkal:ComputeFare()

  Input Provided
  ```
  b14 =
  Booking::Reserve(Station::GetStation("Delhi"),Station::GetStation("Mumbai"),b
  ook,reser, BookingClasses::AC3Tier::Type(),
  BookingCategory::PremiumTatkal::Type(),p11);

  b14->ComputeFare();
  ```
  Output
  ```
  2649
  ```

Gender

- Check working of polymorphic hierarchy from return value of GetName()

  Input Provided
  ```
  const Gender &obj = Gender::Male::Type();
  ```
  Output
  ```
  "Male"
  ```
- Check whether output streaming operator works correctly

  Input Provided
  ```
  const Gender &gTest = Gender::Male::Type();
  ```
  Output
  ```
  "Male"
  ```

GenderTypes<T>

- Check GetName() for Gender::Male
  Input Provided
  `Gender::Male::Type().GetName()`
  Output
  "Male"
- Check GetName() for Gender::Female
  Input Provided
  `Gender::Female::Type().GetName()`
  Output
  "Female"
- Check GetTitle() for Gender::Male
  Input Provided
  `Gender::Male::Type().GetTitle()`
  Output
  "Mr."
- Check GetTitle() for Gender::Female
  Input Provided
  `Gender::Female::Type().GetTitle()`
  Output
  "Ms."
- Test proper working of output stream operator
  Input Provided
  `const Gender::Female &fTest = Gender::Female::Type();`
  Output
  "Female"

Name

- Check whether output streaming operator works correctly
  Input Provided
  `Name n = Name("Bob","Dylan");`
  Output
  "Bob Dylan"

Station

- Check whether output streaming operator works correctly
  Input Provided
  `Station stationTest("Delhi");`
  Output
  "Delhi"
- Check GetDistance() for Stations
  1. One direction

2. Symmetrical opposite direction

Golden Data:

```
{{"Mumbai", "Kolkata"}, 2014},
{{"Mumbai", "Chennai"}, 1338},
{{"Mumbai", "Bangalore"}, 981},
{{"Mumbai", "Delhi"}, 1447},

{{"Delhi", "Kolkata"}, 1472},
{{"Delhi", "Chennai"}, 2180},
{{"Delhi", "Bangalore"}, 2150},
{{"Delhi", "Mumbai"}, 1447},

{{"Kolkata", "Delhi"}, 1472},
{{"Kolkata", "Chennai"}, 1659},
{{"Kolkata", "Bangalore"}, 1871},
{{"Kolkata", "Mumbai"}, 2014},

{{"Bangalore", "Delhi"}, 2150},
{{"Bangalore", "Chennai"}, 350},
{{"Bangalore", "Kolkata"}, 1871},
{{"Bangalore", "Mumbai"}, 981},

{{"Chennai", "Delhi"}, 2180},
{{"Chennai", "Bangalore"}, 350},
{{"Chennai", "Kolkata"}, 1659},
{{"Chennai", "Mumbai"}, 1338}};
```

- Check whether GetName works correctly

  Input Provided

  ```
  Station st5("Kolkata");
  st5.GetName();
  ```

  Output

  ```
  "Kolkata"
  ```

- Check validity of Stations by IsValid()
  1. Empty Station Name

     Input Provided

     ```
     IsValid("");
     ```

     Output

     Exception thrown : `Bad_Station_Name`
  2. Station not present in DataBase

     Input Provided

     ```
     IsValid("Jammu");
     ```

     Output

    Exception thrown : `Bad_Station_Name`

- Check whether GetDistance() throws Exceptions when asked for distances between same station
  Input Provided
  `Station::GetStation("Kolkata").GetDistance(Station::GetStation("Kolkata"));`
  Output
  Exception thrown : `Distance_Not_Defined`
- Check whether GetStation() works correctly
  1. Exception thrown for invalid Station
     Input Provided
     `Station::GetStation("");`
     Output
     Exception thrown : `Bad_Station_Name`
  2. Returns correct station for valid station
     Input Provided
     `Station::GetStation("Kolkata");`
     Output
     `Station("Kolkata");`

Railways

- Check whether all correct stations are stored in list of Stations
  Golden Data
  `{"Bangalore", "Chennai", "Delhi", "Kolkata", "Mumbai"}`
- Check whether sDistStations has correct distance between stations (matching with Golden Output)
  Golden Data

```
    {{"Mumbai", "Kolkata"}, 2014},
     {{"Mumbai", "Chennai"}, 1338},
     {{"Mumbai", "Bangalore"}, 981},
     {{"Mumbai", "Delhi"}, 1447},

     {{"Delhi", "Kolkata"}, 1472},
     {{"Delhi", "Chennai"}, 2180},
     {{"Delhi", "Bangalore"}, 2150},
     {{"Delhi", "Mumbai"}, 1447},

     {{"Kolkata", "Delhi"}, 1472},
     {{"Kolkata", "Chennai"}, 1659},
     {{"Kolkata", "Bangalore"}, 1871},
     {{"Kolkata", "Mumbai"}, 2014},

     {{"Bangalore", "Delhi"}, 2150},
     {{"Bangalore", "Chennai"}, 350},
        {{"Bangalore", "Kolkata"}, 1871},
        {{"Bangalore", "Mumbai"}, 981},

        {{"Chennai", "Delhi"}, 2180},
        {{"Chennai", "Bangalore"}, 350},
        {{"Chennai", "Kolkata"}, 1659},
        {{"Chennai", "Mumbai"}, 1338}};
```

- Check whether testObj.GetDistance() returns correct distance between stations (matching with sDistStations)
  Same Golden Output as above
- Test symmetric ordering of Stations
  Same Golden Output as above
- Check working of IsValid()
  1. Duplicate Station in Stations database
     Output
     Exception thrown : Duplicate_Station
  2. Same pair of stations with a given ordering in distances database not EXACTLY once
     Input Provided
     Output
     Exception thrown : Bad_Railways

  3. Distance between two existing stations is not present in distances database
     Output
     Exception thrown : Incomplete_Distance_Information

  4. Pair with same stations present in distances database
     Output
     Exception thrown : Bad_Railways

  Check working for GetDistance
  1. Queried with the same stations
     Input Provided
     `Railways::Type().GetDistance(Station::GetStation("Kolkata"),Station::GetStation("Kolkata"));`
     Output
     Exception thrown : Distance_Not_Defined
  2. Queried with station not in database
     Input Provided
     `Railways::Type().GetDistance(Station::GetStation("Kolkata"),Station::GetStation("Jammu"))`
     Output

Exception thrown : Bad_Station_Name

Passenger

- Testing is a passenger is valid
    1. Error when both first and last names missing
       Input Provided : IsValid(Name("","","Y"), Date::Today(),Gender::Male::Type(),"123456789999","1234567890",NULL,"");
       Golden Output :Bad_Name Exception thrown
    2. Valid Naming + aadhar + birthday + mobile no - Middle Name missing
       Input Provided IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789999","1234567890",NULL,"")
       Golden Output : No exception thrown
    3. Valid Naming + aadhar + birthday + mobile no - No Name missing
       Input Provided: IsValid(Name("X","Y","Z"), Date::Today(),Gender::Male::Type(),"123456789999","1234567890",NULL,"");
       Golden Output: No exception thrown
    4. Error when Bad Aadhaar - Not 12 digits
       Input Provided: IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"1234567899999","1234567890",NULL,"")
       Golden Output: Bad_Aadhar_Number exception thrown
    5. Error when Bad Aadhar - Non numeric
       Input Provided : IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789a99","1234567890",NULL,"")
       Golden Output: Bad_Aadhar_Number exception thrown
    6. Error when Bad Mobile no - non empty with length not 10
       Input Provided: IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789a99","1234567890",NULL,"")
       Golden Output: Bad_Mobile_Number exception thrown
    7. Error when Bad Mobile no - non empty with non numeric
       Input Provided: IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789999","12314a7890",NULL,"")
       Golden Output: Bad_Mobile_Number exception thrown
    8. Mobile Number is valid
       Input Provided: IsValid(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789999","",NULL,"")
       Golden Output: No exception thrown
    9. Error when Bad Age - Not born yet
       Input Provided: IsValid(Name("X","Y",""), Date::GetDate(1,1,2050),Gender::Male::Type(),"123456789999","1235476890",NULL,"")
       Golden Output: Bad_Age exception thrown
- testing the overloaded == operator
  Input Provided:Passenger p1 = Passenger(Name("X","Y","Z"), Date::Today(),Gender::Male::Type(),"123456789999","1234567890");

Passenger p2 = Passenger(Name("X","Y","Z"), Date::Today(),Gender::Male::Type(),"123456789999","1234567890");
Golden Output : True

- Testing GetPasseneger - Valid Case
  Input Provided: GetPassenger(Name("X","Y",""), Date::Today(),Gender::Male::Type(),"123456789999","1231478190")
  Golden Output: No exception thrown
- Testing GetPasseneger - InValid Case
  Input Provided: GetPassenger(Name("","Y",""), Date::Today(),Gender::Male::Type(),"123456789999","123147890");
  Output: Bad_Passenger exception thrown
- Test Output streaming operator for Passenger
  Input Provided:
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1988),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e")
  Golden Output: "-- Passenger Details --\nName = Priyanka Chopra\nAge = 32\nGender = Female\nAadhar Number = 123456789123\nMobile Number = 0123456789\nDisability Type = Blind\nDisabilityID = e"

BookingClasses

- Test proper working of polymorphic hierarchy from return value of GetName()
  Test Input Provided:
  const BookingClasses &obj = BookingClasses::AC3Tier::Type();
  Golden Output
  obj.GetName()=="AC 3 Tier"
- Check whether output streaming operator works correctly
  Test Input Provided
  const BookingClasses& bTest = AC2Tier::Type();
  Golden Output
  "Travel Class = AC 2 Tier\n : Mode: Sleeping\n : Comfort: AC\n : Bunks: 2\n : Luxury: No\n"

BookingClassTypes<T>

Where T -> ACFirstClassType, ExecutiveChairCarType, AC2TierType, FirstClassType, AC3TierType, ACChairCarType, SleeperType, SecondSittingType

- Test proper working of all simple member functions
  - GetLoadFactor()
    Golden Output is master data of load factor in problem statement
  - GetName()
    Table of Golden Outputs for the 8 sub-types:
    Golden Output is master data of name in problem statement
  - IsAC()
    Golden Output is master data of ac status in problem statement

- IsLuxury()
  Golden Output is master data of luxury status in problem statement
- IsSitting()
  Golden Output is master data of sitting status in problem statement
- GetNumberOfTiers()
  Golden Output is master data of number of tiers in problem statement
- GetReservationCharge()
  Golden Output is master data of reservation charge in problem statement
- GetTatkalLoadFactor()
  Golden Output is master data of tatkal factor in problem statement
- GetMinTatkalCharge()
  Golden Output is master data of min tatkal charge in problem statement
- GetMaxTatkalCharge()
  Golden Output is master data of max tatkal charge in problem statement
- GetMinTatkalDist()
  Golden Output is master data of max tatkal distance in problem statement

- Check whether output streaming operator works correctly
  Test Input Provided:
  `const BookingClasses::AC2Tier& aTest = AC2Tier::Type();`
  Golden Output:
  `"Called From: AC 2 Tier\nTravel Class = AC 2 Tier\n : Mode: Sleeping\n : Comfort: AC\n : Bunks: 2\n : Luxury: No\n"`

BookingCategory

- Test proper working of polymorphic hierarchy from return value of GetName()
  Test Input Provided
  `const BookingCategory &bTest = BookingCategory::Ladies::Type();`
  Golden Output
  `obj.GetName()=="Ladies"`
- Check whether output streaming operator works correctly
  Test Input Provided
  `const BookingCategory &bTest = BookingCategory::Ladies::Type();`
  Golden Output

"Booking Category = Ladies"

- Test proper working of ReserveInCategory(), i.e., whether it returns NULL/non-NULL appropriately
    1. Return pointer to a newly made Booking
       Test Input Provided
       ```
       Passenger p2 = Passenger::GetPassenger(Name("Priyanka","Chopra")
       ,Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456
       789",&Divyaang::Blind::Type(),"e");

       Booking* b1 =
       BookingCategory::General::Type().ReserveInCategory(Station::GetStation(
       "Mumbai"),Station::GetStation("Delhi"),Date::Today(),Date::Today(),
       BookingClasses::ACFirstClass::Type(),p2);
       ```
       Golden Output
       non-NULL

    2. Return NULL when invalid booking (Male of age 12+ in Ladies Category)
       Test Input Provided
       ```
       Passenger p1 =
       Passenger::GetPassenger(Name("Bob","Dylan"),Date::GetDate(5,1,1999),Gen
       der::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),
       "e");
       Booking* b2 =
       BookingCategory::Ladies::Type().ReserveInCategory(Station::GetStation("
       Mumbai"),Station::GetStation("Delhi"),Date::Today(),Date::Today(),
       BookingClasses::ACFirstClass::Type(),p1);
       ```
       Golden Output
       NULL

BookingCategoryTypes<T>

- Test proper working of BookingCategory::General::Eligibility()
  Common Inputs Provided:
  ```
  Passenger p1 =
  Passenger::GetPassenger(Name("Bob","Dylan"),Date::GetDate(5,1,1999),Gen
  der::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),
  "e");
  Passenger p2 =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,195
  0),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind:
  :Type(),"e");
  ```

1. Exception thrown when Date of Reservation after Date of Booking
   Test Input Provided
   `BookingCategory::General::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
   Golden output
   Exception thrown : Bad_Chronology
2. Exception thrown when Date of Reservation more than an year before Booking
   Test Input Provided:
   `BookingCategory::General::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
   Golden Output
   Exception thrown : Bad_Chronology
3. No Exception thrown when all cases above are dissatisfied
   Test Input Provided:
   `BookingCategory::General::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,6,2021));`
   Golden Output:
   No exception thrown

- Test proper working of BookingCategory::Ladies::Eligibility()
  1. Exception thrown when Date of Reservation after Date of Booking
     Input Provided
     `BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
     Output
     Exception thrown : Bad_Chronology
  2. Exception thrown when Date of Reservation more than an year before Booking
     Input Provided
     `BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
     Output
     Exception thrown : Bad_Chronology
  3. Exception thrown when passenger is Male more than 12 years of age
     Input Provided
     `BookingCategory::Ladies::Type().Eligibility(p1,Gender::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(), Date::GetDate(3,6,2021));`
     Output
     Exception thrown : Ineligible_Ladies_Category
  4. No Exception thrown when all cases above are dissatisfied
     Input Provided
     `BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,6,2021));`

    Output
    Exception thrown : None

- Test proper working of BookingCategory::SeniorCitizen::Eligibility()
    1. Exception thrown when Date of Reservation after Date of Booking
       Input Provided
       `BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
       Output
       Exception thrown : Bad_Chronology
    2. Exception thrown when Date of Reservation more than an year before Booking
       Input Provided
       `BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
       Output
       Exception thrown : Bad_Chronology
    3. Exception thrown when passenger is Male less than 60 years of age
       Input Provided
       `BookingCategory::SeniorCitizen::Type().Eligibility(Passenger::GetPassenger(Name("Bob","Dylan"),Date::GetDate(5,1,2020),Gender::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(), Date::GetDate(3,6,2021));`
       Output
       Exception thrown : Ineligible_SeniorCitizen_Category
    4. Exception thrown when passenger is Female less than 58 years of age
       Input Provided
       `BookingCategory::SeniorCitizen::Type().Eligibility(Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(), Date::GetDate(3,6,2021));`
       Output
       Exception thrown : Ineligible_SeniorCitizen_Category
    5. No Exception thrown when all cases above are dissatisfied
       Input Provided
       `BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,6,2021));`
       Output
       Exception thrown : None
- Test proper working of BookingCategory::DivyaangCat::Eligibility()
    1. Exception thrown when Date of Reservation after Date of Booking
       Input Provided
       `BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
       Output

   Exception thrown : Bad_Chronology
2. Exception thrown when Date of Reservation more than an year before Booking
   Input Provided
   `BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
   Output
   Exception thrown : Bad_Chronology
3. Passenger with Divyaang ID and/or Divyaang ID absent
   Input Provided
   `BookingCategory::DivyaangCat::Type().Eligibility(Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Female::Type(),"123456789123","0123456789"), Date::Today(), Date::GetDate(3,6,2021));`
   Output
   Exception thrown : Ineligible_Divyaang_Category
4. No Exception thrown when all cases above are dissatisfied
   Input Provided
   `BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,6,2021));`
   Output
   Exception thrown : None

- Test proper working of BookingCategory::Tatkal::Eligibility()
  1. Exception thrown when Date of Reservation after Date of Booking
     Input Provided
     `BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
     Output
     Exception thrown : Bad_Chronology
  2. Exception thrown when Date of Reservation more than an year before Booking
     Input Provided
     `BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
     Output
     Exception thrown : Bad_Chronology
  3. Reservation done more than 1 day before actual booking timings
     Input Provided
     `BookingCategory::Tatkal::Type().Eligibility(Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(), Date::GetDate(2,4,2022));`
     Output
     Exception thrown : Ineligible_Tatkal_Category

4. No Exception thrown when all cases above are dissatisfied
   Input Provided
   `BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(), Date::Today());`
   Output
   Exception thrown : None

- Test proper working of BookingCategory::PremiumTatkal:Eligibility()
  1. Exception thrown when Date of Reservation after Date of Booking
     Input Provided
     `BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));`
     Output
     Exception thrown : Bad_Chronology
  2. Exception thrown when Date of Reservation more than an year before Booking
     Input Provided
     `BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));`
     Output
     Exception thrown : Bad_Chronology
  3. Reservation done more than 1 day before actual booking timings
     `BookingCategory::PremiumTatkal::Type().Eligibility(Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Male::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(), Date::GetDate(2,4,2022));`
     Output
     Exception thrown : Ineligible_PremiumTatkal_Category
  4. No Exception thrown when all cases above are dissatisfied
     Input Provided
     `BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(), Date::Today());`
     Output
     Exception thrown : None

- Check whether output streaming operator works correctly
  Input Provided
  ```
  const BookingCategory::DivyaangCat &dTest =
  BookingCategory::DivyaangCat::Type();
  ```
  Output
  ```
  "Booking Category = Divyaang"
  ```

Divyaang

- Test proper working of polymorphic hierarchy from return value of GetName()
  Input Provided
  `const Divyaang &obj = Divyaang::Blind::Type();`
  Output
  `obj.GetName()=="Blind"`
- Check whether output streaming operator works correctly
  Input Provided
  `const Divyaang &dTest = Divyaang::Blind::Type();`
  Output
  `"Blind"`

DisabilityTypes<T>

Check GetDivyaangConcessionFactor in a way which includes all Disability Types and all Booking Classes at least once

Where T -> BlindType, OrthopaedicallyHandicappedType, CancerPatientType, TBPatientType

- Check GetDivyaangConncessionFactor() called by BlindType for ACFirstClass
  Input Provided
  `Divyaang::Blind::Type().GetDivyaangConcessionFactor(BookingClasses::ACFirstClass::Type())`
  Output
  `0.50`
- Check GetDivyaangConncessionFactor() called by BlindType for ExecutiveChairCar
  Input Provided
  `Divyaang::Blind::Type().GetDivyaangConcessionFactor( BookingClasses::ExecutiveChairCar::Type());`
  Output
  `0.75`
- Check GetDivyaangConncessionFactor() called by BlindType for FirstClass
  Input Provided
  `Divyaang::Blind::Type().GetDivyaangConcessionFactor( BookingClasses::AC2Tier::Type());`
  Output
  `0.75`
- Check GetDivyaangConncessionFactor() called by BlindType for AC2Tier
  Input Provided
  `Divyaang::Blind::Type().GetDivyaangConcessionFactor( BookingClasses::AC2Tier::Type());`
  Output
  `0.50`

- Check GetDivyaangConncessionFactor() called by BlindType for AC3Tier
  Input Provided
  ```
  Divyaang::Blind::Type().GetDivyaangConcessionFactor(BookingClasses::AC3Tier::
  Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetDivyaangConncessionFactor() called by OrthopaedicallyHandicappedType for AC Chair Car
  Input Provided
  ```
  Divyaang::OrthopaedicallyHandicapped::Type().GetDivyaangConcessionFactor(Book
  ingClasses::ACChairCar::Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetDivyaangConncessionFactor() called by CancerPatientType for Sleeper
  Input Provided
  ```
  Divyaang::CancerPatient::Type().GetDivyaangConcessionFactor(
  BookingClasses::Sleeper::Type()
  ```
  Output
  ```
  1.00
  ```
- Check GetDivyaangConncessionFactor() called by TBPatientType for Second Sitting
  Input Provided
  ```
  Divyaang::TBPatient::Type().GetDivyaangConcessionFactor(
  BookingClasses::SecondSitting::Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetName() called by BlindType
  Input Provided
  ```
  Divyaang::Blind::Type().GetName()
  ```
  Output
  ```
  "Blind"
  ```
- Check GetName() called by OrthopaedicallyHandicappedType
  Input Provided
  ```
  Divyaang::OrthopaedicallyHandicapped::Type().GetName()
  ```
  Output
  ```
  "Orthopaedically Handicapped"
  ```
- Check GetName() called by TBPatientType
  Input Provided
  ```
  Divyaang::CancerPatient::Type().GetName()
  ```
  Output
  ```
  "Cancer Patient"
  ```
- Check GetName() called by CancerPatientType
  Input Provided

`Divyaang::TBPatient::Type().GetName()`
Output
`"TB Patient"`

- Check whether output streaming operator works correctly
  Input Provided
  `const Divyaang::TBPatient &tTest = Divyaang::TBPatient::Type();`
  Output
  `"TB Patient"`

Date

- Check whether output streaming operator works correctly
  Input Provided
  `Date dTest(25,7,2021);`
  Output
  `"25/Jul/2021"`
- Check Date construction with numbers
  Input Provided
  `Date dateObj(1, 1, 2001);`
  Output
  ```
  dateObj.date_ == 1
  dateObj.month_ == static_cast<Month>(1)
  dateObj.year_ == 2001
  ```
- Check copy constructor for Date
  Input Provided
  ```
  Date dateObj(1, 1, 2001);
  Date dateObj2(dateObj);
  ```
  Output
  ```
  dateObj2.date_ == dateObj.date_
  dateObj2.month_ == dateObj.month_
  dateObj2.year_ == dateObj.year_
  ```

- Check whether GetDay() returns correct Day of the month
  Input Provided
  `Date dateObj(1, 1, 2001);`
  Output
  `1`
- Check whether GetMonth() returns correct Month
  Input Provided
  `Date dateObj(1, 1, 2001);`
  Output
  `1`
- Check whether GetYear() returns correct Year
  Input Provided

`Date dateObj(1, 1, 2001);`
Output
`2001`

- Test working of IsLeapYear()
  1. Non-Leap year not divisible by 100
     Input Provided
     `Date dateObj(1, 1, 2001);`
     Output
     false
  2. Non-leap year divisible by 100 but not by 400
     Input Provided
     `Date dateObjy2 = Date(1,1,1900);`
     Output
     false
  3. Leap year divisible by 400
     Input Provided
     `Date dateObjy = Date(1,1,2000);`
     Output
     true
  4. Leap year not divisible by 400
     Input Provided
     `Date dateObj3 = Date(1,1,2004);`
     Output
     true
- Check whether CalculateAge() returns correct Age based on this year (Input Provided is taken as first of January to ensure the golden does not change within 1 year)
  Input Provided
  `Date dateObjy2 = Date(1,1,1900);`
  Output
  `121`
- CalculateSpan() working correctly
  1. when leap years are present in the middle
     Input Provided
     ```
     Date dateObjy2 = Date(1,1,1900);
     Date dateObj(1, 1, 2001);
     dateObjy2.CalculateSpan(dateObj)
     ```
     Output
     `36890`
  2. when leap years are not present in the middle
     Input Provided
     `Date::Today().CalculateSpan(Date::Today())`
     Output
     `0`

- Check Date::Today()
  Gets tested in Application Test
- Check operator ==
  1. When matching
     Input Provided
     `(Date::Today()==Date::Today())`
     Output
     true
  2. When not matching
     Input Provided
     ```
     Date dateObj(1, 1, 2001);
     Date::Today()==dateObjy;
     ```
     Output
     false
- Test correct working of IsValid() for integer Input Provideds
  1. Invalid year (not in 1900-2099)
     Input Provided
     `IsValid(1,1,1000);`
     Output
     Exception Thrown : Invalid_Year
  2. Inavlid month (>12)
     Input Provided
     `IsValid(1,13,2000);`
     Output
     Exception Thrown : Invalid_Month
  3. Invalid month (<12)
     Input Provided
     `IsValid(1,-1,2000);`
     Output
     Exception thrown : Invalid_Month
     Exception Thrown :
  4. Invalid Day (<=0)
     Input Provided
     `IsValid(0,1,2000);`
     Output
     Exception Thrown : Invalid_Day
  5. Invalid Day (29 Days in February in a non-leap year)
     Input Provided
     `IsValid(29,2,2001);`
     Output
     Exception Thrown : Invalid_Day
  6. Valid Day (29 Days in February in a leap year)
     Input Provided
     `IsValid(29,2,2004);`

Output
Exception thrown : None

7. Invalid Day (>30 Days in a month with 30 days)
   Input Provided
   `IsValid(31,4,2001);`
   Output
   Exception Thrown : Invalid_Day
8. Invalid Day (>31 Days in a month with 31 days)
   Input Provided
   `IsValid(32,1,2001);`
   Output
   Exception Thrown : Invalid_Day
9. Valid Day
   Input Provided
   `IsValid(29,2,2004);`
   Output
   Valid Day Tested above

- Test correct working of IsValid() for string Input Provideds
  1. Invalid year (not in 1900-2099)
     Input Provided
     `IsValid("01/01/1000");`
     Output
     Exception thrown : Invalid_Year
  2. Invalid month (>12)
     Input Provided
     `IsValid("01/13/2000");`
     Output
     Exception thrown : Invalid_Month
  3. Invalid month (<12)
     Input Provided
     `IsValid("01/-1/2000");`
     Output
     Exception thrown : Invalid_Month
  4. Invalid Day (<=0)
     Input Provided
     `IsValid("00/01/2000");`
     Output
     Exception thrown : Invalid_Day
  5. Invalid Day (29 Days in February in a non-leap year)
     Input Provided
     `IsValid("29/02/2001");`
     Output
     Exception thrown : Invalid_Day

6. Invalid Day (>30 Days in a month with 30 days)
   Input Provided
   `IsValid("31/04/2001");`
   Output
   Exception thrown : Invalid_Day
7. Valid Day (29 Days in February in a leap year)
   Input Provided
   `IsValid("29/02/2004");`
   Output
   Exception thrown : None
8. Invalid Day (>31 Days in a month with 31 days)
   Input Provided
   `IsValid("32/01/2001");`
   Output
   Exception thrown : Invalid_Day
9. Valid Day
   Input Provided
   `IsValid("29/02/2004");`
   Output
   Valid Day tested above
10. Invalid Format (Not DD/MM/YYYY format with more characters)
    Input Provided
    `IsValid("323/01/2001");`
    Output
    Exception thrown : Invalid_Format
11. Invalid Format (Not DD/MM/YYYY format with less characters)
    Input Provided
    `IsValid("1/1/2001");`
    Output
    Exception thrown : Invalid_Format
12. Invalid Format (Non numeric characters present)
    Input Provided
    `IsValid("a3/1/2001");`
    Output
    Exception thrown : Invalid_Format
13. Invalid Format ('/' not present/replaced)
    Input Provided
    `IsValid("31@1/2001");`
    Output
    Exception thrown : Invalid_Format

- Correct working of GetDate()
  1. Valid Date - string
     Input Provided
     `GetDate("01/01/2001");`

```
   Output
   Date(1,1,2001)
2. Invalid Date - string
   Input Provided
   GetDate("1/1/200");
   Output
   Exception : Bad_Date
3. Valid Date - Numbers
   Input Provided
   GetDate(1,1,2001);
   Output
   Date(1,1,2001)
4. Invalid Date - Numbers
   Input Provided
   GetDate(50,1,1000);
   Output
   Exception : Bad_Date
```

Concessions

We do not check GetConcessions() for every pair.
Instead, we ensure all BookingClasses and Booking Types including all the different subtypes of Divyaang are covered.

- Check GetConcessions for Blind Type and ACFirstClass
  Input Provided
  ```
  Passenger blind =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"12345");

  DivyaangConcession::Type().GetConcessionFactor(blind,
  BookingClasses::ACFirstClass::Type())
  ```
  Output
  ```
  0.5
  ```
- Check GetConcessions for Blind Type and ExecutiveChairCar
  Input Provided
  ```
  Passenger blind =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"12345");
  DivyaangConcession::Type().GetConcessionFactor(blind,
  BookingClasses::ExecutiveChairCar::Type());
  ```
  Output

0.75

- Check GetConcessions for Blind Type and FirstClass
  Input Provided
  ```
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"12345");
  DivyaangConcession::Type().GetConcessionFactor(blind,
  BookingClasses::FirstClass::Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetConcessions for Blind Type and AC2Tier
  Input Provided
  ```
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"12345");
  DivyaangConcession::Type().GetConcessionFactor(blind,
  BookingClasses::AC2Tier::Type());
  ```
  Output
  ```
  0.50
  ```
- Check GetConcessions for Blind Type and AC3Tier
  Input Provided
  ```
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"12345");
  DivyaangConcession::Type().GetConcessionFactor(blind,
  BookingClasses::AC3Tier::Type());
  ```
  Output
  ```
  0.75
  ```

- Check GetConcessions for OrthopaedicallyHandicappedType and ACChairCar
  Input Provided
  ```
  Passenger oh =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::OrthopaedicallyHandicapped::Type(),"12345");
  DivyaangConcession::Type().GetConcessionFactor(oh,
  BookingClasses::ACChairCar::Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetConcessions for CancerPatientType and Sleeper
  Input Provided
  ```
  Passenger cp =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
  ```

```
der::Female::Type(),"123456789123","0123456789",&Divyaang::CancerPatient::Typ
e(),"12345");
DivyaangConcession::Type().GetConcessionFactor(cp,
BookingClasses::Sleeper::Type());
```
Output
```
1.00
```

- Check GetConcessions for TBPatientType and Second Sitting
  Input Provided
  ```
  Passenger tb =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
  der::Female::Type(),"123456789123","0123456789",&Divyaang::TBPatient::Type(),
  "12345");
  DivyaangConcession::Type().GetConcessionFactor(tb,
  BookingClasses::SecondSitting::Type());
  ```
  Output
  ```
  0.75
  ```
- Check GetConcessions for General Booking
  Input Provided
  ```
  GeneralConcession::Type().GetConcessionFactor();
  ```
  Output
  ```
  0.0
  ```

- Check GetConcessions for Ladies Booking
  Input Provided
  ```
  LadiesConcession::Type().GetConcessionFactor(p2);
  ```
  Output
  ```
  0.0
  ```

- Check GetConcessions for Female Senior Citizen
  Input Provided
  ```
  Passenger p2 =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
  der::Female::Type(),"123456789123","0123456789");
  SeniorCitizenConcession::Type().GetConcessionFactor(p2,
  Gender::Female::Type());
  ```
  Output
  ```
  0.5
  ```
- Check GetConcessions for Male Senior Citizen
  Input Provided
  ```
  Passenger p2 =
  Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
  der::Female::Type(),"123456789123","0123456789");
  ```

```
SeniorCitizenConcession::Type().GetConcessionFactor(p2,
Gender::Male::Type());
Output
0.4
```

GeneralConcession

Testing is a subset of Concessions Testing (included there)

LadiesConcession

Testing is a subset of Concessions Testing (included there)

DivyaangConcession

Testing is a subset of Concessions Testing (included there)

SeniorCitizenConcession

Testing is a subset of Concessions Testing (included there)

Application Test

To be done on _DEBUG mode

- Test CONSTRUCTOR for all valid Classes
- Test DESTRUCTOR for all valid Classes
- Test COPY CONSTRUCTOR wherever valid
- Test that program throws expected Exceptions when needed
- Test if all Bookings are executed correctly
- Test Singleton Nature for all Singletons
- Test if List of Bookings is printed correctly