Neha Dalmia 19CS30055

TEST PLAN

Unit Test

BookingClasses

- Check for correct working of polymorphic hierarchy
- Check for correct working of output streaming operator

BookingClassTypes<T>

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

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

BookingCategory

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

BookingCategoryTypes<T>

- Check for correct 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
- Check for correct 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
- Check for correct 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
- Check for correct 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
- Check for correct 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
- Check for correct 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
- Check for correct working of output streaming operator

Booking

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

BookingTypes<T>

- Check for correct working of BookingCategory::General::CheckEligibility() (One valid and one invalid case)
- Check for correct working of BookingCategory::Ladies::CheckEligibility() (One valid and one invalid case)
- Check for correct working of BookingCategory::SeniorCitizen::CheckEligibility()(One valid and one invalid case)
- Check for correct working of BookingCategory::DivyaangCat::CheckEligibility()(One valid and one invalid case)

- Check for correct working of BookingCategory::Tatkal::CheckEligibility()(One valid and one invalid case)
- Check for correct working of BookingCategory::PremiumTatkal:CheckEligibility()(One valid and one invalid case)

- Check for correct working of BookingCategory::Ladies::ComputeFare()
- Check for correct working of BookingCategory::General::ComputeFare()
- Check for correct working of BookingCategory::General::ComputeFare()
- Check for correct working of BookingCategory::SeniorCitizen::ComputeFare()
- Check for correct working of BookingCategory::SeniorCitizen::ComputeFare()
- Check for correct working of BookingCategory::DivyaangCat::ComputeFare()
- Check for correct working of BookingCategory::DivyaangCat::ComputeFare()
- Check for correct working of BookingCategory::Tatkal::ComputeFare()
- Check for correct working of BookingCategory::Tatkal::ComputeFare()
- Check for correct working of BookingCategory::PremiumTatkal:ComputeFare()
- Check for correct working of BookingCategory::PremiumTatkal:ComputeFare()

Divyaang

- Check for correct working of polymorphic hierarchy
- Check for correct working of output streaming operator

DisabilityTypes<T>

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

Where T -> BlindType, OrthopaedicallyHandicappedType, CancerPatientType, TBPatientType

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

Gender

-
- Check working of polymorphic hierarchy
- Check for correct working of output streaming operator

GenderTypes<T>

- Check GetName() for Male
- Check GetName() for Female
- Check GetTitle() for Male
- Check GetTitle() for Female
- Check for correct working of output stream operator

Name

- Check for correct working of output streaming operator

Date

- Check for correct working of output streaming operator
- Check Date construction with numbers
- Check Date construction with string
- Check copy constructor for Date
- Check if GetDay() returns correct Day of the Month
- Check if GetMonth() returns correct Month
- Check if GetYear() returns correct Year
- Check 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
- Check 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
- Check Date::Today()
- Check operator ==
    1. When matching
    2. When not matching
- Check for correct 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
- Check for correct 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

Station

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

Railways

- Check if all correct stations are stored in list of Stations
- Check if sDistStations has correct distance between stations (matching with Golden Output)
- Check if testObj.GetDistance() returns correct distance between stations (matching with sDistStations)
- Check for symmetric ordering of Stations
- Check 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
- Check working for GetDistance
    1. Throws Exception when queried with the same stations
    2. Throws Exception when queried with station not in database

Concessions

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

- Check Get Concessions for Blind Type and ACFirstClass
- Check Get Concessions for Blind Type and ExecutiveChairCar
- Check Get Concessions for Blind Type and FirstClass
- Check Get Concessions for Blind Type and AC2Tier
- Check Get Concessions for Blind Type and ExecutiveChairCar
- Check Get Concessions for Blind Type and AC3Tier
- Check Get Concessions for OrthopaedicallyHandicapped Type and AC Chair Car
- Check Get Concessions for Cancer PatientType and Sleeper
- Check Get Concessions for TBType and Second Sitting
- Check Get Concessions for Ladies Booking
- Check Get Concessions for female Senior Citizen
- Check 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)

SeniorCitizenConcession

Testing is a subset of Concessions Testing (included there)

DivyaangConcession

Testing is a subset of Concessions Testing (included there)

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

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

Neha Dalmia 19CS30055

TEST SUITE

Unit Test

BookingClasses

- Check for correct working of polymorphic hierarchy from return value of GetName()
  Test Input:
  ```
  const BookingClasses &obj = BookingClasses::AC3Tier::Type();
  ```
  Golden Output
  ```
  obj.GetName()=="AC 3 Tier"
  ```
- Check for correct working of Output streaming operator
  Test Input
  ```
  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

- Check for correct working of all simple member functions
  - GetLoadFactor()
    Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
6.5	5.0	4.0	3.0	2.5	2.0	1.0	0.5

  - GetName()
    Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
“AC First Class”	“Executive Chair Car”	“AC 2 Tier”	“First Class”	“AC 3 Tier”	“AC Chair Car”	“Sleeper”	“Second Sitting”

  - IsAC()
    Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
true	true	true	false	true	true	false	false

- IsLuxury()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
true	true	false	true	false	false	false	false

- IsSitting()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
false	true	false	false	false	true	false	true

- GetNumberOfTiers()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
2	0	2	2	3	0	3	0

- GetReservationCharge()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
60	60	50	50	40	40	20	15

- GetTatkalLoadFactor()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	ExecutiveChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.1

- GetMinTatkalCharge()

Table of Golden Outputs for the 8 sub-types:

ACFirstClass	Executive ChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
400	400	400	400	300	125	100	10

- GetMaxTatkalCharge()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	Executive ChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
500	500	500	500	400	225	200	15

- GetMinTatkalDist()
  Table of Golden Outputs for the 8 sub-types:

ACFirstClass	Executive ChairCar	AC2Tier	FirstClass	AC3Tier	ACChairCar	Sleeper	SecondSitting
500	250	500	500	500	250	500	100

- Check for correct working of Output streaming operator
  Test Input:
  ```
  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

- Check for correct working of polymorphic hierarchy from return value of GetName()
  Test Input
  ```
  const BookingCategory &bTest = BookingCategory::Ladies::Type();
  ```
  Golden Output
  ```
  obj.GetName()=="Ladies"
  ```
- Check for correct working of Output streaming operator
  Test Input
  ```
  const BookingCategory &bTest = BookingCategory::Ladies::Type();
  ```
  Golden Output
  ```
  "Booking Category = Ladies"
  ```
- Check for correct working of ReserveInCategory(), i.e., whether it returns NULL/non-NULL appropriately
  1. Return pointer to a newly made Booking
     Test Input
     ```
     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

```
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>

- Check for correct working of BookingCategory::General::Eligibility()
  Common Test Inputs:

```
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 when Date of Reservation after Date of Booking
   Test Input

```
BookingCategory::General::Type().Eligibility(p2, Date::Today(),
Date::GetDate(3,5,1900));
```

Golden Output
Exception thrown : `Bad_Chronology`

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

- Check for correct working of BookingCategory::Ladies::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
     Test Input
     ```
     BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,1900));
     ```
     Golden Output
     Exception thrown : `Bad_Chronology`
  2. Exception when Date of Reservation more than an year before Booking
     Test Input
     ```
     BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,2025));
     ```
     Golden Output
     Exception thrown : `Bad_Chronology`
  3. Exception when passenger is Male more than 12 years of age
     Test Input
     ```
     BookingCategory::Ladies::Type().Eligibility(p1,Gender::Male::Type(),"12
     3456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(),
     Date::GetDate(3,6,2021));
     ```
     Golden Output
     Exception thrown : `Ineligible_Ladies_Category`
  4. No Exception when all cases above are dissatisfied
     Test Input
     ```
     BookingCategory::Ladies::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,6,2021));
     ```
     Golden Output
     Exception thrown : None
- Check for correct working of BookingCategory::SeniorCitizen::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
     Test Input
     ```
     BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,1900));
     ```
     Golden Output

Exception thrown : `Bad_Chronology`

2. Exception when Date of Reservation more than an year before Booking
   Test Input
   ```
   BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));
   ```
   Golden Output
   Exception thrown : `Bad_Chronology`
3. Exception when passenger is Male less than 60 years of age
   Test Input
   ```
   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));
   ```
   Golden Output
   Exception thrown : `Ineligible_SeniorCitizen_Category`
4. Exception when passenger is Female less than 58 years of age
   Test Input
   ```
   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));
   ```
   Golden Output
   Exception thrown : `Ineligible_SeniorCitizen_Category`
5. No Exception when all cases above are dissatisfied
   Test Input
   ```
   BookingCategory::SeniorCitizen::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,6,2021));
   ```
   Golden Output
   Exception thrown : None

- Check for correct working of BookingCategory::DivyaangCat::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
     Test Input
     ```
     BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,1900));
     ```
     Golden Output
     Exception thrown : `Bad_Chronology`
  2. Exception when Date of Reservation more than an year before Booking
     Test Input
     ```
     BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));
     ```
     Golden Output
     Exception thrown : `Bad_Chronology`
  3. Passenger with Divyaang ID and/or Divyaang ID absent
     Test Input

```
BookingCategory::DivyaangCat::Type().Eligibility(Passenger::GetPassenge
r(Name("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Female::Typ
e(),"123456789123","0123456789"), Date::Today(),
Date::GetDate(3,6,2021));
```

Golden Output
Exception thrown : Ineligible_Divyaang_Category

4. No Exception when all cases above are dissatisfied
   Test Input
   ```
   BookingCategory::DivyaangCat::Type().Eligibility(p2, Date::Today(),
   Date::GetDate(3,6,2021));
   ```
   Golden Output
   Exception thrown : None

- Check for correct working of BookingCategory::Tatkal::Eligibility()
  1. Exception when Date of Reservation after Date of Booking
     Test Input
     ```
     BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,1900));
     ```
     Golden Output
     Exception thrown : Bad_Chronology
  2. Exception when Date of Reservation more than an year before Booking
     Test Input
     ```
     BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,2025));
     ```
     Golden Output
     Exception thrown : Bad_Chronology
  3. Reservation done more than 1 day before actual booking timings
     Test Input
     ```
     BookingCategory::Tatkal::Type().Eligibility(Passenger::GetPassenger(Nam
     e("Priyanka","Chopra"),Date::GetDate(5,1,2020),Gender::Male::Type(),"12
     3456789123","0123456789",&Divyaang::Blind::Type(),"e"), Date::Today(),
     Date::GetDate(2,4,2022));
     ```
     Golden Output
     Exception thrown : Ineligible_Tatkal_Category
  4. No Exception when all cases above are dissatisfied
     Test Input
     ```
     BookingCategory::Tatkal::Type().Eligibility(p2, Date::Today(),
     Date::Today());
     ```
     Golden Output
     Exception thrown : None
- Check for correct working of BookingCategory::PremiumTatkal:Eligibility()
  1. Exception when Date of Reservation after Date of Booking
     Test Input
     ```
     BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(),
     Date::GetDate(3,5,1900));
     ```

Golden Output
Exception thrown : `Bad_Chronology`

2. Exception when Date of Reservation more than an year before Booking
   Test Input
   ```
   BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(), Date::GetDate(3,5,2025));
   ```
   Golden 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));
   ```
   Golden Output
   Exception thrown : `Ineligible_PremiumTatkal_Category`
4. No Exception when all cases above are dissatisfied
   Test Input
   ```
   BookingCategory::PremiumTatkal::Type().Eligibility(p2, Date::Today(), Date::Today());
   ```
   Golden Output
   Exception thrown : None

- Check for correct working of Output streaming operator
  Test Input
  ```
  const BookingCategory::DivyaangCat &dTest = BookingCategory::DivyaangCat::Type();
  ```
  Golden Output
  `"Booking Category = Divyaang"`

Booking

- Check for correct working of Reserve
  1. Booking correctly constructed for Ladies Booking
     Test Input
     ```
     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);
     ```
     Golden Output
     non-NULL Booking pointer pointing to a fully constructed Booking object

2. Booking correctly constructed for Tatkal Booking
   Test Input
   ```
   Passenger p2 =
   Passenger::GetPassenger(Name("Nick","Jonas"),Date::GetDate(5,1,1996),Ge
   nder::Male::Type(),"123456789123","0123456789");

   Booking::Reserve(Station::GetStation("Bangalore"),
   Station::GetStation("Chennai"), Date::Today(), Date::Today(),
   BookingClasses::ExecutiveChairCar::Type(),
   BookingCategory::Tatkal::Type(),p2);
   ```
   Golden Output
   non-NULL Booking pointer pointing to a fully constructed Booking object
3. Exception thrown when fromStation = toStation
   Test Input
   ```
   Passenger p1 =
   Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1
   988),Gender::Female::Type(),"123456789123","0123456789",&Divyaang::Blin
   d::Type(),"e");

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

- Check for correct working of Output streaming operator
  Test Input
  ```
  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);
  ```
  Golden 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>

- Check for correct working of BookingCategory::General::CheckEligibility() (One valid and one invalid case)
  Common Test Input

```
Passenger p1 = Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(12,12,1988),Gender::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
     Test Input

```
Booking::GeneralBooking::CheckEligibility(p1, BookingCategory::General::Type(), Date::Today(), Date::GetDate(2,5,2021))
```

     Golden Output

```
true
```

  2. Invalid Booking: Date of Booking is before Date of reservation
     Test Input

```
Booking::GeneralBooking::CheckEligibility(p1, BookingCategory::General::Type(), Date::Today(), Date::GetDate(3,5,1900))
```

     Golden Output
     Exception thrown: `Bad_Chronology`

- Check for correct working of BookingCategory::Ladies::CheckEligibility() (One valid and one invalid case)
  1. Valid Booking
     Test Input

```
Booking::LadiesBooking::CheckEligibility(p1, BookingCategory::Ladies::Type(), Date::Today(), Date::GetDate(2,5,2021))
```

     Golden Output

```
true
```

  2. Invalid Booking: Male of age > 12
     Test Input

```
Booking::LadiesBooking::CheckEligibility(p2, BookingCategory::Ladies::Type(), Date::Today(), Date::GetDate(2,5,2021))
```

     Golden Output

Exception thrown: `Ineligible_Ladies_Category`

- Check for correct working of BookingCategory::SeniorCitizen::CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Test Input

       ```
       Booking::SeniorCitizenBooking::CheckEligibility(Passenger::GetPassenger
       (Name("Jai","Shah"),Date::GetDate(5,1,1950),Gender::Female::Type(),"123
       456789123","0123456789",&Divyaang::Blind::Type(),"e"),
       BookingCategory::SeniorCitizen::Type(), Date::Today(),
       Date::GetDate(2,5,2021))
       ```

       Golden Output
       `true`
    2. Invalid Booking: Male of age < 60
       Test Input
       ```
       Booking::SeniorCitizenBooking::CheckEligibility(p2,
       BookingCategory::SeniorCitizen::Type(), Date::Today(),
       Date::GetDate(2,5,2021))
       ```
       Golden Output
       Exception thrown: `Ineligible_SeniorCitizen_Category`

- Check for correct working of BookingCategory::DivyaangCat::CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Test Input
       ```
       Booking::DivyaangBooking::CheckEligibility(p1,
       BookingCategory::DivyaangCat::Type(), Date::Today(),
       Date::GetDate(2,5,2021))
       ```
       Golden Output
       `true`
    2. Invalid Booking: No disability in passenger
       Test Input
       ```
       Booking::DivyaangBooking::CheckEligibility(p2,
       BookingCategory::DivyaangCat::Type(), Date::Today(),
       Date::GetDate(2,5,2021))
       ```
       Golden Output
       Exception thrown: `Ineligible_Divyaang_Category`
- Check for correct working of BookingCategory::Tatkal::CheckEligibility()(One valid and one invalid case)
    1. Valid Booking
       Test Input
       ```
       Booking::TatkalBooking::CheckEligibility(p1,
       BookingCategory::Tatkal::Type(), Date::Today(), Date::Today())
       ```
       Golden Output

```
true
```

2. Invalid Booking: Date of booking is not within 1 day of date of reservation
   Test Input
   ```
   Booking::TatkalBooking::CheckEligibility(p1,
   BookingCategory::Tatkal::Type(), Date::Today(),
   Date::GetDate(2,7,2021))
   ```
   Golden Output
   Exception thrown: `Ineligible_Tatkal_Category`

- Check for correct working of BookingCategory::PremiumTatkal:CheckEligibility()(One valid and one invalid case)
  1. Valid Booking
     Test Input
     ```
     Booking::PremiumTatkalBooking::CheckEligibility(p1,
     BookingCategory::Tatkal::Type(), Date::Today(), Date::Today())
     ```
     Golden Output
     ```
     true
     ```
  2. Invalid Booking
     Test Input
     ```
     Booking::TatkalBooking::CheckEligibility(p1,
     BookingCategory::PremiumTatkal::Type(), Date::Today(),
     Date::GetDate(2,7,2021))
     ```
     Golden Output
     Exception thrown: `Ineligible_Tatkal_Category`
- Check for correct working of BookingCategory::Ladies::ComputeFare()
  Test Input
  ```
  b4 =
  Booking::Reserve(Station::GetStation("Kolkata"),Station::GetStation("Delhi"),
  book,reser, BookingClasses::AC2Tier::Type(),
  BookingCategory::Ladies::Type(),p11);

  b4->ComputeFare();
  ```
  Golden Output
  2994
- Check for correct working of BookingCategory::General::ComputeFare()
  Test Input
  ```
  b5 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::AC3Tier::Type(),
  BookingCategory::General::Type(),p11);

  b5->ComputeFare();
  ```
  Golden Output
  1849

- Check for correct working of BookingCategory::General::ComputeFare()
  Test Input
  ```
  b6 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::ACFirstClass::Type(),
  BookingCategory::General::Type(),p11);

  b6->ComputeFare();
  ```
  Golden Output
  4763
- Check for correct working of BookingCategory::SeniorCitizen::ComputeFare()
  Test Input
  ```
  b7 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::AC3Tier::Type(),
  BookingCategory::SeniorCitizen::Type(),p21);

  b7->ComputeFare();
  ```
  Golden Output
  1125
- Check for correct working of BookingCategory::SeniorCitizen::ComputeFare()
  Test Input
  ```
  b8 =
  Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
  ook,reser, BookingClasses::ACFirstClass::Type(),
  BookingCategory::SeniorCitizen::Type(),p11);

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

  b9->ComputeFare();
  ```
  Golden Output
  492
- Check for correct working of BookingCategory::DivyaangCat::ComputeFare()
  Test Input

```
b10 =
Booking::Reserve(Station::GetStation("Mumbai"),Station::GetStation("Delhi"),b
ook,reser, BookingClasses::ACFirstClass::Type(),
BookingCategory::DivyaangCat::Type(),p21);

b10->ComputeFare();
```

Golden Output

2411

- Check for correct working of BookingCategory::Tatkal::ComputeFare()

  Test Input

```
b11 =
Booking::Reserve(Station::GetStation("Delhi"),Station::GetStation("Mumbai"),b
ook,reser, BookingClasses::AC3Tier::Type(),
BookingCategory::Tatkal::Type(),p11);

b11->ComputeFare();
```

  Golden Output

  2249

- Check for correct working of BookingCategory::Tatkal::ComputeFare()

  Test Input

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

b12->ComputeFare();
```

  Golden Output

  1198

- Check for correct working of BookingCategory::PremiumTatkal:ComputeFare()

  Test Input

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

b13->ComputeFare();
```

  Golden Output

  1198

- Check for correct working of BookingCategory::PremiumTatkal:ComputeFare()

  Test Input

```
b14 =
Booking::Reserve(Station::GetStation("Delhi"),Station::GetStation("Mumbai"),b
```

```
ook,reser, BookingClasses::AC3Tier::Type(),
BookingCategory::PremiumTatkal::Type(),p11);

b14->ComputeFare();
```

Golden Output
2649

Divyaang

- Check for correct working of polymorphic hierarchy from return value of GetName()
  Test Input
  ```
  const Divyaang &obj = Divyaang::Blind::Type();
  ```
  Golden Output
  ```
  obj.GetName()=="Blind"
  ```
- Check for correct working of Output streaming operator
  Test Input
  ```
  const Divyaang &dTest = Divyaang::Blind::Type();
  ```
  Golden 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
  Test Input
  ```
  Divyaang::Blind::Type().GetDivyaangConcessionFactor(BookingClasses::ACFirstClass::Type())
  ```
  Golden Output
  ```
  0.50
  ```
- Check GetDivyaangConncessionFactor() called by BlindType for ExecutiveChairCar
  Test Input
  ```
  Divyaang::Blind::Type().GetDivyaangConcessionFactor(
  BookingClasses::ExecutiveChairCar::Type());
  ```
  Golden Output
  ```
  0.75
  ```
- Check GetDivyaangConncessionFactor() called by BlindType for FirstClass
  Test Input
  ```
  Divyaang::Blind::Type().GetDivyaangConcessionFactor(
  BookingClasses::AC2Tier::Type());
  ```
  Golden Output
  ```
  0.75
  ```
- Check GetDivyaangConncessionFactor() called by BlindType for AC2Tier

Test Input

```
Divyaang::Blind::Type().GetDivyaangConcessionFactor(
BookingClasses::AC2Tier::Type());
```

Golden Output

```
0.50
```

- Check GetDivyaangConncessionFactor() called by BlindType for AC3Tier

  Test Input

  ```
  Divyaang::Blind::Type().GetDivyaangConcessionFactor(BookingClasses::AC3Tier::
  Type());
  ```

  Golden Output

  ```
  0.75
  ```

- Check GetDivyaangConncessionFactor() called by OrthopaedicallyHandicappedType for AC Chair Ca

  Test Input

  ```
  Divyaang::OrthopaedicallyHandicapped::Type().GetDivyaangConcessionFactor(Book
  ingClasses::ACChairCar::Type());
  ```

  Golden Output

  ```
  0.75
  ```

- Check GetDivyaangConncessionFactor() called by CancerPatientType for Sleeper

  Test Input

  ```
  Divyaang::CancerPatient::Type().GetDivyaangConcessionFactor(
  BookingClasses::Sleeper::Type()
  ```

  Golden Output

  ```
  1.00
  ```

- Check GetDivyaangConncessionFactor() called by TBPatientType for Second Sitting

  Test Input

  ```
  Divyaang::TBPatient::Type().GetDivyaangConcessionFactor(
  BookingClasses::SecondSitting::Type());
  ```

  Golden Output

  ```
  0.75
  ```

- Check GetName() called by BlindType

  Test Input

  ```
  Divyaang::Blind::Type().GetName()
  ```

  Golden Output

  ```
  "Blind"
  ```

- Check GetName() called by OrthopaedicallyHandicappedType

  Test Input

  ```
  Divyaang::OrthopaedicallyHandicapped::Type().GetName()
  ```

  Golden Output

  ```
  "Orthopaedically Handicapped"
  ```

- Check GetName() called by TBPatientType

  Test Input

  ```
  Divyaang::CancerPatient::Type().GetName()
  ```

  Golden Output

  ```
  "Cancer Patient"
  ```

- Check GetName() called by CancerPatientType
  Test Input
  ```
  Divyaang::TBPatient::Type().GetName()
  ```
  Golden Output
  ```
  "TB Patient"
  ```
- Check for correct working of Output streaming operator
  Test Input
  ```
  const Divyaang::TBPatient &tTest = Divyaang::TBPatient::Type();
  ```
  Golden Output
  ```
  "TB Patient"
  ```

Gender

- Check working of polymorphic hierarchy from return value of GetName()
  Test Input
  ```
  const Gender &obj = Gender::Male::Type();
  ```
  Golden Output
  ```
  "Male"
  ```
- Check for correct working of Output streaming operator
  Test Input
  ```
  const Gender &gTest = Gender::Male::Type();
  ```
  Golden Output
  ```
  "Male"
  ```

GenderTypes<T>

- Check GetName() for Gender::Male
  Test Input
  ```
  Gender::Male::Type().GetName()
  ```
  Golden Output
  ```
  "Male"
  ```
- Check GetName() for Gender::Female
  Test Input
  ```
  Gender::Female::Type().GetName()
  ```
  Golden Output
  ```
  "Female"
  ```
- Check GetTitle() for Gender::Male
  Test Input
  ```
  Gender::Male::Type().GetTitle()
  ```
  Golden Output
  ```
  "Mr."
  ```
- Check GetTitle() for Gender::Female
  Test Input
  ```
  Gender::Female::Type().GetTitle()
  ```
  Golden Output
  ```
  "Ms."
  ```

- Check for correct working of Output stream operator
  Test Input
  ```
  const Gender::Female &fTest = Gender::Female::Type();
  ```
  Golden Output
  ```
  "Female"
  ```

Name

- Check for correct working of Output streaming operator
  Test Input
  ```
  Name n = Name("Bob","Dylan");
  ```
  Golden Output
  ```
  "Bob Dylan"
  ```

Date

- Check for correct working of Output streaming operator
  Test Input
  ```
  Date dTest(25,7,2021);
  ```
  Golden Output
  ```
  "25/Jul/2021"
  ```
- Check Date construction with numbers
  Test Input
  ```
  Date dateObj(1, 1, 2001);
  ```
  Golden Output
  ```
  dateObj.date_ == 1
  dateObj.month_ == static_cast<Month>(1)
  dateObj.year_ == 2001
  ```
- Check copy constructor for Date
  Test Input
  ```
  Date dateObj(1, 1, 2001);
  Date dateObj2(dateObj);
  ```
  Golden Output
  ```
  dateObj2.date_ == dateObj.date_
  dateObj2.month_ == dateObj.month_
  dateObj2.year_ == dateObj.year_
  ```

- Check if GetDay() returns correct Day of the month
  Test Input
  ```
  Date dateObj(1, 1, 2001);
  ```
  Golden Output
  ```
  1
  ```
- Check if GetMonth() returns correct Month
  Test Input
  ```
  Date dateObj(1, 1, 2001);
  ```

Golden Output

```
1
```

- Check if GetYear() returns correct Year
  Test Input

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

  Golden Output

  ```
  2001
  ```

- Check for working of IsLeapYear()
  1. Non-Leap year not divisible by 100
     Test Input

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

     Golden Output
     false
  2. Non-leap year divisible by 100 but not by 400
     Test Input

     ```
     Date dateObjy2 = Date(1,1,1900);
     ```

     Golden Output
     false
  3. Leap year divisible by 400
     Test Input

     ```
     Date dateObjy = Date(1,1,2000);
     ```

     Golden Output
     true
  4. Leap year not divisible by 400
     Test Input

     ```
     Date dateObj3 = Date(1,1,2004);
     ```

     Golden Output
     true
- Check if CalculateAge() returns correct Age based on this year (Test Input is taken as first of January to ensure the golden does not change within 1 year)
  Test Input

  ```
  Date dateObjy2 = Date(1,1,1900);
  ```

  Golden Output

  ```
  121
  ```

- CalculateSpan() working correctly
  1. when leap years are present in the middle
     Test Input

     ```
     Date dateObjy2 = Date(1,1,1900);
     Date dateObj(1, 1, 2001);
     dateObjy2.CalculateSpan(dateObj)
     ```

     Golden Output

     ```
     36890
     ```
  2. when leap years are not present in the middle
     Test Input

     ```
     Date::Today().CalculateSpan(Date::Today())
     ```

Golden Output

```
0
```

- Check Date::Today()
  Gets tested in Application Test
- Check operator==
  1. When matching
     Test Input

     ```
     (Date::Today()==Date::Today())
     ```

     Golden Output
     true
  2. When not matching
     Test Input

     ```
     Date dateObj(1, 1, 2001);
     Date::Today()==dateObjy;
     ```

     Golden Output
     false
- Check for correct validation by IsValid() for integer Test Inputs
  1. Invalid year (not in 1900-2099)
     Test Input

     ```
     IsValid(1,1,1000);
     ```

     Golden Output
     Exception Thrown : `Invalid_Year`
  2. Inavlid month (>12)
     Test Input

     ```
     IsValid(1,13,2000);
     ```

     Golden Output
     Exception Thrown : `Invalid_Month`
  3. Invalid month (<12)
     Test Input

     ```
     IsValid(1,-1,2000);
     ```

     Golden Output
     Exception thrown : `Invalid_Month`
     Exception Thrown :
  4. Invalid Day (<=0)
     Test Input

     ```
     IsValid(0,1,2000);
     ```

     Golden Output
     Exception Thrown : `Invalid_Day`
  5. Invalid Day (29 Days in February in a non-leap year)
     Test Input

     ```
     IsValid(29,2,2001);
     ```

     Golden Output
     Exception Thrown : `Invalid_Day`
  6. Valid Day (29 Days in February in a leap year)
     Test Input

`IsValid(29,2,2004);`

Golden Output

Exception thrown : None

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

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

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

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

```
Date(1,1,2001)
```

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

Station

- Check for correct working of Output streaming operator
  Test Input
  ```
  Station stationTest("Delhi");
  ```
  Golden 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 if GetName works correctly
  Test Input
  ```
  Station st5("Kolkata");
  st5.GetName();
  ```
  Golden Output
  ```
  "Kolkata"
  ```
- Check validity of Stations by IsValid()
  1. Empty Station Name
     Test Input
     ```
     IsValid("");
     ```
     Golden Output
     Exception thrown : `Bad_Station_Name`
  2. Station not present in DataBase
     Test Input
     ```
     IsValid("Jammu");
     ```
     Golden Output
     Exception thrown : `Bad_Station_Name`
- Check if GetDistance() throws Exceptions when asked for distances between same station
  Test Input
  ```
  Station::GetStation("Kolkata").GetDistance(Station::GetStation("Kolkata"));
  ```
  Golden Output
  Exception thrown : `Distance_Not_Defined`
- Check if GetStation() works correctly
  1. Exception thrown for invalid Station
     Test Input
     ```
     Station::GetStation("");
     ```
     Golden Output
     Exception thrown : `Bad_Station_Name`
  2. Returns correct station for valid station
     Test Input
     ```
     Station::GetStation("Kolkata");
     ```
     Golden Output
     ```
     Station("Kolkata");
     ```

Railways

- Check if all correct stations are stored in list of Stations
  Golden Data
  ```
  {"Bangalore", "Chennai", "Delhi", "Kolkata", "Mumbai"}
  ```
- Check if 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 if testObj.GetDistance() returns correct distance between stations (matching with sDistStations)
  Same Golden Output as above
- Check for symmetric ordering of Stations
  Same Golden Output as above
- Check working of IsValid()
  1. Duplicate Station in Stations database
     Golden Output
     Exception thrown : `Duplicate_Station`
  2. Same pair of stations with a given ordering in distances database not EXACTLY once
     Test Input
     Golden Output

Exception thrown : `Bad_Railways`

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

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

Check working for GetDistance

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

Concessions

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

- Check GetConcessions for Blind Type and ACFirstClass
  Test Input
  ```
  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())
  ```
  Golden Output
  `0.5`
- Check GetConcessions for Blind Type and ExecutiveChairCar
  Test Input
  ```
  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());
```

Golden Output

```
0.75
```

- Check GetConcessions for Blind Type and FirstClass

  Test Input

```
Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
der::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"123
45");
DivyaangConcession::Type().GetConcessionFactor(blind,
BookingClasses::FirstClass::Type());
```

  Golden Output

```
0.75
```

- Check GetConcessions for Blind Type and AC2Tier

  Test Input

```
Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
der::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"123
45");
DivyaangConcession::Type().GetConcessionFactor(blind,
BookingClasses::AC2Tier::Type());
```

  Golden Output

```
0.50
```

- Check GetConcessions for Blind Type and AC3Tier

  Test Input

```
Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
der::Female::Type(),"123456789123","0123456789",&Divyaang::Blind::Type(),"123
45");
DivyaangConcession::Type().GetConcessionFactor(blind,
BookingClasses::AC3Tier::Type());
```

  Golden Output

```
0.75
```

- Check GetConcessions for OrthopaedicallyHandicappedType and ACChairCar

  Test Input

```
Passenger oh =
Passenger::GetPassenger(Name("Priyanka","Chopra"),Date::GetDate(5,1,1950),Gen
der::Female::Type(),"123456789123","0123456789",&Divyaang::OrthopaedicallyHan
dicapped::Type(),"12345");
DivyaangConcession::Type().GetConcessionFactor(oh,
BookingClasses::ACChairCar::Type());
```

  Golden Output

```
0.75
```

- Check GetConcessions for CancerPatientType and Sleeper
  Test Input
  ```
  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());
  ```
  Golden Output
  ```
  1.00
  ```
- Check GetConcessions for TBPatientType and Second Sitting
  Test Input
  ```
  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());
  ```
  Golden Output
  ```
  0.75
  ```
- Check GetConcessions for General Booking
  Test Input
  ```
  GeneralConcession::Type().GetConcessionFactor();
  ```
  Golden Output
  ```
  0.0
  ```

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

- Check GetConcessions for Female Senior Citizen
  Test Input
  ```
  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());
  ```
  Golden Output
  ```
  0.5
  ```
- Check GetConcessions for Male Senior Citizen
  Test Input

```
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());
```

Golden Output

```
0.4
```

GeneralConcession

Testing is a subset of Concessions Testing (included there)

LadiesConcession

Testing is a subset of Concessions Testing (included there)

SeniorCitizenConcession

Testing is a subset of Concessions Testing (included there)

DivyaangConcession

Testing is a subset of Concessions Testing (included there)

Passenger

- Testing is a passenger is valid
    1. Error when both first and last names missing
       Test Input : `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
       Test Input `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
       Test Input: `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
       Test Input: `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

Test Input : `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
   Test Input: `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
   Test Input: `IsValid(Name("X","Y",""),Date::Today(),Gender::Male::Type(),"123456789999","12314a7890",NULL,"")`
   Golden Output: `Bad_Mobile_Number exception thrown`
8. Mobile Number is valid
   Test Input: `IsValid(Name("X","Y",""),Date::Today(),Gender::Male::Type(),"123456789999","",NULL,"")`
   Golden Output: `No exception thrown`
9. Error when Bad Age - Not born yet
   Test Input: `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
  Test Input:`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
  Test Input: `GetPassenger(Name("X","Y",""),Date::Today(),Gender::Male::Type(),"123456789999","1231478190")`
  Golden Output: `No exception thrown`
- Testing GetPasseneger - InValid Case
  Test Input: `GetPassenger(Name("","Y",""),Date::Today(),Gender::Male::Type(),"123456789999","123147890");`
  Golden Output: `Bad_Passenger exception thrown`
- Test Output streaming operator for Passenger
  Test Input:
  `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"`

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