General Principles followed while formulating the test plan and test suites.

1. The testing phase is divided into 2 parts
    a. Unit Testing
    b. Application Testing

2. For both cases, first of all the "*happy*" paths are tested which includes exhaustive testing of all the methods (including static ones) of a particular class.

3. After that an exhaustive testing of the *"exception"* paths is done.

4. Whenever an exception is caught, and handled, the message "exception caught" is displayed on the screen followed by the exception

5. In most of the unit tests, nothing is printed if the test is successful.

6. However **bugs are never ignored**. In case of a bug, it is always reported.

7. After every test case, the golden output is given.

Class Booking and its hierarchy

```
Station s1 = Station :: createStation("Kolkata");
Station s2 = Station :: createStation("Delhi");
const Date& d1 = Date :: createDate(16, 3, 2001);
const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Date& d2 = Date :: createDate(8, 4, 2021);
const Date& d3 = Date :: createDate(9, 8, 2021);
const Date& d4 = Date :: createDate(12, 3, 2001);
const Date& d5 = Date :: createDate(3, 8, 2022);
const Date& d6 = Date :: createDate(15, 3, 2001);

try {
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), GeneralCategory :: Type(),
p1, d3);       // okay
    cout << "Booking Successfully done\n";
} catch(exception& e) {
    cout << "Error!\n";
    cout << e.what() << '\n';
}

try {
    const Booking* b2 = Booking :: ReserveBooking(s1, s2, d3,
BookingClass :: ACFirstClass :: Type(), GeneralCategory :: Type(),
p1, d2);       // date of booking> reservation
} catch(exception& e) {
    cout << "**Exception Caught!**\n";
    cout << e.what() << '\n';
}

try {
    const Booking* b3 = Booking :: ReserveBooking(s1, s2, d4,
BookingClass :: ACFirstClass :: Type(), GeneralCategory :: Type(),
p1, d6);       // not yet born!
} catch(exception& e) {
    cout << "**Exception Caught!**\n";
    cout << e.what() << '\n';
}

try {
    const Booking* b4 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), GeneralCategory :: Type(),
p1, d5);           //  > one year from booking date
} catch(exception& e) {
    cout << "**Exception Caught!**\n";
```

```
    cout << e.what() << '\n';
}

cout << "***TESTING FOR BOOKING IS DONE****\n";
```

****

Golden Output:

Booking Successfully done
Exception Caught!
Invalid date of booking!
Exception Caught!
Passenger is born after reservation date!
Exception Caught!
Invalid date of booking!
TESTING FOR BOOKING IS DONE*

****

```
Station s1 = Station :: createStation("Kolkata");
Station s2 = Station :: createStation("Delhi");
const Date& d1 = Date :: createDate(16, 3, 2001);
const Passenger p1 = Passenger :: createPassenger("Ankita", "",
"De", "134236789345", d1, Gender :: Female :: Type(),
"8279728914");
const Date& d2 = Date :: createDate(8, 4, 2021);
const Date& d3 = Date :: createDate(9, 8, 2021);

const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), LadiesCategory :: Type(),
p1, d3);
if(abs(b1->GetFare() -4844.5) >= 1e-3)
    cout << "Error in compute fare\n";

cout << "***TESTING FOR LADIES BOOKING IS DONE****\n";
```

****

Golden Output:

TESTING FOR GENERAL BOOKING IS DONE*

****

```
Station s1 = Station :: createStation("Kolkata");
Station s2 = Station :: createStation("Delhi");
const Date& d1 = Date :: createDate(16, 3, 1951);
const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Date& d2 = Date :: createDate(8, 4, 2021);
const Date& d3 = Date :: createDate(9, 8, 2021);

const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), SeniorCitizenCategory ::
Type(), p1, d3);
if(abs(b1->GetFare() -2930.9) >= 1e-3)
    cout << "Error in compute fare\n";

cout << "***TESTING FOR Senior Citizens BOOKING IS DONE****\n";

********************************************************************************
```

Golden Output :
TESTING FOR Senior Citizens BOOKING IS DONE*

****

```
Station s1 = Station :: createStation("Kolkata");
Station s2 = Station :: createStation("Delhi");
const Date& d1 = Date :: createDate(16, 3, 1951);
const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Date& d2 = Date :: createDate(8, 4, 2021);
const Date& d3 = Date :: createDate(9, 4, 2021);

const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), TatkalCategory :: Type(),
p1, d3);
if(abs(b1->GetFare() -5344.5) >= 1e-3)
    cout << "Error in compute fare\n";

cout << "***TESTING FOR TATKAL BOOKING IS DONE****\n";
********************************************************************************
```

Golden Output:

TESTING FOR TATKAL BOOKING IS DONE*

****

```
Station s1 = Station :: createStation("Kolkata");
Station s2 = Station :: createStation("Delhi");
const Date& d1 = Date :: createDate(16, 3, 1951);
const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Date& d2 = Date :: createDate(8, 4, 2021);
const Date& d3 = Date :: createDate(9, 4, 2021);

const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), PremiumTatkalCategory ::
Type(), p1, d3);
if(abs(b1->GetFare() -5286.1) >= 1e-3)
    cout << "Error in compute fare\n";

cout << "***TESTING FOR PREMIUM TATKAL BOOKING IS DONE****\n";
```

****

Golden Output:

TESTING FOR PREMIUM TATKAL BOOKING IS DONE*

****

Class BookingClass and its hierarchy

```
const BookingClass& b = BookingClass :: SecondSitting :: Type();
// meyers singleton
cout << b;                                          // operator <<

if(abs(b.GetReservationCharges() - 15.0) >= 1e-3)
    cout << "Error in GetReservationCharges\n";
if(abs(b.GetLoadFactor() - 0.60) >= 1e-3)
    cout << "Error in GetLoadFactor\n";
if(!b.IsSitting())
    cout << "Error in Getting sit/sleep info\n";
if(b.GetNumberOfTiers() != 3)
    cout << "Error in No of tiers\n";
if(b.GetName() != "Second Sitting")
    cout << "Error in get reservation charges\n";
if(b.IsAC())
    cout << "Error in get AC Info\n";
if(b.IsLuxury())
    cout << "Error in Luxury\n";
if(abs(b.GetMinDistanceForCharge() - 100.0) >= 1e-3)
    cout << "Error in GetMinDistanceForCharge\n";
if(abs(b.GetMaxTatkalCharges() - 15.00) >= 1e-3)
    cout << "Error in GetMaxTatkalCharges\n";
if(abs(b.GetMinTatkalCharges() - 10.00) >= 1e-3)
    cout << "Error in GetMinTatkalCharges\n";
if(abs(b.GetTatkalLoad() - 0.1) >= 1e-3)
    cout << "Error in GetTatkalLoad\n";
if(abs(b.GetPremiumTatkalLoad() - 0.2) >= 1e-3)
    cout << "Error in GetPremiumTatkalLoad\n";

SecondSitting :: changeLuxury(true);
if(!b.IsLuxury())
    cout << "Error in changing Luxury\n";
SecondSitting :: changeLuxury(false);

SecondSitting :: changeLoadFactor(0.70);
if(abs(b.GetLoadFactor() - 0.70) >= 1e-3)
    cout << "Error in changeLoadFactor\n";
SecondSitting :: changeLoadFactor(0.6);

SecondSitting :: changeReservationCharges(20.00);
if(abs(b.GetReservationCharges() - 20.0) >= 1e-3)
```

```
    cout << "Error in changeReservationCharges\n";
SecondSitting :: changeReservationCharges(15.00);

SecondSitting :: changeTatkalLoad(0.2);
if(abs(b.GetTatkalLoad() - 0.2) >= 1e-3)
    cout << "Error in change TatkalLoad\n";
SecondSitting :: changeTatkalLoad(0.1);

SecondSitting :: changeMinTatkalCharge(20.00);
if(abs(b.GetMinTatkalCharges() - 20.00) >= 1e-3)
    cout << "Error in GetMinTatkalCharges\n";
SecondSitting :: changeMinTatkalCharge(10.00);

SecondSitting :: changeMaxTatkalCharge(20.00);
if(abs(b.GetMaxTatkalCharges() - 20.00) >= 1e-3)
    cout << "Error in changeMaxTatkalCharge\n";
SecondSitting :: changeMaxTatkalCharge(15.00);

cout << "****TESTING FOR BOOKINGCLASS IS COMPLETE****\n";
```

****

Golden output:

****TESTING FOR BOOKINGCLASS IS COMPLETE****

AC First Class

: AC: YES

: Bunks: 2

: Luxury: YES

****

```
const BookingClasses<T>& b = Type();            // meyers
singleton
cout << b;                                      // operator <<

if(abs(b.GetReservationCharges() - sReservationCharges) >= 1e-3)
// checking the get methods
    cout << "Error in GetReservationCharges\n";
if(abs(b.GetLoadFactor() - sLoadFactor) >= 1e-3)
    cout << "Error in GetLoadFactor\n";
if(b.IsSitting() != sIsSitting)
    cout << "Error in Getting sit/sleep info\n";
if(b.GetNumberOfTiers() != sNoOfTiers)
    cout << "Error in No of tiers\n";
if(b.GetName() != sName)
    cout << "Error in get reservation charges\n";
if(b.IsAC() != sIsAC)
    cout << "Error in get AC Info\n";
if(b.IsLuxury() != sIsLuxury)
    cout << "Error in Luxury\n";
```

```
if(abs(b.GetMinDistanceForCharge() - sMinDistanceForCharge) >=
1e-3)
    cout << "Error in GetMinDistanceForCharge\n";
if(abs(b.GetMaxTatkalCharges() - sMaxTatkalCharges) >= 1e-3)
    cout << "Error in GetMaxTatkalCharges\n";
if(abs(b.GetMinTatkalCharges() - sMinTatkalCharges) >= 1e-3)
    cout << "Error in GetMinTatkalCharges\n";
if(abs(b.GetTatkalLoad() - sTatkalLoad) >= 1e-3)
    cout << "Error in GetTatkalLoad\n";
if(abs(b.GetPremiumTatkalLoad() - 2*sTatkalLoad) >= 1e-3)
    cout << "Error in GetPremiumTatkalLoad\n";

bool prev = sIsLuxury;
BookingClasses<T> :: changeLuxury(true);                          //
changing checking and resetting!
if(!b.IsLuxury())
    cout << "Error in changing Luxury\n";
BookingClasses<T> :: changeLuxury(prev);

double prev1 = sLoadFactor;
BookingClasses<T> :: changeLoadFactor(0.70);
if(abs(b.GetLoadFactor() - 0.70) >= 1e-3)
    cout << "Error in changeLoadFactor\n";
BookingClasses<T> :: changeLoadFactor(prev1);

double prev2 = sReservationCharges;
BookingClasses<T> :: changeReservationCharges(20.00);
if(abs(b.GetReservationCharges() - 20.0) >= 1e-3)
    cout << "Error in changeReservationCharges\n";
BookingClasses<T> :: changeReservationCharges(prev2);

double prev3 = sTatkalLoad;
BookingClasses<T> :: changeTatkalLoad(0.2);
if(abs(b.GetTatkalLoad() - 0.2) >= 1e-3)
    cout << "Error in change TatkalLoad\n";
BookingClasses<T> :: changeTatkalLoad(prev3);

double prev4 = sMinTatkalCharges;
BookingClasses<T> :: changeMinTatkalCharge(20.00);
if(abs(b.GetMinTatkalCharges() - 20.00) >= 1e-3)
    cout << "Error in GetMinTatkalCharges\n";
BookingClasses<T> :: changeMinTatkalCharge(prev4);

double prev5 = sMaxTatkalCharges;
BookingClasses<T> :: changeMaxTatkalCharge(20.00);
if(abs(b.GetMaxTatkalCharges() - 20.00) >= 1e-3)
    cout << "Error in changeMaxTatkalCharge\n";
BookingClasses<T> :: changeMaxTatkalCharge(prev5);
```

```
cout << "****TESTING FOR " << sName << " IS COMPLETE****\n";
```

****

Golden Output:

```
****TESTING FOR AC First Class IS COMPLETE****
Executive Chair Car
: AC: YES
: Bunks: 0
: Luxury: YES
****TESTING FOR Executive Chair Car IS COMPLETE****
AC 2 Tier
: AC: YES
: Bunks: 2
: Luxury: NO
****TESTING FOR AC 2 Tier IS COMPLETE****
AC 3 Tier
: AC: YES
: Bunks: 3
: Luxury: NO
****TESTING FOR AC 3 Tier IS COMPLETE****
First Class
: AC: NO
: Bunks: 2
: Luxury: YES
****TESTING FOR First Class IS COMPLETE****
AC Chair Car
: AC: YES
: Bunks: 0
: Luxury: NO
****TESTING FOR AC Chair Car IS COMPLETE****
Sleeper
: AC: NO
: Bunks: 3
: Luxury: NO
****TESTING FOR Sleeper IS COMPLETE****
Second Sitting
: AC: NO
: Bunks: 3
: Luxury: NO
```

****

Class BookingCategory and its hierarchy

```
const BookingCategory& b = LadiesCategory :: Type();

cout << b;                  // testing the operator <<

if(b.getName() != "Ladies Category")            // the getName
test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);
const Date d3 = Date :: createDate(16, 3, 1956);
const Date d4 = Date :: createDate(13, 04, 2001);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");
const Passenger p3 = Passenger :: createPassenger("Mamata", "",
"Hajra", "134236789345", d3, Gender :: Female :: Type(),
"9921133112");
const Passenger p4 = Passenger :: createPassenger("Ankita", "",
"", "123456737123", d4, Gender :: Female :: Type());

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p3,
BookingClass :: SecondSitting :: Type());       // for p3 //
checking if polymorphic binding is done or not!
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p4,
BookingClass :: ACFirstClass :: Type());            // for p4
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << e.what() << '\n';
}
```

```
if(LadiesCategory :: IsEligible(p1, Date :: createDate(12, 3,
2021)))                // checking the IsEligible function     p1
is male // here d3 is the reservation date!!
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: Sleeper :: Type());     // p2 is also male!
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR BOOKINGCATEGORY IS DONE****\n";
*****************************************************************************
```

Golden Output:
Booking successfully done!
Exception caught!
Inconsistent booking category of passenger!
TESTING FOR BOOKINGCATEGORY IS DONE*

```
*****************************************************************************
const GeneralCategory& b = GeneralCategory :: Type();           //
here created object is also of the same type

cout << b;                  // testing the operator <<

if(b.getName() != "General Category")            // the getName
test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);
const Date d3 = Date :: createDate(16, 3, 1956);
const Date d4 = Date :: createDate(13, 04, 2001);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");
```

```
const Passenger p3 = Passenger :: createPassenger("Mamata", "",
"Hajra", "134236789345", d3, Gender :: Female :: Type(),
"9921133112");
const Passenger p4 = Passenger :: createPassenger("Ankita", "",
"", "123456737123", d4, Gender :: Female :: Type());

try {
    Station s1 = Station :: createStation("Chennai");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p1,
BookingClass :: ACFirstClass :: Type());       // for p1
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p4,
BookingClass :: ACFirstClass :: Type());            // for p4
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking!\n";
    cout << e.what() << '\n';
}

if(!IsEligible(p1, Date :: createDate(12, 3, 2021)))
// always eligible!
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: Sleeper :: Type());     // p2 is also male!
    cout << "Booking Successfully done\n";                 //
flow doesnt come here
} catch(exception& e) {
    cout << "***Error in booking!***\n";                // here
the flow should never each, since once the basic things are
validated, only the bookingCategory needs validation!
    cout << e.what() << '\n';
}

cout << "***TESTING FOR GENERAL CATEGORY IS DONE****\n";
*****************************************************************************
```

Golden Output:
General Category
Booking successfully done!
Booking Successfully done
TESTING FOR GENERAL CATEGORY IS DONE*

```
*****************************************************************************
const LadiesCategory& b = LadiesCategory :: Type();

cout << b;                    // testing the operator <<

if(b.getName() != "Ladies Category")            // the getName
test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);
const Date d3 = Date :: createDate(16, 3, 1956);
const Date d4 = Date :: createDate(13, 04, 2001);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");
const Passenger p3 = Passenger :: createPassenger("Mamata", "",
"Hajra", "134236789345", d3, Gender :: Female :: Type(),
"9921133112");
const Passenger p4 = Passenger :: createPassenger("Ankita", "",
"", "123456737123", d4, Gender :: Female :: Type());

try {
    Station s1 = Station :: createStation("Bangalore");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p3,
BookingClass :: FirstClass :: Type());       // for p3
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p4,
BookingClass :: ACFirstClass :: Type());            // for p4
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking\n";
    cout << e.what() << '\n';
}

if(!LadiesCategory :: IsEligible(p1, Date :: createDate(12, 3,
2010)))         // MALES less than 12 years of age are also
eligible!
    cout << "Error in eligibility criteria\n";

try {
```

```
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: Sleeper :: Type());     // p2 is also male!
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR LADIES CATEGORY IS DONE****\n";
*********************************************************************************
```

Golden Output:
Ladies Category
Booking successfully done!
Exception caught!
Inconsistent booking category of passenger!
TESTING FOR LADIES CATEGORY IS DONE*

```
*********************************************************************************
const SeniorCitizenCategory& b = SeniorCitizenCategory :: Type();

cout << b;                    // testing the operator <<

if(b.getName() != "Senior Citizens Category")            // the
getName test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);           //
for senior citzen
const Date d3 = Date :: createDate(16, 3, 1956);           // ""
"" ""
const Date d4 = Date :: createDate(13, 04, 2001);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");
const Passenger p3 = Passenger :: createPassenger("Mamata", "",
"Hajra", "134236789345", d3, Gender :: Female :: Type(),
"9921133112");
const Passenger p4 = Passenger :: createPassenger("Ankita", "",
"", "123456737123", d4, Gender :: Female :: Type());
```

```
try {
    Station s1 = Station :: createStation("Bangalore");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: FirstClass :: Type());       // for p2
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p3,
BookingClass :: ACFirstClass :: Type());            // for p3
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking\n";
    cout << e.what() << '\n';
}

if(!SeniorCitizenCategory :: IsEligible(p2, Date :: createDate(12,
3, 2010)))         // Positive
    cout << "Error in eligibility criteria\n";

if(SeniorCitizenCategory :: IsEligible(p1, Date :: createDate(12,
3, 2010)))         // Negative
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Chennai");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p1,
BookingClass :: ACFirstClass :: Type());     // p2 is also male!
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR SENIOR CITIZEN CATEGORY IS DONE****\n";
*****************************************************************************
```

Golden Output:
Senior Citizens Category
Booking successfully done!
Error in eligibility criteria
Exception caught!
Inconsistent booking category of passenger!
TESTING FOR SENIOR CITIZEN CATEGORY IS DONE*

****

```
const DivyaangCategory& b = DivyaangCategory :: Type();

cout << b;                    // testing the operator <<

if(b.getName() != "Divyaang Category")            // the getName
test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);
const Date d3 = Date :: createDate(16, 3, 1956);
const Date d4 = Date :: createDate(13, 04, 2001);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914", "D12344", 1);     // divyaang
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");
const Passenger p3 = Passenger :: createPassenger("Mamata", "",
"Hajra", "134236789345", d3, Gender :: Female :: Type(),
"9921133112");
const Passenger p4 = Passenger :: createPassenger("Ankita", "",
"", "123456737123", d4, Gender :: Female :: Type(), "8922234567",
"d566ygv", 3);        // divyaang

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p1,
BookingClass :: FirstClass :: Type());       // for p1
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p4,
BookingClass :: ACFirstClass :: Type());            // for p4
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking\n";
    cout << e.what() << '\n';
}

if(DivyaangCategory :: IsEligible(p2, Date :: createDate(12, 3,
2010)))         // Should not be eligible!
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
```

```
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p3,
BookingClass :: Sleeper :: Type());     // p2 is not divyaang
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR DIVYAANG CATEGORY IS DONE****\n";
```

****

Golden Output:
Divyaang Category
Booking successfully done!
Exception caught!
Inconsistent booking category of passenger!
TESTING FOR DIVYAANG CATEGORY IS DONE*

****

```
const TatkalCategory& b = TatkalCategory :: Type();

cout << b;                  // testing the operator <<

if(b.getName() != "Tatkal Category")            // the getName
test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);

const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");

try {
    Station s1 = Station :: createStation("Bangalore");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 4, 2021);
```

```
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p1,
BookingClass :: ACFirstClass :: Type());            // for p1
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking\n";
    cout << e.what() << '\n';
}

if(!TatkalCategory :: IsEligible(Date :: createDate(11, 3, 2010),
Date :: createDate(12, 3, 2010)))         // Should be eligible //
1 day!!
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);            //
not within 1 day!!
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: Sleeper :: Type());
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR TATKAL CATEGORY IS DONE****\n";
*********************************************************************************
```

Golden Output:
Tatkal Category
Booking successfully done!
Exception caught!
Inconsistent booking category of passenger!
TESTING FOR TATKAL CATEGORY IS DONE*

```
*********************************************************************************

const PremiumTatkalCategory& b = PremiumTatkalCategory :: Type();

cout << b;                    // testing the operator <<

if(b.getName() != "Premium Tatkal Category")            // the
getName test
    cout << "Error in GetName\n";

const Date d1 = Date :: createDate(16, 3, 2001);
const Date d2 = Date :: createDate(16, 3, 1950);
```

```
const Passenger p1 = Passenger :: createPassenger("Abhinandan",
"", "De", "134236789345", d1, Gender :: Male :: Type(),
"8279728914");
const Passenger p2 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d2, Gender :: Male :: Type(),
"9412056328");

try {
    Station s1 = Station :: createStation("Bangalore");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 4, 2021);
    const Booking* b4 = b.createBooking(s1, s2, d2, d3, p1,
BookingClass :: ACFirstClass :: Type());            // for p1
    cout << "Booking successfully done!\n";
} catch(exception& e) {
    cout << "Error in booking\n";
    cout << e.what() << '\n';
}

if(!TatkalCategory :: IsEligible(Date :: createDate(11, 3, 2010),
Date :: createDate(12, 3, 2010)))         // Should be eligible //
1 day!!
    cout << "Error in eligibility criteria\n";

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);            //
not within 1 day!!
    const Booking* b3 = b.createBooking(s1, s2, d2, d3, p2,
BookingClass :: Sleeper :: Type());
    cout << "Erroneous booking created\n";                  //
flow doesnt come here
} catch(exception& e) {
    cout << "***Exception caught!***\n";
    cout << e.what() << '\n';
}
cout << "***TESTING FOR PREMIUM TATKAL CATEGORY IS DONE****\n";
*********************************************************************************
```

Golden Output:
Premium Tatkal Category
Booking successfully done!
Exception caught!
Inconsistent booking category of passenger!
*****TESTING FOR PREMIUM TATKAL CATEGORY IS DONE****

****

Class Concession
Explanations given in code

```
const Concession& lC = LadiesConcession :: Type();
// polymorphic binding!
cout << lC << '\n';                 // testing the << operator

if(lC.GetName() != "Ladies Concession")         // this is the
only testable method!
    cout << "Error in GetName\n";

cout << "****TESTING FOR CONCESSION IS COMPLETE****\n";
```

****

Golden output:
Ladies Concession
****TESTING FOR CONCESSION IS COMPLETE****
Ladies Concession
****

```
const LadiesConcession& lC = LadiesConcession :: Type();
// polymorphic binding!
cout << lC << '\n';                 // testing the << operator

if(lC.GetName() != "Ladies Concession")         // this is the
only testable method!
    cout << "Error in GetName\n";

if(abs(lC.GetConcession() - 0) >= 1e-3)
    cout << "Error in GetConcession\n";
```

```
LadiesConcession :: ChangeConcession(0.2);          // try to
change concession

if(abs(lC.GetConcession() - 0.2) >= 1e-3)
    cout << "Error in Change Concession\n";

LadiesConcession :: ChangeConcession(0.0);          // changing it
back!

cout << "****TESTING FOR LADIES CONCESSION IS COMPLETE****\n";
```

```
********************************************************************************
```

Ladies Concession

```
****TESTING FOR LADIES CONCESSION IS COMPLETE****
********************************************************************************
```

```
const DivyaangConcession& dC = DivyaangConcession :: Type();
// the only occurence
// cout << dC << '\n';                          // checking the <<
operator

if(dC.GetName() != "Divyaang Concession")           // getname
check
    cout << "Error in GetName function\n";

if(abs(dC.GetConcession(BookingClass :: Sleeper :: Type(),
Divyaang :: BlindDivyaang :: Type()) - 0.75) >= 1e-3)
// checking some random getConcessions
    cout << "Error in getConcession1\n";
if(abs(dC.GetConcession(BookingClass :: ACFirstClass :: Type(),
Divyaang :: OrthopaedicallyHandicappedDivyaang :: Type()) - 0.5)
>= 1e-3)
    cout << "Error in getConcession2\n";
if(abs(dC.GetConcession(BookingClass :: FirstClass :: Type(),
Divyaang :: CancerPatientDivyaang :: Type()) - 0.75) >= 1e-3)
    cout << "Error in getConcession3\n";
if(abs(dC.GetConcession(BookingClass :: SecondSitting :: Type(),
Divyaang :: TBPatientDivyaang :: Type()) - 0.75) >= 1e-3)
    cout << "Error in getConcession4\n";

cout << "****TESTING FOR DIVYAANG CONCESSION IS COMPLETE****\n";
```

```
********************************************************************************
```

Senior Citizens Concession

```
****TESTING FOR SENIOR CITIZENS CONCESSION IS COMPLETE****
********************************************************************************
```

```
const DivyaangConcession& dC = DivyaangConcession :: Type();
// the only occurence
cout << dC << '\n';                          // checking the <<
operator

if(dC.GetName() != "Divyaang Concession")           // getname
check
    cout << "Error in GetName function\n";

if(abs(dC.GetConcession(BookingClass :: Sleeper :: Type(),
Divyaang :: BlindDivyaang :: Type()) - 0.75) >= 1e-3)
// checking some random getConcessions
    cout << "Error in getConcession1\n";
if(abs(dC.GetConcession(BookingClass :: ACFirstClass :: Type(),
Divyaang :: OrthopaedicallyHandicappedDivyaang :: Type()) - 0.5)
>= 1e-3)
    cout << "Error in getConcession2\n";
if(abs(dC.GetConcession(BookingClass :: FirstClass :: Type(),
Divyaang :: CancerPatientDivyaang :: Type()) - 0.75) >= 1e-3)
    cout << "Error in getConcession3\n";
if(abs(dC.GetConcession(BookingClass :: SecondSitting :: Type(),
Divyaang :: TBPatientDivyaang :: Type()) - 0.75) >= 1e-3)
    cout << "Error in getConcession4\n";

cout << "****TESTING FOR DIVYAANG CONCESSION IS COMPLETE****\n";
```

****

Divyaang Concession

****TESTING FOR DIVYAANG CONCESSION IS COMPLETE****

****

Class Date

```
Date d1 = createDate(01, 01, 2001);         // it could have been
directly created although
Date d2 = createDate(2, 03, 2002);          // since it is in the
scope of the class
Date d3 = createDate(02, 3, 2002);          // but we are testing
via the createPassenger method
Date d4(d1);

if(d4 != d1)
    cout << "Error in copy constructor or operator !=\n";       //
it is a copy
if(d3 != d2)
    cout << "Error in operator != \n";

if(d1 > d2)
    cout << "Error in operator >\n";
if(d4 > d1)
    cout << "Error in operator >\n";

try {
    Date d4e = createDate(01, 01, 1890);
    cout << "Program reached an inconsistent state!\n";
// this will never be reached as exception will be thrown a
priori!
} catch(exception& e) {
    cout << "**Exception caught***: ";
    cout << e.what() << '\n';
}
```

```
try {
    Date d4e = createDate(01, 01, 2190);
    cout << "Program reached an inconsistent state!\n";
// this will never be reached as exception will be thrown a
priori!
} catch(exception& e) {
    cout << "**Exception caught***: ";
    cout << e.what() << '\n';
}

try {
    Date d4e = createDate(29, 02, 2003);
    cout << "Program reached an inconsistent state!\n";
// this will never be reached as exception will be thrown a
priori!
} catch(exception& e) {
    cout << "**Exception caught***: ";
    cout << e.what() << '\n';
}

try {
    Date d4e = createDate(31, 4, 2005);
    cout << "Program reached an inconsistent state!\n";
// this will never be reached as exception will be thrown a
priori!
} catch(exception& e) {
    cout << "**Exception caught***: ";
    cout << e.what() << '\n';
}

Date d5 = createDate(28, 2, 2004);
Date d6 = createDate(29, 2, 2004);              // note that this
is valid!
Date d7 = createDate(1, 3, 2004);
Date d8 = createDate(31, 12, 2013);
Date d9 = createDate(1, 1, 2014);

if(addOne(d5) != d6) {                              // testing the
addOne function!
    cout << "Error in addOne";
}
if(addOne(d6) != d7) {
    cout << "Error in addOne";
}
if(!(addOne(d5) != d7)) {                           // This Should
NOT be true!!
    cout << "Error in addOne";
```

```
}
if(addOne(d8) != d9) {
    cout << "Error in addOne";
}

if(computeAge(d5, d8) != 9)                          // checking
the computeAge funciton!
    cout << "Error in compute age\n";
if(computeAge(d7, d9) != 9)
    cout << "Error in computeAge\n";
if(computeAge(d5, d5) != 0)
    cout << "Error in computeAge\n";

cout << "****TESTING FOR DATE IS COMPLETE****\n";
```

****

Golden output:

Exception caught*: Invalid Date!
Exception caught*: Invalid Date!
Exception caught*: Invalid Date!
Exception caught*: Invalid Date!
****TESTING FOR DATE IS COMPLETE****

****

Class Divyaang and Hierarchy

```
const Divyaang& d = Divyaang :: BlindDivyaang :: Type();
// cheking if mereys singleton works
cout << d << '\n';                      // checking if operator <<
works

if(d.GetName() != "Blind Divyaang")
    cout << "Error in Get Name function\n";

if(abs(d.GetConcession(BookingClass :: ACFirstClass :: Type()) -
0.5) >= 1e-3)
    cout << "Error in Get Concession1!\n";
if(abs(d.GetConcession(BookingClass :: AC2Tier :: Type()) - 0.5)
>= 1e-3)
    cout << "Error in Get Concession2!\n";
if(abs(d.GetConcession(BookingClass :: AC3Tier :: Type()) - 0.75)
>= 1e-3)
    cout << "Error in Get Concession3!\n";
if(abs(d.GetConcession(BookingClass :: Sleeper :: Type()) - 0.75)
>= 1e-3)
    cout << "Error in Get Concession4!\n";

cout << "***TESTING FOR CLASS DIVYAANG IS COMPLETE***\n";
```

****
Golden output:
Blind Divyaang

TESTING FOR CLASS DIVYAANG IS COMPLETE
****

```
const DivyaangSubCategories<T>& d = Type();
cout << d << '\n';

if(d.GetName() != sName)
    cout << "Error in Get Name function\n";

if(abs(d.GetConcession(BookingClass :: ACFirstClass :: Type()) -
sACFirstClass) >= 1e-3)
    cout << "Error in Get Concession1!\n";
if(abs(d.GetConcession(BookingClass :: AC2Tier :: Type()) -
sAC2Tier) >= 1e-3)
    cout << "Error in Get Concession2!\n";
if(abs(d.GetConcession(BookingClass :: AC3Tier :: Type()) -
sAC3Tier) >= 1e-3)
    cout << "Error in Get Concession3!\n";
if(abs(d.GetConcession(BookingClass :: Sleeper :: Type()) -
sSleeper) >= 1e-3)
    cout << "Error in Get Concession4!\n";

cout << "***TESTING FOR CLASS " << sName << " IS COMPLETE***\n";
```

****
Golden output:
Blind Divyaang

TESTING FOR CLASS Blind Divyaang IS COMPLETE
Orthopaedically Handicapped Divyaang

TESTING FOR CLASS Orthopaedically Handicapped Divyaang IS COMPLETE
Cancer Patient Divyaang

TESTING FOR CLASS Cancer Patient Divyaang IS COMPLETE
TB Patient Divyaang

TESTING FOR CLASS TB Patient Divyaang IS COMPLETE
Male
****

Class Gender and its hierarchy

```
const Gender& g = Male :: Type();       // creating a child class
with the help of a const reference (avoiding pointers!)

if(g.GetName() != "Male") {                          // a check on
all the members
    cout << "Error in name for male gender\n";
}
if(g.GetTitle() != "Mr.") {
    cout << "Error in salutation for male gender\n";
}
if(!IsMale(g)) {                                     // because if
not male then female
    cout << "Error in IsMale function\n";
}
cout << g << '\n';          // Testing the ostream operator!
cout << "****TESTING FOR GENDER IS COMPLETE****\n";
*******************************************************************************
```

Golden output:

```
****TESTING FOR GENDER IS COMPLETE****
*******************************************************************************
```

```
const Gender& g = Type();
if(g.GetName() != sName){                          // testing the
GetName function
    cout << "Error in GetName\n";
}
if(g.GetTitle() != sSalutation) {                  // the salutation
test!
    cout << "Error in GetSalutation\n";
}
cout << g << '\n';
cout << "****TESTING FOR " << sName << " IS COMPLETE****\n";
*********************************************************************************
```

Golden output:

Male

****TESTING FOR Male IS COMPLETE****

Female

****TESTING FOR Female IS COMPLETE****

****

Class Passenger

```
try {
    const Date d1 = Date :: createDate(12, 3, 2010);
    const Date d2 = Date :: createDate(15, 6, 2019);
    const Date d3 = Date :: createDate(12, 7, 1984);

    const Passenger p1 = createPassenger("Abhinandan", "", "De",
"124568345678", d1, Gender :: Male :: Type(), "8272288222");  //
details are upto mobile no.
    const Passenger p2 = createPassenger("Suryam", "Arnav",
"Kalra", "156359247342", d2, Gender :: Male :: Type(),
"7654890654", "DJNCKKCC", 2);    // all details
    const Passenger p3 = createPassenger("Megha", "", "",
"176245389456", d3, Gender :: Female :: Type());      // no mobile
number also

    cout << "Successfully created all passengers\n";

    if(p1.getDateOfBirth() != d1)
        cout << "Error in getDateOfBirth\n";

    if(p2.getDisabilityIndex() != 2 || p2.getDivyaangId() !=
"DJNCKKCC")
        cout << "Error in getting the divyaang info!\n";
```

```
    if(Gender::IsMale(p3.getGender()))
        cout << "Error in getGender function!\n";

} catch(exception& e) {
    throw;      // rethrow!
}

const Date d1 = Date :: createDate(12, 3, 2010);

try {
    const Passenger p4e = createPassenger("", "", "",
"124568345678", d1, Gender :: Male :: Type(), "8272288222");
// wrong name format
    cout << "Erroneous passenger created!\n";
// flow wont reach this point!
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

try {
    const Passenger p4e = createPassenger("", "Kumar", "",
"124568345678", d1, Gender :: Male :: Type(), "8272288222");
// wrong name format
    cout << "Erroneous passenger created!\n";
// flow wont reach this point!
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

try {
    const Passenger p4e = createPassenger("Abhinandan", "", "De",
"12456A5678", d1, Gender :: Male :: Type(), "8272288222");      //
wrong aadhar format
    cout << "Erroneous passenger created!\n";
// flow wont reach this point!
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

try {
    const Passenger p4e = createPassenger("Abhinandan", "", "",
"124568345678", d1, Gender :: Male :: Type(), "8+7882++22");
// wrong phone number format
    cout << "Erroneous passenger created!\n";
// flow wont reach this point!
```

```
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

try {
    const Passenger p4e = createPassenger("Abhinandan", "", "",
"124568345678", d1, Gender :: Male :: Type(), "827222");      //
wrong phone number format
    cout << "Erroneous passenger created!\n";
// flow wont reach this point!
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}
cout << "****TESTING FOR PASSENGER IS COMPLETE****\n";
*********************************************************************************
```

Golden output

Successfully created all passengers
Exception caught
Invalid format for name!
Exception caught
Invalid format for name!
Exception caught
Invalid aadhar number!
Exception caught
Invalid mobile number!
Exception caught
Invalid mobile number!

****TESTING FOR PASSENGER IS COMPLETE****
****

Class Railways

```
cout << Type();     // checking the Type() function and the
ostream operator

const double error = 1e-3;

if(abs(Type().GetDistance(sStations[1], sStations[2]) - 2150.00)
>= error)              // checking the GetDistance Function
    cout << "Error in getting the distances from GetDistance\n";
if(abs(Type().GetDistance(sStations[1], sStations[4]) - 2180.00)
>= error)
    cout << "Error in getting the distances from GetDistance\n";

if(abs(sDistStations[{sStations[1].GetName(),
sStations[2].GetName()}] - 2150.00) >= error)         // cheking
if inputs were taken correctly
    cout << "Error in storing the distances in sDistStations\n";
if(abs(sDistStations[{sStations[1].GetName(),
sStations[4].GetName()}] - 2180.00) >= error)
    cout << "Error in storing the distances in sDistStations\n";

if(!validStation("Delhi")) {
// checking the validStation function
    cout << "Error in validStation function\n";
}
if(validStation("Kochi")) {
    cout << "Error in validStation function\n";
```

```
}
cout << "****TESTING FOR RAILWAYS IS COMPLETE****\n";
********************************************************************************
```

Golden output

List of stations:
Mumbai
Delhi
Bangalore
Kolkata
Chennai

Distances between stations:
Mumbai Delhi: 1447
Mumbai Bangalore: 981
Mumbai Kolkata: 2014
Mumbai Chennai: 1338
Delhi Bangalore: 2150
Delhi Kolkata: 1472
Delhi Chennai: 2180
Bangalore Kolkata: 1871
Bangalore Chennai: 350
Kolkata Chennai: 1659

****TESTING FOR RAILWAYS IS COMPLETE****
****

Class Station

```
const Station s1 = createStation("Delhi");              // if
createStation works or not
const Station s2 = createStation("Mumbai");
const Station s3 = createStation("Kolkata");

double checker1 = s1.GetDistance(s2);
double checker2 = s1.GetDistance(s3);
double checker3 = s2.GetDistance(s3);

double val1 = Railways :: GetDistance(s1, s2);
double val2 = Railways :: GetDistance(s1, s3);
double val3 = Railways :: GetDistance(s2, s3);

if(abs(checker1-val1) >= 1e-3) {                        //
checking if distances are stores correctly
    cout << "Error in storing distances\n";
}

if(abs(checker2-val2) >= 1e-3) {
    cout << "Error in storing distances\n";
```

```
}

if(abs(checker3-val3) >= 1e-3) {
    cout << "Error in storing distances\n";
}

try {
    const Station s3e = createStation("Kanpur");            //
Station which does not exist!
    cout << "Program is in an inconsistent state\n";
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

try {
    const Station s3e = createStation("");                  //
Station with blank name!
    cout << "Program is in an inconsistent state\n";
} catch(exception& e) {
    cout << "***Exception caught***\n";
    cout << e.what() << '\n';
}

cout << "****TESTING FOR STATION IS COMPLETE****\n";
*********************************************************************************
```

Golden output

Exception caught
Station not found!
Exception caught
Station not found!
****TESTING FOR STATION IS COMPLETE****
****

APPLICATION TESTING

```
try {
    Railways :: createRailways();
} catch(exception& e) {
    cout << "\n***Exception caught!** : ";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    Station s3 = Station :: createStation("Mumbai");
    Station s4 = Station :: createStation("Chennai");
```

```
    Station s5 = Station :: createStation("Kolkata");
    Station s6 = Station :: createStation("Kanpur");
} catch(exception& e) {
    cout << "\n***Exception caught!** : ";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Kolkata");
    Station s2 = Station :: createStation("Delhi");
    const Date& d1 = Date :: createDate(16, 3, 2001);
    const Passenger p1 = Passenger ::
createPassenger("Abhinandan", "", "De", "134236789345", d1, Gender
:: Male :: Type(), "8279728914");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ACFirstClass :: Type(), GeneralCategory :: Type(),
p1, d3);       // correct!
} catch(exception& e) {
    cout << "\n***Exception caught!** : ";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 2001);
    const Passenger p1 = Passenger ::
createPassenger("Abhinandan", "", "De", "134236789345", d1, Gender
:: Male :: Type(), "8279728914");      // correct
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: AC3Tier :: Type(), GeneralCategory :: Type(), p1,
d3);
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1950);
```

```
    const Passenger p1 = Passenger :: createPassenger("Prabhat",
"Kumar", "Hajra", "134236789345", d1, Gender :: Male :: Type(),
"9412056328");     // correct
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: AC3Tier :: Type(), SeniorCitizenCategory ::
Type(), p1, d3);
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1956);
    const Passenger p1 = Passenger :: createPassenger("Mamata",
"", "Hajra", "134236789345", d1, Gender :: Female :: Type(),
"9921133112");
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: Sleeper :: Type(), SeniorCitizenCategory ::
Type(), p1, d3);     // correct
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1980);
    const Passenger p1 = Passenger :: createPassenger("Suryam",
"Arnav", "Kalra", "134236789345", d1, Gender :: Male :: Type(),
"9921133112", "d12233", 1);
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: AC3Tier :: Type(), SeniorCitizenCategory ::
Type(), p1, d3);      // correct
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
```

```
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1980);
    const Passenger p1 = Passenger :: createPassenger("Suryam",
"Arnav", "Kalra", "134236789345", d1, Gender :: Male :: Type(),
"9921133112", "d12233", 3);
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 4, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: FirstClass :: Type(), PremiumTatkalCategory ::
Type(), p1, d3);      // correct!
} catch(exception& e) {
    cout << "****Exception caught!****\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1980);
    const Passenger p1 = Passenger :: createPassenger("Shashank",
"", "Singh", "134236789345", d1, Gender :: Male :: Type(),
"9921133112", "d12233", 3);
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 4, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: SecondSitting :: Type(), SeniorCitizenCategory ::
Type(), p1, d3);      // incorrect!
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Delhi");
    Station s2 = Station :: createStation("Mumbai");
    const Date& d1 = Date :: createDate(16, 3, 1980);
    const Passenger p1 = Passenger ::
createPassenger("Abhinandan", "", "", "134236789345", d1, Gender
:: Male :: Type(), "9921133112", "d12233", 3);
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 8, 2021);
    cout << "\n\n";
```

```
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: ExecutiveChairCar :: Type(), TatkalCategory ::
Type(), p1, d3);         // incorrect!
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

try {
    Station s1 = Station :: createStation("Chennai");
    Station s2 = Station :: createStation("Bangalore");
    const Date& d1 = Date :: createDate(16, 3, 1980);
    const Passenger p1 = Passenger :: createPassenger("Simran",
"Sharma", "", "134236789345", d1, Gender :: Female :: Type(),
"9921133112", "d12233", 3);
    const Date& d2 = Date :: createDate(8, 4, 2021);
    const Date& d3 = Date :: createDate(9, 4, 2021);
    cout << "\n\n";
    const Booking* b1 = Booking :: ReserveBooking(s1, s2, d2,
BookingClass :: AC2Tier :: Type(), PremiumTatkalCategory ::
Type(), p1, d3); // correct!
} catch(exception& e) {
    cout << "Exception caught!\n";
    cout << e.what() << '\n';
}

vector<Booking*>::iterator it;          // traversing through the
list!

for (it = Booking::sBookings.begin(); it <
Booking::sBookings.end(); ++it) {
    cout << *(*it);
}
```

****

Golden output

Added all stations
Updated Distances
Created Railways!
***Exception caught!** : Station not found!

Exception caught!
Inconsistent booking category of passenger!

Exception caught!
Inconsistent booking category of passenger!

Exception caught!

Inconsistent booking category of passenger!

BOOKING SUCCEEDED:
PNR Number : 1
From station : Kolkata
To station : Delhi
Travel Date : 9 Aug 2021
Booking Class : AC First Class
: AC: YES
: Bunks: 2
: Luxury: YES
Booking Category : General Category
Fare : 4844

Passenger details
Name : Mr. Abhinandan De
Aadhar number : 134236789345
Mobile number : 8279728914
Date of birth : 16 Mar 2001
Gender : Male
Divyaang Id : NA
Disability Index : NA

BOOKING SUCCEEDED:
PNR Number : 2
From station : Delhi
To station : Mumbai
Travel Date : 9 Aug 2021
Booking Class : AC 3 Tier
: AC: YES
: Bunks: 3
: Luxury: NO
Booking Category : General Category
Fare : 1849

Passenger details
Name : Mr. Abhinandan De
Aadhar number : 134236789345
Mobile number : 8279728914
Date of birth : 16 Mar 2001
Gender : Male
Divyaang Id : NA
Disability Index : NA

BOOKING SUCCEEDED:
PNR Number : 3
From station : Delhi
To station : Mumbai
Travel Date : 9 Aug 2021
Booking Class : AC 3 Tier
: AC: YES
: Bunks: 3
: Luxury: NO
Booking Category : Senior Citizens Category
Fare : 1125

Passenger details
Name : Mr. Prabhat Kumar Hajra
Aadhar number : 134236789345
Mobile number : 9412056328
Date of birth : 16 Mar 1950
Gender : Male
Divyaang Id : NA
Disability Index : NA

BOOKING SUCCEEDED:
PNR Number : 4
From station : Delhi
To station : Mumbai
Travel Date : 9 Aug 2021
Booking Class : Sleeper
: AC: NO
: Bunks: 3
: Luxury: NO
Booking Category : Senior Citizens Category
Fare : 382

Passenger details
Name : Ms. Mamata Hajra
Aadhar number : 134236789345
Mobile number : 9921133112
Date of birth : 16 Mar 1956
Gender : Female
Divyaang Id : NA
Disability Index : NA

BOOKING SUCCEEDED:
PNR Number : 5
From station : Delhi
To station : Mumbai
Travel Date : 9 Apr 2021
Booking Class : First Class
: AC: NO
: Bunks: 2
: Luxury: YES
Booking Category : Premium Tatkal Category
Fare : 2655

Passenger details
Name : Mr. Suryam Arnav Kalra
Aadhar number : 134236789345
Mobile number : 9921133112
Date of birth : 16 Mar 1980
Gender : Male
Divyaang Id : d12233
Disability Index : 3
BOOKING SUCCEEDED:
PNR Number : 6
From station : Chennai
To station : Bangalore
Travel Date : 9 Apr 2021
Booking Class : AC 2 Tier
: AC: YES
: Bunks: 2
: Luxury: NO
Booking Category : Premium Tatkal Category
Fare : 750

Passenger details
Name : Ms. Simran Sharma
Aadhar number : 134236789345
Mobile number : 9921133112
Date of birth : 16 Mar 1980
Gender : Female
Divyaang Id : d12233
Disability Index : 3

****