Software Engineering CS20006 -- Theory Assignment O5
Nakul Aggarwal 19CS10044
09 April 2021

TEST PLAN DOCUMENT

*A. Unit Test Plan for **Station***

*A.1. Test Scenarios for **Construction of Object(s)***

Test Plan ID	A
Test Suite ID	A.1
Test Case ID	A.1.1
Test Case Summary	Using *Station::CreateStation* method to construct a *Station* object with an *arbitrary non-empty name*
Prerequisite System's State	NIL
Procedure	(1.) Choose a string which has atleast one character other than *whitespace*. (2.) Pass the string as argument to *Station::CreateStation* method. (3.) Surround the function call with *try-catch block*. (4.) Match the *Station::name_* data member of the returned object (if no exception is caught) with the passed *string* arguement.
Test Data	stationNames: *"I am an arbitrary name"*
Expected Result / Golden Output	(1.) No *exception* will be caught. (2.) A *Station* object will be returned. (3.) Value of *Station::name_* data member of the returned object will be same as the argument passed, that is *"I am an arbitrary name"*
Date of Creation	02 April 2021

Test Plan ID	A
Test Suite ID	A.1
Test Case ID	A.1.2
Test Case Summary	Using *Station::CreateStation* method to construct a *Station* object with an *empty*

	name
Prerequisite System's State	NIL
Procedure	(1.) Choose a string which has *zero length* or has no characters other than *whitespace*. (2.) Pass the string as argument to *Station::CreateStation* method. (3.) Surround the function call with *try-catch block*.
Test Data	stationNames: *"", " "*
Expected Result / Golden Output	A *Bad_Station exception* will be caught for both the *test data*.
Date of Creation	02 April 2021

A.2. Test Scenarios for ***Construction of Copies of Object(s)***

Test Plan ID	A
Test Suite ID	A.2
Test Case ID	A.2.1
Test Case Summary	Using *copy constructor* to instantiate *Station* class
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Station* object by passing a *non-empty string* as argument. (2.) Construct a *Station* object by passing this *Station* object as argument. (3.) Compare the attributes of the two *Station* objects.
Test Data	stationNames: *"I am an arbitrary name"*
Expected Result / Golden Output	The *Station* object constructed in (2.) will have the same name as the one in (1.), that is *"I am an arbitrary name"*
Date of Creation	02 April 2021

A.3. Test Scenarios for ***Overloaded Equality Check Operator***

Test Plan ID	A
Test Suite ID	A.3
Test Case ID	A.3.1
Test Case Summary	Comparing two *Station* objects with different names using '==' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a pair of *non-empty string*s with *different values*. (2.) Construct a pair of *Station* objects with these strings respectively. (3.) Compare the two *Station* objects *with* '==' operator and store the result in a *boolean* variable.
Test Data	stationNames: *("Mumbai", "Delhi")*
Expected Result / Golden Output	The value of the *boolean variable* will be *false*.
Date of Creation	02 April 2021

Test Plan ID	A
Test Suite ID	A.3
Test Case ID	A.3.2
Test Case Summary	Comparing two *Station* objects with same names using '==' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a *non-empty* string. (2.) Construct two *Station* objects with this string as arguement. (3.) Compare the two *Station* objects *with* '==' operator and store the result in a *boolean* variable.

Test Data	stationNames: *("Mumbai", "Mumbai")*
Expected Result / Golden Output	The value of the *boolean variable* will be *true*.
Date of Creation	02 April 2021

A.4. Test Scenarios for ***Overloaded Inequality Check Operator***

Test Plan ID	A
Test Suite ID	A.4
Test Case ID	A.4.1
Test Case Summary	Comparing two *Station* objects with different names using '!=' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a pair of *non-empty string*s with *different values*. (2.) Construct a pair of *Station* objects with these strings respectively. (3.) Compare the two *Station* objects *with* '!=' operator and store the result in a *boolean* variable.
Test Data	stationNames: *("Mumbai", "Delhi")*
Expected Result / Golden Output	The value of the *boolean variable* will be *true*.
Date of Creation	02 April 2021

Test Plan ID	A
Test Suite ID	A.4
Test Case ID	A.4.2
Test Case Summary	Comparing two *Station* objects with same names using '!=' operator.
Prerequisite System's State	NIL

Procedure	(1.) Choose a *non-empty* string. (2.) Construct two *Station* objects with this string as arguement. (3.) Compare the two *Station* objects *with* '!=' operator and store the result in a *boolean* variable.
Test Data	stationNames: *("Mumbai", "Mumbai")*
Expected Result / Golden Output	The value of the *boolean variable* will be *false*.
Date of Creation	02 April 2021

*A.5. Test Scenarios for **Overloaded Output Streaming Operator***

Test Plan ID	A
Test Suite ID	A.5
Test Case ID	A.5.1
Test Case Summary	Print a *Station* object onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Station* object passing a *non-empty string* as argument. (2.) Print the constructed object onto the console using *cout* and *output streaming operator* <<.
Test Data	Station name: *"I am an arbitrary name"*
Expected Result / Golden Output	The name of the *Station* (value of *Station::name_* same as the string passed as arguement) that is *"[I am an arbitrary name]"* will be printed onto the console.
Date of Creation	02 April 2021

*A.6. Test Scenarios for **Non Static Member Functions***

Test Plan ID	A
Test Suite ID	A.6
Test Case ID	A.6.1
Test Case Summary	Use the method *Station::GetName*
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Station* object passing a *non-empty string* as argument. (2.) Call the method *Station::GetName* on the object and compare the *Station::name_* data member of the object with the returned value.
Test Data	Station name: *"I am an arbitrary name"*
Expected Result / Golden Output	The name of the *Station* (value of *Station::name_*) will be same as the value returned by the method, that is *"I am an arbitrary name"*
Date of Creation	02 April 2021

Test Plan ID	A
Test Suite ID	A.6
Test Case ID	A.6.2
Test Case Summary	Use the method *Station::GetDistance* -- *correct inputs*
Prerequisite System's State	The singleton instance of Railways is constructed from the default parameters.
Procedure	(1.) Construct a pair of *Station* objects passing *non-empty strings* as arguments. (2.) Call the method *Station::GetDistance* on one of them and pass the other as argument.
Test Data	stationNames: *("Mumbai", "Delhi")*
Expected Result / Golden Output	The value returned will be equal to *1447*

Date of Creation	02 April 2021

Test Plan ID	A
Test Suite ID	A.6
Test Case ID	A.6.3
Test Case Summary	Use the method *Station::GetDistance* -- *erroneous inputs*
Prerequisite System's State	The singleton instance of Railways is constructed from the default parameters.
Procedure	(1.) Construct a pair of *Station* objects passing *non-empty strings* as arguments. (2.) Call the method *Station::GetDistance* on one of them and pass the other as argument. (3.) Surround the function call with *try-catch block*.
Test Data	stationNames: *("Mumbai", "Pune")*
Expected Result / Golden Output	A *Bad_Railways_Distance exception* will be caught.
Date of Creation	02 April 2021

*B. Unit Test Plan for **Railways***

*B.1. Test Scenarios for **Construction of Object(s)***

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.1
Test Case Summary	Call *Railways::SpecialRailways* method with no parameters
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Railways::SpecialRailways* method with no parameters. (2.) Surround the function call with *try-catch block*. (3.) Match the data members of the returned instance with the *default parameters*.
Test Data	NIL
Expected Result / Golden Output	(1.) No *exception* will be caught. (2.) Values of all the data members of the returned instance will be same as the values of the *default parameters*.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.2
Test Case Summary	Call *Railways::SpecialRailways* method with *erroneous arguments* -- *stations vector* has less than 2 stations.
Prerequisite System's State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* of *Station*s

	and a *map* for pairwise distances. (2.) Surround the function call with *try-catch block*.
Test Data	(*stationNames, distStations*)*: ({}, {}), ({"Delhi"}, {})*
Expected Result / Golden Output	A *Bad_Railways_NotEnoughStations exception* will be caught for both the *test data*
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.3
Test Case Summary	Call *Railways::SpecialRailways* method with *erroneous arguments -- duplicate names* in the *stations vector*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* and a *map* as given in *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	(*stationNames, distStations*)*: ({"Mumbai", "Delhi", "Mumbai"}, {{{"Delhi", "Mumbai"}, 1447}})*
Expected Result / Golden Output	A *Bad_Railways_DuplicateStations exception* will be caught.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.4

Test Case Summary	Call *Railways::SpecialRailways* method with *erroneous arguments -- distance between same stations defined*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* and a *map* as given in *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	(*stationNames, distStations*): *({"Mumbai","Delhi"}, {{{"Delhi", "Delhi"}, 5}})*
Expected Result / Golden Output	*Bad_Railways_DistBwSameStationsDefined exception* will be caught.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.5
Test Case Summary	Call *Railways::SpecialRailways* method with *erroneous arguments -- distance between distinct stations defined twice*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* and a *map* as given in *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	(*stationNames, distStations*): *({"Mumbai","Delhi"}, {{{"Delhi", "Mumbai"}, 1447}, {{"Mumbai", "Delhi"}, 1447}})*
Expected Result / Golden Output	*Bad_Railways_RepeatedDefinition exception* will be caught.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.6
Test Case Summary	Call *Railways::SpecialRailways* method with *erroneous arguments -- distance between distinct stations not defined*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* and a *map* as given in *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	(*stationNames, distStations*): *({"Mumbai","Delhi"}, {})*
Expected Result / Golden Output	A *Bad_Railways_NoDefinition exception* will be caught.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.7
Test Case Summary	Call *Railways::SpecialRailways* method with *valid arguments*
Prerequisite System's State	*Railways::SpecialRailways* method has not been called before with *valid non-default or default arguments*
Procedure	(1.) Call *Railways::SpecialRailways* method with two parameters -- a *vector* and a *map* as given in *test data*. (2.) Surround the function call with *try-catch block*. (3.) Match the data members of the returned instance with the *passed arguements*.

Test Data	(*stationNames, distStations*): *({“Mumbai”,“Delhi”}, {{{“Mumbai”, “Delhi”}, 1447}})*
Expected Result / Golden Output	(1.) No *exception* will be caught. (2.) Values of all the data members of the returned instance will be same as the values of the *passed parameters* like in *test data*.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.1
Test Case ID	B.1.8
Test Case Summary	Call *Railways::SpecialRailways* method twice and check if the same object is returned -- *test for singleton class*
Prerequisite System’s State	NIL
Procedure	(1.) Call *Railways::SpecialRailways* method with no parameters and store the returned instance in a *const Railways reference*. (2.) Call *Railways::SpecialRailways* method again with no parameters and store the returned instance in another *const Railways reference*. (3.) Compare the *addresses* of the two *Railways references* using *‘==’* and store the result in a *boolean* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *boolean* variable will be *true*
Date of Creation	03 April 2021

B.2. Test Scenarios for ***Overloaded Output Streaming Operator***

Test Plan ID	B

Test Suite ID	B.2
Test Case ID	B.2.1
Test Case Summary	Print a *Railways* object onto the console using *cout* output stream object.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Get the singleton instance of *Railways* by calling *Railways::SpecialRailways* method (with no parameters) (2.) Print the returned object onto the console using *cout output streaming operator* <<.
Test Data	NIL
Expected Result / Golden Output	The names of all the *Station*s and the pairwise distances between all the *Station*s will be printed onto the console. +++ STATIONS +++ - [Mumbai] - [Delhi] - [Bangalore] - [Kolkata] - [Chennai] -- +++ DISTANCES BETWEEN STATIONS +++ - between Bangalore and Chennai : 350 - between Bangalore and Delhi : 2150 - between Bangalore and Kolkata : 1871 - between Bangalore and Mumbai : 981 - between Chennai and Delhi : 2180 - between Chennai and Kolkata : 1659 - between Chennai and Mumbai : 1338 - between Delhi and Kolkata : 1472 - between Delhi and Mumbai : 1447 - between Kolkata and Mumbai : 2014
Date of Creation	02 April 2021

B.3. *Test Scenarios for* ***Non Static Member Functions***

Test Plan ID	B
Test Suite ID	B.3
Test Case ID	B.3.1
Test Case Summary	Call *Railways::GetDistance* method on the *singleton Railways instance* to get distance between an *erroneous* pair of *Station*s
Prerequisite System's State	*Railways::SpecialRailways* method has not been called before with *valid arguments* other than the *default arguments*.
Procedure	(1.) Get the singleton instance of *Railways* by calling *Railways::SpecialRailways* method (with no parameters) (2.) Call *Railways::GetDistance* method on the *singleton Railways instance* with two *Station* objects as inputs, as given in *test data*. (3.) Surround the function call with *try-catch block*.
Test Data	*stationNames: ("Delhi", "Pune")*
Expected Result / Golden Output	A *Bad_Railways_Distance exception* is caught.
Date of Creation	02 April 2021

Test Plan ID	B
Test Suite ID	B.3
Test Case ID	B.3.2
Test Case Summary	Call *Railways::GetDistance* method on the *singleton Railways instance* to get distance between a *valid* pair of *Station*s
Prerequisite System's State	*Railways::SpecialRailways* method has not been called before with *valid arguments* other than the *default arguments*.
Procedure	(1.) Get the singleton instance of *Railways* by calling *Railways::SpecialRailways*

	method (with no parameters) (2.) Call *Railways::GetDistance* method on the *singleton Railways instance* with two *Station* objects as inputs, as given in *test data*.
Test Data	*stationNames: (“Delhi”, “Mumbai”), (“Mumbai”, “Delhi”)*
Expected Result / Golden Output	The value returned is equal to *1447* for both the *test data*.
Date of Creation	02 April 2021

*C. Unit Test Plan for **Date***

*C.1. Test Scenarios for **Construction of Objects by strings***

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.1
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with an *incorrect format*
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateStrings: "16/11/20", "2020/11/16", "4/11/2020", "04/Nov/2020", "16-11-2020"*
Expected Result / Golden Output	A *Bad_Date_Format exception* will be thrown for all *test data*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.2
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a correct format but *invalid year*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.

Test Data	*dateStrings: “16/11/1889”, “16/11/2100”*
Expected Result / Golden Output	A *Bad_Date_Year exception* will be thrown in both *test data*
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.3
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a correct format but *invalid month*.
Prerequisite System’s State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateStrings: “16/00/2020”, “16/13/2020”*
Expected Result / Golden Output	A *Bad_Date_Month exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.4
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a correct format but *invalid day -- out of bounds*
Prerequisite System’s State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)*

	method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateStrings: “32/11/2020”, “00/11/2020”*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.5
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a correct format but *invalid day* for *February*
Prerequisite System’s State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateStrings: “30/02/2020”, “30/02/2021”, “29/02/2021”*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.6
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a correct format but *invalid day* for a month other than *February*

Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateStrings: "31/04/2020", "31/04/2021", "31/06/2020", "31/09/2020", "31/11/2020"*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.1
Test Case ID	C.1.7
Test Case Summary	Use *Date::CreateDate(const string&)* method to construct a *Date* object with a string that actually represents a date on the calendar
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(const string&)* method with a *string* as argument as given in the test data. (2.) Match the *Date::date_, Date::month_* and *Date::year_* data members of the constructed object with their respective values in the string input.
Test Data	*dateStrings: "01/01/1900", "31/12/2099", "29/02/2020", "28/02/2021", "30/04/2021", "30/04/2020", "30/06/2021", "30/09/2021", "30/11/2021"*
Expected Result / Golden Output	(1.) No exception will be thrown. (2.) The data members of the *Date* objects will be as follows. | *string* | *date_* | *month_* | *year_* |

01/01/1900	1	1	1900
31/12/2099	31	12	2099
29/02/2020	29	2	2020
28/02/2021	28	2	2021
30/04/2021	30	4	2021
30/04/2020	30	4	2020
30/06/2021	30	6	2021
30/09/2021	30	9	2021
30/11/2021	30	11	2021

Date of Creation	02 April 2021

C.2. Test Scenarios for ***Construction of Objects by unsigned integers***

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.1
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with *invalid year*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *triplet of unsigned integers* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateTriplets: (16,11,1889), (16,11,2100)*
Expected Result / Golden Output	A *Bad_Date_Year exception* will be thrown in all *test data*

Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.2
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with *invalid month*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *triplet of unsigned integers* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateTriplets: (16,0,2021), (16,13,2021)*
Expected Result / Golden Output	A *Bad_Date_Month exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.3
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with *invalid day* -- *out of bounds*
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *triplet of unsigned integers* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.

Test Data	*dateTriplets: (32,11,2020), (0,11,2020)*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.4
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with *invalid day* for *February*
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *triplet of unsigned integers* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateTriplets: (30,2,2020), (30,2,2021), (29,2,2021)*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.5
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with *invalid day* for a month other than *February*
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned,*

	unsigned, unsigned) method with a *triplet of unsigned integers* as argument as given in the test data. (2.) Surround the function call with *try-catch block*.
Test Data	*dateTriplets: (31,4,2020), (31,4,2021), (31,6,2020), (31,9,2020), (31,11,2020)*
Expected Result / Golden Output	A *Bad_Date_Day exception* will be thrown.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.6
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object with a triplet of *date, month* and *year* that actually represents a date on the calendar
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *triplet of unsigned integers* as arguments as given in the test data. (2.) Match the *Date::date_, Date::month_* and *Date::year_* data members of the constructed object with the passed arguements.
Test Data	*dateTriplets: (1,1,1900), (31,12,2099), (29,2,2020), (28,2,2021), (30,4,2021), (30,4,2020), (30,6,2021), (30,9,2021), (30,11,2021)*
Expected Result / Golden Output	(1.) No exception will be thrown. (2.) The data members of the *Date* objects will be as follows.

triplet	*date_*	*month_*	*year_*

(1,1,1900)	1	1	1900
(31,12,2099)	31	12	2099
(29,2,2020)	29	2	2020
(28,2,2021)	28	2	2021
(30,4,2021)	30	4	2021
(30,4,2020)	30	4	2020
(30,6,2021)	30	6	2021
(30,9,2021)	30	9	2021
(30,11,2021)	30	11	2021

Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.7
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object by passing only a *valid day and month* as arguments.
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *pair of unsigned integers* as arguments as given in the test data. (2.) Match the *Date::date_* and *Date::month_* data members of the constructed object with the passed arguements and *Date::year_* with the *third default parameter*
Test Data	*(day, month): (10,9)*
Expected Result / Golden Output	The data members (*date_, month_, year_*)

	of the *Date* object will be (10,9,1900)
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.8
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object by passing only a *valid day* as argument.
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method with a *single unsigned integers* as argument as given in the test data. (2.) Match the *Date::date_* data member of the constructed object with the passed arguement and *Date::month_* and *Date::year_* with the *second* and *third default parameters* respectively.
Test Data	*day: 10*
Expected Result / Golden Output	The data members (*date_, month_, year_*) of the *Date* object will be (10,1,1900)
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.2
Test Case ID	C.2.9
Test Case Summary	Use *Date::CreateDate(unsigned, unsigned, unsigned)* method to construct a *Date* object by passing no arguments.
Prerequisite System's State	NIL

Procedure	(1.) Call *Date::CreateDate(unsigned, unsigned, unsigned)* method without any arguements. (2.) Match the *Date::date_*, *Date::month_* and *Date::year_* data members with their *default values*.
Test Data	NIL
Expected Result / Golden Output	The data members (*date_, month_, year_*) of the *Date* object will be (1,1,1900)
Date of Creation	02 April 2021

*C.3. Test Scenarios for **Construction of Copies of Object(s)***

Test Plan ID	C
Test Suite ID	C.3
Test Case ID	C.3.1
Test Case Summary	Using *copy constructor* to instantiate *Date* class
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Date* object by using *Date::CreateDate(const string&).* (2.) Construct a *Date* object by passing this *Date* object as argument. (3.) Compare the attributes of the two *Date* objects.
Test Data	dateString: *"02/04/2021"*
Expected Result / Golden Output	The *Date* object constructed in (2.) will have the same attributes as the one in (1.), that are (2,4,2021) respectively for (*date_, month_, year_*)
Date of Creation	02 April 2021

*C.4. Test Scenarios for **Overloaded Equality Check Operator***

Test Plan ID	C
Test Suite ID	C.4
Test Case ID	C.4.1
Test Case Summary	Comparing two *Date* objects with at least one out of date, month and year different, using '==' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a pair of *triplets*, both of which represent an actual date on calendar in (*D,M,Y*) order and both of them are *unequal*. (2.) Construct a pair of *Date* objects representing these two triplets as dates. (3.) Compare the two *Date* objects *with* '==' operator and store the result in a *boolean* variable.
Test Data	tripletPairs: *((1,1,2020), (1,1,2021))*
Expected Result / Golden Output	The value of the *boolean variable* will be *false*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.4
Test Case ID	C.4.2
Test Case Summary	Comparing two *Date* objects with the same date, month and year, using '==' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a *triplet* that represent an actual date on calendar in (*D,M,Y*) order. (2.) Construct a pair of *Date* objects with the same arguments as this triplet. (3.) Compare the two *Date* objects *with* '==' operator and store the result in a *boolean* variable.

Test Data	tripletPairs: *((1,1,2021), (1,1,2021))*
Expected Result / Golden Output	The value of the *boolean variable* will be *true*.
Date of Creation	02 April 2021

C.5. Test Scenarios for ***Overloaded Inequality Check Operator***

Test Plan ID	C
Test Suite ID	C.5
Test Case ID	C.5.1
Test Case Summary	Comparing two *Date* objects with at least one out of date, month and year different, using '!=' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a pair of *triplets*, both of which represent an actual date on calendar in (*D,M,Y*) order and both of them are *unequal*. (2.) Construct a pair of *Date* objects representing these two triplets as dates. (3.) Compare the two *Date* objects *with* '!=' operator and store the result in a *boolean* variable.
Test Data	tripletPairs: *((1,1,2020), (1,1,2021))*
Expected Result / Golden Output	The value of the *boolean variable* will be *true*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.5
Test Case ID	C.5.2
Test Case Summary	Comparing two *Date* objects with the same

	date, month and year, using '!=' operator.
Prerequisite System's State	NIL
Procedure	(1.) Choose a *triplet* that represent an actual date on calendar in (*D,M,Y*) order. (2.) Construct a pair of *Date* objects with the same arguments as this triplet. (3.) Compare the two *Date* objects *with* '!=' operator and store the result in a *boolean* variable.
Test Data	tripletPairs: *((1,1,2021), (1,1,2021))*
Expected Result / Golden Output	The value of the *boolean variable* will be *false*.
Date of Creation	02 April 2021

C.6. Test Scenarios for ***Overloaded Output Streaming Operator***

Test Plan ID	C
Test Suite ID	C.6
Test Case ID	C.6.1
Test Case Summary	Print a *Date* object onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Date* object passing a valid date in *string* format to *Date::CreateDate(const string&)* as argument. (2.) Print the constructed object onto the console using *cout* and *output streaming operator* <<.
Test Data	dateString: *"01/01/2021"*
Expected Result / Golden Output	"*01/Jan/2021*" will be printed onto the console.
Date of Creation	02 April 2021

*C.7. Test Scenarios for **Overloaded Copy Assignment Operator***

Test Plan ID	C
Test Suite ID	C.7
Test Case ID	C.7.1
Test Case Summary	Using *copy assignment operato '='*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects by using *Date::CreateDate(const string&)*, passing two *distinct* valid dates in *string* format as inputs, as given in *test data.* (2.) Copy the second one to the first using *"="* operator.
Test Data	(*destination, source*): *("01/01/2020", "03/04/2021")*
Expected Result / Golden Output	The data members (*date_, month_, year_*) for the *destination Date object* will have values *(3,4,2021)* respectively
Date of Creation	02 April 2021

*C.8. Test Scenarios for other **Static Member Functions***

Test Plan ID	C
Test Suite ID	C.8
Test Case ID	C.8.1
Test Case Summary	Using *Date::GetTodaysDate*
Prerequisite System's State	NIL
Procedure	(1.) Call *Date::GetTodaysDate* and store the returned *Date* object in a variable. (2.) Match the attributes of the object with the *real date* on the *day this test case is*

	executed.
Test Data	NIL
Expected Result / Golden Output	The data members (*date_, month_, year_*) will have the same values as the date on the system at the time of execution. If executed on *02 April 2021*, the values will be *(2,4,2021)* respectively.
Date of Creation	02 April 2021

C.9. Test Scenarios for ***Non Static Member Functions***

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.1
Test Case Summary	Using *Date::GetDifferenceInYears* -- check for a *positive return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("11/03/2022", "02/04/2021"), ("11/12/2022", "02/04/2021")*
Expected Result / Golden Output	The returned value will be *1* in first and *2* in second *test data*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.2

Test Case Summary	Using *Date::GetDifferenceInYears* -- check for a *zero return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("11/09/2021", "02/04/2021"), ("02/12/2020", "02/04/2021")*
Expected Result / Golden Output	The returned value will be *0* for both *test data*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.3
Test Case Summary	Using *Date::GetDifferenceInYears* -- check for a *negative return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2021", "11/05/2022"), ("02/04/2021", "11/12/2022")*
Expected Result / Golden Output	The returned value will be *-1* in first and *-2* in second *test data*.
Date of Creation	02 April 2021

Test Plan ID	C

Test Suite ID	C.9
Test Case ID	C.9.4
Test Case Summary	Using *Date::GetDifferenceInDays* -- check for a *positive return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInDays(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2021", "02/04/2019"), ("02/04/2021", "01/04/2021")*
Expected Result / Golden Output	The returned value will be *731* for first and *1* for second *test data*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.5
Test Case Summary	Using *Date::GetDifferenceInDays* -- check for a *zero return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2021", "02/04/2021")*
Expected Result / Golden Output	The returned value will be *0*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.6
Test Case Summary	Using *Date::GetDifferenceInDays* -- check for a *negative return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2019", "02/04/2021"), ("01/04/2021", "02/04/2021")*
Expected Result / Golden Output	The returned value will be *-731* in first and *-1* in second *test data*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.7
Test Case Summary	Using *Date::IsAfter* -- check for a *false return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.IsAfter(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2021", "02/04/2021"), ("01/04/2021", "02/04/2021")*
Expected Result / Golden Output	The returned value will be *false* for both the *test data*.

Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.8
Test Case Summary	Using *Date::IsAfter* -- check for a *true return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Date* objects, *d1* and *d2* by passing valid dates in string formats to *Date::CreateDate(const string&) method*. (2.) Call *"d1.GetDifferenceInYears(d2)"* and check the returned value.
Test Data	*(string_d1, string_d2): ("02/04/2021", "01/04/2021")*
Expected Result / Golden Output	The returned value will be *true*.
Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.9
Test Case Summary	Using *Date::IsLeapYear* -- check for a *false return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Date object* by passing valid date in string formats to *Date::CreateDate(const string&) method*. (2.) Call *Date::IsLeapYear* on the object and check the returned value.
Test Data	*dateString: "02/04/2021"*
Expected Result / Golden Output	The returned value will *false*.

Date of Creation	02 April 2021

Test Plan ID	C
Test Suite ID	C.9
Test Case ID	C.9.10
Test Case Summary	Using *Date::IsLeapYear* -- check for a *true return value*
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Date object* by passing valid date in string formats to *Date::CreateDate(const string&) method*. (2.) Call *Date::IsLeapYear* on the object and check the returned value.
Test Data	*dateString: "02/04/2020"*
Expected Result / Golden Output	The returned value will *true*.
Date of Creation	02 April 2021

D. *Unit Test Plan for **BookingClass Hierarchy***

D.1. *Test Scenarios for **Overloaded Output Streaming Operator***

Test Plan ID	D
Test Suite ID	D.1
Test Case ID	D.1.1
Test Case Summary	Print the singleton instance of any *BookingClass static sub-type* object onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Print the instance onto the console using the *cout* output stream object and *output streaming operator* <<.
Test Data	NIL
Expected Result / Golden Output	Details of the booking class *ACFirstClass* will be printed onto the console. +++ DETAILS OF THE BOOKING CLASS +++ - Name : AC First Class - Load factor : 6.5 - No. of tiers : 2 - Is sitting : 0 - Is AC : 1 - Is luxury : 1 - Reservation Charge : 1 - Tatkal Charge : 0.3 - Minimum Distance for Tatkal Charge : 500 - Minimum Tatkal Charge : 500 - Maximum Tatkal Charge : 400
Date of Creation	02 April 2021

D.2. *Test Scenarios for **Non Static Member Functions***

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.1
Test Case Summary	Use *BookingClassTypes<T>::GetName* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetName* method on the instance and check the return value. (3.) Now call *"BookingClass::ExecutiveChairCar::Type()"* to get the *singleton instance* of *ExecutiveChairCar sub-type*. (4.) Call *BookingClassTypes<T>::GetName* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *string "AC First Class"* and *"Executive Chair Car"* respectively
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.2
Test Case Summary	Use *BookingClassTypes<T>::GetLoadFactor* on the singleton instance of any *BookingClass static sub-type*

Prerequisite System’s State	NIL
Procedure	(1.) Call *“BookingClass::ACFirstClass::Type()”* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetLoadFactor* method on the instance and check the return value. (3.) Now call *“BookingClass::AC2Tier::Type()”* to get the *singleton instance* of *AC2Tier sub-type*. (4.) Call *BookingClassTypes<T>::GetLoadFactor* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 6.5* and *4.0* respectively
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.3
Test Case Summary	Use *BookingClassTypes<T>::IsSitting* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System’s State	NIL
Procedure	(1.) Call *“BookingClass::ACFirstClass::Type()”* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::IsSitting* method on the instance and check the return value. (3.) Now call *“BookingClass::FirstClass::Type()”* to get

	the *singleton instance* of *FirstClass sub-type*. (4.) Call *BookingClassTypes<T>::IsSitting* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean false* for both the sub-types.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.4
Test Case Summary	Use *BookingClassTypes<T>::IsAC* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::IsAC* method on the instance and check the return value. (3.) Now call *"BookingClass::AC3Tier::Type()"* to get the *singleton instance* of *AC3Tier sub-type*. (4.) Call *BookingClassTypes<T>::IsAC* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean true* for both the sub-types.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.5
Test Case Summary	Use *BookingClassTypes<T>::IsLuxury* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::IsLuxury* method on the instance and check the return value. (3.) Now call *"BookingClass::ACChairCar::Type()"* to get the *singleton instance* of *ACChairCar sub-type*. (4.) Call *BookingClassTypes<T>::IsLuxury* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean true* and *false* respectively.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.6
Test Case Summary	Use *BookingClassTypes<T>::GetNumberOfTiers* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL

Procedure	(1.) Call *“BookingClass::ACFirstClass::Type()”* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetNumberOfTiers* method on the instance and check the return value. (3.) Now call *“BookingClass::Sleeper::Type()”* to get the *singleton instance* of *Sleeper sub-type*. (4.) Call *BookingClassTypes<T>::GetNumberOfTiers* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *2* and *3* respectively.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.7
Test Case Summary	Use *BookingClassTypes<T>::GetReservationCharge* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System’s State	NIL
Procedure	(1.) Call *“BookingClass::ACFirstClass::Type()”* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetReservationCharge* method on the instance and check the return value. (3.) Now call *“BookingClass::SecondSitting::Type()”* to

	get the *singleton instance* of *SecondSitting sub-type*. (4.) Call *BookingClassTypes<T>::GetReservationCharge* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *double 60.0* and *15.0* respectively.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.8
Test Case Summary	Use *BookingClassTypes<T>::GetTatkalCharge* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetTatkalCharge* method on the instance and check the return value. (3.) Now call *"BookingClass::SecondSitting::Type()"* to get the *singleton instance* of *SecondSitting sub-type*. (4.) Call *BookingClassTypes<T>::GetTatkalCharge* method on this instance and check the return value.
Test Data	NIL

Expected Result / Golden Output	The returned value will be the *double 0.3* and *0.1* respectively
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.9
Test Case Summary	Use *BookingClassTypes<T>::GetMinTatkalCharge* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetMinTatkalCharge* method on the instance and check the return value. (3.) Now call *"BookingClass::SecondSitting::Type()"* to get the *singleton instance* of *SecondSitting sub-type*. (4.) Call *BookingClassTypes<T>::GetMinTatkalCharge* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 400.0* and *10.0* respectively
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2

Test Case ID	D.2.10
Test Case Summary	Use *BookingClassTypes<T>::GetMaxTatkalCharge* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetMaxTatkalCharge* method on the instance and check the return value. (3.) Now call *"BookingClass::SecondSitting::Type()"* to get the *singleton instance* of *SecondSitting sub-type*. (4.) Call *BookingClassTypes<T>::GetMaxTatkalCharge* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 500.0* and *15.0* respectively.
Date of Creation	02 April 2021

Test Plan ID	D
Test Suite ID	D.2
Test Case ID	D.2.11
Test Case Summary	Use *BookingClassTypes<T>::GetMinDistanceForTatkalCharge* on the singleton instance of any *BookingClass static sub-type*
Prerequisite System's State	NIL

Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type*. (2.) Call *BookingClassTypes<T>::GetMinDistanceForTatkalCharge* method on the instance and check the return value. (3.) Now call *"BookingClass::ACChairCar::Type()"* to get the *singleton instance* of *ACChairCar sub-type*. (4.) Call *BookingClassTypes<T>::GetMinDistanceForTatkalCharge* method on this instance and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *500* and *250* respectively.
Date of Creation	02 April 2021

D.3. Test Scenarios for ***Static Member Function***

Test Plan ID	D
Test Suite ID	D.3
Test Case ID	D.3.1
Test Case Summary	Call *BookingClass<T>::Type* method twice for any *BookingClass sub-type* and check if the same object is returned -- *test for singleton class*
Prerequisite System's State	NIL
Procedure	(1.) Call *BookingClass::ACFirstClass::Type* method and store the returned instance in a *const BookingClass reference*. (2.) Call *BookingClass::ACFirstClass::Type* method again and store the returned instance in another *const BookingClass*

	reference. (3.) Compare the *addresses* of the two *BookingClass references* using *'=='* and store the result in a *boolean* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *boolean* variable will be *true*
Date of Creation	03 April 2021

*D.4. Test Scenarios to test **Dynamic Dispatch of Polymorphic Methods***

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.1
Test Case Summary	Use *BookingClassTypes<T>::GetName* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetName* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *string "AC First Class"*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4

Test Case ID	D.4.2
Test Case Summary	Use *BookingClassTypes<T>::GetLoadFactor* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetLoadFactor* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 6.5*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.3
Test Case Summary	Use *BookingClassTypes<T>::IsSitting* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::IsSitting* method on the variable and check the

	return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean false*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.4
Test Case Summary	Use *BookingClassTypes<T>::IsAC* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::IsAC* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean true*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.5
Test Case Summary	Use *BookingClassTypes<T>::IsLuxury* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const*

	BookingClass reference.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::IsLuxury* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *boolean true*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.6
Test Case Summary	Use *BookingClassTypes<T>::GetNumberOfTiers* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetNumberOfTiers* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *2*

Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.7
Test Case Summary	Use *BookingClassTypes<T>::GetReservationCharge* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetReservationCharge* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *double 60.0*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.8
Test Case Summary	Use *BookingClassTypes<T>::GetTatkalCharge* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL

Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetTatkalCharge* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 0.3*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.9
Test Case Summary	Use *BookingClassTypes<T>::GetMinTatkalCharge* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetMinTatkalCharge* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 400.0*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.10
Test Case Summary	Use *BookingClassTypes<T>::GetMaxTatkalCharge* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass sub-type* and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetMaxTatkalCharge* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *double 500.0*
Date of Creation	03 April 2021

Test Plan ID	D
Test Suite ID	D.4
Test Case ID	D.4.11
Test Case Summary	Use *BookingClassTypes<T>::GetMinDistanceForTatkalCharge* on the singleton instance of any *BookingClass static sub-type* upcasted to a *const BookingClass reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingClass::ACFirstClass::Type()"* to get the *singleton instance* of *ACFirstClass*

	sub-type and store it in a *const BookingClass reference* variable. (2.) Call *BookingClassTypes<T>::GetMinDistanceForTatkalCharge* method on the variable and check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be *500*
Date of Creation	03 April 2021

*E. Unit Test Plan for **Divyaang Hierarchy***

*E.1. Test Scenarios for **Overloaded Output Streaming Operator***

Test Plan ID	E
Test Suite ID	E.1
Test Case ID	E.1.1
Test Case Summary	Print the singleton instance of any *Divyaang static sub-type* object onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type*. (2.) Print the instance onto the console using the *cout* output stream object and *output streaming operator* <<.
Test Data	NIL
Expected Result / Golden Output	*Name* and *concessions* for *Blind* divyaang category in different *BookingClass*es will be printed onto the console. Blind Concession for AC First Class: 0.5 Concession for Executive Chair Car: 0.75 Concession for AC 2 Tier: 0.5 Concession for First Class: 0.75 Concession for AC 3 Tier: 0.75 Concession for AC Chair Car: 0.75 Concession for Sleeper: 0.75 Concession for Second Sitting: 0.75
Date of Creation	02 April 2021

*E.2. Test Scenarios for **Non Static Member Functions***

Test Plan ID	E
Test Suite ID	E.2
Test Case ID	E.2.1

Test Case Summary	Use *DivyaangTypes<T>::GetName* on the singleton instance of any *Divyaang static sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type*. (2.) Call *DivyaangTypes<T>::GetName* method on the instance. (3.) Check the return value. (4.) Now call *"Divyaang::TBPatients::Type()"* to get the *singleton instance* of *TBPatients sub-type*. (5.) Call *DivyaangTypes<T>::GetName* method on this instance. (6.) Check the return value.
Test Data	NIL
Expected Result / Golden Output	The returned value will be the *string "Blind"* and *"TB Patients"* respectively
Date of Creation	02 April 2021

Test Plan ID	E
Test Suite ID	E.2
Test Case ID	E.2.2
Test Case Summary	Use *DivyaangTypes<T>::GetConcessionFactor* on the singleton instance of any *Divyaang static sub-type* by passing a *valid BookingClass sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type*. (2.) Call *DivyaangTypes<T>::GetConcessionFactor* method on the instance, passing as argument the singleton instance of a *BookingClass sub-type*, as given in the *test data*.

	(3.) Check the return value.
Test Data	*bookingClassArg: BookingClass::ACFirstClass::Type()*, *BookingClass::FirstClass::Type()*
Expected Result / Golden Output	The returned value will be the *double 0.5* and *0.75* for the first and second *test data* respectively.
Date of Creation	02 April 2021

Test Plan ID	E
Test Suite ID	E.2
Test Case ID	E.2.3
Test Case Summary	Use *DivyaangTypes<T>::GetConcessionFactor* on the singleton instance of any *Divyaang static sub-type* by passing an *invalid BookingClass sub-type*
Prerequisite System's State	An *invalid BookingClass sub-type* should be defined. This must be different from the *8 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *BCTestType*. Initialize all the *static const data members* of *BookingClassTypes<BCTestType>* with arbitrary values (of appropriate data types)
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type*. (2.) Call *DivyaangTypes<T>::GetConcessionFactor* method on the instance, passing as argument the singleton instance of a *BookingClass sub-type*, as given in the *test data*. (3.) Surround the function call with *try-catch block*.
Test Data	*bookingClassArg: BookingClassTypes<BCTestType>::Type()*

Expected Result / Golden Output	A *Bad_Access exception* will be caught
Date of Creation	03 April 2021

Test Plan ID	E
Test Suite ID	E.2
Test Case ID	E.2.4
Test Case Summary	Use *DivyaangTypes<T>::GetConcessionFactor* on the singleton instance of any *Divyaang static sub-type* by passing a *valid BookingClass sub-type*
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::TBPatients::Type()"* to get the *singleton instance* of *Blind sub-type*. (2.) Call *DivyaangTypes<T>::GetConcessionFactor* method on the instance, passing as argument the singleton instance of a *BookingClass sub-type*, as given in the *test data*. (3.) Check the return value.
Test Data	*bookingClassArg: BookingClass::ACFirstClass::Type()*
Expected Result / Golden Output	The returned value will be the *double 0.0*
Date of Creation	03 April 2021

E.3. Test Scenarios for ***Static Member Function***

Test Plan ID	E
Test Suite ID	E.3
Test Case ID	E.3.1
Test Case Summary	Call *Divyaang<T>::Type* method twice for any *Divyaang sub-type* and check if the

	same object is returned -- *test for singleton class*
Prerequisite System's State	NIL
Procedure	(1.) Call *Divyaang::Blind::Type* method and store the returned instance in a *const Divyaang reference*. (2.) Call *Divyaang::Blind::Type* method again and store the returned instance in another *const Divyaang reference*. (3.) Compare the *addresses* of the two *Divyaang references* using *'=='* and store the result in a *boolean* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *boolean* variable will be *true*
Date of Creation	03 April 2021

E.4. Test Scenarios to test ***Dynamic Dispatch of Polymorphic Methods***

Test Plan ID	E
Test Suite ID	E.4
Test Case ID	E.4.1
Test Case Summary	Use *DivyaangTypes<T>::GetName* on the singleton instance of any *Divyaang static sub-type* upcasted to a *const Divyaang reference*
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type* and store it in a *const Divyaang reference* variable. (2.) Call *DivyaangTypes<T>::GetName* method on the variable. (3.) Check the return value.
Test Data	NIL

Expected Result / Golden Output	The returned value will be the *string "Blind"*
Date of Creation	03 April 2021

Test Plan ID	E
Test Suite ID	E.4
Test Case ID	E.4.2
Test Case Summary	Use *DivyaangTypes<T>::GetConcessionFactor* on the singleton instance of any *Divyaang static sub-type* by passing a *valid BookingClass sub-type* upcasted to a *const Divyaang reference*
Prerequisite System's State	NIL
Procedure	(1.) Call *"Divyaang::Blind::Type()"* to get the *singleton instance* of *Blind sub-type* and store it in a *const Divyaang reference* variable. (2.) Call *DivyaangTypes<T>::GetConcessionFactor* method on the variable, passing as argument the singleton instance of a *BookingClass sub-type*, as given in the *test data*. (3.) Check the return value.
Test Data	*bookingClassArg: BookingClass::ACFirstClass::Type()*, *BookingClass::FirstClass::Type()*
Expected Result / Golden Output	The returned value will be the *double 0.5* and *0.75* for the first and second *test data* respectively.
Date of Creation	03 April 2021

F. Unit Test Plan for **Concessions Hierarchy**

F.1. Test Scenarios for **Static Member Functions**

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.1
Test Case Summary	Use *GeneralConcession::GetConcessionFactor*
Prerequisite System's State	NIL
Procedure	Call *GeneralConcession::GetConcessionFactor* method and store the returned value in a *double* variable
Test Data	NIL
Expected Result / Golden Output	Returned value will be *0.0*
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.2
Test Case Summary	Use *LadiesConcession::GetConcessionFactor* by passing a *Passenger* who is ineligible for *Ladies* booking category as arguement
Prerequisite System's State	NIL
Procedure	(1.) Call *LadiesConcession::GetConcessionFactor* method by passing a passenger, as given in *test data*, as argument. (2.) Surround the function call with *try-catch block*.
Test Data	*passenger:*

	Passenger::CreatePassenger(Date::Create Date("15/04/2006"), "Male", "123456789012", "John")
Expected Result / Golden Output	A *Bad_Elligibility exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.3
Test Case Summary	Use *LadiesConcession::GetConcessionFactor* by passing a *Passenger* who is eligible for *Ladies* booking category as arguement
Prerequisite System's State	NIL
Procedure	Call *LadiesConcession::GetConcessionFactor* method by passing a passenger, as given in *test data*, as argument and store the returned value in a *double* variable

Test Data	*passenger:*
	Passenger::CreatePassenger(Date::Create Date("15/04/2010"), "Male", "123456789012", "John")
	Passenger::CreatePassenger(Date::Create Date("15/12/1990"), "Female", "123456789012", "Jane")

Expected Result / Golden Output	For both the *test data* -- (1.) No exception will be thrown (2.) Returned value will be *0.0* for both the *test data*.
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1

Test Case ID	F.1.4
Test Case Summary	Use *SeniorCitizenConcession::GetConcessionFactor* by passing a *Passenger* who is ineligible for *SeniorCitizen* booking category as arguement
Prerequisite System's State	NIL
Procedure	(1.) Call *SeniorCitizenConcession::GetConcessionFactor* method by passing a passenger, as given in *test data*, as argument. (2.) Surround the function call with *try-catch block*.
Test Data	*passenger: Passenger::CreatePassenger(Date::CreateDate("15/04/1963"), "Male", "123456789012", "John")*
Expected Result / Golden Output	A *Bad_Elligibility exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.5
Test Case Summary	Use *SeniorCitizenConcession::GetConcessionFactor* by passing a *Passenger* who is eligible for *SeniorCitizen* booking category as arguement
Prerequisite System's State	NIL
Procedure	Call *SeniorCitizenConcession::GetConcessionFactor* method by passing a passenger, as given in *test data*, as argument and store the returned value in a *double* variable
Test Data	*passenger:*

	Passenger::CreatePassenger(Date::CreateDate("15/04/1958"), "Male", "123456789012", "John")
	Passenger::CreatePassenger(Date::CreateDate("15/04/1961"), "Female", "123456789013", "Jane")

Expected Result / Golden Output	(1.) No exception will be thrown in both the *test data* (2.) Returned value will be *0.4* for the first and *0.5* for the second *test data*
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.6
Test Case Summary	Use *DivyaangConcession::GetConcessionFactor* by passing a *Passenger* who is ineligible for *Divyaang* booking category as arguement
Prerequisite System's State	NIL
Procedure	(1.) Call *DivyaangConcession::GetConcessionFactor* method by passing a passenger and a *BookingClass sub-type*, as given in *test data*, as argument. (2.) Surround the function call with *try-catch block*.
Test Data	*(passenger, bookingClass):* *(Passenger::CreatePassenger(Date::CreateDate("15/04/2020"), "Male", "123456789012", "John"), BookingClass::ACFirstClass::Type())*
Expected Result / Golden Output	A *Bad_Elligibility exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.7
Test Case Summary	Use *DivyaangConcession::GetConcessionFactor* by passing as parameters a *Passenger* who is eligible for *Divyaang* booking category and an *invalid BookingClass sub-type*
Prerequisite System's State	An *invalid BookingClass sub-type* should be defined. This must be different from the *8 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *TestType*. Initialize all the *static const data members* of *BookingClassTypes<TestType>* with arbitrary values (of appropriate data types)
Procedure	(1.) Call *DivyaangConcession::GetConcessionFactor* method by passing a passenger and a *BookingClass sub-type*, as given in *test data*, as argument. (2.) Surround the function call with *try-catch block*.
Test Data	*(passenger, bookingClass): (Passenger::CreatePassenger(Date::CreateDate("15/04/2020"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()), BookingClassTypes<TestType>::Type())*
Expected Result / Golden Output	A *Bad_Access exception* will be caught
Date of Creation	03 April 2021

Test Plan ID	F
Test Suite ID	F.1
Test Case ID	F.1.8

Test Case Summary	Use *DivyaangConcession::GetConcessionFactor* by passing a *Passenger* who is eligible for *Divyaang* booking category and a valid *BookingClass sub-type* as arguements
Prerequisite System's State	NIL
Procedure	Call *DivyaangConcession::GetConcessionFactor* method by passing a passenger and a *BookingClass sub-type*, as given in *test data*, as argument and store the returned value in a *double* variable
Test Data	*(passenger, bookingClass): (Passenger::CreatePassenger(Date::CreateDate("15/04/2020"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()), BookingClass::ACFirstClass::Type()) ,* *(Passenger::CreatePassenger(Date::CreateDate("15/04/2020"), "Male", "123456789012", "John", "", "", "", &Divyaang::TBPatients::Type()), BookingClass::ACFirstClass::Type()) ,* *(Passenger::CreatePassenger(Date::CreateDate("15/04/2020"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()), BookingClass::FirstClass::Type())*
Expected Result / Golden Output	(1.) No exception will be thrown in both the *test data* (2.) Returned value will be *0.5* for the first, *0.0* for the second and *0.75* for the third *test data*
Date of Creation	02 April 2021

*G. Unit Test Plan for **Passenger***

*G.1. Test Scenarios for **Construction of Object(s)***

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.1
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object without *first* and *last* name
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Date::CreateDate("16/11/2001"), "Male", "123456789012")*
Expected Result / Golden Output	A *Bad_Passenger_Name exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.2
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *adhaar number* of length unequal to 12
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch*

	block.
Test Data	*parameters:* *(Date::CreateDate("16/11/2001"), "Male", "12345678901", "John")* *(Date::CreateDate("16/11/2001"), "Male", "1234567890123", "John")*
Expected Result / Golden Output	A *Bad_Passenger_AdhaarNumber exception* will be caught in both the *test data*
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.3
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *adhaar number* of length 12 but having *non-digits*
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Date::CreateDate("16/11/2001"), "Male", "123456A78901", "John")*
Expected Result / Golden Output	A *Bad_Passenger_AdhaarNumber exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1

Test Case ID	G.1.4
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *mobile number* specified but of length unequal to 10
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Date::CreateDate("16/11/2001"), "Male", "123456789012", "John", "", "", "999999999")* *(Date::CreateDate("16/11/2001"), "Male", "123456789012", "John", "", "", "99999999999")*
Expected Result / Golden Output	A *Bad_Passenger_MobileNumber exception* will be caught in both the *test data*
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.5
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *mobile number* of length 10 but having *non-digits*
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch*

	block.
Test Data	*parameters: (Date::CreateDate("16/11/2001"), "Male", "123456789012", "John", "", "", "99999A9999")*
Expected Result / Golden Output	A *Bad_Passenger_MobileNumber exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.6
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *date of birth* in future.
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Date::CreateDate("02/04/2022"), "Male", "123456789012", "John")*
Expected Result / Golden Output	A *Bad_Passenger_DateOfBirth exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.7
Test Case Summary	Using *Passenger::CreatePassenger*

	method to construct a *Passenger* object with *divyaang type specified* but *invalid*.
Prerequisite System's State	An *invalid Divyaang sub-type* should be defined. This must be different from the *4 valid Divyaang sub-types*. To achieve this define a *struct placeholder* with name *DivTestType*. Initialize all the *static const data members* of *DivyaangTypes<DivTestType>* with arbitrary values (of appropriate data types). Write a trivial function definition for *DivyaangTypes<DivTestType>::GetConcessionFactor* that simply returns *0.0*
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Date::CreateDate("01/01/2021"), "Male", "123456789012", "John", "", "", "", &DivyaangTypes<DivTestType>::Type())*
Expected Result / Golden Output	A *Bad_Passenger_DisabilityType exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.8
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with *gender* other than *Male* and *Female*.
Prerequisite System's State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*.

	(2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Date::CreateDate(“01/01/2021”), “NewGender”, “123456789012”, “John”)*
Expected Result / Golden Output	A *Bad_Passenger_Gender exception* will be caught
Date of Creation	02 April 2021

Test Plan ID	G
Test Suite ID	G.1
Test Case ID	G.1.9
Test Case Summary	Using *Passenger::CreatePassenger* method to construct a *Passenger* object with a *valid set of arguements*.
Prerequisite System’s State	NIL
Procedure	(1.) Call *Passenger::CreatePassenger* method by passing the set(s) of parameters as given in the *test data*. (2.) Match the *non-static data members* of the new *Passenger* instance with the *passed* and *default parameters*, as applicable.
Test Data	*parameters:* *(Date::GetTodaysDate(), “Male”, “123456789012”, “John”)* *(Date::CreateDate(“01/01/2017”), “FeMaLe”, “123456789012”, “Jane”)* *(Date::CreateDate(“01/01/2017”), “MaLe”, “123456789012”, “”, “”, “Doe”)* *(Date::CreateDate(“01/01/2017”), “fEMaLe”, “123456789012”, “Jane”, “”, “”, “9874563210”)*

	(Date::CreateDate("01/01/2017"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()) *(Date::CreateDate("01/01/2017"), "mAlE", "123456789012", "John", "Jack", "Doe", "9874563210", &Divyaang::Blind::Type(), "ABC987")*
Expected Result / Golden Output	(1.) No exception will be thrown in all the *test data*. (2.) The values of the data members *(firstName_, middleName_, lastName_, dateOfBirthStringFormat, gender_, adhaarNumber_, mobileNumber_, disabilityType_, disabilityID_)* for the *test data* will be -- *("John", "", "", "02/04/2021", Gender::Male::Type(), "123456789012", "", NULL, "")* Here expected value of *dateOfBirthStringFormat* depends on the *date of execution* of this test case. Here the expected output is written according to the *date of creation* *("Jane", "", "", "01/01/2017", Gender::Female::Type(), "123456789012", "", NULL, "")* *("", "", "Doe", "01/01/2017", Gender::Male.Type(), "123456789012", "", NULL, "")* *("Jane", "", "", "01/01/2017", Gender::Female::Type(), "123456789012", "9874563210", NULL, "")* *("John", "", "", "01/01/2017", Gender::Male::Type(), "123456789012", "", &Divyaang::Blind::Type(), "")* *("John", "Jack", "Doe", "01/01/2017", Gender::Male::Type(), "123456789012", "9874563210", &Divyaang::Blind::Type(), "ABC987")*

Date of Creation	02 April 2021

G.2. Test Scenarios for ***Construction of Copies of Object(s)***

Test Plan ID	G
Test Suite ID	G.2
Test Case ID	G.2.1
Test Case Summary	Using *copy constructor* to instantiate *Passenger* class
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Passenger* object by passing a *valid set* of arguments, as given in *test data.* (2.) Construct a *Passenger* object by passing this *Passenger* object as argument. (3.) Compare all the attributes of the two *Passenger* objects.
Test Data	*sourceParameters: (Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane", "John", "Doe", "9874563210", &Divyaang::Blind::Type(), "ABC123")*
Expected Result / Golden Output	The *Passenger* object constructed in (2.) will have the same attributes as the one in (1.). The *data members* of both the objects will satisfy the following. *(LHS=RHS)*

*firstName_*	*"Jane"*
*middleName_*	*"John"*
*lastName_*	*"Doe"*
*dateOfBirth_*	*Date::CreateDate("01/01/2021")*
*&gender_*	*&Gender::Female::Type()*
*adhaarNumber_*	*"123456789012"*

	<table><tr><td>*mobileNumber_*</td><td>*“9874563210”*</td></tr><tr><td>*disabilityType_*</td><td>*&Divyaang::Blind::Type()*</td></tr><tr><td>*disabilityID_*</td><td>*“ABC123”*</td></tr></table>
Date of Creation	02 April 2021

G.3. Test Scenarios for ***Overloaded Output Streaming Operator***

Test Plan ID	G
Test Suite ID	G.3
Test Case ID	G.3.1
Test Case Summary	Construct and print a *Passenger* object onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Passenger* object by passing a *valid set* of arguments, as given in *test data.* (2.) Print the instance onto the console using the *cout* output stream object and *output streaming operator* <<.
Test Data	*parameters:* *(Date::CreateDate(“01/01/2021”), “Female”, “123456789012”, “Jane”, “John”, “Doe”, “9874563210”, &Divyaang::Blind::Type(), “ABC123”)*
Expected Result / Golden Output	All the details/attributes of the *Passenger* will be printed onto the console. Name = Jane John Doe Adhaar Card No. = 123456789012 Date of Birth = 01/Jan/2021 Gender = Female / Ms. Mobile No. = 9874563210 Disability Type = Blind Disability ID = ABC12

Date of Creation	03 April 2021

G.4. Test Scenarios for ***Non Static Member Functions***

Test Plan ID	G
Test Suite ID	G.4
Test Case ID	G.4.1
Test Case Summary	Use *Passenger::GetGender* method on a *Passenger* object and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Passenger* object by passing a *valid set(s)* of arguments, as given in *test data.* (2.) Call *Passenger::GetGender* method on the object and store the returned value in a *const Gender reference* variable.
Test Data	*parameters:* *(Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane")* *(Date::CreateDate("01/01/2021"), "Male", "123456789012", "John")*
Expected Result / Golden Output	The *address* of the variable will be same as *"&Gender::Female::Type()"* in the first *test data*. The *address* of the variable will be same as *"&Gender::Male::Type()"* in the second *test data*.
Date of Creation	03 April 2021

Test Plan ID	G
Test Suite ID	G.4
Test Case ID	G.4.2
Test Case Summary	Use *Passenger::GetDisabilityType* method

	on a *Passenger* object and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Passenger* object by passing a *valid set(s)* of arguments, as given in *test data.* (2.) Call *Passenger::GetDisabilityType* method on the object and store the returned value in a *"const Divyaang*"* variable.
Test Data	*parameters:* *(Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane", "", "", "", &Divyaang::Blind::Type())* *(Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane")*
Expected Result / Golden Output	The value of the variable will be same as "*&Divyaang::Blind::Type()*" in the first *test data*. The value of the variable will be *NULL* in the second *test data*.
Date of Creation	03 April 2021

Test Plan ID	G
Test Suite ID	G.4
Test Case ID	G.4.3
Test Case Summary	Use *Passenger::GetAge* method on a *Passenger* object and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Construct a *Passenger* object by passing a *valid set(s)* of arguments, as given in *test data.* (2.) Call *Passenger::GetAge* method on the object and store the returned value in an *unsigned int* variable.

Test Data	*parameters:* *(Date::CreateDate("30/11/2020"), "Female", "123456789012", "Jane")* *(Date::CreateDate("30/06/2020"), "Male", "123456789012", "John")* *(Date::CreateDate("16/11/2001"), "Male", "123456789012", "John")*
Expected Result / Golden Output	The returned value will be *0, 1, 19* respectively for *first, second, third test data*.
Date of Creation	03 April 2021

G.5. Test Scenarios for ***Overloaded Equality Check Operator***

Test Plan ID	G
Test Suite ID	G.5
Test Case ID	G.5.1
Test Case Summary	Comparing two *Passenger* objects with at least one out of all the attributes different, using '==' operator.
Prerequisite System's State	NIL
Procedure	(1.) Construct two *Passenger* objects each with a set of parameters as given in *test data*. (2.) Compare the two *Passenger* objects *with* '==' operator and store the result in a *boolean* variable.
Test Data	*parametersPassenger1*: *(Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane", "John", "Doe", "9874563210", &Divyaang::Blind::Type(), "ABC123")* *parametersPassenger2*: *(Date::CreateDate("01/01/2021"), "Female", "123456789012", "Jane", "John", "Doe", "9874563210", &Divyaang::Blind::Type(), "ABC124")*

Expected Result / Golden Output	The value of the *boolean variable* will be *false*.
Date of Creation	05 April 2021

Test Plan ID	G
Test Suite ID	G.5
Test Case ID	G.5.2
Test Case Summary	Comparing two *Passenger* objects with all the attributes same, using ‘==’ operator.
Prerequisite System’s State	NIL
Procedure	(1.) Construct two *Passenger* objects each with a set of parameters as given in *test data*. (2.) Compare the two *Passenger* objects *with* ‘==’ operator and store the result in a *boolean* variable.
Test Data	*parametersPassenger1*: *(Date::CreateDate(“01/01/2021”), “Female”, “123456789012”, “Jane”, “John”, “Doe”, “9874563210”, &Divyaang::Blind::Type(), “ABC123”)* *parametersPassenger2*: *(Date::CreateDate(“01/01/2021”), “Female”, “123456789012”, “Jane”, “John”, “Doe”, “9874563210”, &Divyaang::Blind::Type(), “ABC123”)*
Expected Result / Golden Output	The value of the *boolean variable* will be *true*.
Date of Creation	05 April 2021

H. Unit Test Plan for **Gender Hierarchy**

H.1. Test Scenarios for **Static Member Functions**

Test Plan ID	H
Test Suite ID	H.1
Test Case ID	H.1.1
Test Case Summary	Call *GenderTypes<T>::Type* method twice for any *Gender sub-type* and check if the same object is returned -- *test for singleton class*
Prerequisite System's State	NIL
Procedure	(1.) Call *Gender::Male::Type* method and store the returned instance in a *const Gender reference*. (2.) Call *Gender::Male::Type* method again and store the returned instance in another *const Gender reference*. (3.) Compare the *addresses* of the two *Gender references* using *'=='* and store the result in a *boolean* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *boolean* variable will be *true*
Date of Creation	03 April 2021

Test Plan ID	H
Test Suite ID	H.1
Test Case ID	H.1.2
Test Case Summary	Call *Gender::IsMale* by passing different *Gender sub-types* as arguements and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Call *Gender::IsMale* method with argument as given in the *test data*.

	(2.) Store the return value in a *boolean* variable.
Test Data	*parameter:* *Gender::Male::Type(),* *Gender::Female::Type()*
Expected Result / Golden Output	Value of the *boolean* variable will be *true* and *false* in first and second *test data* respectively.
Date of Creation	03 April 2021

H.2. Test Scenarios for ***Non Static Member Functions***

Test Plan ID	H
Test Suite ID	H.2
Test Case ID	H.2.1
Test Case Summary	Call *Gender::GetName* method on the singleton instance of any of the *Gender sub-types* and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Call *Gender::GetName* method on a *Gender static sub-type instance* as given in the *test data*. (2.) Store the return value in a *string* variable.
Test Data	*subTypeInstance:* *Gender::Male::Type(),* *Gender::Female::Type()*
Expected Result / Golden Output	Value of the *string* variable will be *"Male"* and *"Female"* in first and second *test data* respectively.
Date of Creation	03 April 2021

Test Plan ID	H

Test Suite ID	H.2
Test Case ID	H.2.2
Test Case Summary	Call *GenderTypes<T>::GetTitle* method on the singleton instance of any of the *Gender sub-types* and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Call *GenderTypes<T>::GetTitle* method on a *Gender static sub-type instance* as given in the *test data*. (2.) Store the return value in a *string* variable.
Test Data	*subTypeInstance: Gender::Male::Type(), Gender::Female::Type()*
Expected Result / Golden Output	Value of the *string* variable will be *"Mr."* and *"Ms."* for first and second *test data* respectively.
Date of Creation	03 April 2021

H.3. Test Scenarios for **Overloaded Output Streaming Operator**

Test Plan ID	H
Test Suite ID	H.3
Test Case ID	H.3.1
Test Case Summary	Call *GenderTypes<T>::Type* method for any *Gender sub-type* and print the returned instance onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Call *GenderTypes<T>::Type* method for a *Gender sub-type*, as given in *test data.* (2.) Print the returned instance onto the console using the *cout* output stream

	object and *output streaming operator* <<.
Test Data	*subTypeInstance:* *Gender::Male::Type(),* *Gender::Female::Type()*
Expected Result / Golden Output	Title and name of the gender will be printed onto the console. *"Male / Mr."* and *"Female / Ms."* will be printed for first and second *test data* respectively.
Date of Creation	03 April 2021

*H.4. Test Scenarios to test **Dynamic Dispatch of Polymorphic Methods***

Test Plan ID	H
Test Suite ID	H.4
Test Case ID	H.4.1
Test Case Summary	Call *GenderTypes<T>::GetTitle* method on the singleton instance of any of the *Gender sub-types* upcasted to a *const Gender reference* and check the returned value
Prerequisite System's State	NIL
Procedure	(1.) Call *GenderTypes<T>::GetTitle* method on a *Gender static sub-type instance* (as given in the *test data*) that is stored in a *const Gender reference* variable. (2.) Check the returned *string*.
Test Data	*subTypeInstance:* *Gender::Male::Type(),* *Gender::Female::Type()*
Expected Result / Golden Output	Value of the *string* variable will be *"Mr."* and *"Ms."* for first and second *test data* respectively.
Date of Creation	03 April 2021

*I. Unit Test Plan for **BookingCategory Hierarchy***

*I.1. Test Scenarios for **Static Member Functions***

Test Plan ID	I
Test Suite ID	I.1
Test Case ID	I.1.1
Test Case Summary	Call *BookingCategoryTypes<T>::Type* method twice for any *BookingCategory sub-type* and check if the same object is returned -- *test for singleton class*
Prerequisite System's State	NIL
Procedure	(1.) Call *BookingCategory::General::Type* method and store the returned instance in a *const BookingCategory reference*. (2.) Call *BookingCategory::General::Type* method again and store the returned instance in another *const BookingCategory reference*. (3.) Compare the *addresses* of the two *BookingCategory references* using *'=='* and store the result in a *boolean* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *boolean* variable will be *true*
Date of Creation	03 April 2021

*I.2. Test Scenarios for **Non Static Member Functions***

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.1
Test Case Summary	Call *BookingCategoryTypes<T>::GetName* method for any *BookingCategory sub-type* and check the returned value

Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingCategory::General::Type()"* to get the singleton instance of *General* sub-type. (2.) Call *BookingCategoryTypes<T>::GetName* method on this instance and store the returned value in a *string* variable.
Test Data	NIL
Expected Result / Golden Output	Value of the *string* variable will be *"General"*
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.2
Test Case Summary	Call *BookingCategory::General::IsElligible* method on the *singleton instance* of *General* sub-type with appropriate arguments and check the returned value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object to *BookingCategory::General::IsElligible* method called on the *singleton instance* of *General* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/2020"), "Male", "123456789012", "John"), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *true*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.3
Test Case Summary	Call *BookingCategory::Ladies::IsElligible* method on the *singleton instance* of *Ladies* sub-type with appropriate arguments to check for a *false* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Ladies::IsElligible* method called on the *singleton instance* of *Ladies* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/2008"), "Male", "123456789012", "John"), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *false*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.4
Test Case Summary	Call *BookingCategory::Ladies::IsElligible* method on the *singleton instance* of *Ladies* sub-type with appropriate arguments to check for a *true* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid*

	set of arguments and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Ladies::IsElligible* method called on the *singleton instance* of *Ladies* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/2020"), "Female", "123456789012", "Jane"), Date::CreateDate("01/01/2022"))* *(Passenger::CreatePassenger(Date::CreateDate("30/04/2009"), "Male", "123456789012", "John"), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *true* for both the *test data*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.5
Test Case Summary	Call *BookingCategory::Divyaang::IsElligible* method on the *singleton instance* of *Divyaang* sub-type with appropriate arguments to check for a *false* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Divyaang::IsElligible* method called on the *singleton instance* of *Divyaang* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::Creat*

	eDate("30/06/2020"), "Female", "123456789012", "Jane"), Date::CreateDate("01/01/2022"))
Expected Result / Golden Output	Returned value will be *false*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.6
Test Case Summary	Call *BookingCategory::Divyaang::IsElligible* method on the *singleton instance* of *Divyaang* sub-type with appropriate arguments to check for a *true* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Divyaang::IsElligible* method called on the *singleton instance* of *Divyaang* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/2020"), "Female", "123456789012", "Jane", "", "", "", &Divyaang::Blind::Type()), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *true*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.7

Test Case Summary	Call *BookingCategory::SeniorCitizen::IsElligible* method on the *singleton instance* of *SeniorCitizen* sub-type with appropriate arguments to check for a *false* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::SeniorCitizen::IsElligible* method called on the *singleton instance* of *SeniorCitizen* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/1964"), "Female", "123456789012", "Jane"), Date::CreateDate("01/01/2022"))* *(Passenger::CreatePassenger(Date::CreateDate("30/06/1962"), "Male", "123456789012", "John"), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *false* for both the *test data*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.8
Test Case Summary	Call *BookingCategory::SeniorCitizen::IsElligible* method on the *singleton instance* of *SeniorCitizen* sub-type with appropriate arguments to check for a *true* return value
Prerequisite System's State	NIL

Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::SeniorCitizen::IsElligible* method called on the *singleton instance* of *SeniorCitizen* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel):* *(Passenger::CreatePassenger(Date::CreateDate("30/01/1963"), "Female", "123456789012", "Jane"), Date::CreateDate("01/01/2022"))* *(Passenger::CreatePassenger(Date::CreateDate("30/01/1961"), "Male", "123456789012", "John"), Date::CreateDate("01/01/2022"))*
Expected Result / Golden Output	Returned value will be *true* for both the *test data*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.9
Test Case Summary	Call *BookingCategory::Tatkal::IsElligible* method on the *singleton instance* of *Tatkal* sub-type with appropriate arguments to check for a *false* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Tatkal::IsElligible* method called on the *singleton instance* of *Tatkal* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel):*

	(Passenger::CreatePassenger(Date::CreateDate("30/06/1964"), "Female", "123456789012", "Jane"), Date::CreateDate("01/12/2021"))
Expected Result / Golden Output	Returned value will be *false*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.10
Test Case Summary	Call *BookingCategory::Tatkal::IsElligible* method on the *singleton instance* of *Tatkal* sub-type with appropriate arguments to check for a *true* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::Tatkal::IsElligible* method called on the *singleton instance* of *Tatkal* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/1964"), "Female", "123456789012", "Jane"), Date::CreateDate("04/04/2021"))* *Date* parameter must represent the date next to the *date of execution* of this test case. So change accordingly, otherwise the test will give a false *FAIL*.
Expected Result / Golden Output	Returned value will be *true*.
Date of Creation	03 April 2021

Test Plan ID	I

Test Suite ID	I.2
Test Case ID	I.2.11
Test Case Summary	Call *BookingCategory::PremiumTatkal::IsElligible* method on the *singleton instance* of *PremiumTatkal* sub-type with appropriate arguments to check for a *false* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to *BookingCategory::PremiumTatkal::IsElligible* method called on the *singleton instance* of *PremiumTatkal* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/1964"), "Female", "123456789012", "Jane"), Date::CreateDate("01/12/2021"))*
Expected Result / Golden Output	Returned value will be *false*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.12
Test Case Summary	Call *BookingCategory::PremiumTatkal::IsElligible* method on the *singleton instance* of *PremiumTatkal* sub-type with appropriate arguments to check for a *true* return value
Prerequisite System's State	NIL
Procedure	Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object (as given in *test data*) to

	BookingCategory::PremiumTatkal::IsElligible method called on the *singleton instance* of *PremiumTatkal* sub-type; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate("30/06/1964"), "Female", "123456789012", "Jane"), Date::CreateDate("04/04/2021"))* *Date* parameter must represent the date next to the *date of execution* of this test case. So change accordingly.
Expected Result / Golden Output	Returned value will be *true*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.13
Test Case Summary	Call *BookingCategory::Ladies::SelectBooking* method on the *singleton instance* of *Ladies* sub-type with *erroneous terminal stations*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *"BookingCategory::Ladies::Type()"* to get the singleton instance of *Ladies* sub-type. (2.) Call *BookingCategory::Ladies::SelectBooking* method on this instance with the parameters as given in the *test data*. (3.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Pune"), Date::CreateDate("01/12/2021"),*

	BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2020"), "Male", "123456789012", "John"), Date::GetTodaysDate())
Expected Result / Golden Output	A *Bad_Booking_UndefinedTerminals exception* will be caught.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.14
Test Case Summary	Call *BookingCategory::Ladies::SelectBooking* method on the *singleton instance* of *Ladies* sub-type with *erroneous date of booking*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *"BookingCategory::Ladies::Type()"* to get the singleton instance of *Ladies* sub-type. (2.) Call *BookingCategory::Ladies::SelectBooking* method on this instance with the parameters as given in the *test data*. (3.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/01/2021"), BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2020"), "Male", "123456789012", "John"), Date::GetTodaysDate())* *(Station::CreateStation("Mumbai"),*

	Station::CreateStation("Delhi"), Date::GetTodaysDate(), BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2020"), "Male", "123456789012", "John"), Date::GetTodaysDate()) *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2022"), BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2020"), "Male", "123456789012", "John"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* will be caught for all the *test data*.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.15
Test Case Summary	Call *BookingCategory::Ladies::SelectBooking* method on the *singleton instance* of *Ladies* sub-type with *invalid BookingClass sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.* An *invalid BookingClass sub-type* should be defined. This must be different from the *8 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *TestTypeBC*. Initialize all the *static const data members* of *BookingClassTypes<TestTypeBC>* with arbitrary values (of appropriate data types)
Procedure	(1.) Call *"BookingCategory::Ladies::Type()"*

	to get the singleton instance of *Ladies* sub-type. (2.) Call *BookingCategory::Ladies::SelectBooking* method on this instance with the parameters as given in the *test data*. (3.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClassTypes<TestTypeBC>::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2020"), "Male", "123456789012", "John"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_BookingClass exception* will be caught.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.16
Test Case Summary	Call *BookingCategory::Ladies::SelectBooking* method on the *singleton instance* of *Ladies* sub-type with *Passenger ineligible for Ladies*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *"BookingCategory::Ladies::Type()"* to get the singleton instance of *Ladies* sub-type. (2.) Call *BookingCategory::Ladies::SelectBooking* method on this instance with the parameters as given in the *test data*.

	(3.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2001"), "Male", "123456789012", "John"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_Passenger exception* will be caught.
Date of Creation	03 April 2021

Test Plan ID	I
Test Suite ID	I.2
Test Case ID	I.2.17
Test Case Summary	Call *BookingCategoryTypes<T>::SelectBooking* method on the *singleton instance* of all *BookingCategory* sub-types with *valid arguments* -- to check if the *sub-routine* of this method for all *BookingCategory* sub-types calls *BookingTypes<T>::CreateSpecialBooking* method of the appropriate *Booking sub-type* only.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *BookingCategoryTypes<T>::Type* method to get the singleton instance of a *BookingCategory* sub-type, as given in *test data*. (2.) Call *BookingCategoryTypes<T>::SelectBooking* method on this instance with the parameters as given in the *test data*.

	(3.) Call method *BookingTypes<T>::GetType* on the returned object and check the returned value.
Test Data	*bookingCategorySubTypes:* *BookingCategory::General::Type(),* *BookingCategory::Ladies::Type(),* *BookingCategory::SeniorCitizen::Type(),* *BookingCategory::Divyaang::Type(),* *BookingCategory::Tatkal::Type(),* *BookingCategory::PremiumTatkal::Type()* *parameters:* *(Station::CreateStation("Mumbai"),* *Station::CreateStation("Delhi"),* *Date::CreateDate("05/04/2021"),* *BookingClass::ACFirstClass::Type(),* *Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane", "", "", "", &Divyaang::Blind::Type()),* *Date::GetTodaysDate())* These parameters are valid for *BookingCategoryTypes<T>::SelectBooking* method call on any *BookingCategory sub-type*. The third parameter however should be chosen next date to the *Date of Execution* of this *test case*.
Expected Result / Golden Output	(1.) No exception will be thrown for any *test data*. (2.) The returned *string* value be *"General", "Ladies", "Senior Citizen", "Divyaang", "Tatkal", "Premium Tatkal"* for *first, second, third, fourth, fifth* and *sixth test data* respectively.
Date of Creation	04 April 2021

Note: Correct initialization of data members will be checked in Unit Test Plan for Booking Hierarchy

I.3. Test Scenarios for ***Overloaded Output Streaming Operator***

Test Plan ID	I
Test Suite ID	I.3

Test Case ID	I.3.1
Test Case Summary	Call *BookingCategoryTypes<T>::Type* method for any *BookingCategory sub-type* and print the returned instance onto the console using *cout* output stream object.
Prerequisite System's State	NIL
Procedure	(1.) Call *BookingCategoryTypes::General::Type* method to get the singleton instance of *General* sub-type*.* (2.) Print the returned instance onto the console using the *cout* output stream object and *output streaming operator* <<.
Test Data	NIL
Expected Result / Golden Output	Name of the *sub-type*, that is *"General"* will be printed onto the console.
Date of Creation	03 April 2021

I.4. Test Scenarios to test ***Dynamic Dispatch of Polymorphic Methods***

Test Plan ID	I
Test Suite ID	I.4
Test Case ID	I.4.1
Test Case Summary	Call *BookingCategoryTypes<T>::GetName* method on the singleton instance of any *BookingCategory* sub-type upcasted to a *const BookingCategory reference*.
Prerequisite System's State	NIL
Procedure	(1.) Call *"BookingCategory::General::Type()"* and store the returned instance in a *const BookingCategory reference* variable. (2.) Call *BookingCategoryTypes<T>::GetName* method on this variable and check the

	returned value.
Test Data	NIL
Expected Result / Golden Output	The value of the returned *string* will be *“General”*
Date of Creation	04 April 2021

Test Plan ID	I
Test Suite ID	I.4
Test Case ID	I.4.2
Test Case Summary	Call *BookingCategory::General::IsElligible* method on the *singleton instance* of *General* sub-type, upcasted to a *const BookingCategory reference*, with appropriate arguments and check the returned value
Prerequisite System’s State	NIL
Procedure	(1.) Call *“BookingCategory::General::Type()”* and store the returned instance in a *const BookingCategory reference* variable. (2.) Construct a *Passenger* object with a *valid set of arguments* and pass that along with a *Date* object to *BookingCategory::General::IsElligible* method called on this variable; and check the returned value
Test Data	*(passenger, dateOfTravel): (Passenger::CreatePassenger(Date::CreateDate(“30/06/2020”), “Male”, “123456789012”, “John”), Date::CreateDate(“01/01/2022”))*
Expected Result / Golden Output	Returned value will be *true*.
Date of Creation	04 April 2021

Test Plan ID	I
Test Suite ID	I.4
Test Case ID	I.4.3
Test Case Summary	Call *BookingCategoryTypes<T>::SelectBooking* method on the *singleton instance* of *BookingCategory* sub-type, upcasted to a *const BookingCategory reference*, with appropriate arguments and check the type of the returned instance
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *"BookingCategory::General::Type()"* and store the returned instance in a *const BookingCategory reference* variable. (2.) Call *BookingCategoryTypes<T>::SelectBooking* method on this variable with the arguments as given in the *test data* and store the returned instance in *const Booking** variable. (3.) Call *BookingTypes<T>::GetType* method on *const Booking** variable and check the returned value.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())*
Expected Result / Golden Output	Value of the returned *string* will be *"General"*
Date of Creation	04 April 2021

J. Unit Test Plan for **Booking Hierarchy**

J.1. Test Scenarios for **Construction of Objects**

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.1
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for *undefined terminal stations*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Pune"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))*
Expected Result / Golden Output	A *Bad_Booking_UndefinedTerminals exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.2
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for *date of booking*

	in past.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/01/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::GetTodaysDate(), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))*
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.3
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for *date of booking after 365 days from the present day*.
Prerequisite System's State	*The singleton instance of Railways is*

	constructed from the default parameters.
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("04/05/2022"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))*
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.4
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for an *invalid BookingCategory sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.* An *invalid BookingCategory sub-type* should be defined. This must be different from the *6 valid BookingCategory sub-types*. To achieve this define a *struct placeholder* with name *BookCatTestType*. Initialize all the *static const data members* of *BookingCategoryTypes<BookCatTestType>* with arbitrary values (of appropriate data types). Write trivial function definitions for

	BookingCategoryTypes<BookCatTestType>::IsElligible and *BookingCategoryTypes<BookCatTestType>::SelectBooking* that simply return *true* and *NULL* respectively.
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategoryTypes<BookCatTestType>::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))*
Expected Result / Golden Output	A *Bad_Booking_BookingCategory exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.5
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for an *invalid BookingClass sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.* An *invalid BookingClass sub-type* should be defined. This must be different from the *6 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *BookClassTestType*. Initialize all the *static const data members* of

	BookingClassTypes<BookClassTestType> with arbitrary values (of appropriate data types).
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClassTypes<BookClassTestType>::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"))*
Expected Result / Golden Output	A *Bad_Booking_BookingClass exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.6
Test Case Summary	Use *Booking::CreateBooking* to construct a *Booking sub-type* object for an *ineligible Passenger*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"),*

	Station::CreateStation(“Delhi”), Date::CreateDate(“01/12/2021”), BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::Create Date(“01/01/1956”), “Male”, “123456789012”, “John”))
Expected Result / Golden Output	A *Bad_Booking_Passenger exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.1
Test Case ID	J.1.7
Test Case Summary	Call *Booking::CreateBooking* with *valid arguments* for all *BookingCategory sub-types* -- to check if the *sub-routine* of this method for all *BookingCategory* sub-types constructs an instance of the corresponding *Booking sub-type* only.
Prerequisite System’s State	*The singleton instance of Railways is constructed from the default parameters.* No *Booking sub-type* was instantiated before the execution of this *test case*.
Procedure	(1.) Call “*Booking::CreateBooking(Station::CreateSt ation(“Mumbai”), Station::CreateStation(“Delhi”), Date::CreateDate(“05/04/2021”), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::Create Date(“01/01/1956”), “Female”, “123456789012”, “Jane”, “”, “”, “”, &Divyaang::Blind::Type()))*” The third parameter however should be chosen next date to the *Date of Execution* of this *test case* for *Tatkal* and *PremiumTatkal* BookingCategory sub-types. For others, any other valid *Date* in future within 1 year from the *date of execution* would work.

	Call the method *multiple times*, each time changing the *BookingCategory* parameter; cover all the *BookingCategory sub-types* given in the *test data*. (2.) Check all the *non-static data members* for the returned instance, except *Booking::fare_*. (3.) Use *BookingTypes<T>::GetType* method on it to ensure that the object is an instance of the *Booking sub-type* corresponding to the passed *BookingCategory sub-type*.
Test Data	*bookingCategorySubTypes:* *BookingCategory::General::Type(),* *BookingCategory::Ladies::Type(),* *BookingCategory::SeniorCitizen::Type(),* *BookingCategory::Divyaang::Type(),* *BookingCategory::Tatkal::Type(),* *BookingCategory::PremiumTatkal::Type()*
Expected Result / Golden Output	(1.) No exception will be thrown for any *test data*. (2.) The *non-static data members* for all the *test data* satisfy the following relations. *(LHS = RHS)*

*fromStation_*	*Station::CreateStation("Mumbai")*
*toStation_*	*Station::CreateStation("Delhi")*
*dateOfBooking_*	*Date::CreateDate("05/04/2021")*
*&bookingClass_*	*&BookingClass::ACFirstClass::Type()*
attributes of *passenger_*	attributes of *Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane", "", "", "", &Divyaang::Blind::Type())*
*dateOfReservation_*	*Date::GetTodaysDate()*

Value of *dateOfBooking* depends on the *Date of Execution* of this *test case*.

(3.) Value of *"&bookingCategory_"* will be *"&BookingCategory::General::Type()", "&BookingCategory::Ladies::Type()", "&BookingCategory::SeniorCitizen::Type()", "&BookingCategory::Divyaang::Type()", "&BookingCategory::Tatkal::Type()", "&BookingCategory::PremiumTatkal::Type()"* for *first, second, third, fourth, fifth* and *sixth*

	test data respectively. (4.) The value of *pnr_* will be *1, 2, 3, 4, 5, 6* for *first, second, third, fourth, fifth* and *sixth test data* respectively. (5.) The *string* returned by *BookingTypes<T>::GetType* method will be *"General", "Ladies", "SeniorCitizen", "Divyaang", "Tatkal", "PremiumTatkal"* for *first, second, third, fourth, fifth* and *sixth test data* respectively.
Date of Creation	04 April 2021

J.2. Test Scenarios for other ***Static Member Functions***

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.1
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for *undefined terminal stations*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"),* *Station::CreateStation("Pune"),* *Date::CreateDate("01/12/2021"),* *BookingClass::ACFirstClass::Type(),* *BookingCategory::General::Type(),*

	Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())
Expected Result / Golden Output	A *Bad_Booking_UndefinedTerminals exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.2
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for *date of booking in past*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/01/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::GetTodaysDate(), BookingClass::ACFirstClass::Type(),*

	BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate(“01/01/1956”), “Female”, “123456789012”, “Jane”), Date::GetTodaysDate())
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.3
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for *date of booking after 365 days from the present day*.
Prerequisite System’s State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation(“Mumbai”), Station::CreateStation(“Delhi”), Date::CreateDate(“04/05/2022”), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate(“01/01/1956”), “Female”, “123456789012”, “Jane”), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* will be caught.

Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.4
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for an *invalid BookingCategory sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.* An *invalid BookingCategory sub-type* should be defined. This must be different from the *6 valid BookingCategory sub-types*. To achieve this define a *struct placeholder* with name *BookCatTestType*. Initialize all the *static const data members* of *BookingCategoryTypes<BookCatTestType>* with arbitrary values (of appropriate data types). Write trivial function definitions for *BookingCategoryTypes<BookCatTestType>::IsElligible* and *BookingCategoryTypes<BookCatTestType>::SelectBooking* that simply return *true* and *NULL* respectively.
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategoryTypes<BookCatTestType*

	>::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())
Expected Result / Golden Output	A *Bad_Booking_BookingCategory exception* will be caught.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.5
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for a *valid but incompatible BookingCategory sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategoryTypes::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_BookingCategory exception* will be caught.

Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.6
Test Case Summary	Use *Booking::GeneralBooking::CreateSpecialBooking* to construct a *GeneralBooking sub-type* object for an *invalid BookingClass sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.* An *invalid BookingClass sub-type* should be defined. This must be different from the *6 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *BookClassTestType*. Initialize all the *static const data members* of *BookingClassTypes<BookClassTestType>* with arbitrary values (of appropriate data types).
Procedure	(1.) Call *Booking::GeneralBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClassTypes<BookClassTestType>::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_BookingClass exception* will be caught.

Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.2
Test Case ID	J.2.7
Test Case Summary	Use *Booking::LadiesBooking::CreateSpecialBooking* to construct a *LadiesBooking sub-type* object for an *ineligible Passenger*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::LadiesBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Surround the function call with *try-catch block*.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Male", "123456789012", "John"), Date::GetTodaysDate())*
Expected Result / Golden Output	A *Bad_Booking_Passenger exception* will be caught.
Date of Creation	04 April 2021

J.3. Test Scenarios for ***Non Static Member Functions***

Test Plan ID	J
Test Suite ID	J.3

Test Case ID	J.3.1
Test Case Summary	Test *BookingTypes<T>::GetType* method on an instance of any *Booking sub-type*
Prerequisite System's State	The singleton instance of Railways is constructed from the default parameters.
Procedure	(1.) Call *Booking::LadiesBooking::CreateSpecialBooking* method with the parameters as given in the *test data*. (2.) Call *BookingTypes<T>::GetType* method on the constructed object and check the return value.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())*
Expected Result / Golden Output	The value of the returned *string* will be *"Ladies"*.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.2
Test Case Summary	Test *Booking::GeneralBooking::ComputeFare* method on instances of *GeneralBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*

Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::AC3Tier::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* *(Station::CreateStation("Chennai"), Station::CreateStation("Kolkata"), Date::CreateDate("01/12/2021"), BookingClass::FirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))*
Expected Result / Golden Output	The value of *Booking::fare_* will be *4763, 1849, 2539* for *first, second* and *third test data* respectively.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.3

Test Case Summary	Test *Booking::LadiesBooking::ComputeFare* method on instances of *LadiesBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Female", "123456789012", "Jane"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::FirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2010"), "Male", "123456789012", "Jane"))*
Expected Result / Golden Output	The value of *Booking::fare_* will be *4763, 2221* for *first* and *second test data* respectively.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.4
Test Case Summary	Test

	Booking::SeniorCitizenBooking::ComputeFare method on instances of *SeniorCitizenBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::SeniorCitizen::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1961"), "Female", "123456789012", "Jane"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::SeniorCitizen::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1959"), "Male", "123456789012", "John"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::AC3Tier::Type(), BookingCategory::SeniorCitizen::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1959"), "Male", "123456789012", "John"))*
Expected Result / Golden Output	The value of *Booking::fare_* will be *2411, 2882, 1125* for *first, second* and *third test data* respectively.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.5
Test Case Summary	Test *Booking::DivyaangBooking::ComputeFare* method on instances of *DivyaangBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Divyaang::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Divyaang::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John", "", "", "", &Divyaang::TBPatients::Type()))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::AC2Tier::Type(),*

	BookingCategory::Divyaang::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type()))
Expected Result / Golden Output	The value of *Booking::fare_* will be *2411, 4763, 1497* for *first, second* and *third test data* respectively.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.6
Test Case Summary	Test *Booking::TatkalBooking::ComputeFare* method on instances of *TatkalBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("05/04/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Tatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* *(Station::CreateStation("Chennai"), Station::CreateStation("Bangalore"), Date::CreateDate("05/04/2021"), BookingClass::ACChairCar::Type(), BookingCategory::Tatkal::Type(),*

	Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John")) *(Station::CreateStation("Chennai"), Station::CreateStation("Bangalore"), Date::CreateDate("05/04/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Tatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* Here the *third parameter* depends on the *date of execution* of this test case. Choose *date next to the date of execution*.
Expected Result / Golden Output	The value of *Booking::fare_* will be *5263, 515, 1198* for *first, second* and *third test data* respectively.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.3
Test Case ID	J.3.7
Test Case Summary	Test *Booking::PremiumTatkalBooking::ComputeFare* method on instances of *PremiumTatkalBooking sub-type*.
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data*. (2.) Check *Booking::fare_* data member for the returned instance.
Test Data	*parameters:* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("05/04/2021"),*

	BookingClass::ACFirstClass::Type(), BookingCategory::PremiumTatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John")) *(Station::CreateStation("Chennai"), Station::CreateStation("Bangalore"), Date::CreateDate("05/04/2021"), BookingClass::ACChairCar::Type(), BookingCategory::PremiumTatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* *(Station::CreateStation("Chennai"), Station::CreateStation("Bangalore"), Date::CreateDate("05/04/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::PremiumTatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))* Here the *third parameter* depends on the *date of execution* of this test case. Choose *date next to the date of execution*.
Expected Result / Golden Output	The value of *Booking::fare_* will be *5763, 640, 1198* for *first, second* and *third test data* respectively.
Date of Creation	04 April 2021

*J.4. Test Scenarios for **Overloaded Output Streaming Operator***

Test Plan ID	J
Test Suite ID	J.4
Test Case ID	J.4.1
Test Case Summary	Print a *Booking sub-type* object onto the console using *cout* output stream object.
Prerequisite System's State	*The singleton instance of Railways is*

	constructed from the default parameters.
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters given in the *test data.* (2.) Print the returned object onto the console using *cout* and *output streaming operator* <<.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Male", "123456789012", "John", "Jack", "Doe", "9874563210", &Divyaang::CancerPatients::Type(), "ABC987"))*
Expected Result / Golden Output	All details/attributes of the booking and the passenger will be printed onto the console. BOOKING SUCCESSFUL : PNR Number = 1 From Station = Mumbai To Station = Delhi Reservation Date = 04/Apr/2021 Travel Date = 01/Dec/2021 Travel Class = AC First Class : Mode : Sleeping : Comfort : AC : Bunks : 2 : Luxury : Yes Booking Category = General Fare = 4763 Name = John Jack Doe Adhaar Card No. = 123456789012 Date of Birth = 01/Jan/1956 Gender = Male / Mr. Mobile No. = 9874563210 Disability Type = Cancer Patients Disability ID = ABC987 The value of *PNR number* depends on how many times a *Booking sub-type* was instantiated before executing this *test case*.

	The value of *Reservation Date* depends on the *date of execution* of this test case.
Date of Creation	04 April 2021

J.5. Test Scenarios to test **Dynamic Dispatch of Polymorphic Methods**

Test Plan ID	J
Test Suite ID	J.5
Test Case ID	J.5.1
Test Case Summary	Test *BookingTypes<T>::GetType* method on an instance of any *Booking sub-type* upcasted to *const Booking**
Prerequisite System's State	The singleton instance of Railways is constructed from the default parameters.
Procedure	(1.) Call *Booking::LadiesBooking::CreateSpecialBooking* method with the parameters as given in the *test data* and store the returned value in a *const Booking** variable. (2.) Call *BookingTypes<T>::GetType* method on it and check the return value.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1956"), "Female", "123456789012", "Jane"), Date::GetTodaysDate())*
Expected Result / Golden Output	The value of the returned *string* will be *"Ladies"*.
Date of Creation	04 April 2021

Test Plan ID	J
Test Suite ID	J.5
Test Case ID	J.5.2
Test Case Summary	Test *BookingTypes<T>::ComputeFare* method on an instance of any *Booking sub-type* upcasted to *const Booking**
Prerequisite System's State	*The singleton instance of Railways is constructed from the default parameters.*
Procedure	(1.) Call *Booking::CreateBooking* method with the parameters as given in the *test data* and store the returned value in a *const Booking** variable. (2.) Call *BookingTypes<T>::ComputeFare* method on it and check the return value.
Test Data	*parameters: (Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::General::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John"))*
Expected Result / Golden Output	The returned value will be *4763*.
Date of Creation	04 April 2021

K. ***Application Test Plan***

K.1. *Test Scenarios for* ***Variable BookingClass Sub-Types***

Test Plan ID	K
Test Suite ID	K.1
Test Case ID	K.1.1
Test Case Summary	Exhaustively check for *Booking*s with every *BookingClass* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with varying *BookingClass* sub-types, with arguments adhering to the *test data*. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.

Test Data	*BookingClass sub-types:*
	BookingClass::ACFirstClass::Type()
	BookingClass::ExecutiveChairCar::Type()
	BookingClass::AC2Tier::Type()
	BookingClass::FirstClassType::Type()
	BookingClass::AC3Tier::Type()
	BookingClass::ACChairCar::Type()
	BookingClass::Sleeper::Type()
	BookingClass::SecondSitting::Type()
	
	Terminal Stations: Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*.
	
	BookingCategory sub-type: Keep it constant at *BookingCategory::General::Type()*
	
	Passenger instance: *Passenger information* must adhere to the *constraints*.
	
	Date of Booking/Travel: Choose any arbitrary *Date* in the future, within next *one*

	year. Total Sub Levels -- 8
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

K.2. Test Scenarios for **Variable Terminal Stations**

Test Plan ID	K
Test Suite ID	K.2
Test Case ID	K.2.1
Test Case Summary	Exhaustively check for *Booking*s with every *ordered pair of terminal stations*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects, exhaustively covering all the *ordered pairs* of distinct terminal *Station*s, with arguments adhering to the *test data*. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly. (3.) Check the *symmetric ordering* of *Station*s by the virtue of which two bookings with same *BookingClass sub-type, BookingCategory sub-type*, *Passenger* and *unordered pair* of *terminal Station*s but different *ordered pair* of *terminal Station*s must have identical *booking fares*.
Test Data	*Terminal Stations:* There are 5 *Station*s in the *default Indian Railways* -- *Mumbai, Delhi, Bangalore, Kolkata, Chennai*. Exhaustively enumerate each ordered pair of distinct terminal *Station*s.

	(Mumbai, Delhi) *(Mumbai, Bangalore)* **….** *(Delhi, Mumbai)* *(Delhi, Bangalore)* **….** **….** *(Chennai, Bangalore)* *(Chennai, Kolkata)* *BookingClass sub-types:* Choose arbitrarily any of the *8 BookingClass sub-types*. But, it must be same for the two *ordered pairs* corresponding to every *unordered pair*. That is, for the *sub-levels (Mumbai, Delhi)* and *(Delhi, Mumbai) BookingClass sub-type* must be the same. *BookingCategory sub-type:* Keep it constant at *BookingCategory::General::Type()* *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year.* *Total Sub Levels -- 20*
Expected Result / Golden Output	Each *sub-level* will execute without any hindrance. *Symmetric ordering* of *Station*s is consistent with the *booking fare amount*.
Date of Creation	05 April 2021

K.3. Test Scenarios for ***Variable BookingCategory Sub-Types***

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.1

Test Case Summary	Check for different scenarios of *Booking*s with *General BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* object(s) with arguments adhering to the *test data*. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John")* *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*.</u> *Total Sub Levels -- 1*
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.2

Test Case Summary	Check for different scenarios of *Booking*s with *Ladies BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. Exhaustively cover both the *Gender* sub-types. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::Ladies::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::CreateDate("01/01/2011"), "Male", "123456789012", "John")* *Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Female", "123456789012", "Jane")* *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year.* *Total Sub Levels -- 2*
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.3
Test Case Summary	Check for different scenarios of *Booking*s with *SeniorCitizen BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. Exhaustively cover both the *Gender* sub-types*.* (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::SeniorCitizen::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Passenger::CreatePassenger(Date::CreateDate("01/01/1960"), "Male", "123456789012", "John")* *Passenger::CreatePassenger(Date::CreateDate("01/01/1961"), "Female", "123456789012", "Jane")* *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*</u>. *Total Sub Levels -- 2*

Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.4
Test Case Summary	Check for different scenarios of *Booking*s with *Divyaang BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. Exhaustively cover all the *Divyaang* sub-types. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::Divyaang::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "", &Divyaang::Blind::Type())* *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "",*

	&Divyaang::OrthopaedicallyHandicapped::Type()) *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "", &Divyaang::TBPatients::Type())* *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "", &Divyaang::CancerPatients::Type())* *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year*. *Total Sub Levels -- 4*
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.5
Test Case Summary	Check for different scenarios of *Booking*s with *Tatkal BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*.

	BookingCategory sub-type: BookingCategory::Tatkal::Type() *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::Create Date("01/01/2019"), "Male", "123456789012", "John")* *Date of Booking/Travel*: Specifically choose the *Date* next to the *date of execution* of this test case. *Total Sub Levels -- 1*
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.3
Test Case ID	K.3.6
Test Case Summary	Check for different scenarios of *Booking*s with *PremiumTatkal BookingCategory* sub-type
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Print all the constructed *Booking* objects onto the console and ensure all bookings are executing correctly.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*.

	BookingCategory sub-type: BookingCategory::PremiumTatkal::Type() *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::Create Date("01/01/2019"), "Male", "123456789012", "John")* *Date of Booking/Travel*: Specifically choose the *Date* next to the *date of execution* of this test case. *Total Sub Levels -- 1*
Expected Result / Golden Output	Each *sub-level* should execute without any hindrance.
Date of Creation	05 April 2021

*K.4. Test Scenarios for **Erroneous Passenger Information***

Test Plan ID	K
Test Suite ID	K.4
Test Case ID	K.4.1
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *name constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*.

	(3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*.</u> *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Passenger::CreatePassenger(Date::Create Date("01/01/2019"), "Male", "123456789012")* *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Passenger_Name exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.4
Test Case ID	K.4.2
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *adhaar number constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method

	to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*.</u> *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Passenger::CreatePassenger(Date::CreateDate(“01/01/2019”), “Male”, “12345678901”, “John”)* *Passenger::CreatePassenger(Date::CreateDate(“01/01/2019”), “Male”, “1234567890123”, “John”)* *Passenger::CreatePassenger(Date::CreateDate(“01/01/2019”), “Male”, “12345678901A”, “John”)* *Total Sub Levels -- 3*
Expected Result / Golden Output	A *Bad_Passenger_AdhaarNumber exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

Test Plan ID	K

Test Suite ID	K.4
Test Case ID	K.4.3
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *mobile number constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "987456321")* *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John", "", "", "98745632100")* *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male",*

	"123456789012", "John", "", "", "987456321A") *Total Sub Levels -- 3*
Expected Result / Golden Output	A *Bad_Passenger_MobileNumber exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.4
Test Case ID	K.4.4
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *date of birth constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year*.

	Passenger instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::Create Date("01/01/2023"), "Male", "123456789012", "John")* *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Passenger_DateOfBirth exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.4
Test Case ID	K.4.5
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *valid disability type constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.* An *invalid Divyaang sub-type* should be defined in the *application*. This must be different from the *4 valid Divyaang sub-types*. To achieve this define a *struct placeholder* with name *AppTestDiv*. Initialize all the *static const data members* of *DivyaangTypes<AppTestDiv>* with arbitrary values (of appropriate data types). Write a trivial function definition for *DivyaangTypes<AppTestDiv>::GetConcessionFactor* that simply returns *0.0*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*.

	(3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::Create Date("01/01/2019"), "Male", "123456789012", "John", "", "", "", &DivyaangTypes<AppTestDiv>::Type())* *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Passenger_DisabilityType exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

Test Plan ID	K
Test Suite ID	K.4
Test Case ID	K.4.6
Test Case Summary	Check for *Booking* request when the *Passenger* does not satisfy the *gender constraint*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*

Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Stations* in each sub-level, between which distance is *well-defined*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year*. *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Others", "123456789012", "John")* *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Passenger_Gender exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	05 April 2021

K.5. Test Scenarios for ***Erroneous Booking Requests***

Test Plan ID	K
Test Suite ID	K.5
Test Case ID	K.5.1

Test Case Summary	Check for *Booking* request when the distance between *terminal Station*s are *undefined*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* (*Mumbai, Pune*), (*Delhi, Delhi*) *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*.</u> *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Total Sub Levels -- 2*
Expected Result / Golden Output	A *Bad_Booking_UndefinedTerminals exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	06 April 2021

Test Plan ID	K
Test Suite ID	K.5
Test Case ID	K.5.2

Test Case Summary	Check for *Booking* request when the *date of booking* is *undefined*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Date of Booking/Travel*: past -- *Date::CreateDate("02/04/2021")* present -- *Date::GetTodaysDate()* future -- *Date::CreateDate("01/09/2022")* *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Total Sub Levels -- 3*
Expected Result / Golden Output	A *Bad_Booking_DateOfBooking exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	06 April 2021

Test Plan ID	K
Test Suite ID	K.5
Test Case ID	K.5.3

Test Case Summary	Check for *Booking* request when the *booking class is invalid*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.* An *invalid BookingClass sub-type* should be defined. This must be different from the *8 valid BookingClass sub-types*. To achieve this define a *struct placeholder* with name *AppTestBookClass*. Initialize all the *static const data members* of *BookingClassTypes<AppTestBookClass>* with arbitrary values (of appropriate data types)
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type: BookingClassTypes<AppTestBookClass>::Type()* *BookingCategory sub-type: BookingCategory::General::Type()* *Terminal Stations:* Choose any two arbitrary *Station*s in each sub-level, between which distance is *well-defined*. *Date of Booking/Travel*: Choose any arbitrary *Date* in the future, within next *one year.* *Passenger* instance: *Passenger information* must adhere to the *constraints*. *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Booking_BookingClass exception* should be caught in the *application* and an appropriate message should be printed

	onto the console.
Date of Creation	06 April 2021

Test Plan ID	K
Test Suite ID	K.5
Test Case ID	K.5.4
Test Case Summary	Check for *Booking* request when the *booking category is invalid*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.* An *invalid BookingCategory sub-type* should be defined. This must be different from the *6 valid BookingCategory sub-types*. To achieve this define a *struct placeholder* with name *AppTestBookCat*. Initialize all the *static const data members* of *BookingCategoryTypes<AppTestBookCat>* with arbitrary values (of appropriate data types). Write trivial function definitions for *BookingCategoryTypes<AppTestBookCat>::IsElligible* and *BookingCategoryTypes<AppTestBookCat>::SelectBooking* that simply return *true* and *NULL* respectively.
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments adhering to the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*BookingClass sub-type:* Choose arbitrarily any of the *8 BookingClass sub-types*. *BookingCategory sub-type: BookingCategoryTypes<AppTestBookCat>::Type()*

	Terminal Stations: Choose any two arbitrary *Station*s in each sub-level, <u>between which distance is *well-defined*</u>. *Date of Booking/Travel*: Choose any arbitrary *Date* <u>in the future, within next *one year*</u>. *Passenger* instance: <u>*Passenger information* must adhere to the *constraints*</u>. *Total Sub Levels -- 1*
Expected Result / Golden Output	A *Bad_Booking_BookingCategory exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	06 April 2021

Test Plan ID	K
Test Suite ID	K.5
Test Case ID	K.5.5
Test Case Summary	Check for *Booking* request when the *passenger is ineligible for the booking category*.
Prerequisite System's State	The *singleton instance* of *Railways* is constructed from the *default parameters.*
Procedure	(1.) Use *Booking::CreateBooking* method to construct *Booking* objects with arguments as given in the *test data*. (2.) Surround the function call with *try catch block*. (3.) Print the *error type/message* onto the console if an *exception* is caught.
Test Data	*Parameters*: *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"),*

	BookingClass::ACFirstClass::Type(), BookingCategory::Ladies::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2000"), "Male", "123456789012", "John")) *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::SeniorCitizen::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1962"), "Male", "123456789012", "John"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::SeniorCitizen::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/1960"), "Female", "123456789012", "Jane"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Divyaang::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(), BookingCategory::Tatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John"))* *(Station::CreateStation("Mumbai"), Station::CreateStation("Delhi"), Date::CreateDate("01/12/2021"), BookingClass::ACFirstClass::Type(),*

	BookingCategory::PremiumTatkal::Type(), Passenger::CreatePassenger(Date::CreateDate("01/01/2019"), "Male", "123456789012", "John")) Following *arguments* can be chosen arbitrarily, adhering to the given norms. - *BookingClass sub-type:* Choose arbitrarily any of the *8 valid BookingClass sub-types*. - *Terminal Stations:* Choose any two arbitrary *Station*s, between which distance is *well-defined*. *Total Sub Levels -- 6*
Expected Result / Golden Output	A *Bad_Booking_Passenger exception* should be caught in the *application* and an appropriate message should be printed onto the console.
Date of Creation	06 April 2021

*K.6. Test Scenarios for **Checking Construction / Destruction***

Test Plan ID	K
Test Suite ID	K.6
Test Case ID	K.6.1
Test Case Summary	Tracking calls to various *constructors* and *destructors*. These tests will be implemented alongside the other test cases in the *Unit Test Plan K*
Prerequisite System's State	Put a *print message* in the *constructors* and *destructors* of all the *classes* (except *abstract base classes*) under *_DEBUG*.
Procedure	Execute all *test cases* in *Unit Test Plan K*.
Test Data	NIL
Expected Result / Golden Output	The *construction* and *destruction* activity

	will be printed onto the console when the project is compiled under *debug build*. - Before the *main* function is entered, the *static data member* of *Railways* class that consists of a *vector* of *Station*s should be initialized and hence *5 Station objects* will be constructed. - All *automatic* (*Date, Station, Passenger, user-defined Exceptions*), *dynamic* (*Booking sub-types*) and *static* (*instances of all the singleton classes*) objects are constructed in the function *ApplicationTestPlan.* - All the *singleton classes* are instantiated at most once *(one or zero calls to the constructor)*. These classes include -- *BookingCategory sub-types, BookingClass sub-types, Gender sub-types, Divyaang sub-types* and *Railways*. So they truly behave as *singletons*. - Call to constructor of *Passenger* is always preceded by call to constructor of *Date*. - Call to constructor of *Booking sub-type* is always preceding by a call to the constructors of *Station, Date* and *Passenger*. - *Booking* objects (*dynamically allocated*) are destructed in the order of construction. - Call to the *destructor* of a *Booking sub-type* is followed by a call to the *destructors* of *Date, Passenger* and *Station*. - Call to *destructor* of a *Passenger* is followed by a call to the *destructor* of *Date*.

	- The *automatic* objects (and *locals*) get *destructed* when the function *ApplicationTestPlan* finishes. - All *static* objects get *destructed* after *main* finishes. *Total Sub Levels -- 10*
Date of Creation	08 April 2021