Name : Abhinandan De
Roll No: 19CS10069

Note :

1. All classes overload the stream operator for easily debugging and display of information.

2. All classes have a static UnitTest() method which supports testing of the class.

3. Some classes eg: Booking which don't have further derived classes have been made polymorphic to extend the design.

4. Encapsulation is increased as much as possible. Attributes have been made private, constructors/destructors are made private for singletons. Copy assignment operators and copy constructors are blocked in those cases.

5. Static members are used for modelling attributes which are common to all instances of a class.

6. Const-ness is introduced wherever possible according to the need.

7. All static members have been named starting with s and follow camelCase.

8. All members of a class follow camelCase and end with the “_” symbol.

9. Refer to class diagrams for better view and to get a good grasp of the HLD.

A. class Railways

This class is designed to be a singleton using ***Meyer's singleton method***.

Private methods/members:

1. sStations : Static data member representing the different stations in the Railway Network

2. sDistStations : Static data member representing the distance between any two stations given in sStations.

3. sValidated : Static data member which confirms whether the input data has been validated or not.

These are declared to be static, although it wouldn't have really mattered since we are creating only one instance.

4. The **constructor**, **copy constructor** and **copy assignment operator** are put in the private section to avoid creation of multiple instances.

Public members/methods:

1. Type(): static method which returns the only instance of the class iff the input data has been validated..

2. **Validate():** static method in which the input is supplied and the validation is done only once when we access the Railways through the Type( ) method. Else it throws an exception of a suitable type.

3. **GetDistance(const Station&, const Station&)** :: returns the distance between two stations given.

Note: The Railways :: Type() returns a const instance of the class as no more stations can be added in the future. It is also assumed that the distance between stations does not change. It returns an instance iff the input data is valid!

B. class Station

This is a class which represents a blueprint of a railway station.

Private members/methods:

1. **name_** : stores the name of the station.
2. **The constructor**.

Note: Here the constructor is placed in the private section in order to ensure the validity of the object created so that the constructor gets called iff all the attributes of the constructor are valid.

Public members/methods:

1. GetName( ): returns the name of the station.

2. GetDistance(const Station&): returns the distance between the current station to the station passed in the function.

3. The **copy constructor** and the **copy assignment operator**.

4. createStation(. . .) : static method which checks for validity of the inputs and then creates an instance of the object iff all the inputs are valid. Else it throws an exception of a suitable type.

Note: The **copy constructor** and the **copy assignment operator** are placed in public as we have already created an instance of the object and we are just trying to copy it.

C. class Date

This stores the date of booking.

Private methods/members:

1. date_ , month_, year_ : 3 integers representing the date, month and year respectively.
2. smonthNames : static vector containing the month Names for the corresponding month number.
3. The constructor.

Note: Here the constructor is placed in the private section in order to ensure the validity of the object created so that the constructor gets called iff all the attributes of the constructor are valid.

Public methods/members:

1. createDate(. . .): a static method which takes in the month, date and year as input and creates an instance iff it results in a valid date. Else it throws an exception of a suitable type.

2. static bool checkValid(int date, int month, int year) : Checks if a date is valid.

3. A **copy assignment operator**, **copy constructor** and a **destructor**.

4. bool operator!= (const Date& d) const: This is a comparator for dates.

5. leapYear(int) : Static method which checks if a year is a leap year (for validity of Feb(29)).

6 computeAge(Date&, Date&) : Static method which computes the age of a passenger by checking the difference between the dates.

7. operator >(Date&, Date&) : static method returns the greater date of the two given.

Note: The **copy constructor** and the **copy assignment operator** are placed in public as we have already created a valid instance of the object and we are just trying to copy it.

D. class BookingClass

For BookingClasses, a **flat level hierarchy** has been followed :
class BookingClass is the **abstract base class.**
The following classes inherit from class BookingClass:

1. **class ACFirstClass**
2. **class ExecutiveChairCar**
3. **class AC2Tier**
4. **class AC3Tier**
5. **class FirstClass**
6. **class ACChairCar**
7. **class Sleeper**
8. **class SecondSitting**

Naturally, these are **singletons** as they are used to model a concept rather than an object. These are modelled using ***Meyer's singleton.***

However since these classes mostly differ in terms of their static attributes, a mixture of **parametric and inclusion polymorphism** has been used to model this hierarchy.

template<typename T> class BookingClasses , has been created and it *inherits* from **BookingClass.**

D.1 class BookingClass

Private data members/methods

1. The **constructor**.
2. Structures representing the types of booking classes which are used for parametric polymorphism. eg: `struct ACFirstClass_{}`
3. **Copy constructor** and **copy assignment operator**.
4. A **virtual destructor**

Public data members/methods.

1. Typedefs for all booking classes to assist readability of code.
eg: `typedef BookingClasses<ACFirstClass_> ACFirstClass;`
2. The following **pure virtual functions** to be overridden in the child classes. Names are descriptive:
    - **GetLoadFactor()**
    - **IsSitting()**
    - **GetNumberOfTiers()**
    - **IsAC()**
    - **GetName()**
    - **IsLuxury()**

D.2 template<typename T> class BookingClasses

Private data members/methods

1. The **constructor**.
2. Static data members for describing the attributes. **Names are descriptive.**

- **sIsAC (const)**
- **sNoOfTiers (const)**
- **sIsSitting (const)**
- **sName (const)**
- **sLoadFactor**
- **sIsLuxury**
- **sReservationCharges**
- **sMinTatkalCharges**
- **sMaxTatkalCharges**
- **sMinDistanceForCharge**

3. **Copy constructor** and **copy assignment operator**.
4. Destructor.

Public data members/methods

1. Type() : Static method which returns the only instance of the class.
2. The following static methods are used to change the non-const static members. Names are descriptive.

- **changeLuxury()**
- **changeLoadFactor()**
- **changeReservationCharges()**
- **changeMinTatkalCharges()**
- **changeMaxTatkalCharges()**
- **changeMinDistanceForCharge()**

3. The following **overridden virtual methods** are used to get the values of the static members. Names are descriptive.

- **GetLoadFactor()**
- **IsSitting()**
- **GetNumberOfTiers()**
- **GetName()**
- **IsAC()**
- **IsLuxury()**

E. class Gender

Class Gender is an **abstract base class** from which there are two specializations:

- class Male
- class Female

Since these classes differ only in terms of their gender, a mixture of **parametric polymorphism** along with **inclusion polymorphism** has been used to model them.

Naturally, these are **singletons** as they are used to model a concept rather than an object. These are modelled using ***Meyer's singleton.***

E.1 Class Gender

Private members/methods :

1. name_ : const string that stores the name of the gender
2. The **Male** and **Female Types**. (eg: `struct Maletype{ }`)

Protected members/methods :

1. The **constructor**
2. The **virtual destructor.**

Public members/methods

1. **GetTitle()** : virtual method that gets the salutation specific to the title.
2. IsMale(.) : checks whether the gender is male or not.
3. Typedefs to improve the readability of code.

The child classes have been modelled using a template as
template <typename T>
class GenderTypes : public Gender

For this class:
Private members/methods:
1. The constructor, virtual destructor, copy constructor.
2. sName, sSalutation : static attributes that store the name and salutation of the gender.

Public members/methods:
1. Type( ): used to get the only instance for the class
2. GetTitle( ) : overrides the same method from the parent class.

F. class Passenger

It is the blueprint of a passenger.

Private methods/members:

1. firstName_ , **middleName_**, **lastName** : Stores the first name.
2. **aadharNumber_** : Stores the aadhar number.
3. **mobileNumber_** : Stores the mobile number.
4. dateOfBirth_ : Stores the date of birth.
5. **gender_** : Stores the gender.
6. **age_** : Stores the age of the passenger after computation.
7. **disabilityIndex_** : Value stored here is indicative of the type of disability of the passenger.
8. divyaangId_ : Optional divyaang id.
8. The **constructor**.

Note:

Convention for disability index:
0: No disability (DEFAULT VALUE).
1: Blind
2: Orthopaedically Handicaped
3: Cancer Patient
4: TB Patient

Public members/methods:

1. createPassenger(. . .) : static method which takes in all the parameters of the Passenger and tries to validate it according to the specifications. If validated, a Passenger is created by calling the constructor. Else it throws an exception of a suitable type.

2. The **copy constructor**, **copy assignment operator**, **destructor**.

G. class Booking

Booking is an **abstract base class**.
This class represents a blueprint of a journey ticket.

There is a **flat level hierarchy** modelled with **inclusion polymorphism** and following classes inherit from booking:

1. **GeneralBooking**
2. **LadiesBooking**
3. **SeniorCitizenBooking**
4. **DivyaangBooking**
5. **TatkalBooking**
6. **PremiumTatkalBooking**

Class Booking

private members/methods:

1. fromStation_, toStation_ : (const) The source and destination for the booking.
2. Passenger_: (const) The passenger whose booking is done.
3. Pnr_ : (const) The unique identifier for the ticket
4. Fare_ : the fare for the booking
5. dateOfReservation_ , dateOfBooking (const for now): dates of journey and booking.
6. Message_: The remarks for booking.
7. Some static members include:
  7.1 sBaseFarePerKM (base fare per km) : This can be changed later.
  7.2 sBookingPNRSerial : this stores the current available PNR.
  7.3 sLuxuryTaxPercent : Tax to be levied on luxury coaches only.
8. The constructor.

public members/methods:

1. sBookings: This is a list of the bookings confirmed
2. ReserveBooking(...) : Static method which takes in the input for creating a booking and then calls the createBooking( ) method for the corresponding method of the bookingCategory supplied if input is valid. If not validated, it throws an exception of a suitable type.
3. ComputeFare() computes the fare based on the BookingClass (**pure virtual**).
4. ChangeAcSurcharge(), ChangeBaseFare() and **ChangeLuxuryTaxPercent()** are responsible for changing the sACSurcharge, sBaseFarePerKM and sLuxuryTaxPercent respectively.
5. It has a **virtual destructor.**

Note : After ReserveBooking( )calls the corresponding createBooking( ) method it **virtually constructs** the **specialized booking** for the corresponding booking category. This is done by following the ***virtual construction*** procedure specified in the requirements.

class GeneralBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

class LadiesBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

class SeniorCitizenBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

class DivyaangBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

class TatkalBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

class PremiumTatkalBooking

public members/methods:

1. The constructor.
2. ComputeFare( ): method overridden method from parent class.
3. The destructor.

Note : here since the logic for computation of fare is different for all classes. eg: some add to the fare while some reduce the fare. Hence **virtual polymorphism with overriding** has been used.

H. Class BookingCategory

BookingCategoryis an **abstract base class**.
This class represents a blueprint of a booking category

There is a **flat level hierarchy** modelled with **inclusion polymorphism** and following classes inherit from booking:

1. **GeneralCategory**
2. **LadiesCategory**
3. **SeniorCitizenCategory**
4. **DivyaangCategory**
5. **TatkalCategory**
6. **PremiumTatkalCategory**

Again, these are **singletons** as they are used to model a concept rather than an object. These are modelled using ***Meyer's singleton.***

Note: The hierarchies of BookingCategory and Booking run parallel to each other. We construct the corresponding booking for each category using **virtual construction**.

class BookingCategory

Protected data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking( ): pure virtual function, will be used for creating booking.

class GeneralCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..): the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

class LadiesCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..):the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

class SeniorCitizenCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..): the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

class DivyaangCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..): the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

class TatkalCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..):the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

class PremiumTatkalCategory

Private data members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public data members/methods:

1. Type( ): returns the only instance of the class.
2. CreateBooking(..): the overridden version, which is used for creating booking.
3. IsEligible(..): static method to check if the category is eligible or not.

I. class Concessions

Class concessions has been modelled as the **base class.**

This has been implemented as a **flat level hierarchy** using **ad-hoc polymorphism**.

Naturally, these are **singletons** as they are used to model a concept rather than an object. These are modelled using ***Meyer's singleton.***

classes that inherit from class Concessions:

- **Class GeneralConcession**
- **Class LadiesConcession**
- **Class DivyangConcession**
- **Class SeniorCitizensConcessions**

Class Concessions:

Protected members/methods:

1. The **constructor**
2. The **virtual destructor.**

Public members/methods:

1. Type( ): static member that returns the ONLY instance.

Class GeneralConcession:

Private members/methods

1. The **constructor**, the **copy assignment operator**, the **copy constructor**.
2. The **virtual destructor.**

Public members/methods:

1. Type( ): static member that returns the ONLY instance.
2. GetConcession ( ) : method is vague here since there is no concession, however in special cases this may be modified to give concessions to all as special offers.

Class LadiesConcession:

Private members/methods

1. The **constructor**, the **copy assignment operator**, the **copy constructor**.
2. The **virtual destructor**.

Public members/methods:

1. Type( ): static member that returns the ONLY instance.
2. GetConcession (BookingClass&, Passenger&, Date& dateOfReservation) : takes in the passenger, checks for concession and grants concession if applicable

Class DivyangConcession:

Private members/methods

1. The **constructor**, the **copy assignment operator**, the **copy constructor**.
2. The **virtual destructor**.

Public members/methods:

1. Type( ): static method that returns the ONLY instance.
2. 8 overloaded versions of **GetConcessions()** that differ in terms of the booking class.

Note: There are 8 overloaded versions of the GetConcessions function and each has been done with templates so that each function may be for calling the GetConcession method in the respective Divyaang category (there are 4 categories).

Class SeniorCitizensConcessions:

Private members/methods

1. The **constructor**, the **copy assignment operator**, the **copy constructor**.
2. The **virtual destructor**.

Public members/methods:

1. Type( ): static member that returns the ONLY instance.
2. GetConcession(BookingClass& b, Passenger& p, Date& dateOfReservation) : takes in the passenger, checks for concession and grants concession if applicable

J. class Divyaang

Class Divyaang has been modelled as the **abstract base class.**

This has been implemented as a **flat level hierarchy** using **inclusion-parametric polymorphism**.

Naturally, these are **singletons** as they are used to model a concept rather than an object. These are modelled using ***Meyer's singleton.***

classes that inherit from class Divyaang:

- **Class BlindDivyaang**
- **Class OrthopaedicallyHandicappedDivyaang**
- **Class CancerPatientDivyaang**
- **Class TBPatientDivyaang**

Class Divyaang

Private members/methods :

1. Structs for **BlindDivyaang, OrthopaedicallyHandicappedDivyaang, CancerPatientDivyaang** and **TBPatientDivyaang.** (eg: `struct BlindDivyaang_{ }`)

Protected members/methods :

1. The **constructor**
2. The **virtual destructor.**

Public members/methods

1. Type( ): static method that returns the ONLY instance.
2. Typedefs to improve the readability of code.
3. 8 overloaded versions of **GetConcessions** (one for each booking class) which are pure virtual functions.

The child classes have been modelled using a template as:

template <typename T>
class DivyaangSubCategories: public Divyaang

For each of the child classes:

Private members/methods:

1. The **constructor**
2. The **virtual destructor.**
3. 8 static attributes for concessions (one for each of the booking classes).

Public members/methods:

1. All the 8 **GetConcession methods** are overridden.
2. Type ( ) : static method which returns the only instance of the class.

K. Exception classes that inherit from class exception.

class exception:

1. **class Bad_Date** (thrown if all requirements for class Date are not satisfied)

2. **class Bad_Railways**
   a. **class Bad_Station**. (Station not available)
   b. **class Station_Duplication**. (Station name duplication)
   c. **class Distance_Duplication**. (Distance between stations is duplicate.)

3. **class Bad_Passenger**
   a. **class Bad_Name.** (name requirements not satisfied)
   b. **class Bad_Moblie_Number:** (mobile number is of invalid format)
   c. **class Bad_Aadhar_Number:** (aadhar number is of invalid format)

4. **class Bad_Booking**
   a. **class Bad_Date_Of_Reservation:** (date of reservation is not within 1 year of booking.
   b. **class Inconsistent_Booking_Category**: (booking category is inconsistent)
   c. **class Bad_Age_Of_Passenger**: (passenger isn’t born before reservation date)

*All these classes override the virtual method **const char* what()** from the **parent class exception.***

This what() method returns the description of the exception.

Apart from this, there is a **constructor**, **copy constructor** and a **virtual destructor** for all exception classes to ensure polymorphism and dynamic dispatch.