Railway Booking System

Design Document

Assignment 4

Software Engineering

CS20006

Suhas Jain

19CS30048

Design of Classes and the Object Orientation in the program

1. Class Station

i. private data members

- **const string name_** : Attribute to store the name of each Station. As the name of the object will not change during the execution of program it has been made constant.

ii. private member functions / operator functions

- **Station(const string&)** : Constructor from string
- **Station& operator=(const Station&)** : Blocked copy assignment operator
- **Station(const Station&)** : Blocked copy constructor

iii. public member functions

- **~Station()** : Destructor
- **static Station& CreateStation(const string&)** : Returns startion object after checking the string for any errors.
- **string GetName() const** : Returns the name of a Station object.
- **int GetDistance(const Station&) const** : It calls the GetDistance method of the Railways class to get the distance between two stations.
- **friend ostream& operator<<(ostream&, const Station&)** : Output streaming operator.

2. Class Railways

Singleton class

i. private data members

- **static const vector<Station> sStations** : A vector of Station objects that stores the master data, i.e., the list of all Stations. As the name of the stations will not change during the execution of program it has been made constant. It is static so it can be accessed anywhere in the program.
- **static const map<pair<string, string>, int> sDistStations** : This stores the pairwise distances between all Stations. It is implemented as a map<pair<string, string>, int>.
  It is static so it can be accessed anywhere in the program.

ii. private member functions / operator functions

- **Railways()** : Constructor
- **~Railways()** : Destructor
- **Railways(const Railways&)** : Blocked copy constructor
- **Railways& operator=(const Railways&)** : Blocked copy assignment operator

iii. public member functions

- **static const Railways& IndianRailways** : This function implements the Meyer's Singleton and returns the singleton object. It has to be made static because this function itself returns the single instance. If it is not static, then for the first call to this function we will not have any instance, and we will not be able to call it. Hence it has to be made static.
- **Station GetStation(const string& name) const** : It retrives the data from sStattions and returns the reference to that station. It has been made const as it does not need to be changed during the execution of the program.

- **int GetDistance(const Station&, const Station&)** : This is a const function as it only returns the distance between two Stations and does not need to change the state of any object. The arguments as usual are passed a const references. This function retrieves the distance between Stations from the map sDistStations by using the pair of Station names as the key.
- **friend ostream& operator<<(ostream&, const Railways&)** : Output streaming operator.

3. Class Date

i. private data members

- **static const vector<string> sMonthNames** : A vector of strings to store the names of the 12 months. It is made static as we need it at multiple times and places during the execution. It is const as it will never change as the names of the 12 months cannot change.
- **static const vector<string> sDayNames** : : A vector of strings to store the names of the 7 days of the week. It is made static as we need it at multiple instances during the execution. It is const as it will never change as the names of the days of the week cannot change.

  These have been made constants they will not change after the object has been created:
  - **const int date_**
  - **const Month month_**
  - **const int year_**

ii. private member functions / operator functions

- **Date(int, int, int)** : Constructor which takes three integers as arguments.
- **Date& operator=(const Date&)** : Blocked copy assignment operator.

iii. public member functions

- **static Date& CreateDate(const string)** : A static function that returns a Date object after checking if the string passed to it has any errors. If not, then it invokes the constructor of the Date class.
- **Date(const Date&)** : Copy Constructor with usual semantics. It has been defined as it is needed in the initializer list of the class Booking.
- **~Date()** : Destructor.
- **Day day() const** : : This function returns an integer from 0-6 depending on the day of the week with 0 denoting Sunday.
- **static Date Today()** : static function to return the Date of the present day.
- **friend int operator-(const Date&, const Date&)** : Friend function that overloads the - operator to return the difference between two dates
- **bool operator<=(const Date&)** : Overloaded operator to compare two dates.
- **bool operator==(const Date&)** : Equality operator to check equality of two dates
- **friend ostream& operator<<(ostream&, const Date&)** : Output Streaming Operator.

4. Class and Hierarchy of Gender

i. class Gender (base)

a. private data members

- **const string name_** : const data member that stores the name of the gender.

b. protected data members

- **Gender(const string& name)** : Constructor.
- **virtual ~Gender()** : Virtual Destructor for a polymorphic hierarchy.

c. **public member**

- **const string& GetName() const** : Function to get the name of the Gender. It is made const as it does not change the state of any object.
- **virtual const string GetTitle() const = 0** : pure virtual function to make the class abstract.
- **static bool IsMale(const Gender&)** : static function to check if an object is of Male or Female type.

ii. **class Male and Female made using template GenderTypes<T>**

Male and Female are singleton classes modelled using static sub-typing (templates)

a. **private data members**

static constants. These are not needed in the application space and are hence initialized in Gender.cpp itself:

- **static const string sName**
- **static const string sSalutation**

b. **private member functions / operator functions**

- **GenderTypes(const string& name = GenderTypes<T>::sName) :** Constructor.
- **~GenderTypes() : Destructor.**

c. **public member functions**

- **static const GenderTypes<T>& Type() :** Function to return the singleton object.
- **const string GetTitle() const** : Overriden function to get the salutation for a gender

5. Class Passenger

i. **private data members**

These are the data members to store information about a Passenger. All are made const because none of them can change once entered:

- **const Name name**
- **const Date dateOfBirth**
- **const Gender& gender**
- **const string& aadhaar**
- **const string& mobile**
- **const Divyaang& disability**
- **const string& disabilityID**

ii. **private member functions / operator functions**

- **Passenger(const Name, const Date, const Gender&, const string&, const string&, const Divyaang&, const string&)** : Constructor.

iii. **public member functions**

- **static Passenger& CreatePassenger(const Name, const Date, const Gender&, const string&, const string&, const Divyaang&, const string&)** : A static function that returns a Passenger object after checking if the attributes passed to it have any errors. If not, then it invokes the constructor of the Passenger class.
- **const Date GetDateOfBirth() const :** Returns the dateOfBirth . Made const as it will not change the state of the object.
- **const Gender& GetGender() const :** Returns the gender . Made const as it will not change the state of the object.
- **const Divyaang& GetDisability() const :** Returns the disability. Made const as it

will not change the state of the object.

6. Class and Hierarchy of BookingClass

The class BookingClass and its 8 child classes are modelled using a mix of static sub-typing and inclusion polymorphism.

i. class BookingClass

a. private data members

- **const string name** : const data member that stores the name of the booking class.

b. protected member functions

- **BookingClass(const string& name)** : Constructor.
- **virtual ~BookingClass()** : Virtual Destructor for a polymorphic hierarchy.

c. public member functions

- **const string& GetName() const** : Function to get the name of the BookingClass. It is
- made const as it does not change the state of any object.
- **friend ostream& operator<<(ostream&, const BookingClass&)** : Output Streaming Operator.

There are a number of pure virtual functions which can be listed down as follows :

- **virtual bool IsAC() const = 0**
- **virtual bool IsLuxury() const = 0**
- **virtual bool IsSitting() const = 0**
- **virtual double GetLoadFactor() const = 0**
- **virtual int GetNumberOfTiers() const = 0**
- **virtual double GetReservationCharge() const = 0**
- **virtual double GetTatkalFactor() const = 0**
- **virtual double GetTatkalMinCharge() const = 0**
- **virtual double GetTatkalMaxCharge() const = 0**
- **virtual int GetMinTatkalDistance() const = 0**

Now, we have 8 booking classes, These are singleton classes modelled using static sub-typing (templates):

- ACFirstClass
- ExecutiveChairCar
- AC2Tier
- FirstClass
- AC3Tier
- ACChairCar
- Sleeper
- SecondSitting

ii. Modelling of these 8 classes using the template BookingClassType<T>

a. private data members / member functions / operator functions

- **static const string sName** : static constant to set the name of the booking class.

These static constants will never be changed, hence they are intialized in the library itself:

- **static const bool sIsSitting**
- **static const bool sIsAC**
- **static const int**
- **sNumberOfTiers**
- **static const double sLoadFactor, static const bool sIsLuxury, static const double, sReservationCharge, static const double, sTatkalLoadFactor, static const double, sMinTatkalCharge, static const double, sMaxTatkalCharge, static const int,**

sMinTatkalDistance : These are changeable static constants, hence they are intialized in the application space

b. private member functions / operator functions

.

- **BookingClassType(const string& name = BookingClassType<T>::sName)** : Constructor.
- **~BookingClassType()** : Destructor.

c. public member functions

- **static const BookingClassType<T>& Type()** : Function to return the singleton object.

All the pure virtual functions are again redefined to make these classes concrete classes.

7. Class and Hierarchy of BookingCategory

The class BookingCategory and its child classes are modelled using static sub-typing by inclusion and parametric polymorphism.

i. class BookingCategory

a. private data members / member functions / operator functions

- **const string name** : const data member that stores the name of the booking category.

b. protected member functions

- **BookingCategory(const string& name**) : Constructor.
- **virtual ~BookingCategory()** : Virtual Destructor for a polymorphic hierarchy.

c. public member functions

- **const string& GetName() const** : Function to get the name of the BookingCategory.
- **virtual bool IsEligible(Passenger&) const = 0** : Pure virtual function, it serves to check the eligibility of a person in the leaf classes (booking categories).
- **friend ostream& operator<<(ostream&, const BookingCategory&)** : Output Streaming Operator.

Now, BookingCategory has 5 concerete singleton child classes - General, Ladies, SeniorCitizen, Tatkal, PremiumTatkal modelled as templates using the name BookingCategoryType<T>. Also, BookingCategory has an abstract child class Divyaang which has its own hierarchy as described in the next section.

ii. The 5 concrete Booking Categories denoted by the template BookingCategoryType<T>

a. **private data members / member functions / operator functions**

- **static const string sName** : static constant to set the name of the booking class.
- **BookingCategoryType(const string& name BookingCategoryType<T>::sName) :** Constructor.
- **~BookingCategoryType()** : Destructor.

b. public member functions

- **static const BookingCategoryType<T>& Type()** : Function to return the singleton object.

- **bool IsEligible(Passenger&) const** : Implementing the pure virtual function defined in BookingCategory.
- **Booking* CreateBooking(...)** : A function made in accordance with the virtual construction idiom to invoke the correct constructor of the Booking hierarchy in accordance with the class in the BookingCategory hierarchy.

8. Class and Hierarchy of Divyaang

Divyaang is just an abstract class derived from the class BookingCategory.
It has 4 concerete singleton child classes - Blind, OrthoHandicapped, Cancer, TB , each standing for a disability type. These are modelled by DivyaangType<T>.

i. class Divyaang

a. public member functions

- **friend ostream& operator<<(ostream&, const Divyaang&)** : Output Streaming Operator

ii. 4 concrete Divyaang Categories denoted by the template DivyaangType<T>

a. private data members / member functions / operator functions

- **static const string sName** : static constant to set the name of the booking class.
- **DivyaangType(const string& name = DivyaangType<T>::sName**) : Constructor.
- **~DivyaangType()** : Destructor.

b. public member functions

- **static const DivyaangType<T>& Type()** : Function to return the singleton object.
- **bool IsEligible(Passenger&) const** : Implementing the pure virtual function defined in BookingCategory.
- **Booking* CreateBooking(...)** : A function made in accordance with the virtual construction idiom to invoke the correct constructor of the Booking hierarchy in accordance with theclass in the BookingCategory hierarchy.

9. Class and Hierarchy of Concessions

This class stores the data regarding different concessions that are given for different types of booking categories. The clild classes of the Concessions class have very few elements in common, hence it is modelled using ad-hoc polymorphism.

i. class Concessions (base)

a. private data members

- **const string name_** : const string variable to store name of the Concessions class.

b. protected data members

- **Concessions (string&)** : Constructor.
- **~Concessions()** : Destructor.

There are 4 singleton classes of the base class:

ii. class GeneralConcession

a. private data members

- **static const double sFactor** : static constant to store the concession factor for the general category, which will be 0 for all cases.
- **GeneralConcessions(const String&)** : Constructor.
- **~ GeneralConcession()** : Destructor.

b. public memver functions

- **static const GeneralConcession& Type()** : function to return the singleton object.
- **friend ostream& operator<<(ostream&, const GeneralConcession)** : Output streaming operator.
- **double GetFactor()** : static function that will return the the concession factor for the general category, which will be 0 for all cases.

iii. class LadiesConcession

a. private data members

- **static const double sFactor** : static constant to store the concession factor for the ladies category, which will be 0 for all cases.
- **GeneralConcessions(const String&)** : Constructor.
- **~ GeneralConcession()** : Destructor.

b. public memver functions

- **static const LadiesConcession& Type()** : function to return the singleton object.
- **friend ostream& operator<<(ostream&, const LadiesConcession)** : Output streaming operator.
- **double GetFactor()** : static function that will return the the concession factor for the ladies category, which will be 0 for all cases.

iv. class SeniorCitizenConcession

a. private data members

- **static const double sFactorMale** : static constant to store the concession factor for a male senior citizen.
- **static const double sFactorMale** : static constant to store the concession factor for a female senior citizen.
- **GeneralConcessions(const String&)** : Constructor.
- **~ GeneralConcession()** : Destructor.

b. public memver functions

- **static const LadiesConcession& Type()** : function to return the singleton object.
- **friend ostream& operator<<(ostream&, const LadiesConcession)** : Output streaming operator.
- **double GetFactor()** : static function that will return the the concession factor for the senior citizen category.

v. class DivyaangConcession

a. private data members

- **static const map<pair<BookingClass*, Divyaang*>, double> sFactor :** The concession factor for Divyaang also depends on BookingClass and the type of Divyaang the object is so we store the data in the form of map.
- **GeneralConcessions(const String&)** : Constructor.

- **~ GeneralConcession()** : Destructor.

b. public memver functions

- **static const LadiesConcession& Type()** : function to return the singleton object.
- **friend ostream& operator<<(ostream&, const LadiesConcession)** : Output streaming operator.
- **double GetFactor()** : static function that will return the the concession factor for the an object of type Divyaang and a particular booking class.

10.Class and Hierarchy of Booking

i. class Booking

a. private data members

- **static int sBookingPNRSerial** : A static variable to keep track of the next PNR to be allocated. It is made static as we need a single copy of this for all the Booking objects. Hence it cannot be object-specific. It is not made const as its value increases by one every time a new Booking is made.

b. protected data members / member functions

- **const Station fromStation** : Made const as once the booking is made, the station from which the booking is done will never change.
- **const Station& toStation** : Made const as once the booking is made, the station upto
  which the booking is done will never change.
- **const Date dateOfBooking** : This is also made const as date of travel will not change once the booking has been done.
- **const BookingClass& bookingClass** : Made const as the booking class will not change after the booking. It has been made a reference because all booking classes are singleton classes, hence there is only one instance (object) for each of them. Thus, whenever we need to use a booking class object, we will have to use a reference to that single instance.
- **const BookingCategory& bookingCategory** : Similar to bookingClass , this is also made const.
- **Passenger passenger** : Passenger information, as a reference to the object.
- **int fare** : The fare computed for the booking using the ComputeFare() method.
- **const int PNR** : Made const as the PNR is kind of a unique ID for each booking, and hence will always remain constant for the booking. It is not made const as its value increases by one every time a new booking is made.
- **Booking(const Booking&)** : Blocked copy constructor.
- **Booking& operator=(const Booking&)** : Blocked copy assignment operator.
- **Booking(...)** : Constructor.
- **~Booking()** : Virtual Destructor for a polymorphic hierarchy.

c. public data members / member functions

- **static const double sBaseFarePerKM** : static constant that is initialized from the application space.
- **static vector<Booking*> sBookings** : This is a vector of pointers to objects of the Booking class. This is basically a list of all the Bookings made till now. It is made

static as we need a single copy of the list irrespective of the objects. It is however not const as it gets updated whenever a new Booking is made.

- **friend ostream& operator<<(ostream&, const Booking**&) : Output streaming operator.
- **Booking* ReserveBooking(...)** : Function which is called from the application space to create a booking. This, in accordance with the virtual construction idiom calls the appropriate CreateBooking(...) function of the BookingCategory hierarchy.
- **virtual int ComputeFare() const = 0** : Pure virtual function which implements the fare computation logic.

Similar to the BookingCategory hierarchy, Booking has 5 concrete child classes - GeneralBooking, LadiesBooking, SeniorCitizenBooking, TatkalBooking, PremiumTatkalBooking modelled as templates using the name BookingCategoryType<T>.
Also, Booking has an abstract child class DivyaangBooking.
DivyaangBooking further has 4 concrete child classes –
BlindBooking,
OrthoHandicappedBooking,
CancerBooking,
TBBooking.
All the concrete leaf classes have the following member functions : are singleton classes and implement the ComputeFare() function.

- Constructor.
- Destructor.
- **Type()** - To get the singleton object.
- **int ComputeFare() const** - Implements the fare computation logic