RAILWAY BOOKING APPLICATION DESIGN

-Abhilash Datta 19CS30001

LOW-LEVEL DESIGN (LLD)

The following practices were kept in mind :

1. **Encapsulation**
    - Encapsulation has been maximized for every class
    - Private access specifiers are used for all data members that are not needed by derived classes. Protected members used otherwise.
    - Public access specifier used for interface methods and static constants and friend functions only.

2. **STL Containers**
    - STL containers (like vector, map, hash-map, list, etc.) and their iterators have been used. Arrays have not been used.
    - Iterators have been used for STL containers. Bare for loops have not been used unless unavoidable.

3. **Pointers & References**
    - The use of pointers has been minimised. Pointers are used only if null-able entities are needed.
    - If pointers are used for dynamically allocated objects (this has been minimized), they have been deleted at an appropriate position in the program.
    - const references have been used wherever possible.

The following design principles were strictly followed while making the High-level Design :

- Flexible & Extensible Design
    - The design is made to be flexible. It is easy to change the changeable parameters (like base rate, load factor etc.) easily from the application space. It does not need re-building of the library of classes.
    - The design has been made extensible. It is easy to add new classes anywhere. And does not require a re-coding.

- Minimal Design
    - Used typedefs wherever possible to improve readability
    - Only the stated models and behaviour have been coded. No extra class or method that is not required is added to the system .
    - Less code, less error principle has been followed.

- Reliable Design
    - Reliability should be a priority. Everything should work as designed and coded.
    - Data members, methods and objects should be made constant wherever possible.
    - Parameters should have appropriately defaulted wherever possible

- Testable Design
    - Every class should support the output streaming operator for checking intermittent output if needed.
    - Every class should be unit tested.
    - Test Applications should be designed for testing the application on different scenarios of use.

CODING GUIDELINES FOLLOWED

- Use CamelCase for naming variables, classes, types, and functions
- Every name should be indicative of its semantics
- Start every variable with a lowercase letter
- Start every function and class with an uppercase letter
- Use a trailing underscore _ for every data member
- Use a leading 's' for every static data member
- Do not use any global variable or function (except main(), and friends)
- No constant value should be written within the code - should be put in the application as static
- Prefer to pass parameters by value for build-in type and by const reference for UDT
- Every polymorphic hierarchy must provide a virtual destructor in the base class
- Prefer C++ style casting (like static_cast(x) over C Style casting (like (int))
- The project should compile without any compiler warning
- Indent code properly

HIGH-LEVEL DESIGN (HLD)

- **Class Date**

  STATIC MEMBERS

  Private
  1. enum Month : stores the name of the months

  -NON-STATIC MEMBERS

  Private
  1. const unsigned int day : stores date
  2. const unsigned int month : stores month
  3. const unsigned int year : stores year

  -NON-STATIC METHODS

  Public
  1. Date(unsigned int, unsigned int, unsigned int) : Constructor to initialize date object when month = MM

  2. Date(unsigned int, string, unsigned int) : Type 2 constructor to initialize data object when Month is inputted as MMM

  3. ~Date() : Destructor

  4. friend ostream & operator << (ostream &out, const Date &) : Friend Function to display the Date

  5. friend bool operator == (const Date &, const Date &) : Friend Function to check if 2 dates are the same

  6. void validateDate(): to check the validity of date

  7. int computeAge(): for computing age

8. double computeSpan(): to compute span of year

- **Class Station**

  NON-STATIC MEMBERS

  Private
  1. const string name : string to store the unique name of the station

  NON-STATIC METHODS

  Public
  1. Station(const string &) : Constructor to initialise the object of class Station with station name as parameter

  2. string GetName() : Function to return the name of the station as string

  3. int GetDistance(string) : Function to return to distance of the station object from the station named as the string passed as parameter

  4. friend ostream & operator << (ostream &out, const Station &) : Friend Function to print the details of the Station object

  5. ~Station() : Destructor

- **Class Railways**

  STATIC MEMBERS

  Private
  1. const vector<Station> sStations : vector of strings to store the names of all stations

  2. const unordered_map< pair < Station, Station>, int > sDistStations : map to store the distance between 2 stations (same station pair and invalid pairs have not been initialised and)

  STATIC METHODS

Private
1. int GetDistance(const string &a, const string &b) : function to return the distance between 2 stations named a and b

2. unordered_map< pair < Station, Station >, int > createMap() : Function to initialise the Map sDistStations
(Manually entry)

3. Railways & IndianRailways() : Function to return the singleton object of class Railways

-NON-STATIC METHODS

Private
1. Railways() : Class constructor

Public
1. ~Railways() : Class Destructor

- **Class BookingClass** [ABSTRACT]

HIERARCHY DESIGN

1. A single level inclusion-polymorphism based class BookingClassType is declared to help avoid using inclusion polymorphism for various type classes .

2. BookingClassTypes is a template based class which depends on the structs

3. Parametric Polymorphism is implemented for all the type of booking classes types with the BookingClassType as the Base Class

-METHOD AND MEMBER TYPE JUSTIFICATION

1. All return methods are declared as virtual so that they can be defined in the inherited class BookingClassType which are then further inherited by the leaf classes by parameterised inheritance

2. members corresponding to the static members in BookingClassType are declared in BookingClass to store the booking Class details

HIERARCHY

BookingClass
- |-> BookingClassType (Template based class) [single level inclusion polymorphism]
  - |-> ACFirstClass
  - |-> ExecutiveChairCar
  - |-> AC2Tier
  - |-> FirstClass
  - |-> AC3Tier
  - |-> ACChairCar
  - |-> Sleeper
  - |-> SecondSitting

  - [NOTE : Parametric polymorphism for the above leaf classes]

-NON-STATIC MEMBERS

Private

1. const string name_ : to store the name of the booking class
2. const double fareLoadFactor_ : to store load factor
3. const bool isSeat_ : to store whether the class has berth or not
4. const bool isAC_ : to store whether the class type is AC/Non-AC
5. const int numOfTiers_ : To store the number of tiers in the Booking Class Type
6. const bool isLuxury_ : To store whether the booking class type is luxury/non-luxury
7. const double reservationCharge : To store the booking class type reservation charge
8. const double tatkalLoadFactor : To store the tatkal load factor related to the booking class type
9. const double minTatkalCharge : to store the minimum tatkal charge
10. const double maxTatkalCharge : to store the maximum tatkal charge
11. const int minTatkalDistance : to store the minimum distance after which tatkal charge is charged for the booking class type

12. STRUCTS
    - ACFirstClassType
    - ExecutiveChairCarType
    - AC2TierType
    - FirstClassType
    - AC3TierType
    - ACChairCarType
    - SleeperType

SecondSittingType

NON-STATIC METHODS

Public

1. friend ostream & operator << (ostream &out, const BookingClasses &b) : Friend Function to print the details of the Booking Class

2. virtual string GetName () const : Function to return Booking Class Type [only for inheritance]

3. virtual double GetLoadFactor () const : Function to return Load Factor of Booking Class [only for inheritance]

4. virtual bool IsSitting () const : to return whether the Class Type is Sitting or not [only for inheritance]

5. virtual bool IsAC () const : to return whether the booking class type is AC/non-AC [only for inheritance]

6. virtual int GetNumberOfTiers () const : to return the number of tiers for the booking class type [only for inheritance]

7. virtual bool IsLuxury () const : To return Luxury status for Booking class type [only for inheritance]

8. virtual double GetTatkalLoadFactor() : To return the tatkal load factor [only for inheritance]

9. virtual double GetMinTatkalLoadFactor() : To return the minimum tatkal load factor [only for inheritance]

10. virtual ~BookingClasses() : virtual destructor to allow inclusion inheritance [only for inheritance]

- **Class BookingClassTypes** : PUBLICLY INHERITED FROM BOOKING CLASS [TEMPLATE CLASS] [LEAF CLASSES INHERITED BY PARAMETERISED INHERITANCE]

    -STATIC MEMBERS

    Private
    1. const string name_ : to store the name of the booking class
    2. const double loadFactor_ : to store load factor
    3. const bool berth_ : to store whether the class has berth or not
    4. const bool ac_ : to store whether the class type is AC/Non-AC
    5. const int noOfTiers_ : To store the number of tiers in the Booking Class Type
    6. const bool luxury_ : To store whether the booking class type is luxury/non-luxury
    7. const double reservationCharge : To store the booking class type reservation charge
    8. const double tatkalLoadFactor : To store the tatkal load factor related to the booking class type
    9. const double minTatkalCharge : to store the minimum tatkal charge
    10. const double maxTatkalCharge : to store the maximum tatkal charge
    11. const int minTatkalDistance : to store the minimum distance after which tatkal charge is charged for the booking class type

    -NON-STATIC METHODS

    Protected
    1. BookingClassType() : Class Constructor
    2. ~BoookingClassType() : Class Destructor

    Public :
    [virtual functions form BookingClass declared]

    1. string GetName () const : Function to return Booking Class Type

    2. double GetLoadFactor () const : Function to return Load Factor of Booking Class

    3. bool IsSitting () const : to return whether the Class Type is Sitting or not

    4. bool IsAC () const : to return whether the booking class type is AC/non-AC

5. int GetNumberOfTiers () const : to return the number of tiers for the booking class type

6. bool IsLuxury () const : To return Luxury status for Booking class type

7. double GetTatkalLoadFactor() : To return the tatkal load factor

8. double GetMinTatkalLoadFactor() : To return the minimum tatkal load factor

-STATIC METHODS

1. BookingClassType& Type() : To return the singleton class

- **Class BookingCategory** [Abstract Base Class]

HIERARCHY DESIGN

1. A single level inclusion-polymorphism based class BookingCategoryType with BookingCategory as base class is declared to help avoid using inclusion polymorphism for various type classes .

2. BookingCategoryType is a template based class which depends on the structs

3. Parametric Polymorphism is implemented for all the type of disabilities with the BookingCategoryType as the Base Class

-METHOD AND MEMBER TYPE JUSTIFICATION

1. All return methods are declared as virtual so that they can be defined in the inherited class BookingCategoryType which are then further inherited by the leaf classes by parameterised polymorphism

2. members corresponding to the static members in BookingCategoryType are declared in BookingCategry to store the Disability type details

HIERARCHY

BookingCategory

|-> BookingCategoryTypes
    |-> General
    |-> Concessional
        |-> Ladies
        |-> SeniorCitizens
        |-> Divyaang
            |->Blind
            |->TB
            |->Orthohandicapped
            |->Cancer
    |-> Priority
        |-> Tatkal
        |-> PremiumTatkal

NON-STATIC MEMBERS

1. string name_ : to store the name of the category

2. STRUCTS
    General
    Concessional
    Priority

NON-STATIC METHODS

Protected
1. BookingCategory() : class constructor
2. ~BookingCategory() : class destructor

Public
1. virtual string GetName() : return name of the booking category
2. virtual bool isEligible(Passenger) : to check the eligibility of the passenger wrt the Booking Category

- **Class General** [inherited from BookingCategory]

  NON-STATIC METHODS

  Public
  1. GeneralBooking createBooking(): to create an instance of GeneralBooking and to compute fare.

- **Class Concessional** [inherited from BookingCategory]

  HIERARCHY DESIGN
  Single level ad-hoc polymorphism is being used for the leaf classes of concession

  HIERARCHY

  Concessions
  |-> Ladies
  |-> SeniorCitizen
  |-> Divyaang

  NON-STATIC METHODS

  Protected
  1. Concessions() : class constructor
  2. ~Concessions() : class destructor

  -STATIC METHODS

  Public :
  1. Concessions & Type() ; to return the object
  2. double GetCFactor() : to return the concession factor

  ---LEAF CLASSES
  -- GENERAL CONCESSION : PUBLICLY INHERITED FROM CONCESSIONS

  STATIC MEMBERS
  Private
  const double sGeneralCFactor : to store the concession factor for General
  STATIC METHODS
  Public
  GeneralConcession & Type () : to return singleton class object
  double GetConcessionFactor(Passenger) : to return concession factor

  NON-STATIC METHODS
  Private
  GeneralConcession() : class constructor
  ~GeneralConcession() : class destructor

- **Class Ladies**: PUBLICLY INHERITED FROM CONCESSIONAL

  STATIC MEMBERS
  Private
  const double sLadiesCFactor : to store the concession factor for Ladies

  STATIC METHODS
  Public
  LadiesConcession & Type () : to return the singleton class object
  double GetConcessionFactor(Passenger) : to return the concession factor for Ladies

  NON-STATIC METHODS
  Private
  Ladies() : class constructor
  ~Ladies() : class destructor

- **Class SeniorCitizen**: PUBLICLY INHERITED FROM CONCESSIONAL

  STATIC MEMBERS
  Private
  double sSeniorCitizenCFactor : to store the concession factor for Senior Citizen

  STATIC METHODS
  Public
  SeniorCitizenConcession & Type () : to return the singleton class object
  double GetConcessionFactor(Passenger) : to return the concession factor for Senior Citizen

  NON-STATIC METHODS
  Private
  SeniorCitizen() : class constructor
  ~SeniorCitizen() : class destructor

- **Class Divyaang** PUBLICLY INHERITED FROM CONCESSIONAL

  HIERARCHY DESIGN

1. A single level inclusion-polymorphism based class DisabilityType is declared to help avoid using inclusion polymorphism for various type classes .

2. DisabilityType is a template based class which depends on the structs

3. Parametric Polymorphism is implemented for all the type of disabilities with the DisabilityType as the Base Class

METHOD AND MEMBER TYPE JUSTIFICATION

1. All return methods are declared as virtual so that they can be defined in the inherited class DisabilityType which are then further inherited by the leaf classes by parameterised polymorphism

2. members corresponding to the static members in DisabilityType are declared in Divyaang to store the Disability type details

HIERARCHY

Divyaang
- |-> DisabilityType [SINGLE LEVEL INCLUSION INHERITANCE]
  - |-> Blind
  - |-> TB
  - |-> Cancer
  - |-> Orthohandicapped

[PARAMETERISED INHERITANCE FOR THE ABOVE LEAF CLASSES]
NON-STATIC MEMBERS

1. string name_ : name of disability

2. double divyaangConcessionFactor_ : value of concession factor

3. STRUCTS
   - BlindType
   - TBPatientType
   - CancerPatientType
   - OrthopaedicallyHandicappedType

-NON-STATIC METHODS

Protected :

1. Divyaang() : class constructor
2. virtual ~Divyaang() : class destructor

[VIRTUAL FUNCTIONS ONLY FOR INHERITANCE]

Public :

3. virtual string GetName() : to return the name of the disability type

- **Class DisabilityTypes** : PUBLICLY INHERITED FROM DIVYAANG
[TEMPLATE CLASS] [FOR LEAF CLASS INHERITANCE BY PARAMETERIZED POLYMORPHISM]

    STATIC MEMBERS

    Private

    1. const double sACFirstClassCFactor : to store the concession factor for AC First Class
    2. const double sExecutiveChairCarCFactor : to store the concession factor of Executive Chair Car
    3. const double sAC2TierCFactor : To store the concession factor of AC 2 Tier
    4. const double sFirstClassCFactor : To store the concession factor of First Class
    5. const double sAC3TierCFactor : To store the concession factor of AC 2 Tier
    6. const double sACChairCarCFactor : To store the concession factor of AC Chair Car
    7. const double sSleeperCFactor : To store the concession factor of Sleeper
    8. const double sSecondSittingCFactor : To store the concession faction of Second Sitting

    NON-STATIC METHODS

    Private

    1. DisabilityType() : class constructor
    2. ~DisabilityType() : class destructor

    Public
    3. string GetName() : to return the name of the disability type

STATIC METHODS

Public
1. DisabilityType & Type() : Returning the singleton object of leaf classes

- **Class Passenger**

NON-STATIC MEMBERS
Private

1. const Name name_ : to store the name of the Passenger
2. const Date dateOfBirth : to store the date of birth of the passenger
3. const Gender gender_ : to store the gender of the passenger
4. const string mobileNo_ : to store the mobile number of the passenger
5. const string aadharNo_ : to store the unique Aadhar ID of the passenger
6. const Divyaang disabilityType_ : to store the disability type of the passenger
7. const string disabilityID_ : to store the disability ID of the disabled passenger

- NON-STATIC MEMBERS

Public
1. Passenger() : class constructor
2. ~Passenger() : class destructor
3. friend ostream & operator << (ostream &, Passenger &) : to display the passenger details

-HIERARCHY

Passenger HAS-A Date
Passenger HAS-A Name
[Name : string firstName_, middleName_, lastName_]
Passenger HAS-A Divyaang

- **Class Booking**

HIERARCHY DESIGN

1. A single level inclusion-polymorphism based class BookingType with Booking as the base class is declared to help avoid using inclusion polymorphism for various leaf classes .

2. BookingType is a template based class which depends on the structs

3. Parametric Polymorphism is implemented for all the type of Booking Types with the BookingType as the Base Class

METHOD AND MEMBER TYPE JUSTIFICATION

1. All return methods are declared as virtual so that they can be defined in the inherited class BookingType which are then further inherited by the leaf classes by parameterised polymorphism

2. members corresponding to the static members in BookingType are declared in Booking to store the Disability type details

HIERARCHY

Booking
|-> BookingType
|-> GeneralBooking
|-> ConcessionalBooking
|-> LadiesBooking
|-> SeniorCitizenBooking
|-> DivyaangBooking
|->BlindBooking
|->TBBooking
|->OrthohandicappedBooking
|->CancerBooking
|-> PriorityBooking
|-> TatkalBooking
|-> PremiumTatkalBooking

- STATIC MEMBERS
Public
vector<Booking*> sBookings : to store all the bookings conducted

Protected
int sGeneratePNR : to generate PNR by incrementing at every constructor call
double sBaseFarePerKm : to store the base fare per kilometer

NON-STATIC MEMBERS

Protected
1. const Station toStation_ : to store the final travel station
2. const Station fromStation_ : to store the initial travel station
3. const Date dateOfBooking_ : to store the date of Booking
4. const Date dateOfReservation_ : to store the date of Reservation
5. const BookingClass bookingClass_ : to store the Booking Class
6. const Passenger passenger_ : to store the passenger details
7. const pnr_ : to store the Booking PNR
8. int fare_ : to store the travel fare
9. bool bookingStatus_ : to store the booking status
10. string bookingMessage_ : to store the message for successful booking
11. STRUCTS
    GeneralBooking
    ConcessionalBooking
    PriorityBooking

NON-STATIC METHODS

Protected
Booking() : Class Constructor
virtual ~Booking() : Class Destructor

Public
virtual int ComputeFare() : to compute and return the travel fare
Booking & Reservation(Station, Station, BookingClasses, Date, Date, Passenger, BookingCategory) : to make a reservation and returning the booking object which stores the details

- **Class BookingTypes** : PUBLICLY INHERITED FROM BOOKING

NON-STATIC MEMBERS
Private
BookingCategory bookingCategory_ : to store the booking Category

NON-STATIC METHODS
Public

1. BookingType(Station, Station, BookingClasses, Date, Date, Passenger, BookingCategory) : class constructor
2. ~BookingType() ; class constructor
3. int ComputeFare() : to compute and return the travel fare
4. isEligible(Passenger) : to check the eligibility of the passenger wrt to a specific booking category

[The hierarchy is same as that of BookingCategory Class but they all have an extra computeFare function]

- **Class Exceptions** : PUBLICLY DERIVED FROM STD::EXCEPTIONS

  STATIC METHODS

  1. Bad_Date : to check the validity of a given date and to catch and report any errors
  2. Bad_Railways : to check the validity of the class Railways and to catch and report any errors
  3. Bad_Passenger : to check the validity of the given data of a passenger and to catch and report any errors
  4. Bad_Booking : to check the validity of the given data of a booking and to catch and report any errors

THE END