Name: Debanjan Saha **Roll No: 19CS30014**

DESIGN

This document contains the design plan for the Railway Reservation System (LLD+HLD)

Below are the various classes that are required for implementing the system:

Station

1. Station contains the ***name*** of the station as an attribute.
2. The class has a static method **CreateStation** which takes the name of the station and creates an instance of it.
3. It has a **constant GetName** method which returns the name of station.
4. It has a **constant GetDistance** method which returns distance between 2 stations.
5. The class has it's **output stream overloaded** to print the attributes and allow for easy debugging.

Date

1. Date contains the ***date***, ***month***, and ***year*** as attributes.
2. The class has its **output stream overloaded** to print the attributes and allow for easy debugging.
3. The class Date has a static method **CreateDate** which takes data about the Date and **returns an instance** of Date if the data pertaining to a valid Date, else throws Bad_Date.
4. It has the **equality operator overloaded** to check if 2 dates are equal.
5. There is a method **isWithinAYear** to check whether a date is within one year of the current date and a method **ComputeAge** which is used to find age of a person at a given date.
6. Less than(<) operator is overloaded which checks whether a date is less than another date.

Passenger

1. It holds the information about the passenger: ***firstName, middleName, lastName, date of birth, gender, aadhar, mobile number, disability type, disabilityID*.** All these are **constant** attributes.

2. For creating passengers, a static method **CreatePassenger** is called in which various error handling is done, checking whether first name or last name exists, date of reservation is greater than date of birth. A valid 12 digit aadhaar number, and valid 10 digit mobile number(if given) is entered. If any of these checks fail, then a **Bad_Passenger** is thrown, else an instance of Passenger is created and returned.

3. Getter methods for name(**GetName**)**,** disability(**GetDisability**), gender(**GetGender**) and date of birth(**GetDateOfBirth**) have been used as these details are required for verification and validation at various places.

Disability (Divyaang)

1. It's a **parametric polymorphic hierarchy**, as all subcategories of Divyaang differ by name, and provides better code reusability.

2. BlindType, OrthopaedicType, CancerType and TBType are the subcategories of Divyaang.

3. DisabilityTypes is inherited from Disability and has a template parameter T which generates the subcategories of Disabilities it.

BookingClasses

1. BookingClasses are largely similar except the attributes values, therefore to improve code reusability it has been implemented using **parametric polymorphic hierarchy**.

2. The subclasses have **static constant** members which are initialised respectively for different booking classes.

3. It's attributes have been made constants.

4. Getter methods are provided for attributes: **IsAc, IsSitting, IsLuxury, GetNumberOfTiers, GetReservationCharges** and for various Tatkal costs.

5. The leaf classes are Meyer's singletons.

Railways

1. It contains two maps ***sDistStations*** and ***sStations***. sStations contains the list of stations in the master data and sDistStations holds the distance between pairs of stations.

2. It has a method **GetStation** which returns an instance of Station given the station name from **sStations**. **GetDistance** takes names of 2 stations as inputs and returns the distance between them.

3. **IndianRailways** is a **static method** which returns the singleton instance of railways. It's implemented using Meyer's Singleton.

4. **AddStation** and **AddDistance** are private methods which are used while creating the database to add stations and distances between pairs of stations. They handle **exceptions regarding duplication of stations and distance between stations** by checking the maps about whether the station exists or not and distance between stations respectively.

Exceptions

1. Multiple exceptions classes are inherited from std::exception: **Bad_Passenger, Bad_Date, Bad_Railways** and **Bad_Booking.**

2. These are thrown when validity checks are failed, errors are thrown. These don't allow the system to go into an inconsistent state.

3. Some exceptions classes are further specialised into **subclasses** to be more indicative of the errors they represent. **Bad_Passenger** has subclasses to represent whether there was an error in data for: *name, age, aadhaar, mobile,etc*.

Booking:

1. Booking has been designed without any hierarchy for simplicity and reducing repetitive lines of codes. I have not created multiple subclasses of Booking as I have already handled the business logic for creating a new booking inside the **CalculateFare** methods of each concrete BookingCategory subclasses.So, it seemed pretty simpler not to make a parallel hierarchy for Booking depending on BookingCategory according to my design.

2. Booking Class **HAS-A *fromStation*** (source station for booking), ***toStation***(destination station for booking), ***dateOfBooking***, ***bookingCategory***,***passenger***, ***dateOfReservation***,***sBookings***(static vector for storing all bookings done), ***sBookingPNRSerial*** (static PNR auto generated ID) and **fare.**

3. For creating Booking, a static method **CreateBooking** is called with all the relevant details of booking and an error handling is done inside this method before invoking the constructor of the class. Various types of errors in input can be handled here like checking for a valid dateOfBooking and dateOfReservation, eligibility for a particular bookingCategory for the given Passenger etc.

4. There is also a public Getter method **GetFare** to get the Fare for a particular booking after its computation.

5. The class has it's **output stream overloaded** to print each booking as and when appropriate.

<u>BookingCategory:</u>

1. It is an abstract base class having a single attribute: ***name.***

2. It has a Getter method **GetName** for returning its name

3. It has two virtual methods: **CalculateFare-** It is called from its concrete subclasses for generating fares based on the different business logic for different passengers as we get them in the CreateBooking method of Booking Class and returns the calculated fare.

4. It also has a public method **isEligible**- for checking eligibility of a passenger to get a successful booking for a particular bookingCategory in its subclasses .

5. The **business logic** given is **implemented** by BookingCategory which accepts instances of Passenger and BookingClass, calculating the fare.

The Hierarchy of BookingCategory is divided into 3 subclasses - General,ConcessionCategory and PriorityCategory.The details of each of these are as follows:

a. <u>General:</u>

    1. It has the implementation of **CalculateFare** which returns the fare according to its business logic and also makes **any valid passenger Eligible** for this category through the method **isEligible**.

2. It has a public **static** method **Type()** to return a single **static constant** object of this class.

3. The class has it's **output stream overloaded** to print its details.

b. ConcessionCategory:

1. ConcessionCategory is part of the hierarchy of BookingCategory. It has 3 subclasses derived from it: **Ladies**, **Divyaang** and **SeniorCitizen.**

2. All of the above subclasses have a public **static** method **Type()** to return a single **static constant** object of their respective classes.

3. The subclasses have their **output stream overloaded** to print about its details.

4. All 3 categories refer to a type of booking categories that have concessions. The respective categories all have their respective concession datas. Here Divyaang is a subclass where BookingClass and Disability information is **mapped** to the **concession percentage.**

5. The leaf classes are singleton objects that are used to check eligibility of a booking and get concession percentage based on class.

c. PriorityCategory:

1. It is an abstract base class for extending the hierarchy to **Tatkal** and **PremiumTatkal** classes respectively**.**

2. Tatkal and PremiumTatkal have a public **static** method **Type()** to return a single **static constant** object of the respective classes.

3. **CalculateFare** and **isEligible** methods are implemented in Tatkal and PremiumTatkal as described above.

4. Tatkal and PremiumTatkal have their **output stream overloaded** to print their details.