Software Engineering

Class Design and their Attributes:

Date:

1. It is a concrete class which stores date in DD/MMM/YY format. eg 11/Jan/2022.
2. month number and month name in MMM format is mapped using a static vector Month
3. check for the validity of the date
4. dates before 1900 and after 2100 are considered invalid

Attributes:

date_: int
month_: int
month_name_: string
year_: int
sMonthsNames: vector<string>

Operations:

GiveDate(): unsigned int
GiveMonth(): string
Give Year(): unsigned int
methods to return the date, month, year respectively
validate() : bool

Station

1. It is a concrete class
2. stores the name of the station in private variable name_.

Attributes:

name_: string

Operations:

getName() - method which returns the name of the station. It is a const function.

getDistance() - method to return the distance of current station from the station passed in the parameter. Calls the getDistance method of Railways class. Const function.

Gender

It is a enumeration class used to identify gender of the passenger

Railways

Attributes
:

sStations: vector<Station>
sDistStations:vector<vector<int>>
sStToInd: map<string, int>

Operations:

getDistance(string, string) - method which returns the distance between two stations.

1. It is a Meyer's singleton Class because there can be just one Railway network which is IndianRailways
2. has a static member vector of Stations, sStations which stores the list of all the station
3. has a static member sDistanceStations which is a undirected graph with distance between stations as weight
It maps pair of stations to the distance between them
4. While constructing the object on the first call it reads the 'station.txt' file and extracts the information of all the stations.
5. In the constructor we have made the five stations in the IndianRailways and added them to the vector.

Passenger

1. This stores the all the information like name, aadhar, DOB etc of the passenger
2. All the Attributes will be of const type, once created it cannot be changed.

Attributes:

name: string
dateOFBirth: Date
gender: gender
aadhar: string
mobile: string
disability_type: string
disability_ID: int

Operations:

Check_Name(): bool
Check_Aadhar(): bool
Check_Moblie(): bool
Check_Disability(): bool
Comp_Age(): int
Comp_DisType(): string

Booking

It contains all the information regarding a booking

Attributes:

PNR_: int
BaseFare: double
sBaseFarePerKm: float
sBookingPNRSerial: int
sACSurcharge: float
sLuxuryTaxPercent: float
BookingStatus_: bool
bookingMessage_: string
fromStation: station
toStation: station
date: Date
bookingClass: BookingClasses& ->pointer to booking class of the booking
bookingCategory: BookingCategory&
fareComputed_: int
selfPNRnumber_: int
sBookings: vector<Booking* >

Operations:

NextPNR(): int
GetFromStation(): Station
GetToStation(): Station
GetDate(): Date
GetFare(): int
GetSelfPNR(): int
GetBookingStatus(): string
GetBookingClassName(): string
ComputeFare(): int
DoBooking(): void

1. Constructor adds the newly created booking obj to the sBookings.
2. In this class we store all the information about the ticket.
3. sBookings is a vector to store all the bookings. It is public because it needs to be accessed from main
4. All the information like source station, destination station, PNR value, booking status, booking message, booking class, etc are stored as private.

5. sPNR a static integer which counts the number of bookings and assigns the current sPNR to PNR of the current booing and then increments it by one
6. We create a constructor which takes from station, to station, date and type of compartment and make a ticket and print the information accordingly.
7. Once all data is received and verification is done, a booking object is created and ComputeFare() calculates the fare for the passenger. If found invalid, booking is terminated.

Connections:

> Passenger object has all the details of the passenger once entered. First of all, all the user details are verified, like optional behavior of first, mid and last name, string length of aadhar and mobile number, etc.
> It is then connected to the station class. All station and railway related information must be correct for booking to be done, example from and to station.
> Booking class ,and booking category are obviously connected to “booking classes” class in the class diagram. This is because we need to set the booking class and category from the input data provided. Like compartment, luxury details, tatkal details etc.

Booking Classes

Attributes:

loadFactor: float
name: string
tatkalLoadfactor: float -min TatkalCharge: int
maxTatkalCharge: int
min TatkalDist: int

Operations:

getLoadFactor(): double
GetName(): string
IsStiting(): bool
ISAC(): bool
GetNumberOfTiers():int
isLuxury(): bool

It is implemented using Parametric Polymorphism along with Inclusion Polymorphism.
The BookingClass class has all the information about the type of the ticket the user want to make.
We have used inheritence to make the design flexible and easily modifiable in the future.

Now we will look into various inherited child of the BookingClass which are:
1. ACFirstClass
2. AC2Tier
3. AC3Tier
4. ACChairCar

5. FirstClass
6. Sleeper
7. SecondSitting

All these classes share the same attributes and methods

Attributes:

name: string
loadFactor: double

Methods:

Type(): ACFirstClass&
IsLuxury(): bool
IsAC(): bool
GetLoadFactor(): double
GetName(): string

They have overridden IsAC, IsLuxury, GetLoadFactor, GetName.
In these classes which are inherited from BookingClass, we have stored different information about the compartments, like its type, mode, comfort, whether it is AC or not, load factor and whether it is luxurious or not.

It is an abstract class. with no methods or attributes. Other types of disabilities are derived from this class.

The following classes have been derived from Divyang:

TB:
- Attributes
  - concessionFactor: float
- Operations:
  - getConcessionFactor(string): float

Cancer:
- Attributes
  - concessionFactor: float
- Operations:
  - getConcessionFactor(string): float

Orthopedic:
- Attributes

concessionFactor: float
Operations:
getConcessionFactor(string): float

Blind:
Attributes
concessionFactor: float
Operations:
getConcessionFactor(string): float

All the four classes, TB, Cancer, Orthopedic, Blind, are same with respective concessionFactor
as attributes and getConcessionFactor method to return respective concessionFactor. These are
all static members and attributes because there is no need to create new objects of these classes.

BookingCategory:

Attributes:

name: string
concessionFactor: float

Operations:

getName(): string
getConcessionFactor(): float

This is an abstract class, Other Booking Categories are derived from this class. Functions are not defined here but in derived classes with required specifications. Functions and attributes are self explanatory.

The following classes have been derived from BookingCategory:

Senior Citizen:
Attributes
concessionFactor: float
Operations:
getConcessionFactor(Gender): float

Ladies:
Attributes
concessionFactor: float
Operations:
getConcessionFactor(): float

Tatkal:
  Attributes
    concessionFactor: float
  Operations:
    getConcessionFactor(Gender): float

PremiumTatkal:
  Attributes
    concessionFactor: float
  Operations:
    getConcessionFactor(): float

General:
  Attributes
    concessionFactor: float
  Operations:
    getConcessionFactor(): float

All these classes are the same, just their implementation details defer. It overrides methods of the base class. All the methods and attributes are static. No need for creation of new objects.

-->All the booking Categories have its own business logic and all the derived classes overrides the methods and attributes or parent class according to their respective business logic. Final fare is calculated independently for each derived Business Categories accordingly.

Concession

For keeping the information of concessions for General and Concessional Booking according to respective Booking Categories, a Concession class is created. For this, we note that concession is dependent on BookingCategory, BookingClass, and gender, age & ability type (divyangjan) of the Passenger.

It has the following

Attributes:
concession_mat: map<pair<string, string>, float>

Methods:
getConcessionFactor(string, string): float

The concession_mat stores the float value of concession based on the booking category and divyang status.
Get concession factor simply returns the float value from the concession map.
No attribute or method is required to be static.

Exceptions

It’s hierarchy is bad date, bad passenger, bad railways, bad booking.

This helps to catch the various exceptions in the different classes.
The following exceptions class are derived from standard exceptions of C++

1. bad_date - This detects all types of exceptions in dates like date provided in a wrong format, or current or past date is provided. If software throws this exception, the user will be asked to re enter the date again.
2. bad_passenger- This detects all types of exceptions in passenger details like complete name not provided, aadhar or contact no. in wrong format, etc. If this exception is thrown, the user will be asked to enter again.
3. bad_railways- This detects all types of exceptions in railways like wrong name. If software throws this exception, user will be asked to re enter date again.
4. bad_booking- This detects all types of exceptions in the booking process. Like wrong name railway station selection. If software throws this expection, users will be asked to re enter the date again.