Railway Booking System Design

Railway Booking System

- Station class /station-class
- Railways class /railways-class
- Date class /date-class
- BookingClass class /bookingclass-class
- Divyaang class /divyaang-class
- Concession class /concession-class
- BookingCategory class /bookingcategory-class
- Passenger class /passenger-class
- Booking class /booking-class
- Exception classes /exception-classes

Station class

- `Station` is simple data class
- identified by name (`string`)
- `const std::string name_` attribute is kept private and and can be accessed with a get-method
- copyable class
- throws no exception
- `operator==` works by comparing name `string` s, which is obvious because a `Station` is identified by its name
- `operator<` is defined to support storing in `std::set<Station>`

Railways class

- `Railways` is a singleton class, implemented as a *Meyer's singleton*.
- `static` singleton object is accessed a `Railways::IndianRailways()`
- list of stations and list of distances are stored in `const std::set` and `const std::map` datatypes respectively (stored as `private` `static` attributes
- master data of the railways is hard coded
- `static bool ValidStation(const Station &station);` method validates a station which is used in `Booking`
- small methods are made `inline`
- all the methods and attributes are `static`

Date class

- `Date` class is a data class, yet has a significant amount of logic
- `Day` , `Month` , `Year` are `typedef` ed `unsigned short` in `Date` class (`public`)
- `day_` , `month_` and `year_` are `const` , `public` attributes
- copyable class (copy constructor is defined)
- Constructor is kept private. A `Date` object can be created by the `static Date Construct(Day day, Month month, Year year) noexcept(false);` method which valdiates the date or copy constructor
- Validation is done by
  - checking whether year is in the *1900 - 2099*
  - month in range *1-12*
  - day is checked according to month (leap year is handled)
  - in case of construction from a `string` , *dd/mm/yyyy* and *dd/MMM/yyyy* formats are valid

  `Bad_Date` exception is thrown when date is not valid
- `operator==` is defined
- `operator<` , `operator>` are defined to check which dates preceeds among 2 dates
- A supplementary `class DateDelta` is implemented to represent a duration betwen two dates
  `operator-` in `Date` class returns `DateDelta` object. `DateDelta` class has its own `Construct` method with its own validations.
- Month names are stored in a `static std::map`

BookingClass class

- `BookingClass` hirearchy is implemented as a flat *inclusion-parametric polymorphism*
- each template instance is singleton (implemented as a *Meyer's singleton*) and are accessed with `static const BookingClass &Type();` method
- `sFareLoadFactor` , `sIsLuxury` , `sReservationCharge` , `sMinimumTatkalCharges` , `sMaximumTatkalCharges` , `sMinimumDistanceForTatkalCharge` , `sTatkalChargeRate` , `sBlindConcessionFactor` , `sOrthopaedicallyHandicappedConcessionFactor` , `sCancerPatientsConcessionFactor` , `sTBPatientsConcessionFactor` are `static` variables (`private`) which need to be initialised in the application code. This is done to change to facilate changing values without re-compiling the library. They can be accessed with get-methods
- `ACFirstClass` , `ExecutiveChairCar` , `AC2Tier` , `FirstClass` , `AC3Tier` , `ACChairCar` , `Sleeper` , `SecondSitting` are the template instances in `BookingClass::` scope
- All get-methods are `virtual`
- implemented as *Meyer's singleton*

Divyaang class

- `Divyaang` hirearchy is implemented as a flat *inclusion-parametric polymorphism*
- each template instance is singleton (implemented as a *Meyer's singleton*) and are accessed with `static const BookingClass &Type();` method
- `Divyaang` class is used to represent a disability type
- agregated in `class Passenger` i.e, `Passenger` *HAS* `Divyaang` (*nullable* thought)
- `Blind` , `OrthopaedicallyHandicapped` , `CancerPatients` , `TBPatients` are `typedef` ed tempalte instances scoped in `Divyaang::`
- `static const Divyaang &Type();` method to get a singleton object reference
- This `Divyaang` hirearchy is different form `DivyaangCategory` which is a specialisation of `Concession` class

Concession class

- `Concession` class is a specialistaion of `class BookingCategory` and both are *abstract*
- `class Ladies` , `class SeniorCitizen` and `template<typename D> class DivyaangCategory` are specialisations of `Concession` class
- `DivyaangCategory<Divyaang::Blind>` , etc are the classes for different Divyaang types . i.e, the `Divyaang` hirearchy is utilised here
- all leaf classes have implementation of `virtal float CalculateFare` method (defined in `class BookingCategory`) which takes loadedFare and other paramaters and returns fare after applying concession or tatkaal fare.
- No *parametric polymorphism* is impelemented here

BookingCategory class

- `BookingCategory` is an *abstract* class
- `General`, `Concession` and `Priority` are specialisations of `BookingCategory` among which, only `General` is a leaf class and rest are abstract
- *paramentric polymorphism* doesn't give much benifit here as there is not much common code amon the three classes. Hence *inclusion polymorphism* is implemented
- `public` method:
  ```
  virtual float CalculateFare(float loadedFare, const BookingClass &bookingClass, const Passenger &passenger, unsigned distance) const = 0;
  ```
  is implemented in the leaf clasess of the hirearchy. This method is called during booking to get concessioned or additionally charged fare from the loaded fare. Virtual constructor idiom si used here. similarly,
  ```
  virtual bool IsEligible(const Passenger &passenger, const Date &dateOfBooking, const Date &dateOfReservation) const = 0;
  ```
  method too.

Passenger class

- `Passenger` is a data class
- There is no much logic here except the validations
- supplementary classes like `PhoneNumber` , `AadhaarNumber` are implemnted with their validations in `PassengerDetails::` scope
- uncopyable
- constructor is kept `private` and a `static` method `Construct` is implemented which validates the data before constructing a `Passenger` object.
- In case of invalid data `Bad_Passenger` exception is thrown or its specialisations like `Bad_Aadhaar`

Booking class

- There is no extensive hirearchy in `Booking` class
- `BookingBase` <|------ `Booking` are the classes in the so called hirearchy
- This diffferentiation is done to seperate bussiness logic and taking details in the previous version of Railway Booking System. It is just that the hirearchy is maintained to save rewriting code
- `BookingBase` contains get-methods and is an abstract class
- this is not even an inclusion polymorphism
- Constructor of `Booking` is kept `private` and a `static` method `Construct` is implemented which validated the data (like eligibilty of the person to the booking category selected, existence of the given station in `Railways` etc)
- `Bad_Booking` error is thrown when a valdiation fails

Exception classes

- `Exception` class hirearchy is staightforward inheritance
- `Exception` class is inherited form `std::exception` and `what()` method is implemented
- `Bad_Date` , `Date_Booking` , `Bad_Passenger` , `Bad_Railways` inherit from `Exception` . They may have further specialisations