**1)Customer class**

1. public int customerID(PK)

2. public string Name

3. public String address

4. public string phone\_number

Define a parameterized constructor in the Customer to initialize values to the above fields.

**Constructor:** Customer(int customerId,String name,String address,String phone\_number)

**CustomerDataException class:**

Define a custom exception class by the name “CustomerDataException".If any of the fields are invalid (for example, Name is empty, or phone\_number is invalid), it will throw CustomerDataException.Handles errors related to invalid customer data.

**CustomerUtility class** with methods:

**public List<Customer> customerData(Customer obj)**

This method returns the list of customer objects. Contains records of customer table in database we used.

**public void AddCustomer(Customer obj)**

This method used add the customer object to list.

**public void DisplayAllCustomers()**

This displays all customers details present in list.

**public Boolean IsCustomerExists(int customerId)**

This methods return true or false.If customer is exists in list return true or else false.It takes arguments has customerId and checks exists or not.

**2)MenuClass:**

Long restaurantId;(FK)

String name;

String description;

Double price;

**Parametrized constructor** : Menu(String name,String description,Double price)

**MenuDataException Class:**

It ensures that menu details such as the name, description, and price are valid. If invalid data is entered, a custom exception (MenuDataException) is thrown

**Menu Utility class:**

**public List<Menu> getMenuData()**

This method returns the list of menu items that are currently stored.

**public void addMenu(Menu menu) throws MenuDataException**

This method adds a new Menu object to the list of menus. It will validate the menu data before adding it. If the menu data is invalid, it throws a MenuDataException.

**public void displayAllMenus()**

This method displays the details of all menu items stored in the list.

**public boolean isMenuExists(Long menuId)**

This method checks if a menu item exists in the list by searching for the provided menuId. It returns true if the menu item is found, otherwise false.

**3) Restaurant class:**

private int restaturantId;

            private String name;

            private String location;

            private String cusine;

            private String contact;

            private float rating;

**Constructor :** Restaurant(int id,String name,String location,String cusine,String contact,float rating)

**RestaurantDataException Class**

This is a custom exception class used to handle invalid input for the Restaurant class. The exception will be thrown if any of the required fields are invalid.

**RestaurantUtility Class**

The RestaurantUtility class will provide helper methods for managing the list of restaurants.

**public List<Restaurant> getRestaurantData()**

This method returns the list of restaurant objects.

**public void addRestaurant(Restaurant restaurant) throws RestaurantDataException**

This method adds a new Restaurant object to the list of restaurants after validating the data. If the data is invalid, it throws a RestaurantDataException.

**public void displayAllRestaurants()**

This method displays the details of all the restaurants stored in the list.

**public boolean isRestaurantExists(int restaurantId)**

This method checks if a restaurant exists in the list by searching for the provided restaurantId. It returns true if the restaurant is found, otherwise it returns false.

**4)Order class:**

Private int orderId;

Private int customerId;(FK)

Private int restaurantId;(FK)

Private String deliveryAddress:

Private String double totalPrice;

Private String String status;

**Constructor:** order(int orderId,String deliveryAddress,double totalPrice,String status)

* Both customerId,restaurantId are fields are inherited from customer,restaurant classes

**OrderException class:**

This exception used to invalid order data.

**public OrderException(String message)**

**OrderUtility class:**

**Public static void validateTotalPrice(double totalPrice) throws OrderException**

used to validate total price.

**Public static void validateStatus(String status) throws OrderException**

validate order status

**Public static void processOrder(Order order)**

method to simulate order processing

**5)Payment class:**

Private int paymentId;

Private int orderId;(FK)

Private double amount;

Private String payment\_method;

**Constructor:**Payment(int paymentId,double amount,String payment\_method)

* orderId are inherited from order class.

**PaymentException class:**

This throws exception when invalid payment will happen.

**public PaymentException(String message)**

**PaymentUtility class:**

**public static void validateAmount(double amount) throws PaymentException**

Method to validate the payment amount

**public static void validatePaymentMethod(String paymentMethod) throws PaymentException**

Method to validate the payment method (supports "Credit Card", "PayPal", "Cash")

**public static void processPayment(Payment payment)**

Method to simulate payment processing

**6)Rating class:**

Private int ratingId;

Private int orderId;(FK)

Private int stars;

Private String comment;

**Constructor:**Rating(int ratingId,int stars,String comment)

* orderId are inherited from order class.

**RatingException Class**

custom exception class for handling issues related to the rating, such as invalid stars or missing comments.

**public RatingException(String message)**

**RatingUtility Class**

**public static void validateComment(String comment) throws RatingException**

Method to validate that the comment is not empty

**public static void processRating(Rating rating)**

Method to simulate processing the rating

**public static void validateStars(int stars) throws RatingException**

Method to validate the number of stars (it must be between 1 and 5)

**7)Driver class:**

Private int driverId;

Private String name;

Private String phoneNumber;

Private String vehicleType;

**Constructor:**Driver(int driverId,String name,String phoneNumber,String vehicleType)

* restaurantId are inherited from restaurant class;

**DriverException Class:**

**public DriverException(String message)**

This custom exception class will handle errors related to the Driver class, such as invalid phone numbers or unsupported vehicle types.

**DriverUtility Class:**

**public static void validatePhoneNumber(String phoneNumber) throws DriverException**

Method to validate phone number (it must be 10 digits)

**public static void validateVehicleType(String vehicleType) throws DriverException**

Method to validate vehicle type (supports "Car", "Bike", "Scooter")

**public static void assignDriverToRestaurant(Driver driver)**

Method to simulate driver assignment .

**8)Promotion class:**

Private int promotionId;

Private int restaurantId;(FK)

Private String promotionDetails;

Private String startDate;

Private String endDate;

**Constructor:** public Promotion(int restaurantId, String restaurantName, String restaurantAddress, int promotionId, String promotionDetails, String startDate, String endDate)

**PromotionException Class:**

A custom exception class for handling issues related to the promotion, such as invalid dates or missing promotion details.

**public PromotionException(String message)**

**PromotionUtility Class:**

**public static void validateDates(String startDate, String endDate) throws PromotionException**

Method to validate that the start date is before the end date

**public static void validatePromotionDetails(String promotionDetails) throws PromotionException**

Method to validate that promotion details are not empty

**public static void applyPromotion(Promotion promotion)**

Method to simulate applying a promotion to a restaurant

**9)Delivery class:**

Private int deliveryId;

Private int orderId;(FK)

Private int driverId;

Private String deliveryStatus;

**Constructor:**Delivery(deliveryId,driverId,deliveryStatus)

**DeliveryException Class:**

A custom exception class to handle errors related to deliveries, such as invalid status or missing driver details.

**public DeliveryException(String message)**

**DeliveryUtility Class:**

**public static void validateDeliveryStatus(String deliveryStatus) throws DeliveryException**

Method to validate delivery status (supports "Pending", "In Transit", "Delivered")

**public static void assignDriverToDelivery(Delivery delivery) throws DeliveryException**

Method to assign a driver to a delivery

**public static void trackDelivery(Delivery delivery)**

Method to simulate delivery tracking.

**Inheritance:**

**Hierarchical Inheritance:User,Customer,driver,restaurant**

**User(super class):**

public abstract class User {

protected String id;

protected String name;

protected String phoneNumber;

public User(String id, String name, String phoneNumber) {

this.id = id;

this.name = name;

this.phoneNumber = phoneNumber;

}

Getters and setters

//Abstract method

}

**Customer(subclass of User):**

public class Customer extends Person {

private String email;

private String address;

private String paymentMethod;

public Customer(String id, String name, String phoneNumber, String email, String address, String paymentMethod) {

super(id, name, phoneNumber);

this.email = email;

this.address = address;

this.paymentMethod = paymentMethod;

}

// Getters and Setters

}

import java.util.ArrayList;

import java.util.List;

**Restaurant(subclass of user):**

public class Restaurant extends Person {

private int restaurantId;

private String name;

private String location;

private String cuisineType;

private double rating;

// Constructor

public Restaurant(int personID, String name, String phoneNumber, String address, String cuisineType, double rating) {

super(personID, name, phoneNumber);

this.address = address;

this.cuisineType = cuisineType;

this.rating = rating;

}

// Getters and Setters

}

**OrderService:(some classes)**

public class OrderService {

private List<Order> orders = new ArrayList<>();

public Order placeOrder(Customer customer, Restaurant restaurant, List<OrderDetails> orderDetails) {

String orderId = "ORD" + (orders.size() + 1);

Order order = new Order(orderId, customer, restaurant, orderDetails);

orders.add(order);

return order;

}

public void cancelOrder(String orderId) {

orders.removeIf(order -> order.getOrderId().equals(orderId));

}

public List<Order> getOrders() {

return orders;

}

}

**Main Program:**

import java.util.ArrayList;

import java.util.List;

public class FoodDeliveryApp {

public static void main(String[] args) {

Customer customer = new Customer("112", "Sunny", "1234567890", "sunny@example.com", "4-156/1", "Credit Card");

Restaurant restaurant = new Restaurant("1001", "Paradise", "445 uppal", "0987654321", "indian", 4.5);

Menu pizza = new Menu("1001", "Margherita Pizza", 8.99);

Menu pasta = new Menu("1002", "Pasta Primavera", 10.99);

List<OrderDetails> orderDetails = new ArrayList<>();

orderDetails.add(new OrderDetails("OD001", pizza, 2));

orderDetails.add(new OrderDetails("OD002", pasta, 1));

OrderService orderService = new OrderService();

Order order = orderService.placeOrder(customer, restaurant, orderDetails);

System.out.println("Order placed successfully! Order ID: " + order.getOrderId());

}

}