***ONLINE FOOD DELIVERY***



***TEAM NUMBER: 36***

### **Abstract**

Online food delivery services have rapidly transformed the culinary and dining landscape over the past decade. This service model, which allows consumers to order food from a variety of local restaurants through a digital platform, has experienced significant growth due to advancements in technology, changing consumer preferences, and the global shift towards convenience-oriented lifestyles. Platforms such as Uber Eats, DoorDash, and Grubhub have become household names, providing a seamless and efficient way for customers to enjoy their favorite meals without leaving their homes. This introduction explores the evolution, mechanics, and impact of online food delivery services, highlighting their benefits, challenges, and future prospects.

### **Introduction**

The emergence of online food delivery services represents a significant shift in the food and beverage industry. Initially rooted in traditional takeaway models, these services have evolved to offer a comprehensive and sophisticated approach to dining, leveraging the power of digital technology. The basic premise involves a customer placing an order through a website or mobile app, which is then prepared by a partnered restaurant and delivered by a network of couriers.

The rapid proliferation of smartphones and increased internet penetration have been pivotal in the rise of online food delivery platforms. These services cater to the growing demand for convenience, enabling consumers to access a wide variety of cuisines from the comfort of their homes. Furthermore, the COVID-19 pandemic has accelerated this trend, as lockdowns and social distancing measures led to a surge in demand for home-delivered meals.

**Summary of Entities and thier Attributes:**

**Order**: A request made by the customer for food from a restaurant, processed through the online platform.

* **Attributes**: Order ID, customer ID, restaurant ID, list of items, total price, order status (e.g., pending, confirmed, delivered), order time, delivery time.

**Rating**: A feedback mechanism allowing customers to evaluate their experience with the food, restaurant, and delivery service.

* **Attributes**: Rating ID, order ID, customer ID, restaurant ID, driver ID, score (e.g., 1-5 stars), review text, timestamp.

**Customer**: The individual who places the order through the online platform.

* **Attributes**: Customer ID, name, contact information, address, payment details, order history, preferences.

**Payment**: The process of transferring money from the customer to the restaurant via the online platform.

* **Attributes**: Payment ID, order ID, customer ID, amount, payment method (e.g., credit card, digital wallet), transaction status, timestamp.

**Menu**: The individual responsible for picking up the order from the restaurant and delivering it to the customer.

* **Attributes**: Driver ID, name, contact information, vehicle details, current location, delivery history, rating.

**Restaurant**: The food establishment that prepares the meals ordered by customers through the online platform.

* **Attributes**: Restaurant ID, name, location, contact information, cuisine type, menu, rating, operating hours.

**Delivery**: The process of transporting the food from the restaurant to the customer's specified address.

**Attributes**: Delivery ID, order ID, driver ID, pickup time, delivery time, delivery status (e.g., in transit, delivered), delivery address.

SUMMARY OF ENTITIES RELATIONSHIP :

**Customer** 1-to-Many relationship with Order

**Customer** 1-to-Many relationship with Payment

**Customer** 1-to-Many relationship with Delivery

**Order** 1-to-1 relationship with Payment

**Order** Many-to-1 relationship with Customer

**Order** 1-to-1 relationship with Delivery

**Payment** 1-to-1 relationship with Order

**Payment** Many-to-1 relationship with Customer

**Menu** Many-to-Many relationship with Order

**Menu** 1-to-1 relationship with Restaurant

**Restaurant** 1-to-Many relationship with Order

**Restaurant** 1-to-1 relationship with Menu

**Driver** 1-to-Many relationship with Delivery

**Delivery** Many-to-1 relationship with Driver

**FUNCTIONAL REQUIREMENTS FOR UML DIAGRAM**

Functional requirements in UML (Unified Modeling Language) diagrams are depicted through various types of diagrams that each serve different purposes in modeling the behavior and structure of a system.

**Use Case Diagram**:

* **Purpose**: To capture the functional requirements of a system from an end-user perspective.
* **Elements**:
  + **Actors**: Represent users or other systems that interact with the system.
  + **Use Cases**: Represent the functional requirements (i.e., what the system should do).
  + **Relationships**: Include associations, extensions, and inclusions between use cases and actors.

**Activity Diagram**:

* **Purpose**: To model the workflow and the sequence of activities in a system.
* **Elements**:
  + **Activities**: Represent individual tasks or functions.
  + **Transitions**: Show the flow from one activity to another.
  + **Decision Points**: Represent branching based on conditions.
  + **Start and End Nodes**: Indicate the beginning and end of the process.

**Sequence Diagram**:

* **Purpose**: To illustrate how objects interact in a particular sequence to achieve a function.
* **Elements**:
  + **Objects/Participants**: Represent the different entities involved.
  + **Messages**: Show the communication between objects.
  + **Lifelines**: Represent the lifespan of an object during the interaction.
  + **Activation Bars**: Indicate when an object is active or processing.

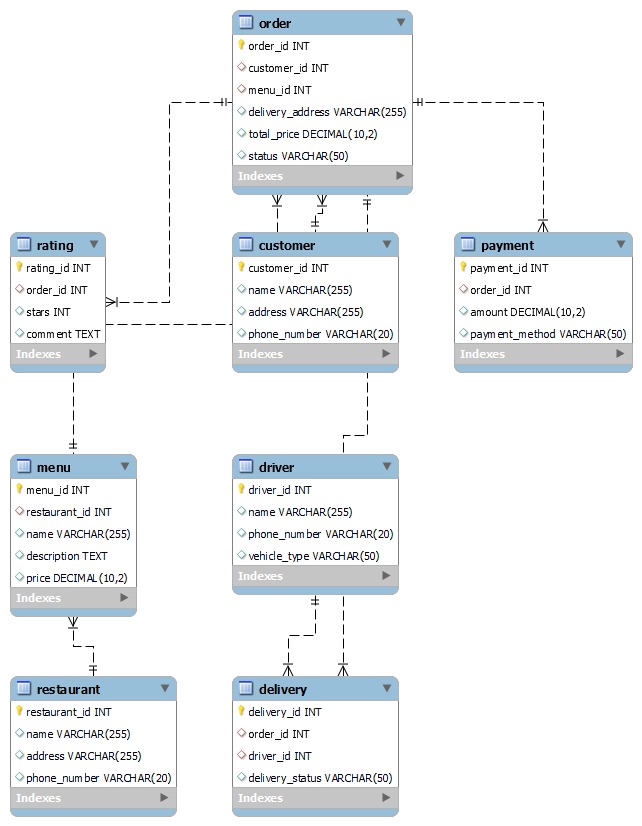
**State Machine Diagram**:

* **Purpose**: To model the states an object goes through in response to events.
* **Elements**:
  + **States**: Represent different states of an object.
  + **Transitions**: Show the movement from one state to another.
  + **Events**: Trigger transitions between states.

**Class Diagram**:

* **Purpose**: To model the static structure of the system by showing its classes, attributes, operations, and relationships.
* **Elements**:
  + **Classes**: Represent the objects or concepts.
  + **Attributes**: Define the properties of a class.
  + **Operations**: Define the behaviors or functions of a class.
  + **Relationships**: Include associations, generalizations, and dependencies between classes.

ER DIAGRAM:



Queries to Create Dabases

create database Food;

CREATE TABLE Food.Restaurant (

RestaurantID INT PRIMARY KEY,

Name VARCHAR(50),

Address VARCHAR(100),

City VARCHAR(50),

State VARCHAR(50),

ZipCode VARCHAR(10)

);

INSERT INTO Food.Restaurant (RestaurantID, Name, Address, City, State, ZipCode)

VALUES

(1, 'Burger King', '123 Main St', 'New York', 'NY', '10001'),

(2, 'Pizza Hut', '456 Elm St', 'Los Angeles', 'CA', '90001'),

(3, 'McDonalds', '789 Oak St', 'Chicago', 'IL', '60601'),

(4, 'Subway', '321 Maple St', 'Houston', 'TX', '77001'),

(5, 'Dominoes', '901 Pine St', 'Philadelphia', 'PA', '19101');

CREATE TABLE Food.Customer (

CustomerID INT PRIMARY KEY,

Name VARCHAR(50),

Phone VARCHAR(20),

Address VARCHAR(100),

);

INSERT INTO Food.Customer (CustomerID, FirstName, LastName, Email, Phone, Address, City, State, ZipCode)

VALUES

(1, 'John', 'Doe', 'johndoe@example.com', '123-456-7890', '123 Main St', 'New York', 'NY', '10001'),

(2, 'Jane', 'Smith', 'janesmith@example.com', '987-654-3210', '456 Elm St', 'Los Angeles', 'CA', '90001'),

(3, 'Bob', 'Johnson', 'bobjohnson@example.com', '555-123-4567', '789 Oak St', 'Chicago', 'IL', '60601'),

(4, 'Alice', 'Williams', 'alicewilliams@example.com', '321-987-6543', '321 Maple St', 'Houston', 'TX', '77001'),

(5, 'Mike', 'Davis', 'mikedavis@example.com', '901-555-1234', '901 Pine St', 'Philadelphia', 'PA', '19101');

CREATE TABLE Food.Order (

OrderID INT PRIMARY KEY,

CustomerID INT,

RestaurantID INT,

OrderDate DATE,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),

FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID)

);

INSERT INTO Food.Order (OrderID, CustomerID, RestaurantID, OrderDate, TotalAmount)

VALUES

(1, 1, 1, '2024-05-01', 20.00),

(2, 2, 2, '2024-05-02', 30.00),

(3, 3, 3, '2024-05-03', 25.00),

(4, 4, 4, '2024-05-04', 18.00),

(5, 5, 5, '2024-05-05', 22.00);

CREATE TABLE Food.Menu (

MenuID INT PRIMARY KEY,

RestaurantID INT,

ItemName VARCHAR(50),

Price DECIMAL(10, 2),

FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID)

);

INSERT INTO Food.Menu (MenuID, RestaurantID, ItemName, Price)

VALUES

(1, 1, 'Burger', 5.00),

(2, 1, 'Fries', 3.00),

(3, 2, 'Pizza', 12.00),

(4, 3, 'McNuggets', 6.00),

(5, 4, 'Sub Sandwich', 8.00);

CREATE TABLE Food.Driver (

DriverID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20),

LicenseNumber VARCHAR(20)

);

INSERT INTO Food.Driver (DriverID, FirstName, LastName, Email, Phone, LicenseNumber)

VALUES

(1, 'John', 'Doe', 'johndoe@example.com', '123-456-7890', 'DL123456'),

(2, 'Jane', 'Smith', 'janesmith@example.com', '987-654-3210', 'DL987654'),

(3, 'Bob', 'Johnson', 'bobjohnson@example.com', '555-123-4567', 'DL555123'),

(4, 'Alice', 'Williams', 'alicewilliams@example.com', '321-987-6543', 'DL321987'),

(5, 'Mike', 'Davis', 'mikedavis@example.com', '901-555-1234', 'DL901555');

CREATE TABLE Food.Delivery (

DeliveryID INT PRIMARY KEY,

OrderID INT,

DriverID INT,

DeliveryDate DATE,

DeliveryTime TIME,

FOREIGN KEY (OrderID) REFERENCES Food.Order(OrderID),

FOREIGN KEY (DriverID) REFERENCES Driver(DriverID)

);

INSERT INTO Food.Delivery (DeliveryID, OrderID, DriverID, DeliveryDate, DeliveryTime)

VALUES

(1, 1, 1, '2024-05-01', '12:00:00'),

(2, 2, 2, '2024-05-02', '13:00:00'),

(3, 3, 3, '2024-05-03', '14:00:00');

CREATE TABLE Food.Payment (

PaymentID INT PRIMARY KEY,

OrderID INT,

PaymentMethod VARCHAR(50),

PaymentDate DATE,

Amount DECIMAL(10, 2),

FOREIGN KEY (OrderID) REFERENCES Food.Order(OrderID)

);

INSERT INTO Food.Payment (PaymentID, OrderID, PaymentMethod, PaymentDate, Amount)

VALUES

(1, 1, 'Credit Card', '2024-05-01', 20.00),

(2, 2, 'Cash', '2024-05-02', 30.00),

(3, 3, 'PayPal', '2024-05-03', 25.00),

(4, 4, 'Debit Card', '2024-05-04', 18.00),

(5, 5, 'Apple Pay', '2024-05-05', 22.00);

CREATE TABLE Food.Rating (

RatingID INT PRIMARY KEY,

OrderID INT,

Rating DECIMAL(2, 1),

Review VARCHAR(200),

FOREIGN KEY (OrderID) REFERENCES Food.Order(OrderID)

);

INSERT INTO Food.Rating (RatingID, OrderID, Rating, Review)

VALUES

(1, 1, 4.5, 'Great food and service!'),

(2, 2, 4.0, 'Food was good, but delivery was slow.'),

(3, 3, 5.0, 'Excellent experience!'),

(4, 4, 3.5, 'Food was okay, but not impressed.'),

(5, 5, 4.8, 'Fast delivery and delicious food!');

CREATE TABLE Food.Promotion (

PromotionID INT PRIMARY KEY,

RestaurantID INT,

PromotionName VARCHAR(50),

Description VARCHAR(200),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (RestaurantID) REFERENCES Restaurant(RestaurantID)

);

INSERT INTO Food.Promotion (PromotionID, RestaurantID, PromotionName, Description, StartDate, EndDate)

VALUES

(1, 1, 'Buy One Get One Free', 'Buy one burger, get one free!', '2024-05-01', '2024-05-31'),

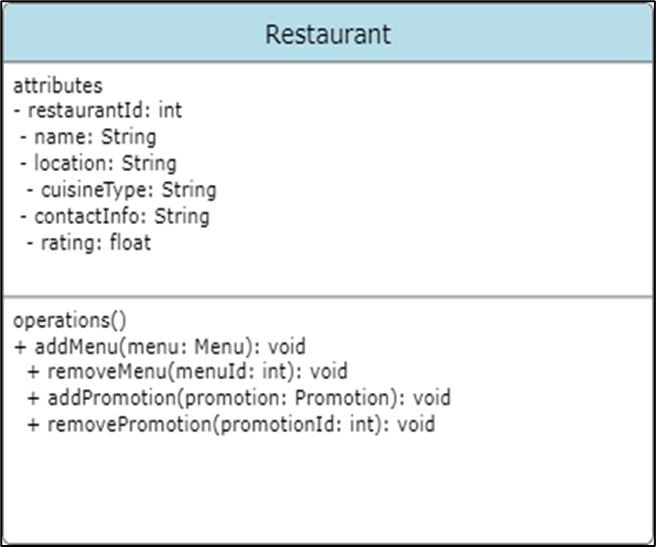
(2, 2, '10% Off All Orders', 'Get 10% off all orders over $20!', '2024-05-05', '2024-06-05'),

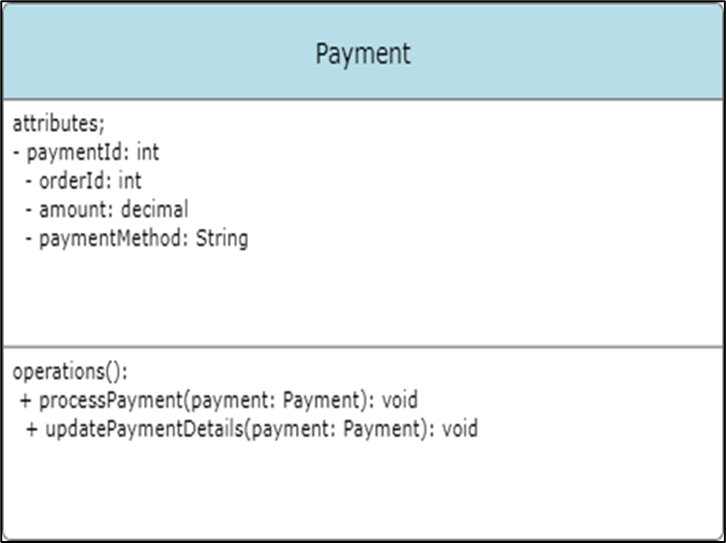
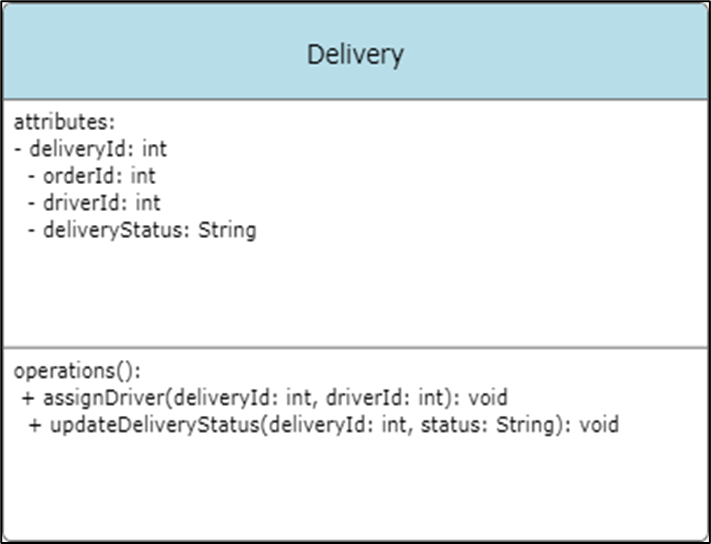
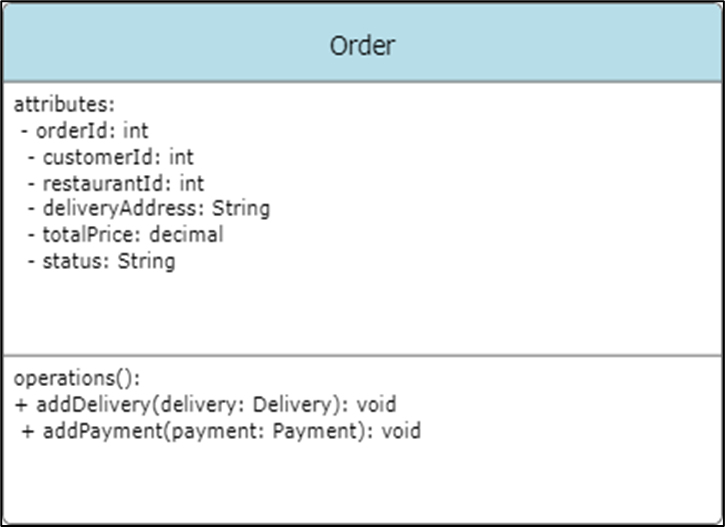
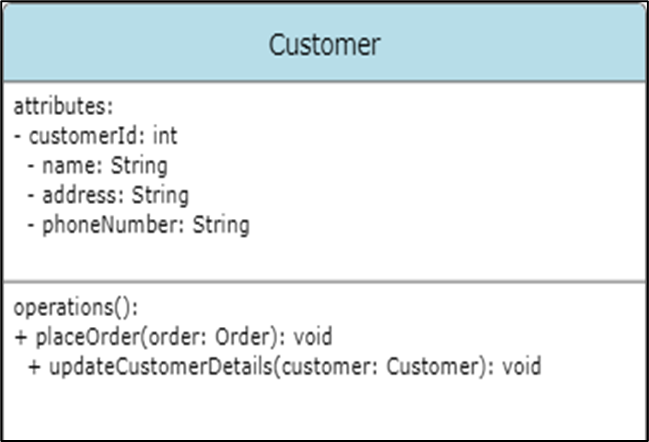
(3, 3, 'Free Delivery', 'Get free delivery on all orders over $30!', '2024-05-10', '2024-06-10'),

(4, 4, 'Kids Eat Free', 'Kids eat free with the purchase of an adult meal!', '2024-05-15', '2024-06-15'),

(5, 5, '20% Off All Menu Items', 'Get 20% off all menu items!', '2024-05-20', '2024-06-20');

UML/CLASS DIAGRAM





**All class java file codes**

**Main code:**

**Main code:** package fooddelivery;

import java.util.Date;

public class Main {

public static void main(String[] args) {

// Creating an instance of the Restaurant class

//

obj,,restaurant.setRestaurantID(x);restrestaurant.setName(ganesh);

//like this write for every getter and setter

//public int getRestaurantID() {

//return restaurantID;

// }

// public void setRestaurantID(int restaurantID) {

//this.restaurantID = restaurantID;

//}

// public String getName() {

//return name;

//}

// public void setName(String name) {

//this.name = name;

// }

// public String getAddress() {

// return address;

//}

Restaurant restaurant1 = new Restaurant(1, "Burger King", "123 Main St", "New York", "NY", "10001");

Restaurant restaurant2 = new Restaurant(2, "Pizza Hut", "456 Elm St", "Los Angeles", "CA", "90001");

Restaurant restaurant3 = new Restaurant(3, "McDonald's", "789 Oak St", "Chicago", "IL", "60601");

Restaurant restaurant4 = new Restaurant(4, "Subway", "321 Maple St", "Houston", "TX", "77001");

Restaurant restaurant5 = new Restaurant(5, "Domino's", "901 Pine St", "Philadelphia", "PA", "19101");

Restaurant restaurant6 = new Restaurant(6, "KFC", "234 Cedar Ave", "San Francisco", "CA", "94101");

Restaurant[] restaurants = {restaurant1, restaurant2, restaurant3, restaurant4, restaurant5, restaurant6};

for (int i = 0; i < restaurants.length; i++) {

System.out.println("Restaurant Details:");

System.out.println("ID: " + restaurants[i].getRestaurantID());

System.out.println("Name: " + restaurants[i].getName());

System.out.println("Address: " + restaurants[i].getAddress());

System.out.println("City: " + restaurants[i].getCity());

System.out.println("State: " + restaurants[i].getState());

System.out.println("Zip Code: " + restaurants[i].getZipCode());

System.out.println();

}

System.out.print("---------CUSTOMER PART-------\n");

Customer customer1 = new Customer(1, "John", "Doe", "[johndoe@example.com](mailto:johndoe@example.com)", "123-456-7890", "123 Main St", "New York", "NY", "10001");

Customer customer2 = new Customer(2, "Jane", "Smith", "[janesmith@example.com](mailto:janesmith@example.com)", "987-654-3210", "456 Elm St", "Los Angeles", "CA", "90001");

Customer customer3 = new Customer(3, "Bob", "Johnson", "[bobjohnson@example.com](mailto:bobjohnson@example.com)", "555-123-4567", "789 Oak St", "Chicago", "IL", "60601");

Customer customer4 = new Customer(4, "Alice", "Williams", "[alicewilliams@example.com](mailto:alicewilliams@example.com)", "321-987-6543", "321 Maple St", "Houston", "TX", "77001");

Customer customer5 = new Customer(5, "Mike", "Davis", "[mikedavis@example.com](mailto:mikedavis@example.com)", "901-555-1234", "901 Pine St", "Philadelphia", "PA", "19101");

Customer[] customers = {customer1, customer2, customer3, customer4, customer5};

for (int i = 0; i < customers.length; i++) {

customers[i].display();

}

System.out.print("---------ORDER PART-------\n");

Order order1 = new Order(1, 1, 1, new Date(), 29.99);

Order order2 = new Order(2, 2, 2, new Date(), 15.99);

Order order3 = new Order(3, 3, 3, new Date(), 9.99);

Order order4 = new Order(4, 4, 4, new Date(), 19.99);

Order order5 = new Order(5, 5, 5, new Date(), 12.99);

Order[] orders = {order1, order2, order3, order4, order5};

for (int i = 0; i < orders.length; i++) {

orders[i].display();

//orders[i] is used to access the Order object at index i in the orders array.//

System.out.print("---------Menu PART-------\n");

Menu menu1 = new Menu(1, 1, "Cheeseburger", 5.99);

Menu menu2 = new Menu(2, 1, "Veggie Burger", 6.99);

Menu menu3 = new Menu(3, 2, "Pepperoni Pizza", 8.99);

Menu menu4 = new Menu(4, 2, "Cheese Pizza", 7.99);

Menu menu5 = new Menu(5, 3, "Big Mac", 4.99);

Menu menu6 = new Menu(6, 3, "McNuggets", 3.99);

Menu[] menus = {menu1, menu2, menu3, menu4, menu5, menu6};

for (int j = 0; j < menus.length; j++) {

menus[j].display();

// System.out.println("Driver Details: " +

// "ID: " + driver.getDriverID() +

// ", First Name: " + driver.getFirstName() +

// ", Last Name: " + driver.getLastName() +

// ", Email: " + driver.getEmail() +

// ", Phone: " + driver.getPhone() +for one driver1 details

// ", License Number: " + driver.getLicenseNumber());

// }for this we need to follows above comments

// }

System.out.print("---------Driver PART-------\n");

Driver driver1 = new Driver(1, "John", "Doe", "[john.doe@example.com](mailto:john.doe@example.com)", "555-1234", "D12345678");

Driver driver2 = new Driver(2, "Jane", "Smith", "[jane.smith@example.com](mailto:jane.smith@example.com)", "555-5678", "D87654321");

Driver driver3 = new Driver(3, "Jim", "Brown", "[jim.brown@example.com](mailto:jim.brown@example.com)", "555-8765", "D45678901");

Driver driver4 = new Driver(4, "Jake", "White", "[jake.white@example.com](mailto:jake.white@example.com)", "555-4321", "D98765432");

Driver driver5 = new Driver(5, "Jill", "Green", "[jill.green@example.com](mailto:jill.green@example.com)", "555-6789", "D12340987");

Driver[] drivers = {driver1, driver2, driver3, driver4, driver5};

for (int k = 0; k < drivers.length; k++) {

drivers[k].display();

}

}

System.out.print("---------Delivery PART-------\n");

Delivery delivery1 = new Delivery(1, 1, 1, new Date(), "10:30 AM");

Delivery delivery2 = new Delivery(2, 2, 2, new Date(), "11:00 AM");

Delivery delivery3 = new Delivery(3, 3, 3, new Date(), "11:30 AM");

Delivery delivery4 = new Delivery(4, 4, 4, new Date(), "12:00 PM");

Delivery delivery5 = new Delivery(5, 5, 5, new Date(), "12:30 PM");

// Storing Delivery instances in an array

Delivery[] deliveries = {delivery1, delivery2, delivery3, delivery4, delivery5};

for (int k = 0; k < deliveries.length; k++) {

deliveries[k].display();

}

System.out.print("---------Payment PART-------\n");

Payment payment1 = new Payment(1, 1, "Credit Card", new Date(), 29.99);

Payment payment2 = new Payment(2, 2, "PayPal", new Date(), 19.99);

Payment payment3 = new Payment(3, 3, "Cash", new Date(), 24.99);

Payment payment4 = new Payment(4, 4, "Credit Card", new Date(), 34.99);

Payment payment5 = new Payment(5, 5, "Debit Card", new Date(), 22.99);

Payment[] payment = {payment1,payment2,payment3,payment4,payment5};

for (int k = 0; k < payment.length; k++) {

payment[k].display();

}

Rating rating1 = new Rating(1, 1, "5 stars", "Great service!");

Rating rating2 = new Rating(2, 2, "4 stars", "Good food, but late delivery");

Rating rating3 = new Rating(3, 3, "3 stars", "Average experience");

Rating rating4 = new Rating(4, 4, "5 stars", "Excellent food quality");

Rating rating5 = new Rating(5, 5, "2 stars", "Poor customer service");

// Displaying the details of each rating

System.out.println("Rating Details:");

rating1.display();

rating2.display();

rating3.display();

rating4.display();

rating5.display();

}

}

}

**Customer code:**

package fooddelivery;

public class Customer {

private int customerID;

private String firstName;

private String lastName;

private String email;

private String phone;

private String address;

private String city;

private String state;

private String zipCode;

// Constructors

public Customer() {}

public Customer(int customerID, String firstName, String lastName, String email, String phone, String address, String city, String state, String zipCode) {

this.customerID = customerID;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phone = phone;

this.address = address;

this.city = city;

this.state = state;

this.zipCode = zipCode;

}

public int getCustomerID() {

return customerID;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getEmail() {

return email;

}

public String getPhone() {

return phone;

}

public String getAddress() {

return address;

}

public String getCity() {

return city;

}

public String getState() {

return state;

}

public String getZipCode() {

return zipCode;

}

// Setters

public void setCustomerID(int customerID) {

this.customerID = customerID;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public void setEmail(String email) {

this.email = email;

}

public void setPhone(String phone) {

this.phone = phone;

}

public void setAddress(String address) {

this.address = address;

}

public void setCity(String city) {

this.city = city;

}

public void setState(String state) {

this.state = state;

}

public void setZipCode(String zipCode) {

this.zipCode = zipCode;

}

public void display() {

System.out.println("Customer Details:");

System.out.println("ID: " + customerID);

System.out.println("First Name: " + firstName);

System.out.println("Last Name: " + lastName);

System.out.println("Email: " + email);

System.out.println("Phone: " + phone);

System.out.println("Address: " + address);

System.out.println("City: " + city);

System.out.println("State: " + state);

System.out.println("Zip Code: " + zipCode);

System.out.println();

}

}

**Delivery code:**

package fooddelivery;

import java.util.Date;

public class Delivery {

private int deliveryID;

private int orderID;

private int driverID;

private Date deliveryDate;

private String deliveryTime;

public Delivery(int deliveryID, int orderID, int driverID, Date deliveryDate, String deliveryTime) {

this.deliveryID = deliveryID;

this.orderID = orderID;

this.driverID = driverID;

this.deliveryDate = deliveryDate;

this.deliveryTime = deliveryTime;

}

public int getDeliveryID() {

return deliveryID;

}

public void setDeliveryID(int deliveryID) {

this.deliveryID = deliveryID;

}

public int getOrderID() {

return orderID;

}

public void setOrderID(int orderID) {

this.orderID = orderID;

}

public int getDriverID() {

return driverID;

}

public void setDriverID(int driverID) {

this.driverID = driverID;

}

public Date getDeliveryDate() {

return deliveryDate;

}

public void setDeliveryDate(Date deliveryDate) {

this.deliveryDate = deliveryDate;

}

public String getDeliveryTime() {

return deliveryTime;

}

public void setDeliveryTime(String deliveryTime) {

this.deliveryTime = deliveryTime;

}

public void display() {

System.out.println("Delivery Details: " +

"Delivery ID: " + deliveryID

+

", Order ID: " + orderID

+

", Driver ID: " + driverID

+

", Delivery Date: " + deliveryDate

+

", Delivery Time: " + deliveryTime);

}

}

**Driver code:**

package fooddelivery;

public class Driver {

private int driverID;

private String firstName;

private String lastName;

private String email;

private String phone;

private String licenseNumber;

public Driver(int driverID, String firstName, String lastName, String email, String phone, String licenseNumber) {

this.driverID = driverID;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phone = phone;

this.licenseNumber = licenseNumber;

}

public int getDriverID() {

return driverID;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getEmail() {

return email;

}

public String getPhone() {

return phone;

}

public String getLicenseNumber() {

return licenseNumber;

}

public void setDriverID(int driverID) {

this.driverID = driverID;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public void setEmail(String email) {

this.email = email;

}

public void setPhone(String phone) {

this.phone = phone;

}

public void setLicenseNumber(String licenseNumber) {

this.licenseNumber = licenseNumber;

}

public void display() {

System.out.println("Driver Details:");

System.out.println("ID: " + driverID);

System.out.println("First Name: " + firstName);

System.out.println("Last Name: " + lastName);

System.out.println("Email: " + email);

System.out.println("Phone: " + phone);

System.out.println("License Number: " + licenseNumber);

System.out.println();

}

}

**Menu code:**

package fooddelivery;

public class Menu {

private int menuID;

private int restaurantID;

private String itemName;

private double price;

public Menu(int menuID, int restaurantID, String itemName, double price) {

this.menuID = menuID;

this.restaurantID = restaurantID;

this.itemName = itemName;

this.price = price;

}

// Getters

public int getMenuID() {

return menuID;

}

public int getRestaurantID() {

return restaurantID;

}

public String getItemName() {

return itemName;

}

public double getPrice() {

return price;

}

// Setters

public void setMenuID(int menuID) {

this.menuID = menuID;

}

public void setRestaurantID(int restaurantID) {

this.restaurantID = restaurantID;

}

public void setItemName(String itemName) {

this.itemName = itemName;

}

public void setPrice(double price) {

this.price = price;

}

// Display method

public void display() {

System.out.println("Menu Details:");

System.out.println("Menu ID: " + menuID);

System.out.println("Restaurant ID: " + restaurantID);

System.out.println("Item Name: " + itemName);

System.out.println("Price: $" + price);

System.out.println();

}

}

**Order code:**

package fooddelivery;

import java.util.Date;

public class Order {

private int orderID;

private int customerID;

private int restaurantID;

private Date orderDate;

private double totalAmount;

public Order() {}

public Order(int orderID, int customerID, int restaurantID, Date orderDate, double totalAmount) {

this.orderID = orderID;

this.customerID = customerID;

this.restaurantID = restaurantID;

this.orderDate = orderDate;

this.totalAmount = totalAmount;

}

public void display() {

System.out.println("Order Details:");

System.out.println("Order ID: " + orderID);

System.out.println("Customer ID: " + customerID);

System.out.println("Restaurant ID: " + restaurantID);

System.out.println("Order Date: " + orderDate);

System.out.println("Total Amount: " + totalAmount);

System.out.println();

}

}

**Payment code:**

package fooddelivery;

import java.util.Date;

public class Payment {

private int paymentID;

private int orderID;

private String paymentMethod;

private Date paymentDate;

private double amount;

// Constructor

public Payment(int paymentID, int orderID, String paymentMethod, Date paymentDate, double amount) {

this.paymentID = paymentID;

this.orderID = orderID;

this.paymentMethod = paymentMethod;

this.paymentDate = paymentDate;

this.amount = amount;

}

// Getters and setters

public int getPaymentID() {

return paymentID;

}

public void setPaymentID(int paymentID) {

this.paymentID = paymentID;

}

public int getOrderID() {

return orderID;

}

public void setOrderID(int orderID) {

this.orderID = orderID;

}

public String getPaymentMethod() {

return paymentMethod;

}

public void setPaymentMethod(String paymentMethod) {

this.paymentMethod = paymentMethod;

}

public Date getPaymentDate() {

return paymentDate;

}

public void setPaymentDate(Date paymentDate) {

this.paymentDate = paymentDate;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public void display() {

System.out.println("Payment Details:");

System.out.println("Payment ID: " + paymentID);

System.out.println("Order ID: " + orderID);

System.out.println("Payment Method: " + paymentMethod);

System.out.println("Payment Date: " + paymentDate);

System.out.println("Amount: $" + amount);

System.out.println();

}

}

**Rating code:**

package fooddelivery;

public class Rating {

private int ratingID;

private int orderID;

private String rating;

private String review;

// Constructor

public Rating(int ratingID, int orderID, String rating, String review) {

this.ratingID = ratingID;

this.orderID = orderID;

this.rating = rating;

this.review = review;

}

// Getters

public int getRatingID() {

return ratingID;

}

public int getOrderID() {

return orderID;

}

public String getRating() {

return rating;

}

public String getReview() {

return review;

}

// Setters

public void setRatingID(int ratingID) {

this.ratingID = ratingID;

}

public void setOrderID(int orderID) {

this.orderID = orderID;

}

public void setRating(String rating) {

this.rating = rating;

}

public void setReview(String review) {

this.review = review;

}

// Display method

public void display() {

System.out.println("Rating Details:");

System.out.println("Rating ID: " + ratingID);

System.out.println("Order ID: " + orderID);

System.out.println("Rating: " + rating);

System.out.println("Review: " + review);

System.out.println();

}

}

**Restaurant code:**

package fooddelivery;

public class Restaurant {

private int restaurantID;//instance variable

private String name;

private String address;

private String city;

private String state;

private String zipCode;

public Restaurant(int restaurantID, String name, String address, String city, String state, String zipCode) {

this.restaurantID = restaurantID;

this.name = name;

this.address = address;

this.city = city;

this.state = state;

this.zipCode = zipCode;

}

// Getters and setters

public int getRestaurantID() {

return restaurantID;

}

public void setRestaurantID(int restaurantID) {//int restaurantID local variable

this.restaurantID = restaurantID;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getZipCode() {

return zipCode;

}

public void setZipCode(String zipCode) {

this.zipCode = zipCode;

}

public void display() {

System.out.println("Restaurant Details:");

System.out.println("ID: " + restaurantID);

System.out.println("Name: " + name);

System.out.println("Address: " + address);

System.out.println("City: " + city);

System.out.println("State: " + state);

System.out.println("Zip Code: " + zipCode);

System.out.println();

}

}

CHALLENGES LIST:

### **Challenges in Creating UML Diagrams:**

**Abstraction and Detail Balance**: Striking the right balance between abstraction and detail can be challenging. Too much detail can clutter the diagram, while too little can obscure its meaning.

**Complex Relationships**: Representing complex relationships between classes, such as many-to-many associations or inheritance hierarchies, can sometimes be challenging to visualize clearly.

**Changing Requirements**: UML diagrams should ideally reflect the current state of the system. However, as requirements change, updating the diagrams to reflect these changes can be time-consuming.

**Tool Limitations**: Some UML modeling tools may have limitations in representing certain aspects of the system or may not support all UML diagram types.

**Consistency**: Ensuring consistency across multiple diagrams and within the same diagram (e.g., consistent naming conventions, styles, and notations) can be challenging, especially in large projects.

**Challenges in Writing Java Classes:**

**Understanding Requirements**: Clearly understanding the requirements of the system and translating them into an appropriate class design can be challenging, especially if the requirements are vague or constantly changing.

**Design Patterns**: Choosing the right design patterns and applying them effectively to solve design problems can be challenging, especially for developers with limited experience.

**Encapsulation and Cohesion**: Ensuring that classes are well-encapsulated and have high cohesion while maintaining low coupling can be challenging, especially in complex systems.

**Performance Considerations**: Designing classes that are efficient in terms of memory usage and runtime performance can be challenging, especially for resource-intensive applications.

**Testing and Debugging**: Writing classes that are easy to test and debug can be challenging, especially if the class has complex interactions with other classes or external dependencies.

### **Challenges in Designing ER Diagrams:**

**Understanding the Domain**: Clearly understanding the domain and its entities, attributes, and relationships can be challenging, especially in complex domains or domains that are unfamiliar to the designer.

**Normalization**: Ensuring that the database schema is normalized to eliminate redundancy and minimize update anomalies can be challenging, especially for designers with limited experience in database design.

**Performance Optimization**: Designing an efficient database schema that minimizes the number of joins and maximizes query performance can be challenging, especially for large-scale databases.

**Concurrency Control**: Designing a database schema that supports concurrent access and ensures data consistency under concurrent updates can be challenging, especially for databases with high concurrency requirements.

**Data Integrity**: Ensuring data integrity constraints (such as referential integrity, entity integrity, and domain integrity) are enforced correctly in the database schema can be challenging, especially in complex data models.