



# **INFINITE DELIVERY SERVICE SYSTEM GROUP NO-05**

**T.Chankavi - EC/2021/079**

**V.Mathivany - EC/2021/072**

**A.Siboshan - EC/2021/067**

**S.Ganamoorthy - EC/2021/063**

**C.Prinshiga - EC/2021/045**

## **Group No-05**

## **Group Members**

	<b>Student Number</b>	<b>Student Name</b>
<b>01</b>	<b>EC/2021/079</b>	<b>T.Chankavi</b>
<b>02</b>	<b>EC/2021/072</b>	<b>V.Mathivany</b>
<b>03</b>	<b>EC/2021/063</b>	<b>S.Ganamoorthy</b>
<b>04</b>	<b>EC/2021/067</b>	<b>A.Siboshan</b>
<b>05</b>	<b>EC/2021/045</b>	<b>C.Prinshiga</b>

## *Introduction*

In the modern economy, delivery services are essential due to the growth of e-commerce and on-demand services. As a project, developing a delivery service system enables us to work on a solution that has real-world applications. User interfaces for both clients and delivery staff are part of a delivery service system. This provides the chance to focus on the customer experience, making sure the system is simple to use, satisfies the needs of many clients, and is user-friendly. In conclusion, working on a delivery service system project in Java offers a wealth of educational opportunities that include a variety of theoretical and practical topics relevant to current software development.

A delivery service system can be used in various situations where the transportation and delivery of goods or services are involved.

### **Some common use cases are:**

- Customers place orders online, and the delivery system ensures the efficient dispatch and delivery of products to the specified locations.
- Restaurants receive orders through the system, and delivery personnel are assigned to pick up and deliver food to customers.
- Companies offering courier services can use the system to manage the transportation and delivery of parcels from one location to another.
- Any local service that involves delivering goods, such as groceries, flowers, or hardware supplies, can benefit from a well-designed delivery service system.

## Goals and Objectives

### Goals and Objectives for a Delivery Service System

#### 1) User Authentication and Management:

- Goal: Ensure secure access to the system for users.
- Objective: Implement a user authentication system to verify the identity of users and manage their access to the system.

#### 2) Customer Ordering Process:

- Goal: Facilitate customers in placing orders for delivery.
- Objective: Allow customers to log in, provide personal information, and place orders for items, generating unique order numbers.

#### 3) Order Placement Process:

- Goal: Capture and store detailed information about customer orders.
- Objective: Allow users to input item details, quantity, and delivery information, creating an order object with a unique order number.

#### 4) Courier Selection Process:

- Goal: Assign a courier for the delivery of orders.
- Objective: Provide a list of available couriers, allow users to select a courier by ID, and display details of the selected courier.

#### 5) Charge Calculation:

- Goal: Determine delivery charges based on specified criteria.
- Objective: Calculate charges considering factors like weight, distance, and fixed charges, and display the breakdown of charges to the user.

#### 6) Payment Process:

- Goal: Enable customers to make payments for their orders.
- Objective: Implement different payment methods (cash on delivery, online payment), provide bank details for online payments, and guide users through the payment process.

#### 7) Real-Time Access Display:

- Goal: Provide users with real-time access information.
- Objective: Display a message indicating the user's login time, serving as a dummy implementation for real-time access information.

## 8) Order Number Generation:

- Goal: Generate unique order numbers for each order.
- Objective: Develop a mechanism to create unique order numbers for tracking and identification purposes.

## 9) Courier Details Display:

- Goal: Inform customers about the selected courier for their order.
- Objective: Display details of the selected courier, including name, ID, and contact number.

## 10) System Exit:

- Goal: Allow users to gracefully exit the system.
- Objective: Implement an option to exit the system when the user chooses to do so.

## *The primary target audience for a online delivery service system.*

### Customers:

- Individuals or businesses looking to order goods or services for delivery.

### Couriers/Delivery Personnel:

- Individuals responsible for picking up and delivering orders to customers.

### Administrators/Managers:

- Those managing the online delivery service platform, overseeing orders, and maintaining the system.

### System Operators:

- Individuals responsible for the day-to-day operation and maintenance of the delivery system.

# classes

1.USER CLASS

2.CUSTOMER CLASS

3.ORDER CLASS

4.COURIER CLASS

5.CHARGE CLASS

6.PAYMENT METHOD CLASS

7.MAIN CLASS

## *Attributes and Functions in every class*

### **1. \*User Class\*:**

#### *- Attributes:*

- username: Stores the username of the user.
- password: Stores the password of the user.

#### *- Functions:*

- User(String username, String password): Constructor to initialize username and password.
- authenticate(String enteredUsername, String enteredPassword): Method to authenticate the user based on entered username and password.

### **2. \*Customer Class\* (inherits from User):**

#### *- Attributes:*

- Inherits username and password from the User class.
- name: Stores the name of the customer.
- address: Stores the address of the customer.
- mobileNumber: Stores the mobile number of the customer.

#### *- Functions:*

- Customer(String username, String password, String name, String address, String mobileNumber): Constructor to initialize customer data.
- Getter methods for name, address, and mobileNumber.

### 3. **\*Order Class\*:**

- *Attributes:*

- orderNumber: Stores the order number.
- itemName: Stores the name of the item in the order.
- itemDetails: Stores details about the item.
- quantity: Stores the quantity of the item.
- deliveryPersonName: Stores the name of the delivery person.
- deliveryAddress: Stores the delivery address.
- deliveryPersonContact: Stores the contact number of the delivery person.

- *Functions:*

- Order(...): Constructor to initialize order details.
- Getter methods for various attributes.

### 4. **\*Courier Class\*:**

- *Attributes:*

- name: Stores the name of the courier person.
- id: Stores the ID of the courier person.
- contactNumber: Stores the contact number of the courier person.

- *Functions:*

- Courier(String name, int id, String contactNumber): Constructor to initialize courier details.
- Getter methods for attributes.



## 5. **\*Charge Class\*:**

### - *Attributes:*

- Constants for various charges like BASE\_CHARGE, MINIMUM\_WEIGHT\_CHARGE, etc.

- distances: Array storing distances to Colpetty for each town.

### - *Functions:*

- calculateCharge(double netWeight, int selectedTown): Calculates the delivery charge based on weight and distance.

## 6. **\*PaymentMethod Class\*:**

### - *Attributes:*

- method: Stores the payment method.

### - *Functions:*

- PaymentMethod(String method): Constructor to initialize payment method.

- Getter and setter methods for method.

## 7. **\*Main Class\*:**

- Contains the main method to run the program.

- Handles user interaction through console input.

- Contains methods to handle customer ordering, order placement, courier selection, charge calculation, and payment process.

## *OOP Principles*

### **Encapsulation:**

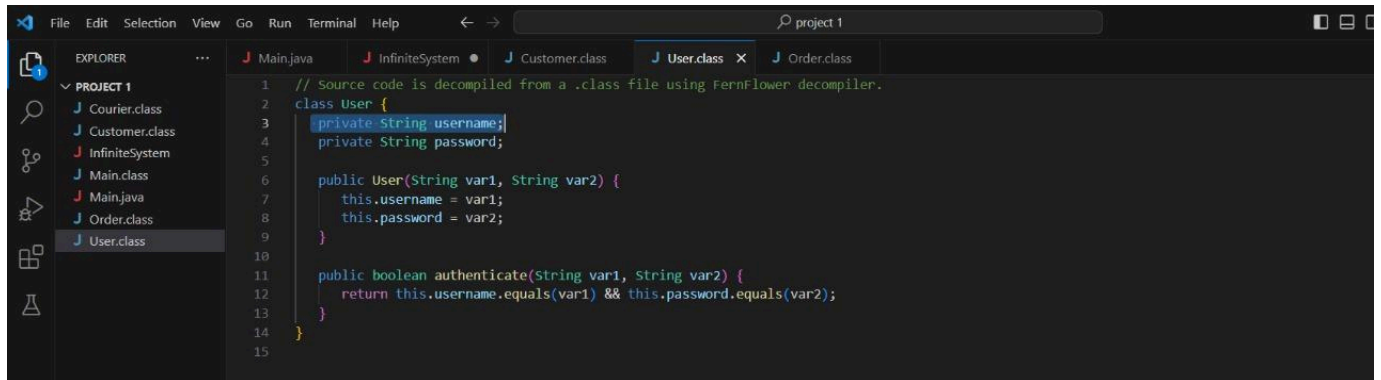
In our system, encapsulation serves as a powerful tool for organizing and securing our data. The classes User, Customer, Order, Courier, Charge, and PaymentMethod employ private instance variables, shielding their internal details. Through carefully crafted public methods, known as getters and setters, we provide controlled access to these variables. This encapsulation strategy not only enhances data security but also fosters a clean and manageable interface for interacting with our objects. By encapsulating the intricacies within our classes, we improve code maintainability and minimize the risk of unintended interference.

### **Inheritance:**

Building upon the foundation of encapsulation, our system leverages the concept of inheritance. Specifically, the Customer class extends the User class, establishing an "is-a" relationship, signifying that a customer is a type of user. This elegant mechanism promotes code reuse by allowing the Customer class to inherit the properties and methods of the User class while still maintaining its distinct attributes and functionalities. Inheritance enhances our codebase's scalability and maintainability by avoiding redundancy and encouraging a modular approach to design.

# THE CODING OF OOP PRINCIPLES PART OF OUR CODING

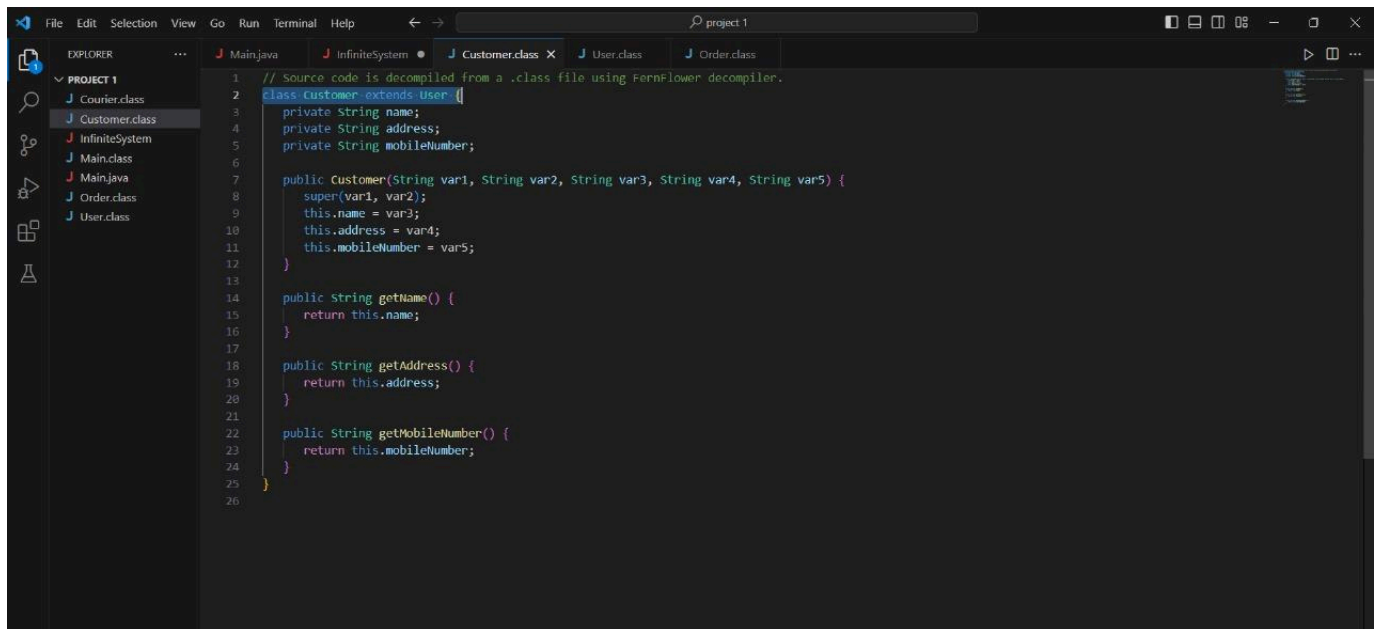
## Encapsulation



The screenshot shows an IDE window with the 'User.class' file open. The Explorer panel on the left lists several files: Courier.class, Customer.class, InfiniteSystem, Main.class, Main.java, Order.class, and User.class. The main editor area displays the following Java code for the User class:

```
1 // Source code is decompiled from a .class file using FernFlower decompiler.
2 class User {
3     private String username;
4     private String password;
5
6     public User(String var1, String var2) {
7         this.username = var1;
8         this.password = var2;
9     }
10
11     public boolean authenticate(String var1, String var2) {
12         return this.username.equals(var1) && this.password.equals(var2);
13     }
14 }
15
```

## Inheritance



The screenshot shows the same IDE window, but now the 'Customer.class' file is open. The Explorer panel on the left shows the same file list. The main editor area displays the following Java code for the Customer class, which inherits from the User class:

```
1 // Source code is decompiled from a .class file using FernFlower decompiler.
2 class Customer extends User {
3     private String name;
4     private String address;
5     private String mobileNumber;
6
7     public Customer(String var1, String var2, String var3, String var4, String var5) {
8         super(var1, var2);
9         this.name = var3;
10        this.address = var4;
11        this.mobileNumber = var5;
12    }
13
14    public String getName() {
15        return this.name;
16    }
17
18    public String getAddress() {
19        return this.address;
20    }
21
22    public String getMobileNumber() {
23        return this.mobileNumber;
24    }
25 }
26
```

## Evaluation

The provided Java program implements a simple online delivery system. Here's an evaluation of the program:

The code is well-organized with clear class definitions for User, Customer, Order, Courier, Charge, and PaymentMethod.

Proper use of inheritance is employed to create a Customer class extending the User class.

Constants are appropriately defined for charge calculations.

The program covers various aspects of the delivery system, including customer authentication, order placement, courier selection, charge calculation, and payment processing.

It uses a simple set of predefined usernames and passwords for authentication, and the credentials are checked against this set.

Dummy data is provided for order details, courier selection, and payment information, making it suitable for a demonstration.

The program handles incorrect username or password entries during the customer authentication process, prompting the user to try again.

There's a default case in the switch statement to handle invalid menu choices.

Data such as distances and courier details are represented using arrays and switch statements, providing an organized way to store and access this information.

The Charge class is well-implemented to calculate delivery charges based on weight, distance, and predefined rates.

The charge calculation is well-documented with comments, making it easy to understand.

Two payment methods (cash on delivery and online payment) are implemented, and the user is given instructions based on their choice.

A dummy method (displayRealTimeAccess()) is included to simulate

real-time access logging, demonstrating potential for further enhancements in logging functionality.

Certain methods, such as `displayRealTimeAccess()`, `generateOrderNumber()`, and `displayCourierDetails()`, are marked as dummy implementations, indicating that these could be further developed for a production-ready system.

The program could benefit from incorporating exception handling to enhance error management.

Validation of user inputs could be added for robustness.

Consider using more advanced data structures, such as `HashMap`, for storing and retrieving user credentials.

Enhance the program to support a dynamic list of couriers and towns, making it more adaptable for different scenarios.

### *Modifications what we planned*

#### **Modifications**

Instead of using a hardcoded array for usernames and passwords, consider implementing a more secure authentication mechanism. A database allows for better management of user information and provides a more robust and secure authentication process.

Implement a proper payment processing system with third-party APIs or libraries for handling real transactions. Real-world applications require secure and reliable payment processing.

Enhance the user interface with clear prompts, instructions, and a more user-friendly design. Improving the user interface enhances the overall user experience and makes the system more accessible and user-friendly.

Implement security measures such as input validation, secure communication, and authorization to protect against common security threats. Security is crucial in online systems.

## *The Challenges what we faced*

### **Challenges**

Challenge: Handling concurrent access and data consistency.

Solution: Implement proper synchronization mechanisms or consider using a database management system that handles concurrent access.

Challenge: Testing the system thoroughly.

Solution: Implement a comprehensive testing strategy, including unit tests, integration tests, and system tests, to identify and resolve potential issues.

Challenge: Ensuring system scalability.

Solution: Design the system with scalability in mind, such as using modular and efficient algorithms and considering potential bottlenecks.

## *Conclusion*

In this Java-based delivery service system, users can seamlessly interact with an online platform to initiate and manage the delivery process. Customers can place orders, authenticate their identity, and provide necessary information for delivery. The system incorporates features such as order creation, courier selection, charge calculation based on weight and distance, and payment processing with options like cash on delivery or online payment. Real-time access and order details are displayed during the ordering process, providing a user-friendly experience. The system also allows for the selection of courier personnel and calculates charges based on predefined rates. Overall, this delivery service system is designed to offer a convenient, efficient, and secure platform for customers to place and manage their orders while providing essential tools for administrators and operators to oversee and maintain the system.

## Team Work

**The project has been distributed among a team of 5 individuals to ensure efficient collaboration and timely completion. The list of works assumed for each person is like;**

- ★ C.Prinshiga-Assigned to write programs for the customer,order classes.
- ★ V.Mathivany - Reserved to develop code for the courier class.
- ★ A.Siboshan - Distributed delivery class to write coding.
- ★ S.Ganamoorthy - Allowed to develop the payment and delivery status classes.
- ★ T.Chankavi-Allocated to design coding for the delivery service part and to overall troubleshooting.