**Introduction**

Imagine a world where you can browse, select, and purchase any item you desire, all form the comfort of your home. This is the reality of online shopping, a technological marvel that has revolutionized the way we buy and sell goods. But have you ever wondered what goes on behind the scenes of theses digital marketplace? How do these systems manage products, track inventory, and process orders seamlessly?

The Concept of e-commerce has grown exponentially since its inception in the 1990s. What started as simple online catalogues has evolved into sophisticated platforms handling millions of transactions daily. These systems are built on complex software architectures that coordinate various components such as product databases, user accounts, shopping carts, and payment gateways. Understanding the basics of how these systems work is not only fascinating but also crucial in our increasingly digital world.

The Online Shopping System implemented in Java, designed to demonstrate the fundamental concepts and components that form the backbone of e-commerce platforms. By exploring this basic model, we aim to demystify the inner workings of online shopping systems, providing a clear understanding of how products, carts, and customers interact in a digital marketplace. This educational tool serves as a stepping stone for beginners in programming and software design, offering hands-on experience with the essential elements of e-commerce systems while highlighting the potential for future enhancements and complexities in real-world applications.

**Case Analysis**

The rapid growth of e-commerce has created a need for robust, efficient, and user-friendly online

shopping systems. However, developing such systems presents several challenges that need to be addressed.

Many struggle to grasp how different component of an online shopping system interact, from managing product inventories to processing customer orders. Firstly, existing e-commerce solutions are often too complex for educational purposes, making it difficult for beginners to learn and experiment with basic e-commerce concepts. This complexity can be overwhelming and may discourage newcomers from pursuing further studies in this field. Secondly, there’s a gap between theoretical knowledge of programming concepts and their practical application in real-world scenarios like an online shopping systems. Students often find it challenging to translate classroom learning into functional code that simulates actual e-commerce operations. Furthermore, many educational resources fail to provide a comprehensive yet simplified model of an online shopping system that demonstrates core functionalities while remaining accessible to beginners. This lack of suitable learning tools hinders the effective teaching and learning of e-commerce system development. Addressing these challenges requires the development of a carefully designed, beginner-friendly Online Shopping System that can serve as both an educational tool and a foundation for more advanced e-commerce projects.

**Documentation**

The creation of the Online Shopping System highlights the power of teamwork and effective collaboration in achieving a unified goal. By harnessing each team member's unique skills and following a structured development process, we successfully built a program that addresses the diverse needs of online shoppers. This documentation underscores the importance of clear communication, defined roles, and iterative development in delivering a reliable and user-friendly e-commerce solution.

**Brainstorming**

During our brainstorming, we discuss about the development of Online Shopping System (OSS). One of the many reason why we gathered for brainstorming is to distribute roles for this case study, discuss how can we address the potential challenges. We acknowledged that we need to follow the instructions when creating the OSS program. Then we decided created a group chat in messenger so we can then schedule meetings making sure that all of us is available when whenever we decided to have a meeting, and everybody in this group can contribute to this case study.

We all discussed the significance of the real-life examples in this case study. They suggested adding actual situations as examples. One of my teammates brought up the importance of having clear documentation to detail the workings of the system and the interactions between different roles. This will help readers better grasp the intention of the Online Shopping System and how it can advantage organizations. We all reached a consensus that the addition of examples and clear explanations would enhance our case study.

**Pictures**

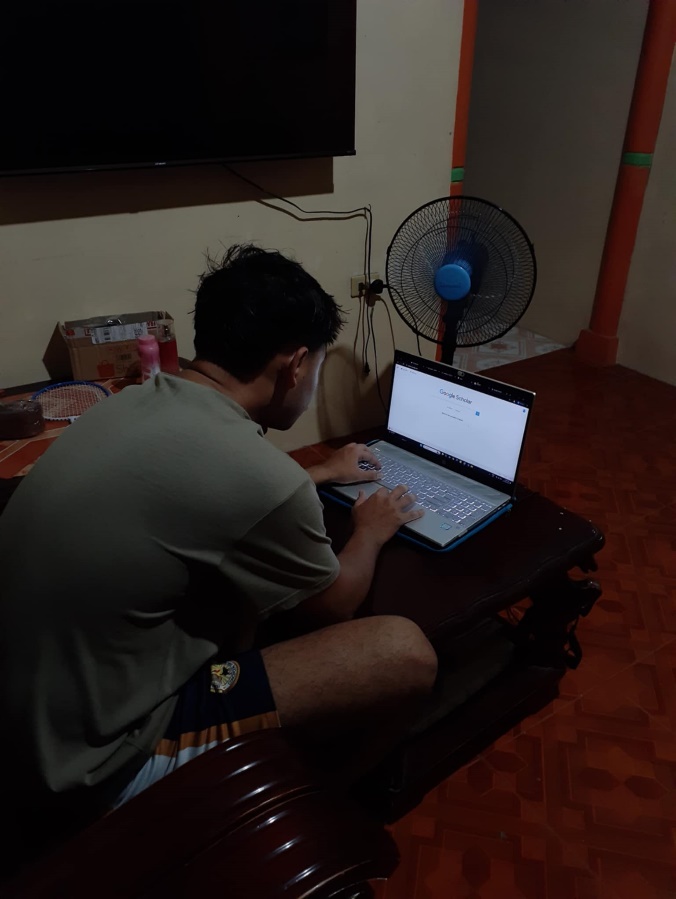


Figure 1. Researching 

Figure 2. starting to program the code.

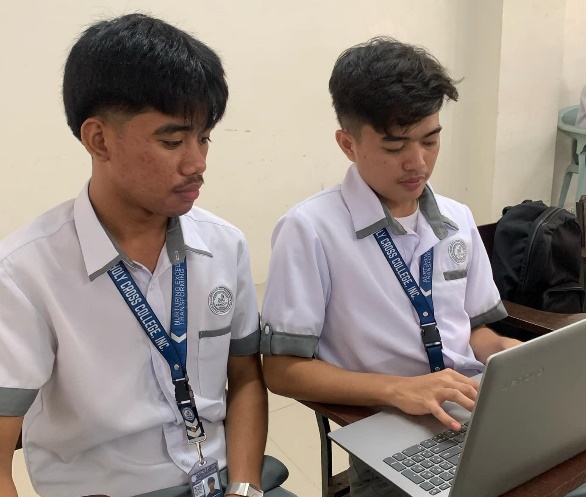
Figure 3. checking any issues with the code. 

Figure 4. Finalizing the code making sure everything works perfectly fine.

**Solution**

This solution uses object-oriented principles in Java, specifically encapsulation and modular design, to model an online shopping system. The Product class encapsulates details about products, such as name, price, and stock level. The Cart class manages a collection of products, allowing items to be added, removed, and cleared. It also calculates the total price of all items in the cart.

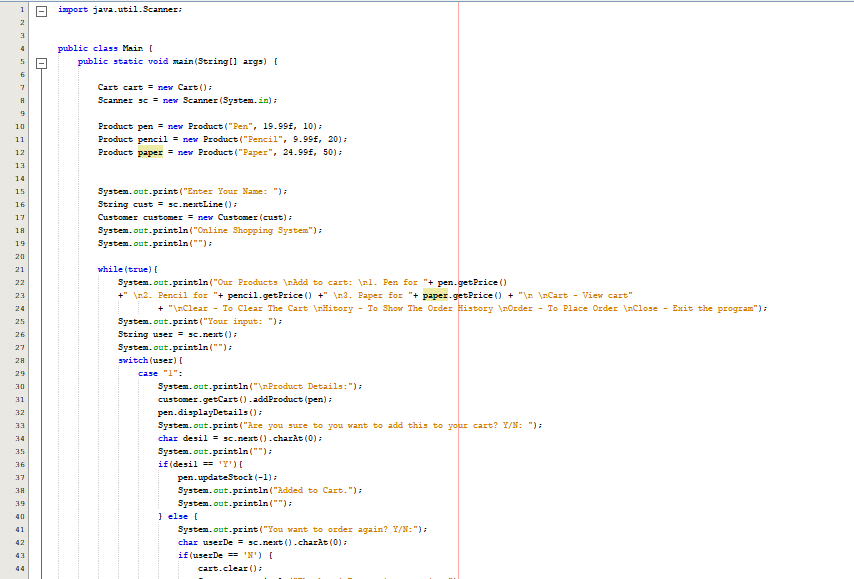
The Customer class holds the customer’s name, a cart, and an order history, enabling them to add products to their cart, view their order history, and place orders. The Customer class has methods to interact with the cart and manage the customer’s shopping experience in a structured way. This separation of responsibilities makes the code easy to maintain and extend.

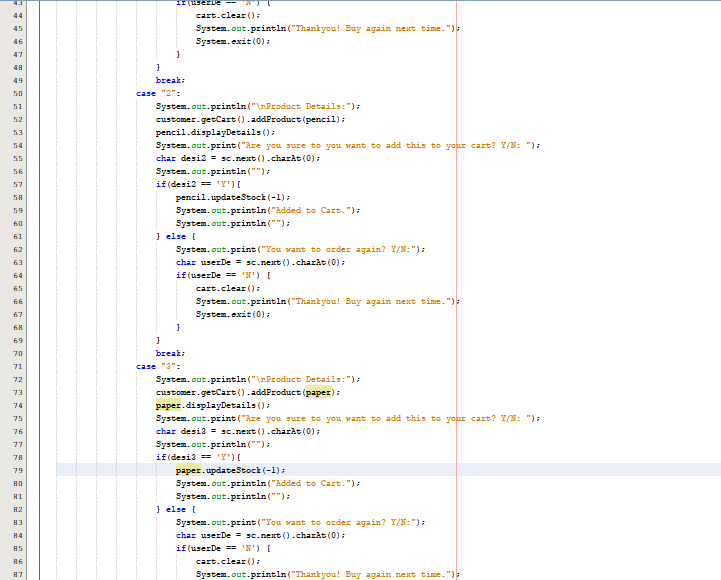
The program uses the Scanner class to take user input, allowing the customer to dynamically add products to their cart, view the cart, place an order, and more. This input-driven approach allows for a dynamic and interactive user experience.

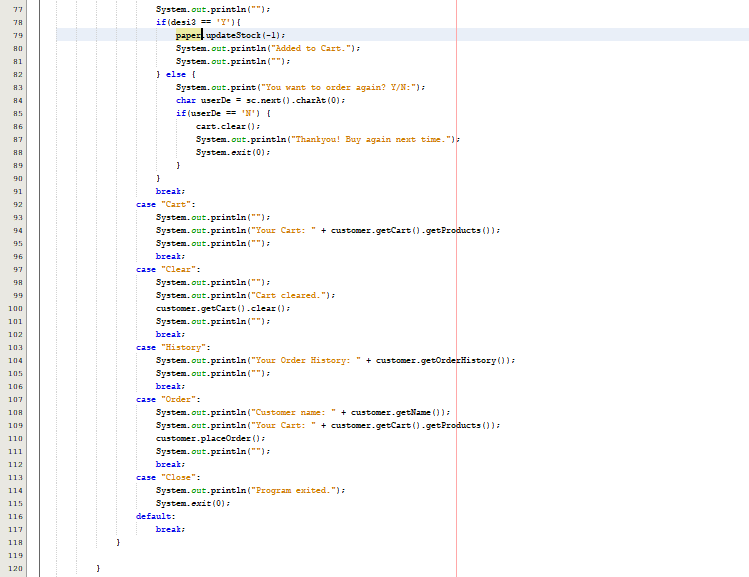
**Screenshot of Input and Output**

**Input**

This the main class, where the creation of product and customer objects.







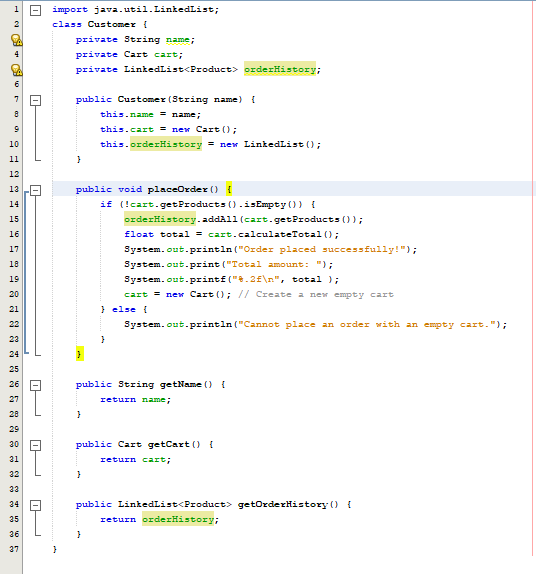
This is the Product class, where the product constructor located, and we had to use Override to address a problem.

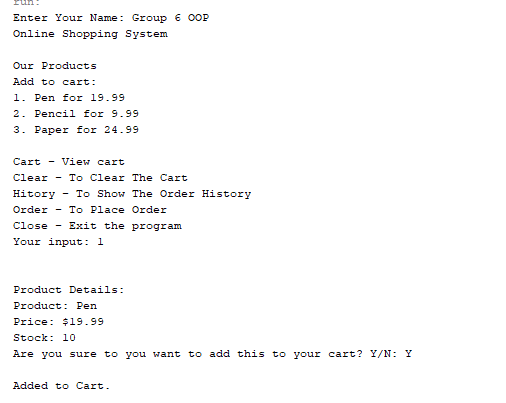


The cart class, each customer can add product to cart and order.

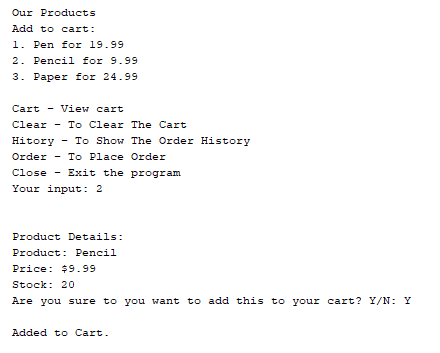


The customer class, where the customer constructor, cart object, and order history located.

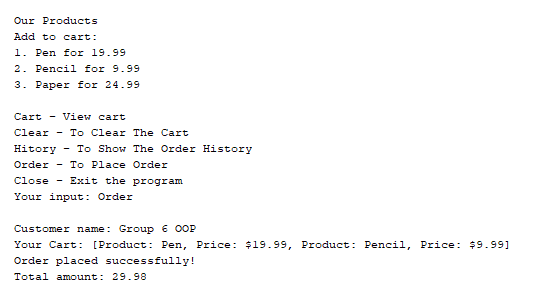


**Output**

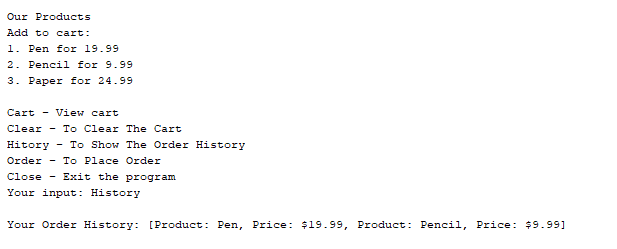
The user added a pen, and the system ask the if the user is sure about adding this product to the cart.



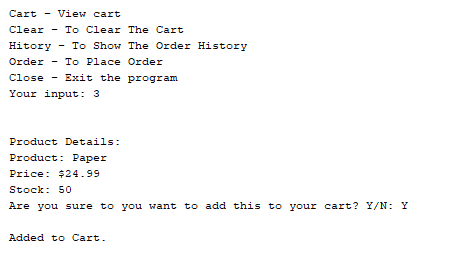
The user added another product which is the pencil, and the system ask again if the user is sure about adding this product to the cart.



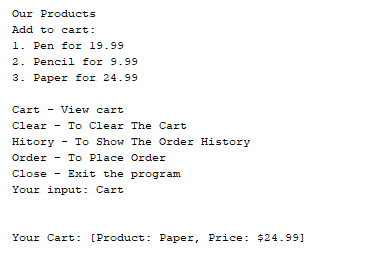
The user ordered the products, in here we can see the total amount.



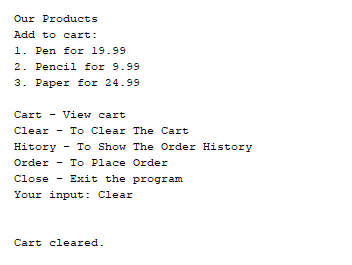
The user wanted to see the order history.



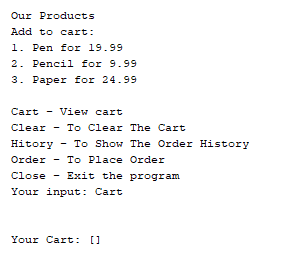
The user added another product (Paper).



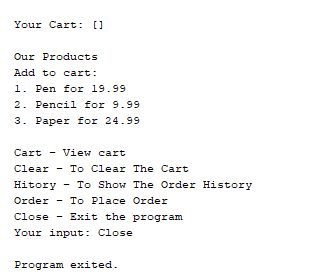
The user viewed the cart.



Then, user wanted to clear the cart.



When the user viewed the cart. The cart is empty.



User proceeds to exit the program

**Conclusion**

The Online Shopping System offers a valuable educational tool for aspiring programmers and software developers interested in e-commerce platforms. By providing a simplified yet functional model of an online store, it successfully bridges the gap between theoretical knowledge and practical application in the field of e-commerce system development.

The system's core components Product, Cart, and Customer classes effectively demonstrate fundamental concepts of object-oriented programming and data structure usage in a real-world context. Through its hands-on approach, the system demystifies the complexities of e-commerce operations, making them accessible to beginners while laying a solid foundation for more advanced studies. The modular design not only facilitates easy understanding but also encourages experimentation, allowing learners to gradually tackle more complex features as their skills improve. While the current implementation is intentionally simple, focusing on console-based interactions and core functionalities, it serves as a springboard for future enhancements.

As e-commerce continues to grow and evolve, tools like this Online Shopping System play a crucial role in preparing the next generation of developers to meet the challenges of building and maintaining robust, efficient, and user-friendly online marketplaces. By providing a clear, interactive model of e-commerce basics, this system not only educates but also inspires, paving the way for innovative solutions in the ever-expanding digital marketplace.

**Reference**

https://www.geeksforgeeks.org/convert-linkedlist-to-string-in-java/

<https://www.infoworld.com/article/2170611/code-reuse-and-object-oriented-systems.html>

Ahmad, M.F. and Khozium, M.O., JAVA BASED IMPLEMENTATION OF AN ONLINE HOME DELIVERY SYSTEM.

https://naveen-metta.medium.com/demystifying-inheritance-in-java-unlocking-the-power-of-code-reusability-e0a88af8e767

**Source Code**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Cart cart = new Cart();

Scanner sc = new Scanner(System.in);

Product pen = new Product("Pen", 19.99f, 10);

Product pencil = new Product("Pencil", 9.99f, 20);

Product paper = new Product("Paper", 24.99f, 50);

System.out.print("Enter Your Name: ");

String cust = sc.nextLine();

Customer customer = new Customer(cust);

System.out.println("Online Shopping System");

System.out.println("");

while(true){

System.out.println("Our Products \nAdd to cart: \n1. Pen for "+ pen.getPrice()

+" \n2. Pencil for "+ pencil.getPrice() +" \n3. Paper for "+ paper.getPrice() + "\n \nCart - View cart \nClear - To Clear The Cart \nHitory - To Show The Order History \nOrder - To Place Order \nClose - Exit the program");

System.out.print("Your input: ");

String user = sc.next();

System.out.println("");

switch(user){

case "1":

System.out.println("\nProduct Details:");

customer.getCart().addProduct(pen);

pen.displayDetails();

System.out.print("Are you sure to you want to add this to your cart? Y/N: ");

char desi1 = sc.next().charAt(0);

System.out.println("");

if(desi1 == 'Y'){

pen.updateStock(-1);

System.out.println("Added to Cart.");

System.out.println("");

} else {

System.out.print("You want to order again? Y/N:");

char userDe = sc.next().charAt(0);

if(userDe == 'N') {

cart.clear();

System.out.println("Thankyou! Buy again next time.");

System.exit(0);

}

}

break;

case "2":

System.out.println("\nProduct Details:");

customer.getCart().addProduct(pencil);

pencil.displayDetails();

System.out.print("Are you sure to you want to add this to your cart? Y/N: ");

char desi2 = sc.next().charAt(0);

System.out.println("");

if(desi2 == 'Y'){

pencil.updateStock(-1);

System.out.println("Added to Cart.");

System.out.println("");

} else {

System.out.print("You want to order again? Y/N:");

char userDe = sc.next().charAt(0);

if(userDe == 'N') {

cart.clear();

System.out.println("Thankyou! Buy again next time.");

System.exit(0);

}

}

break;

case "3":

System.out.println("\nProduct Details:");

customer.getCart().addProduct(paper);

paper.displayDetails();

System.out.print("Are you sure to you want to add this to your cart? Y/N: ");

char desi3 = sc.next().charAt(0);

System.out.println("");

if(desi3 == 'Y'){

paper.updateStock(-1);

System.out.println("Added to Cart.");

System.out.println("");

} else {

System.out.print("You want to order again? Y/N:");

char userDe = sc.next().charAt(0);

if(userDe == 'N') {

cart.clear();

System.out.println("Thankyou! Buy again next time.");

System.exit(0);

}

}

break;

case "Cart":

System.out.println("");

System.out.println("Your Cart: " + customer.getCart().getProducts());

System.out.println("");

break;

case "Clear":

System.out.println("");

System.out.println("Cleared cart.");

customer.getCart().clear();

System.out.println("");

break;

case "History":

System.out.println("Your Order History: " + customer.getOrderHistory());

System.out.println("");

break;

case "Order":

System.out.println("Customer name: " + customer.getName());

System.out.println("Your Cart: " + customer.getCart().getProducts());

customer.placeOrder();

System.out.println("");

break;

case "Close":

System.out.println("Program exited.");

System.exit(0);

default:

break;

}

}

}

}

class Product {

private String productName;

private float price;

private int stock;

public Product(String productName, float price, int stock) {

this.productName = productName;

this.price = price;

this.stock = stock;

}

public void displayDetails() {

System.out.println("Product: " + productName);

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

System.out.println("Stock: " + stock);

}

public void updateStock(int amount) {

this.stock += amount;

}

public String getProductName() {

return productName;

}

public float getPrice() {

return price;

}

public int getStock() {

return stock;

}

@Override

public String toString() {

return "Product: " + productName + ", Price: $" + price;

}

}

import java.util.LinkedList;

class Cart {

private LinkedList<Product> products;

public Cart() {

this.products = new LinkedList<>();

}

public void addProduct(Product product) {

products.add(product);

}

public void removeProduct(Product product) {

products.remove(product);

}

public float calculateTotal() {

float total = 0;

for (Product product : products) {

total += product.getPrice();

}

return total;

}

public LinkedList<Product> getProducts() {

return products;

}

public void clear() {

products.clear();

}

}

import java.util.LinkedList;

class Customer {

private String name;

private Cart cart;

private LinkedList<Product> orderHistory;

public Customer(String name) {

this.name = name;

this.cart = new Cart();

this.orderHistory = new LinkedList();

}

public void placeOrder() {

if (!cart.getProducts().isEmpty()) {

orderHistory.addAll(cart.getProducts());

float total = cart.calculateTotal();

System.out.println("Order placed successfully!");

System.out.print("Total amount: ");

System.out.printf("%.2f\n", total );

cart = new Cart(); // Create a new empty cart

} else {

System.out.println("Cannot place an order with an empty cart.");

}

}

public String getName() {

return name;

}

public Cart getCart() {

return cart;

}

public LinkedList<Product> getOrderHistory() {

return orderHistory;

}

}

**Contributions**

Torres

* Code
* Introduction
* Screenshot of Input and Output
* Conclusion

Maniago

* Documentation
* Brainstorming

De Leon

* Solution
* Case Analysis