Project Report



E commerce management system

**Submitted to:**

Syed Burhan Uddin

**Department of Computer Sciences**

**2025 CS-B**

**Course: Object Oriented Programming**

**Made by:**

* Rasba Mazhar (243571)
* Saleha Khurram (243529)

# **Introduction:**

## **Overview****:**

The E-Commerce Management System is a **console-based application** designed to simulate an online shopping platform. It allows **Admins** to manage products and categories, and **Users** to browse, search, and purchase products and Employee to manage the whole system as well as to deal with process of selection of orders, to view the product and categories and to help the user have a better experience while he initiate order. The system emphasizes **object-oriented programming** and **file-based data storage** (using .txt files).

**Problem Statement:**

The goal of this project is to develop a **console-based E-Commerce Management System using C++** that allows users to browse and search products, and enables admins to manage products and categories.

The system supports **user roles** (Admin, User, Employee) with appropriate access, and uses **file handling** (fstream) for data storage instead of a database. It demonstrates **object-oriented programming** concepts such as classes, encapsulation, and modularity.

The project aims to simulate a simplified e-commerce platform, helping learners understand the structure and logic behind such systems while practicing C++ fundamentals.

**Objectives:**

**User Management**

* Sign Up and Log In for Users, Admins, and Employees
* Input validation for usernames and passwords
* Role-based access control (different menus for each role)

**Product Management**

* Add, update, or delete products (Admin only)
* View all products with details (name, price, category, etc.)
* Search products by name or category

**Category Management**

* Add new product categories
* Update or remove existing categories
* Link products to specific categories

**Search Functionality**

* Search products by partial or full name
* Filter products by category

**File-Based Data Storage**

* User data stored in users.txt, admin.txt, and employee.txt
* Product and category data stored in products.txt and categories.txt
* Uses fstream for persistent storage without a database

**Console Interface**

* Interactive, menu-driven interface
* Clear navigation for all user types
* Feedback messages for actions (success/failure)

**Validation & Security (Basic)**

* Password rules: minimum length, must include a digit
* Prevent empty or duplicate usernames
* Role verification during login

## **Scope:**

### **Included Features:**

### Searching for Product, displaying categories, product and then reviewing the product bought. Admin can delete employee, view them and users and also search for product and the categories. User can order product, view the cart, update the quantity of the cart, cancel order, can remove an item, can make payment, can see the status of the order and can review them. Employee can cancel the order, view the order, check the low stock, update the stock, can add, update and delete Category and products.

### **Limitations:**

## **Text File Storage:** Data is stored in plain .txt files. This is not scalable for large numbers of users, products, or transactions.

## **No Indexing/Search Optimization:** Searching through text files linearly becomes slow as the data grows.

## **Lack of Concurrency Support:** If multiple users access or modify the data at the same time, file corruption or data inconsistency may occur.

* **No Real Payment Integration:** Payment is simulated; there is no connection to real payment gateways (e.g., PayPal, Stripe).
* **Order Lifecycle is Simplified:** There is no shipment tracking, delivery status, or refund process.
* **No Notification System:** Users do not receive email or system alerts about orders, payments, or account changes.
* **Minimal Error Handling:** There's little handling for invalid input, missing files, or failed operations.
* **No Transaction Rollback:** If an error occurs during a file update, there's no rollback mechanism to maintain consistency.
* **Assumes File Integrity:** The system assumes that all input files are correctly formatted and not corrupted.

## **Significance:**

* **Practical Use of OOP Concepts**: The system effectively applies key OOP principles such as **encapsulation**, **inheritance**, and **polymorphism**.
* For example, the base class User is inherited by Admin, Employee, and Customer, showcasing code reuse and extension.
* **File Handling in C++**: Demonstrates how data persistence can be achieved using text files with fstream, simulating a rudimentary database system.
* **Password Masking and Validation**: Introduces beginner-level user input protection and user authentication techniques.
* **Role-Based Access Control**: Different menus and access levels for Admin, Employee, and Customer reinforce the concept of access control in software systems.
* **Product Management**: Adding, displaying, searching, and deleting products and categories.
* **User Registration/Login**: User authentication with password masking and validation.
* **Order and Cart Management**: Users can add products to their cart, place orders, and simulate payments.
* **Review System**: Users can give feedback on products, mimicking real-world review mechanisms.
* **Admin Controls**: Admins can manage users, employees, products, and view sales statistics.

# **Methodology:**

## **Programming Environment:**

* **Integrated Development Environment (IDE)**: Dev C++ 6.3
* **Compiler**: GCC (GNU Compiler Collection)

## **Libraries:**

### **Standard Template Library (STL):**

## **#include <iostream>**

## **Purpose:** This header file is part of the C++ Standard Library and is used for input and output operations.

## **Key Components:**

## **cout**: Used to output data to the console (standard output).

## **cin:** Used to read input from the console (standard input).

## **cerr:** Used to output error messages to the console.

## **#include <fstream>**

## **Purpose**: This header file provides functionalities for file handling in C++. It allows you to read from and write to files.

## **Key Components**:

## **ifstream**: Used for reading from files (input file stream).

## **ofstream**: Used for writing to files (output file stream).

## **fstream**: Used for both reading and writing to files.

## **3. #include <string>**

## **Purpose**: This header file provides support for the **std::string** class, which is used to handle strings in C++.

## **Key Components**:

## **string**: A class that represents a sequence of characters and provides various functions for string manipulation (e.g., concatenation, substring, searching).

## **4. #include <cstdlib>**

## **Purpose**: This header file includes functions for performing general utility functions, such as memory allocation, process control, conversions, and random number generation.

## **Key Components**:

## **system()**: A function that can be used to execute system commands (e.g., **system("CLS")** clears the console screen on Windows).

## **system()**: A function that pauses the program until the user press a key to perform the next task.

## **5. #include <conio.h>**

## **Purpose**: This header file is specific to certain compilers (like Turbo C++ and some versions of Microsoft Visual C++) and provides functions for console input/output.

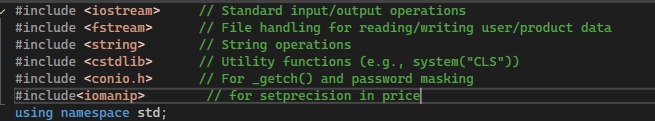
## **Key Components**:

## **\_getch()**: A function that reads a single character from the keyboard without echoing it to the console. This is often used for password input where you don't want the characters to be displayed.

**6.#include<iomanip>**

* **Purpose**: This header file is used in C++ compiler to give precision and accuracy of values**.**
* **Key Componenets:**
  + - **setw():** A function that is used to set width between variables of data during output that is shown on the screen.
    - **Setprecision():** A function used to give precision to the values to give more efficient answers during the output shown.

**Snippet:**



**Classes and Modules:**

**Class**: Masking

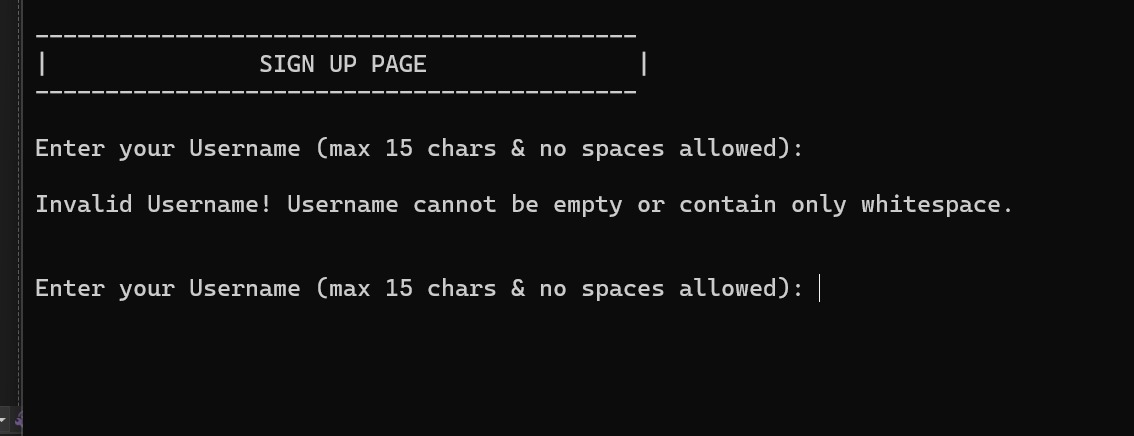
In this class we are making a function called [static string getHiddenPassword() ] which is used in the Login and sign up function to hide the password enter by the user, employee and admin at the time of Login and sign up. In this function **\_getch()** which is used to read every character from the keyboard and show it on the console but in this function only ( \* ) is shown tha hides the password.

**Snippet:**



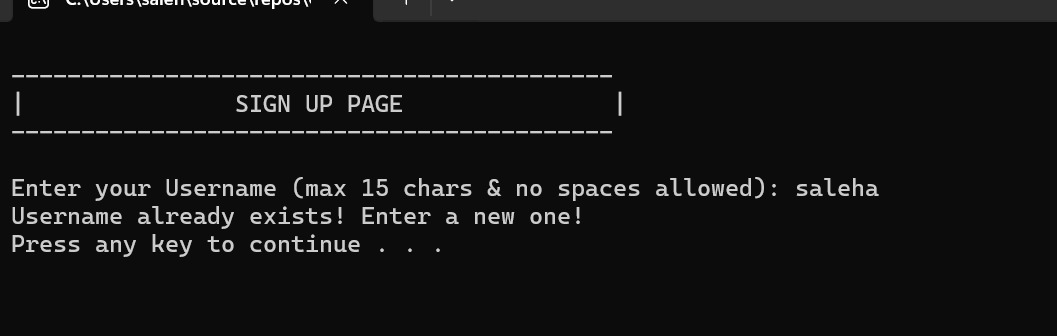
**Class:** Validation

* In Validation class **static bool isEmptyOrWhitespace()** is used to make sure that file is not empty it will return false if the file is empty.

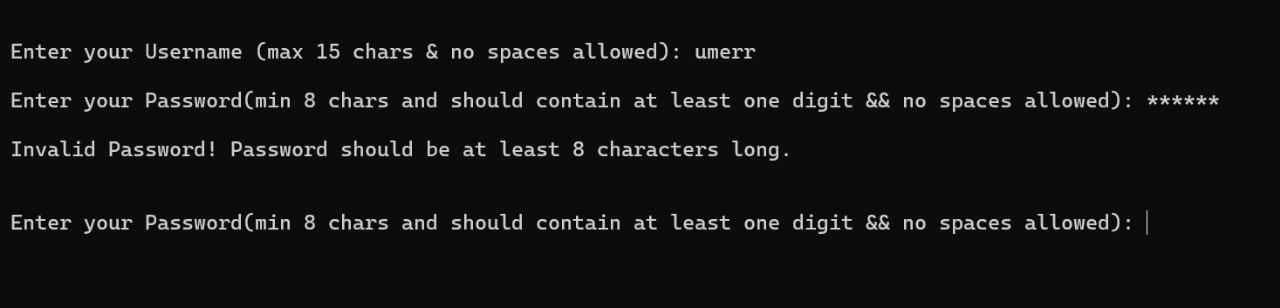


* **static int orderId():** This function is designed to generate and manage a unique order ID.
* **static bool checkproID():** This function is used to validate the input if the same id is entered it should tell the user to give another id for input.
* **static bool containsDigit():** this function is used to mae sure that the password entered by the user has digits in it.
* **static bool validUsername(): This** function limits the username entered to me of not more then of 15 characters.

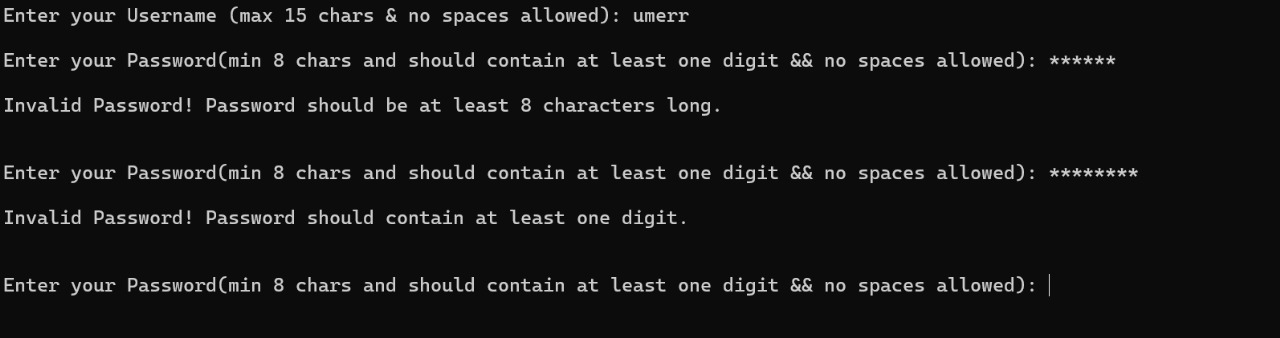
**static bool checkUser():** it make sure that the name entered by the user does not already exits in the file and if it does it shows message and tell to re-enter the username. It also save data based on the roles in different files.



* **static bool validPass():** it makes sure that the password has atleast 8 character and contains atleast one digit.

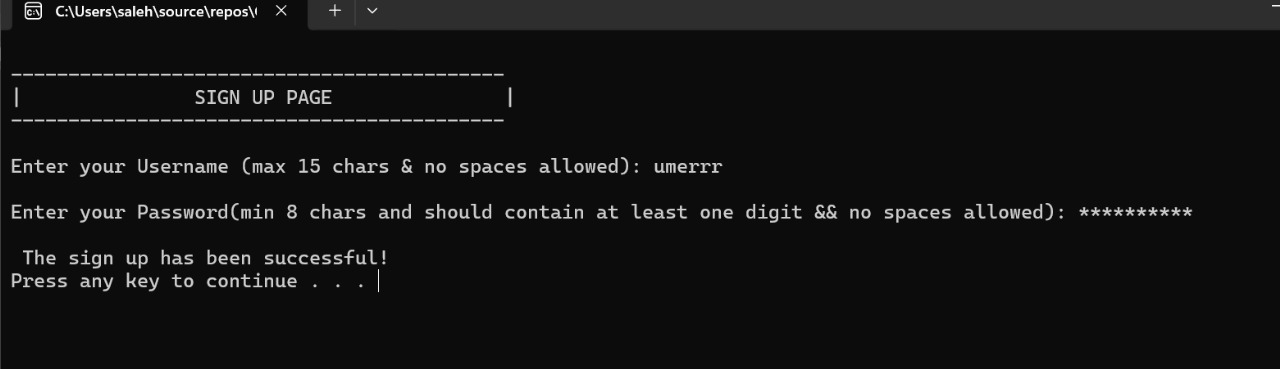


* **static bool validInteger()**: it checks if there is at least a digit in the password in the given input by the user.



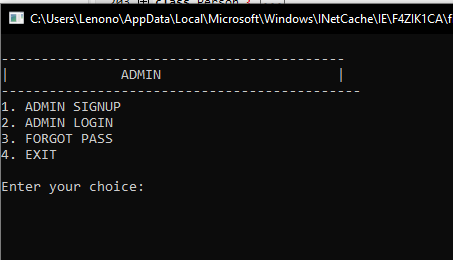
**Class:** Menu

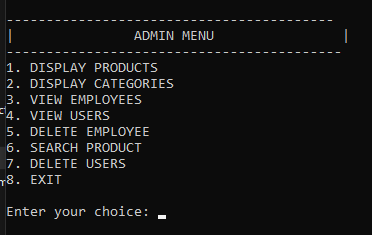
* **static bool menu\_up():** is a menu made for user so they can signup and login by selecting a choice. If the user has forgotten their password they can use forgotten password by chossing it from the menu and can reset their password.



* **static bool start\_up():** It is a menu for Admin and is the same as what is for User.

2

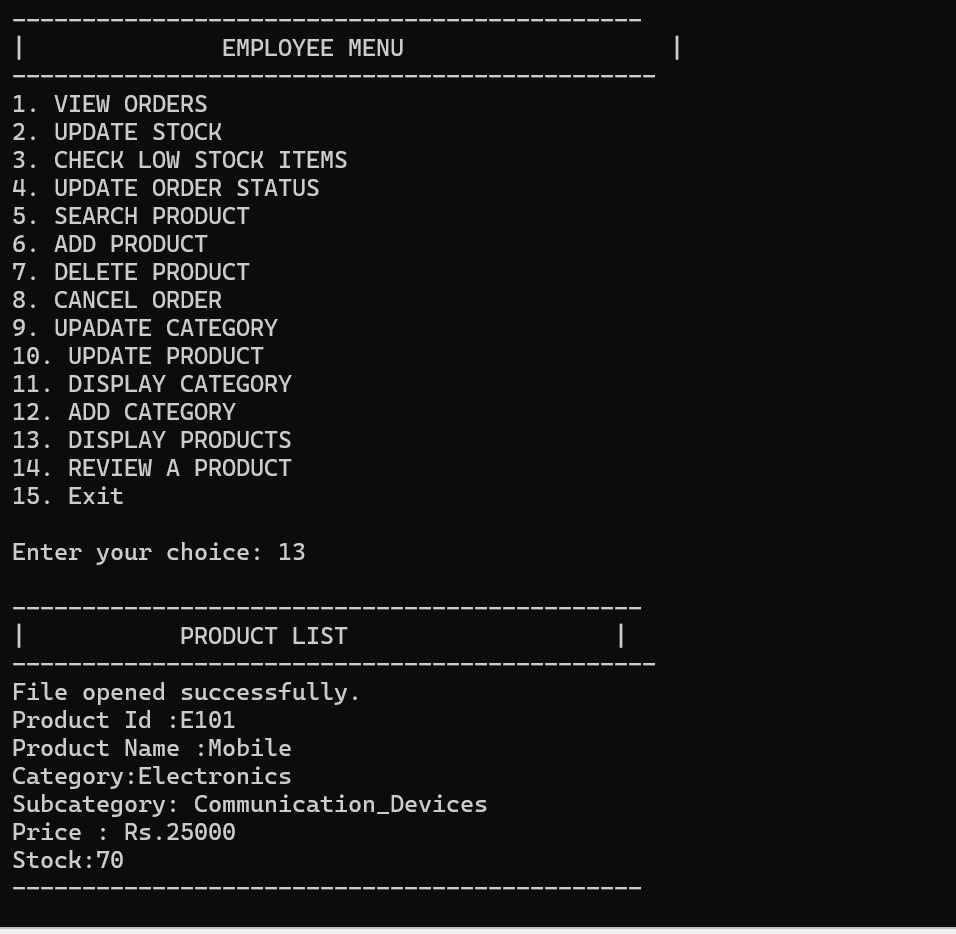


* **static bool empMenu\_up():** It is menu for Employee and has the same options as for user and admin.
* **static void showadminmenu():** It is the menu of the Admin that tell the major functionalities that admin is able to perform in the System.
* 
* **static void showUserMenu():** It is the menu for User and it tell us all the functionalities and access that a user has in the system and how it is improved by users interaction with it.
* **static void empMenu():** It is the menu of Employee that works under Admin and can assist and guide the user to make a better choice. Enployee can also alter the functionalities of user by changing them.

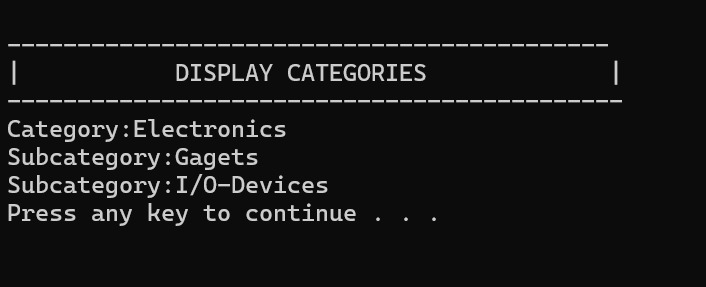
**Class:** Product

In this class, data members are defined as string proID, string name, float price, int stock, string category, string subcategory, float rating; that are used to store the data about the product and the data that is required to save in the file at the time of input and output. A constructor along with setters and getters of the member functions are made so that can be assigned and stored.

* **void displayProducts():** This function is used toread data from the product files and display it on the console.



* **void displayCategories():** This function is used to read data from the category file and display the categories along with their subcategory on the console.



* **void reviewProducts():** Aproduct id is used to matchwith the input id to match from the product file and a review of the product along with its id is written in the review file.
* **void viewReviews():** The review file is used to read reviews of the input id and show it on the console.

**Class:** CartOrder

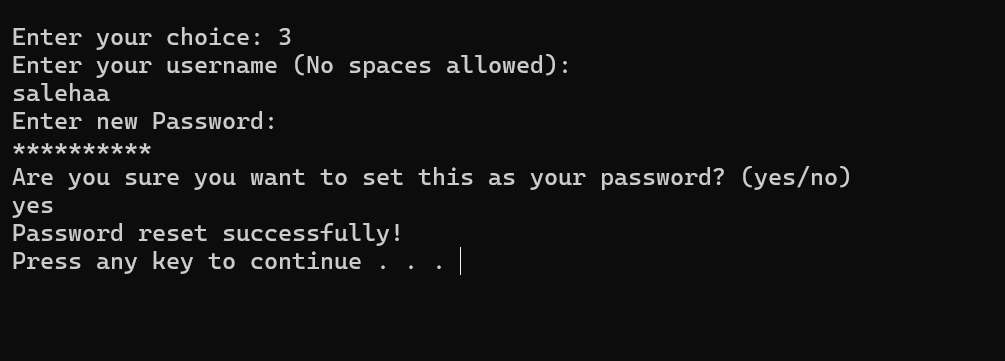
In this class, data member string orderStatus, int orderID, string payStatus are used to store data about the order. A destructor, constructor and setters and getters of the data members are made in the class.

* **void addItem():** The addItem function allows a user to add a product to their shopping cart by reading product details from a file (products.txt) and updating or creating a cart file (cart.txt) accordingly. It checks if the product exists, prompts for the quantity, and either updates the quantity in the cart or adds the product if it doesn't already exist.
* **void viewCart():** The viewCart function displays the contents of the shopping cart by reading from cart.txt and printing each product's details, while notifying the user if the cart is empty.
* **void removeItem():**The removeItem function allows the user to remove a specified product from the cart by reading the cart file, skipping the item to be removed, and writing the remaining items to a temporary file, which is then renamed to update the cart.
* **void updateQuantity():** The updateQuantity function allows the user to change the quantity of a specific product in the cart by reading from cart.txt, updating the quantity if the product is found, and writing the updated cart to a temporary file that replaces the original.
* **void updateOrderStatus():** function updates the status of a specific order in **orders.txt** by reading the file, modifying the status if the order ID matches, and writing the updated information to a temporary file that replaces the original. If the specified order ID is not found, it notifies the user accordingly.
* **void selectItems():** The selectItems function allows the user to choose a specified number of products from their cart to place an order by first displaying the cart contents and then prompting for product IDs. It checks if the selected items exist in the cart, writes the order details to orders.txt, and generates a unique order ID, notifying the user of successful additions or if no valid items were found.
* **void** **viewOrder():**  The function displays all orders by reading from **orders.txt** and printing the details of each order, including the order ID, product information, quantity, order status, and payment status. If the file is successfully opened, it iterates through the orders and formats the output for the user.
* **void** **cancelord():** The function allows the user to cancel an order by entering the order ID, checking if it exists in **orders.txt**, and ensuring that the payment status is "Unpaid." If found, it updates the order status to "Cancelled" in a temporary file, which then replaces the original orders file, and notifies the user of the cancellation or if the order ID was not found.
* **void removeCart():** The removeCart function removes items from the shopping cart that have been marked as "Paid" in orders.txt. It reads the cart items, checks each against the orders to see if they have been paid, and writes only the unpaid items to a temporary file, which then replaces the original cart file, effectively removing the paid items from the cart.
* **void makePayment():** The makePayment function processes payment for a specified order by reading from orders.txt, calculating the total amount for unpaid items, and updating their payment status to "Paid" in a temporary file. If the order ID is valid and contains unpaid items, it completes the payment, replaces the original orders file with the updated one, and calls removeCart to remove any paid items from the cart, notifying the user of the successful payment.

**Class:** Person

This class is the parent class to admin , employee and user class as its public functionalities are inherited in the derived classes.

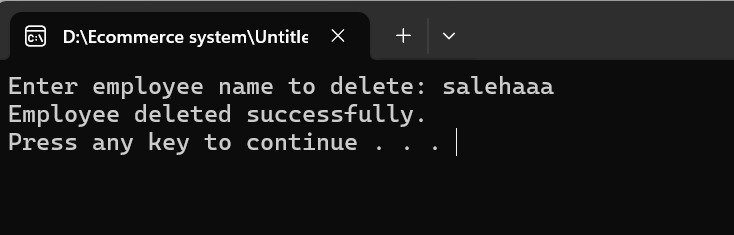
* **static void forgotPass():** The forgotPass function allows users (User , Admin, or Employee) to reset their passwords by entering their username and providing a new password. It checks if the username exists in the corresponding file (users.txt, admin.txt, or employee.txt), validates the new password, and updates the file with the new password if confirmed; otherwise, it retains the old password. If the username is not found, it notifies the user accordingly.

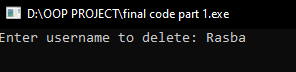


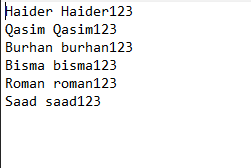
* **static void signup():** This signup function allows a user (Admin, Employee, or User) to create an account by entering a valid username and password. It validates the input, then saves the credentials to a role-specific file (users.txt, employee.txt, or admin.txt).
* **static bool login():** The login function verifies a user's credentials (Admin, Employee, or User) by checking the entered username and password against the corresponding file. If matched, login is successful; otherwise, it prompts to try again.

**Class:** Admin

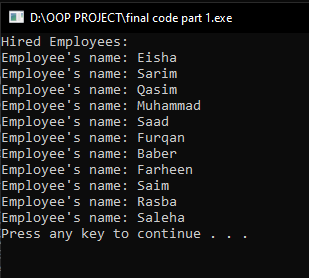
The class defines the functionalities that serves as Admin role in the system. The functionalities are also some of the major functionalities that alter and improve the system. The data members are inherited from the Person class.

* **void delete\_employee():** Deletes a specific employee by name from employee.txt by copying all other records to a temporary file. If the employee is found, their record is skipped during the copy, effectively deleting them.
* 
* **void delete\_users():** Deletes a user from users.txt by matching the entered username and rewriting the file without that user. A temporary file temp.txt is used to store the updated list and replaces the original after deletion.

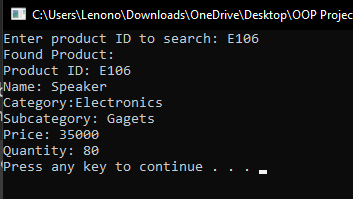




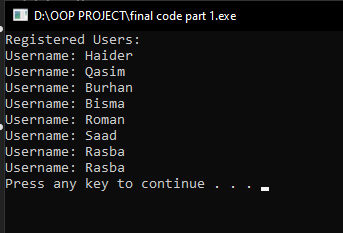
* **void view\_employee():** Reads and displays all employee usernames from employee.txt. Skips any empty or malformed entries and informs the user if the file can't be opened.



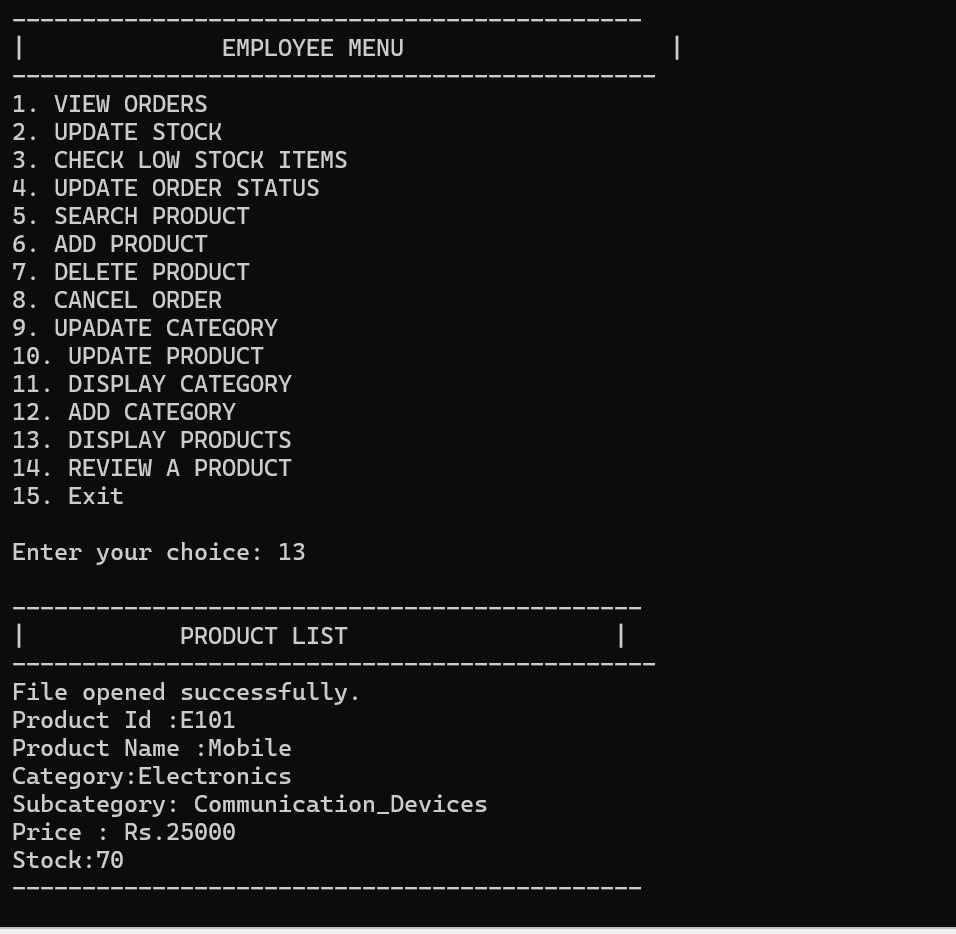
* **void search\_product():** Searches products.txt for a product using its product ID. If found, it displays all product details like name, category, price, and stock quantity.



* **void view\_users():** Displays all registered usernames by reading from users.txt. Skips incomplete records and shows only valid entries with both username and password.



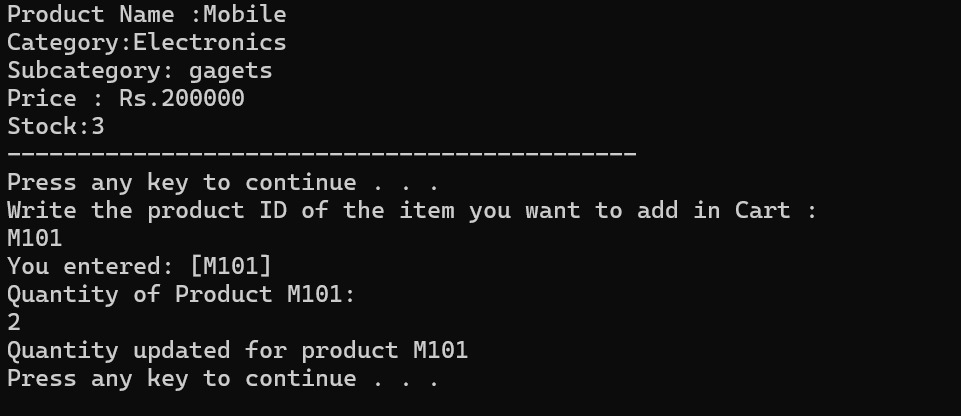
* **void displayProducts():** This function is used toread data from the product files and display it on the console.

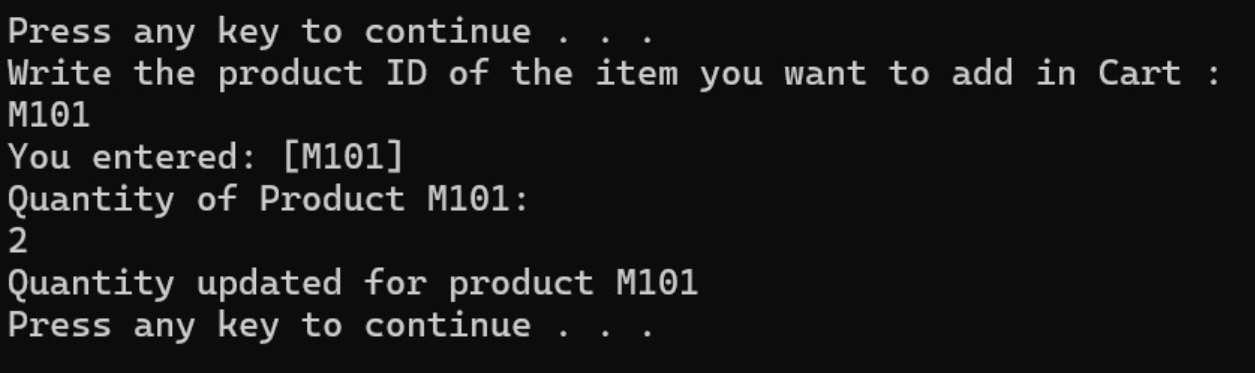


**Class:** User

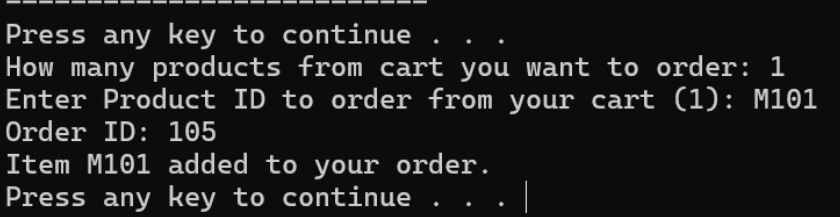
In the user class, Functionalities are inherited from the Person class and also from the CartOrder class in the user class. An object of the CartOrder is created and Functionalities are called inside the public functionalities of user class. This task is simply done by composition.

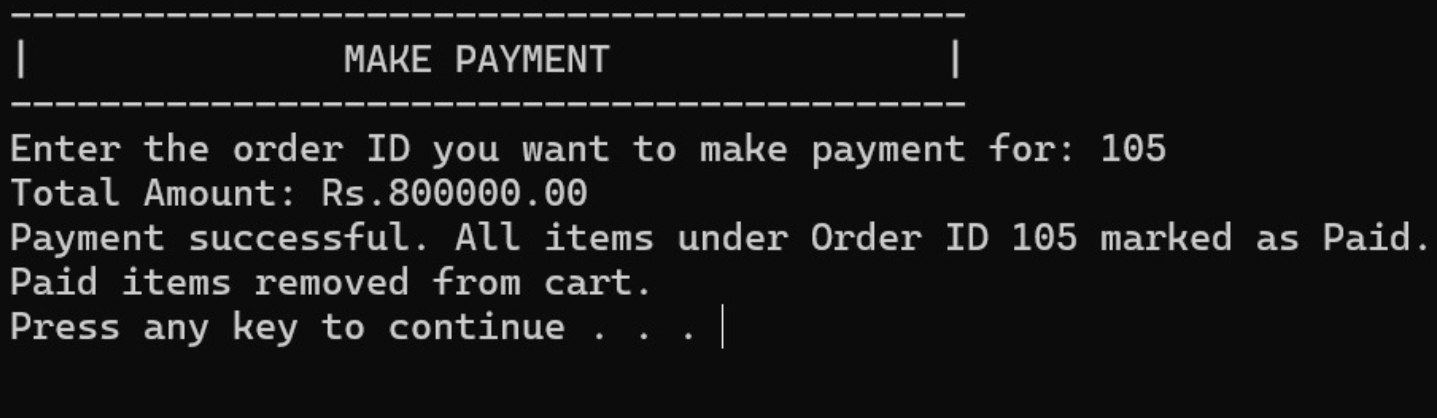
**Snippet:**

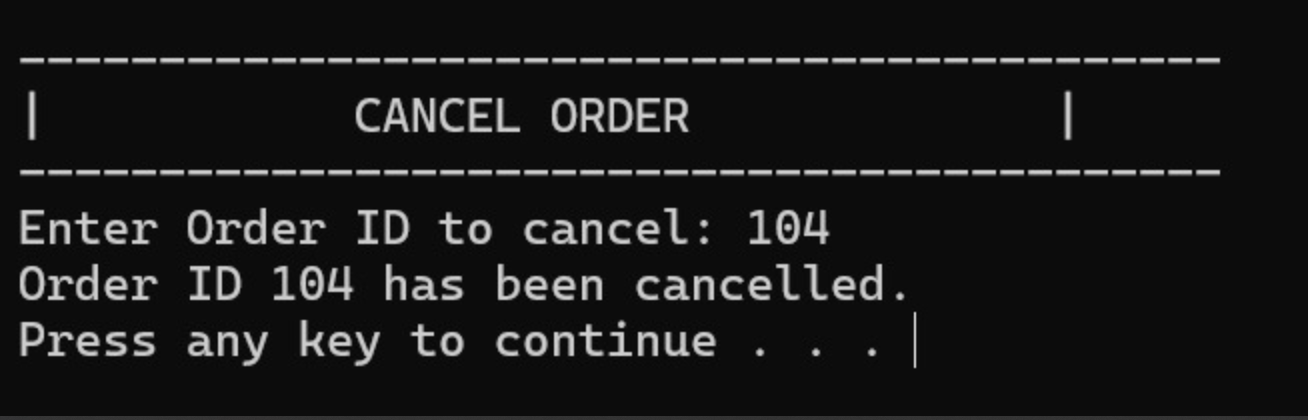


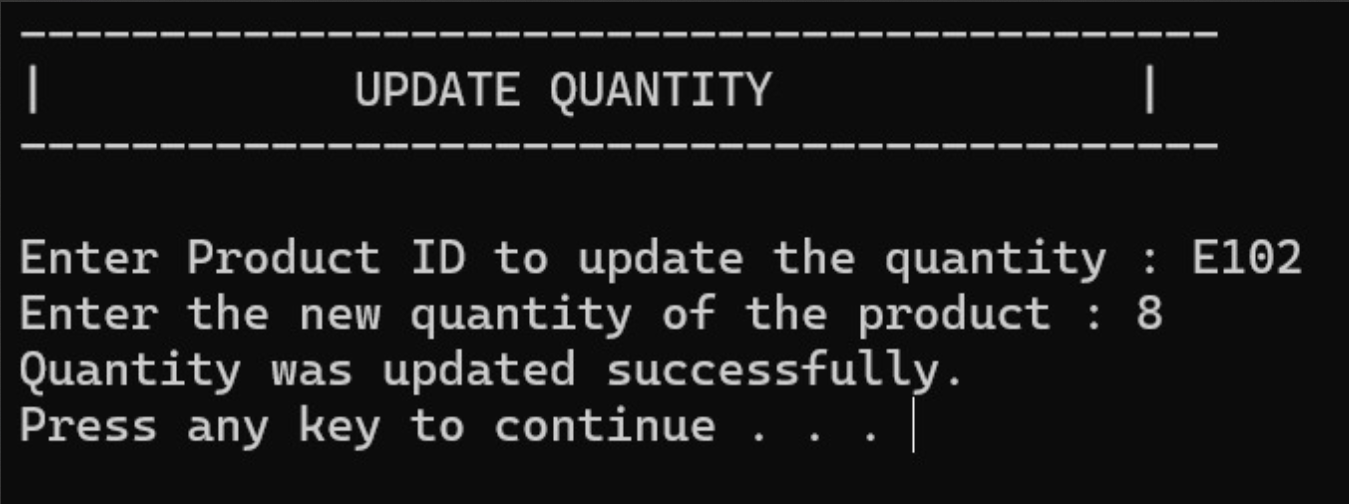


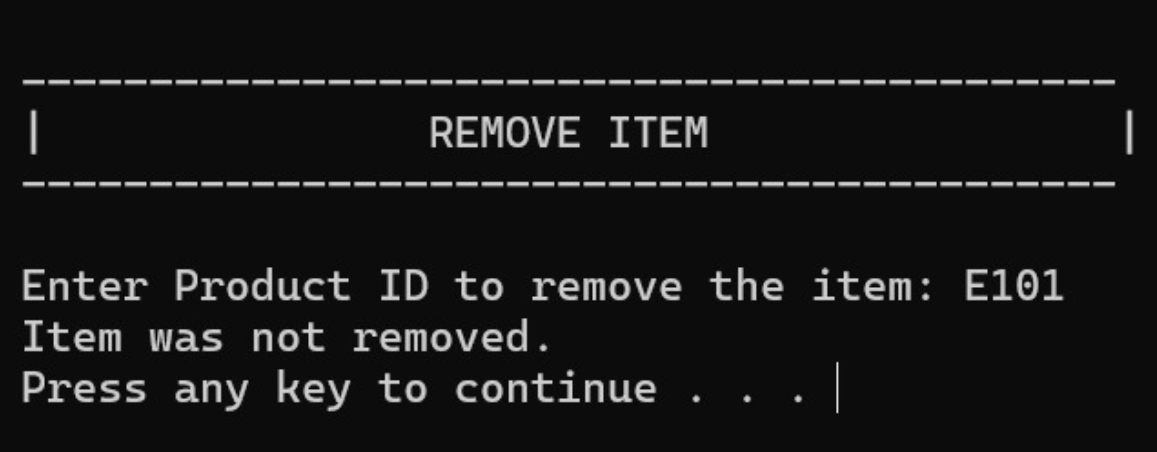


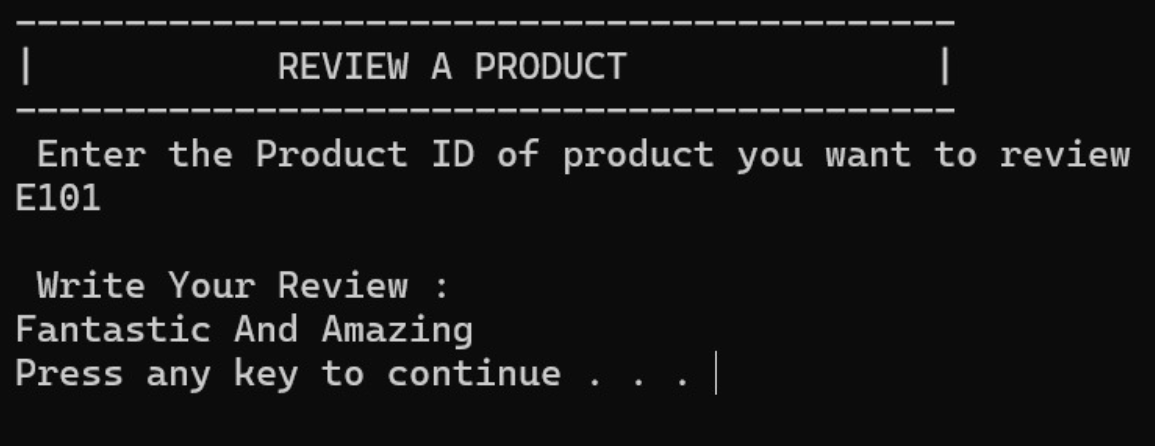


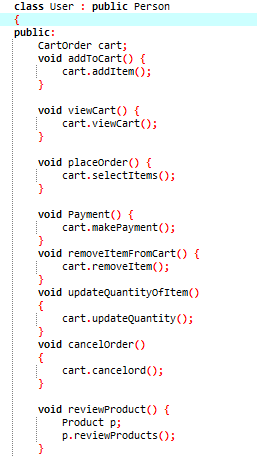








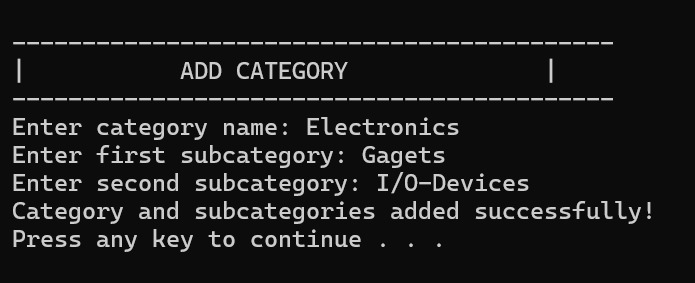


****

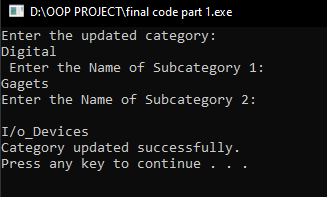
**Class:** Employee

This class has also inherited from the Person class all of its functionalities and used them in the code. The Employee class has many functionalities that hold a great grip in the system.

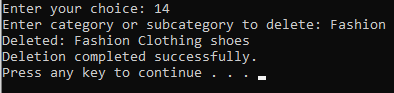
* **void addCategory():** This function allows the admin to create a new product category with two subcategories. The admin inputs the category name and both subcategories, which are then stored in the cat.txt file for future reference. This helps organize products for better navigation and classification.



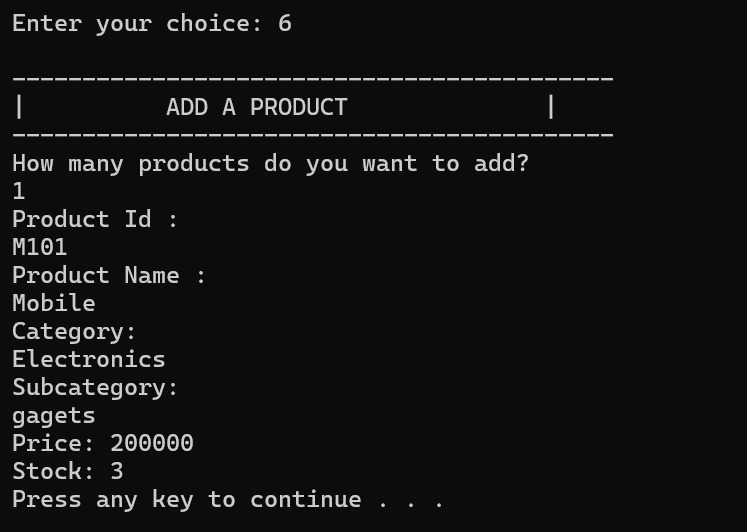
* **void updatecategory():** This function is used to update an existing category or its subcategories. It reads all categories from cat.txt, finds the matching category, and updates its information based on new admin input. The updated data replaces the old one by writing to a temporary file and renaming it back to cat.txt.



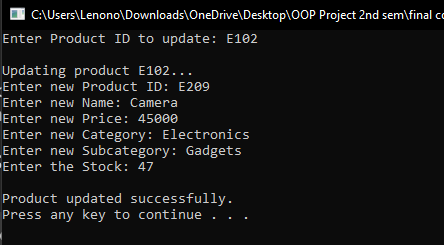
* **void deletecategory():** The admin can delete an existing category or subcategory using this function. It reads from cat.txt, omits the matching category/subcategory line, and writes the rest to a temporary file. After processing, the original file is replaced with the new one to reflect the changes.



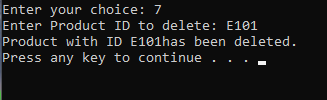
* **void addproduct():** This function allows the admin to add new products to the system. The admin is prompted to enter the product ID, name, category, subcategory, price, and stock quantity. The entered data is appended to products.txt, where all product information is stored.



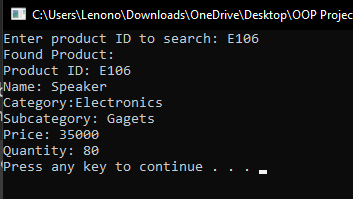
* **void updateProduct():** This function is used to modify the details of an existing product by searching with its product ID. Once found, the admin can update the name, category, subcategory, price, and stock. The old entry is replaced in products.txt with the new updated record.



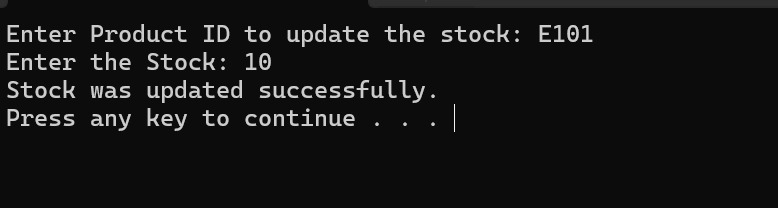
* **void deleteProduct():** Admins can remove products from the system using this function. It searches for the product by its ID in products.txt and excludes it from the updated list written to a temporary file. This cleaned data is then saved back to products.txt, effectively deleting the product.



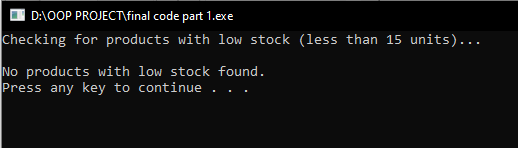
* **void search\_product():** This feature allows anyone (admin or employee) to search for a product using its ID. If a match is found in products.txt, it displays the product’s name, category, subcategory, price, and current stock. If not found, an appropriate message is displayed.



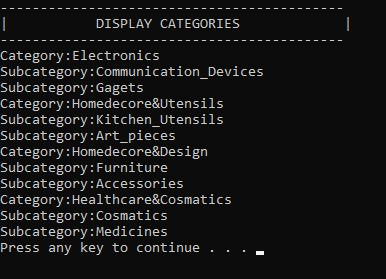
* **void updatestock():** Used to change the stock quantity of a specific product. After entering the product ID, the system locates the record and prompts for a new stock value. The product’s stock is then updated and saved back to products.txt.



* **void checkLowStock():** This function helps identify products that are running low on stock (less than 15 units). It reads through products.txt, filters those with low stock, and displays their details so the admin or employee can restock them before they run out.

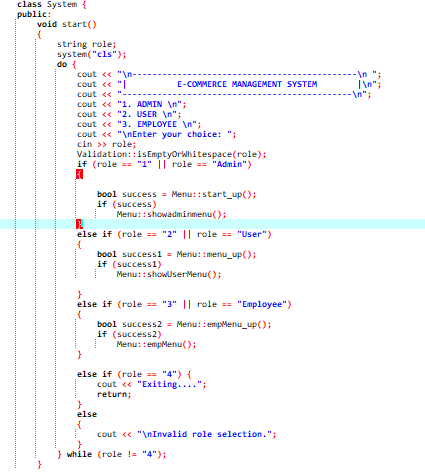


* **Void displaycategories():** This function reads data from the file of cat.txt and display categories along with its subcategories on console.

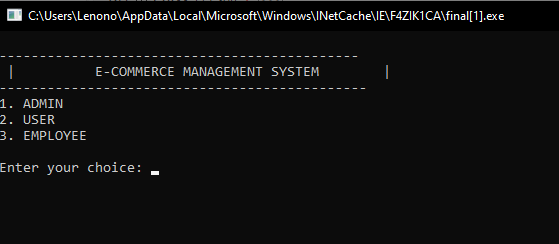


**Class:** System

This class shows the main menu of the system and displays it on console and user can select from it based on their roles.

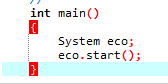


**Output :**

****

**Main menu:**

An object of the system class is called in the menu, which makes the whole program to execute and show the output of the system on the console.



**UML of the E-commerce system:**

