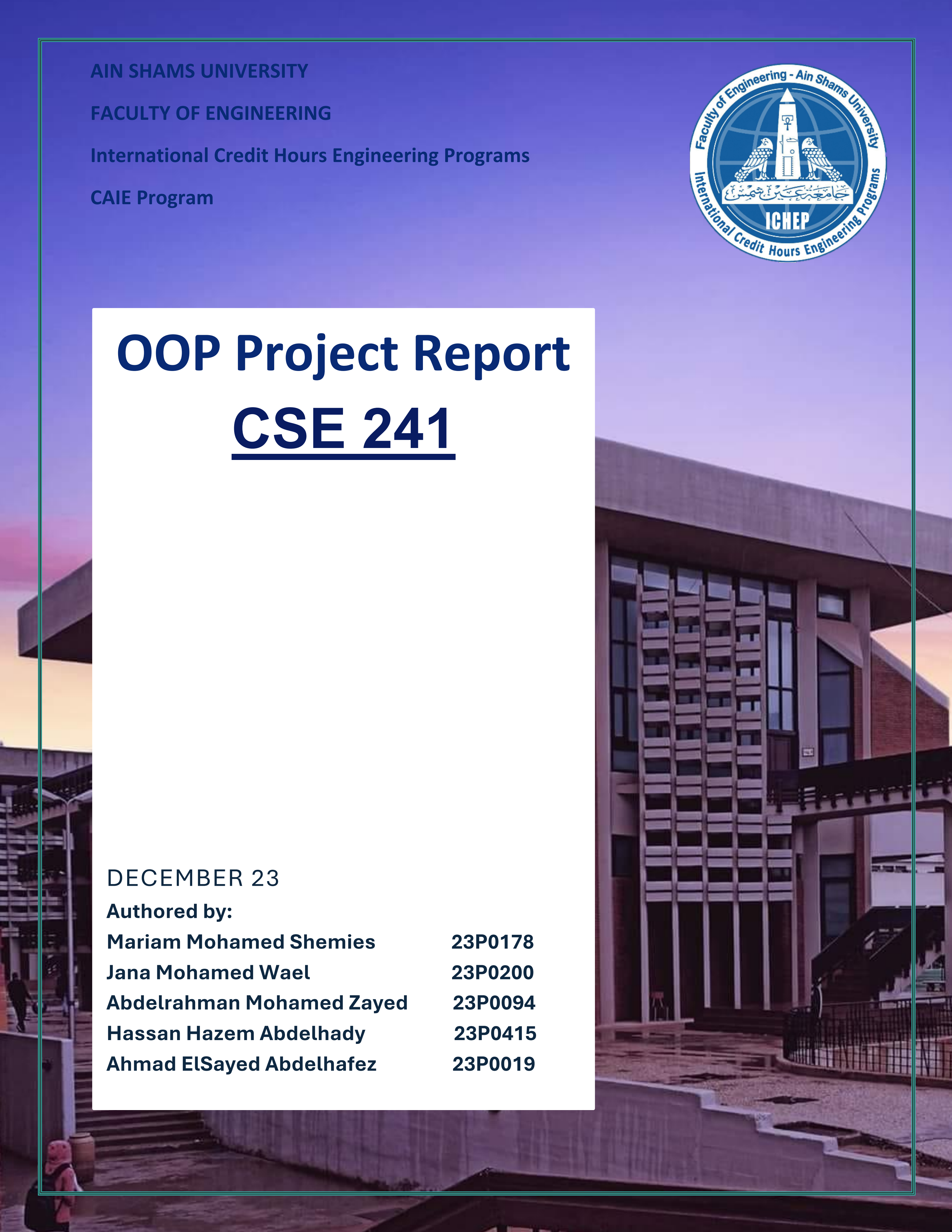


AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

International Credit Hours Engineering Programs

CAIE Program

OOP Project Report

CSE 241

DECEMBER 23

Authored by:

Mariam Mohamed Shemies	**23P0178**
Jana Mohamed Wael	**23P0200**
Abdelrahman Mohamed Zayed	**23P0094**
Hassan Hazem Abdelhady	**23P0415**
Ahmad ElSayed Abdelhafez	**23P0019**

Contents

Introduction

This project is an implementation of an e-commerce system developed using object-oriented programming (OOP) principles. The system is designed to manage various components such as users, products, orders, and administrative functionalities. It incorporates modular and reusable code, ensuring scalability and ease of maintenance. The application integrates a user-friendly interface with robust backend logic to deliver a seamless shopping experience for customers and effective management tools for administrators.

Overview

This project is an e-commerce application that employs object-oriented programming (OOP) principles to manage and organize its components. The system consists of core business logic, user interfaces, and utility functionalities. Below is a detailed explanation of the project's components and their relationships.

Core Classes (Business Logic)

1. Admin

- Purpose: Manages administrator-specific data and functionalities.
- Attributes: Stores admin details such as username, password, and privileges.
- Methods: Includes functions for managing products, categories, and user accounts.

2. Cart

- Purpose: Represents a shopping cart for users.
- Attributes: Maintains a list of products, quantities, and total price.
- Methods: Includes add/remove product functionality and calculates the total cost.

3. Category

- Purpose: Handles product categories.
- Attributes: Stores category names and descriptions.
- Methods: Includes methods to retrieve and display category-related data.

4. Customer

- Purpose: Represents customer details and behaviors.
- Attributes: Includes user ID, name, email, address, and order history.
- Methods: Handles actions like updating profiles, browsing products, and placing orders.

5. Order

- Purpose: Represents customer orders.
- Attributes: Contains order ID, products, quantities, total amount, and status.
- Methods: Handles order creation, updating status, and displaying order details.

6. Product

- **Purpose: Represents product data.**
- **Attributes: Includes product ID, name, price, description, and category.**
- **Methods: Supports adding, updating, and retrieving product information.**

Controller Classes (User Interface Logic)

7. AdminController

- **Purpose: Manages admin-specific UI actions.**
- **Methods: Provides functionalities for logging in as admin, viewing reports, and managing resources.**

8. CartController

- **Purpose: Handles user interactions with their shopping cart.**
- **Methods: Includes adding/removing items, updating quantities, and viewing cart details.**

9. CustomerController

- **Purpose: Manages customer-specific UI interactions.**
- **Methods: Supports browsing products, viewing profiles, and interacting with the shopping cart.**

10. LoginController

- **Purpose: Handles user login.**
- **Methods: Validates credentials and redirects users to the appropriate interface (Admin/Customer).**

11. PlaceOrderController

- **Purpose: Facilitates the order placement process.**
- **Methods: Confirms order details, processes payments, and updates the order database.**

12. ProfileController

- **Purpose: Allows customers to view and edit their profiles.**
- **Methods: Supports updating personal information and changing passwords.**

13. SignupController

- **Purpose: Handles new user registration.**
- **Methods: Collects and validates user details and saves them to the database.**

14. updateController

- **Purpose: Handles data updates.**
- **Methods: Updates product details or customer information as required.**

Utility and Supporting Classes

15. Database

- Purpose: Manages database connections and queries.
- Methods: Includes functionalities for executing SQL commands and retrieving data.

16. Displayable

- Purpose: An interface for making objects displayable in the UI.
- Methods: Likely includes a display() method to format objects for the frontend.

17. CRUD

- Purpose: Provides basic database operations (Create, Read, Update, Delete).
- Methods: Simplifies database interaction across different classes.

Entry Point

18. EcommerceApplication

- **Purpose: The main class that initializes and launches the application.**
- **Methods: Sets up the application environment and starts the primary UI.**

FXML and UI Integration

The project includes multiple FXML files for defining user interfaces. Controllers link these layouts to the business logic. Examples include:

- **Login Screen: Managed by LoginController.**
- **Admin Dashboard: Managed by AdminController.**
- **Cart View: Managed by CartController.**

Relationships Between Classes

1. Inheritance:

- **Admin and Customer inherit from Person.**
- **Displayable serves as an interface for UI components.**

2. Composition:

- **Cart contains a list of Product objects.**
- **Order includes customer and product details.**

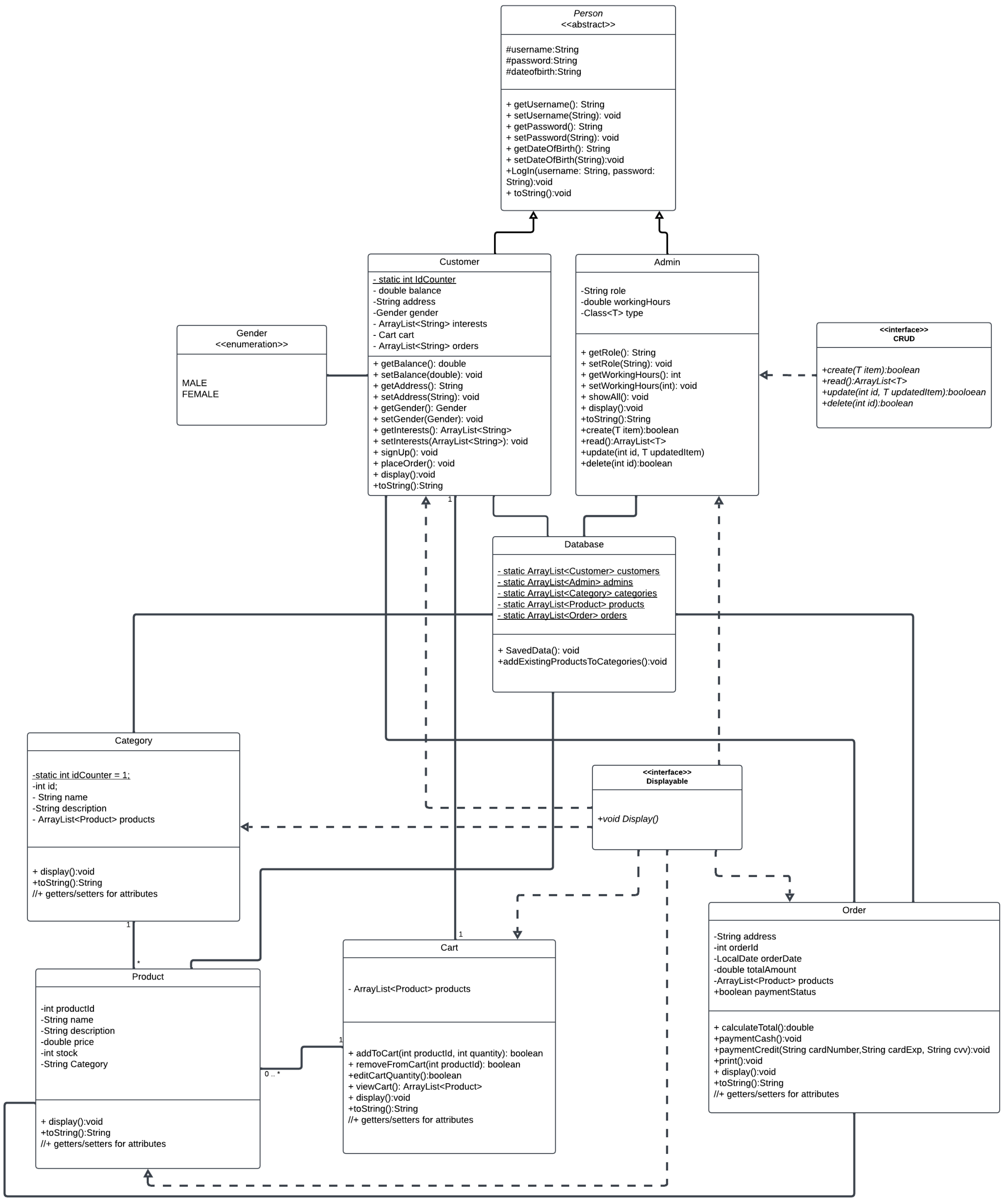
3. Association:

- **Customer is associated with Order and Cart.**
- **Admin interacts with Product and Category for management purposes.**

Additional Notes

- **Error Handling: Likely managed within controllers and utility classes.**
- **Extensibility: The use of interfaces and modular design suggests the system can be extended with minimal effort.**

UML Diagram


Person
<<abstract>>
#username:String
#password:String
#dateofbirth:String
+ getUsername(): String
+ setUsername(String): void
+ getPassword(): String
+ setPassword(String): void
+ getDateOfBirth(): String
+ setDateOfBirth(String):void
+LogIn(username: String, password: String):void
+ toString():void
Customer
- static int IdCounter
- double balance
-String address
-Gender gender
- ArrayList<String> interests
- Cart cart
- ArrayList<String> orders
+ getBalance(): double
+ setBalance(double): void
+ getAddress(): String
+ setAddress(String): void
+ getGender(): Gender
+ setGender(Gender): void
+ getInterests(): ArrayList<String>
+ setInterests(ArrayList<String>): void
+ signUp(): void
+ placeOrder(): void
+ display():void
+toString():String
Admin
-String role
-double workingHours
-Class<T> type
+ getRole(): String
+ setRole(String): void
+ getWorkingHours(): int
+ setWorkingHours(int): void
+ showAll(): void
+ display():void
+toString():String
+create(T item):boolean
+read():ArrayList<T>
+update(int id, T updatedItem)
+delete(int id):boolean
Gender
<<enumeration>>
MALE
FEMALE
<<interface>>
CRUD
+create(T item):boolean
+read():ArrayList<T>
+update(int id, T updatedItem):booloean
+delete(int id):boolean
1
Database
- static ArrayList<Customer> customers
- static ArrayList<Admin> admins
- static ArrayList<Category> categories
- static ArrayList<Product> products
- static ArrayList<Order> orders
+ SavedData(): void
+addExistingProductsToCategories():void
Category
-static int idCounter = 1;
-int id;
- String name
-String description
- ArrayList<Product> products
+ display():void
+toString():String
//+ getters/setters for attributes
<<interface>>
Displayable
+void Display()
1
*
Product
-int productId
-String name
-String description
-double price
-int stock
-String Category
+ display():void
+toString():String
//+ getters/setters for attributes
0 .. *
1
1
Cart
- ArrayList<Product> products
+ addToCart(int productId, int quantity): boolean
+ removeFromCart(int productId): boolean
+editCartQuantity():boolean
+ viewCart(): ArrayList<Product>
+ display():void
+toString():String
//+ getters/setters for attributes
Order
-String address
-int orderId
-LocalDate orderDate
-double totalAmount
-ArrayList<Product> products
+boolean paymentStatus
+ calculateTotal():double
+paymentCash():void
+paymentCredit(String cardNumber,String cardExp, String cvv):void
+print():void
+ display():void
+toString():String
//+ getters/setters for attributes

Conclusion

This e-commerce application demonstrates the effective use of object-oriented programming principles to create a structured, scalable, and maintainable system. By modularizing the application into business logic, controllers, and utilities, it ensures separation of concerns and facilitates future development. The integration of FXML for the user interface further enhances the user experience while maintaining clean backend logic. This project showcases the importance of OOP concepts in developing real-world software solutions, balancing functionality with maintainability.