CSC8404 – Advanced Programming in Java Coursework Report

Author: Swapnil Sagar **Student ID: 250620502** **Date: October 17, 2025**

1. Introduction

For this coursework, I created a Java system to handle online orders for a PC retailer. The main aim wasn't just to build something that works but to apply the key ideas we covered in the CSC8404 module, like object-oriented design, abstraction, and safe programming practices. While working on it, I tried to follow the same design principles taught in lectures, such as programming to interfaces, using inheritance properly, implementing the factory pattern, and writing defensive and immutable code. This report explains what I did, why I did it, and how it connects to what we learned in class.

2. Design and Implementation

Throughout development, I focused on keeping my code clean, flexible, and easy to maintain. I didn't just want to get it running, I wanted to understand why each design choice mattered.

2.1 Interface-Based Design

One of the first things I did was define all the main parts of the system like PCModel, Order, Customer, CreditCard, and OrderSystem, as interfaces. I remember from the "Programming to the Interface" lecture that this makes the code more flexible. It really helped because the OrderSystem didn't need to care whether it was dealing with a preset or a custom model both followed the same interface.

2.2 Inheritance and Abstraction

To avoid repeating code, I made an abstract class called AbstractPCModel. It handled all the common things between PresetModel and CustomModel, like model names and shared methods. This made the structure cleaner and easier to extend later. It also helped me understand the real value of inheritance and abstraction, not just as theory but as a way to write smarter code.

2.3 Factory Pattern

When I got to the part about making sure credit card numbers and custom model names were unique, I used the Factory Pattern just like we discussed in the "Object Factories" lecture. I built two factories, CreditCardFactory and CustomModelFactory, that take care of creating and managing objects. They use a static map to store existing instances so no duplicates are created. This design also stopped other parts of the code from creating objects directly, keeping everything consistent and safe.

2.4 Immutability and Defensive Programming

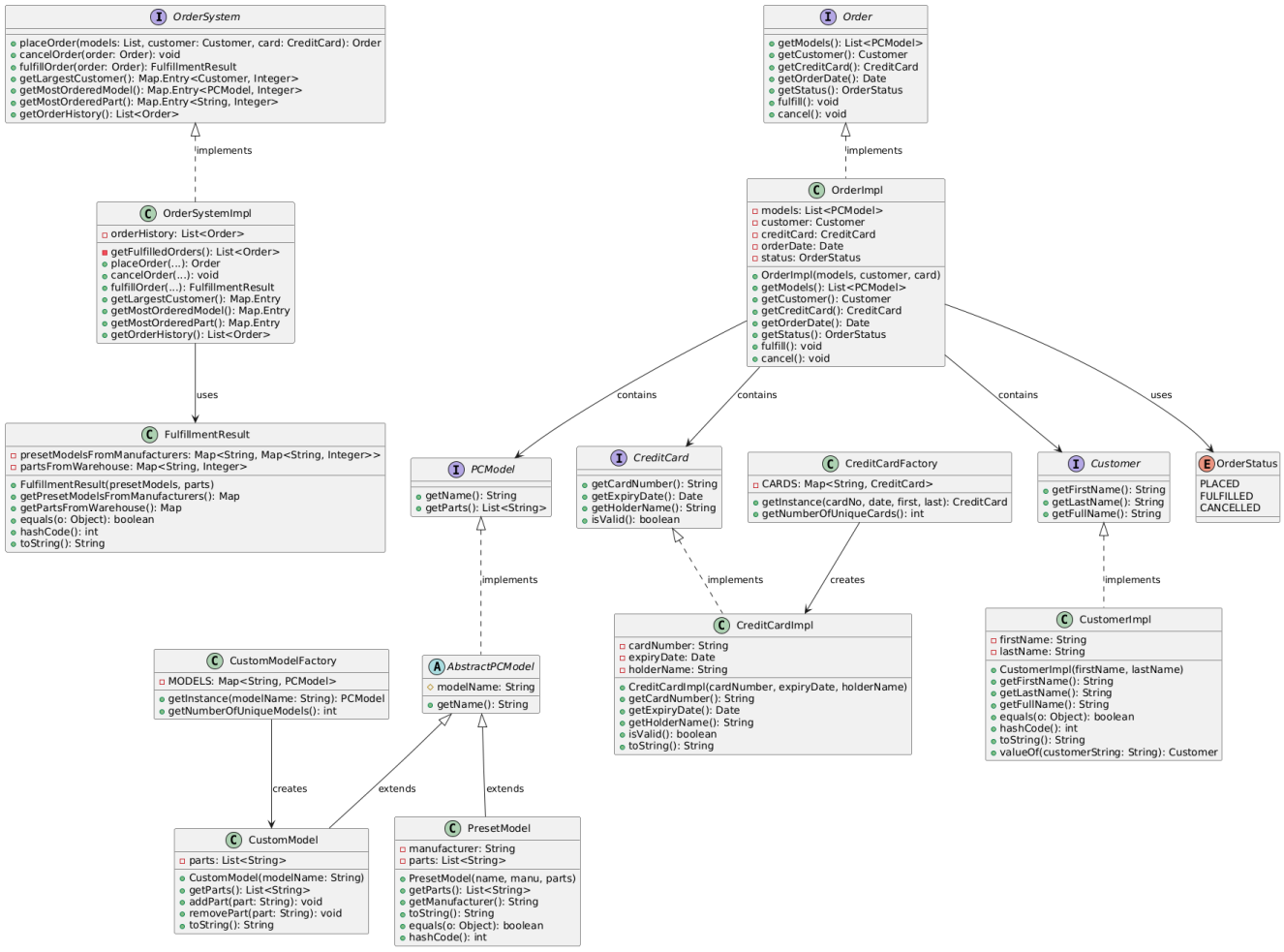
Another big focus was writing code that couldn't be easily broken. I made classes like Customer, PresetModel, and FulfillmentResult immutable, meaning once they're created, their values can't change. I also made defensive copies of things like lists and dates so the internal data couldn't be modified from outside. This was something emphasized in the "Good Practice and Safer Programming" lecture and actually implementing it helped me understand why it's so important in real projects.

3. Overview and UML diagram

3.1 System Overview

Here's a quick look at the main parts of my system and what they do:
- OrderSystem / OrderSystemImpl: Handles placing, cancelling, and fulfilling orders.
- PCModel, PresetModel, CustomModel: Represent different types of computers.
- Customer / CreditCard: Simple, immutable objects storing customer and payment info.
- CustomModelFactory / CreditCardFactory: Create and manage unique instances.
- Order / OrderStatus: Manage each order's state and history.

3.2 UML diagram



4. Conclusion

Doing this coursework gave me a chance to turn theory into practice. I got to see how programming to interfaces really improves flexibility, how abstraction and inheritance make code reusable, and how factory patterns simplify object management. Making my classes immutable and writing defensive code also gave me confidence in building systems that are safe and predictable. Finally, setting up JUnit tests helped me catch errors early and confirmed that my code followed the coursework rules. Overall, this project helped me connect everything we learned in CSC8404 with real coding experience, and I feel more confident now in designing and implementing object-oriented systems on my own.