Car Wash Management System

SuperShine is a self-service carwash. Your job is to develop a car wash system for *SuperShine.*

This is how the business work at *SuperShine* and high-level requirements for the new system:

- Customers buy an electronic WashCard that contains a value of washes for 200-1000 kr. (this is **not** part of the system).
- When customers want to wash their car, they insert their card into the machine and chose a specific wash type. The price of the wash is then deducted from the card.
- Customers can recharge their card with an amount up to 1000 kr. by inserting their credit card and their WashCard into the machine.
- Customers can check the amount left on their card by inserting it into the machine.
- Customers can have a receipt printed if they wish.
- The owner would like to have statistics of the washes.
- The system should be able to handle wash types and prices

Wash types:

- Economy kr. 50
- Standard kr. 80
- De Luxe kr. 120

SuperShine has a special “Early Bird” discount on weekdays of 20%, except De Luxe, if the wash starts before 2 p.m. (14:00)

Software Analysis & Design (UML)

- Identify **Actors** and draw a **Use Case Diagram**
- Create a **Domain Model**
- Write **Use Case descriptions**

Software Construction (Java)

Based on the above diagrams, you must implement the use cases one at a time[1]. This "divide and conquer" approach will address fundamental challenges such as 1) what shall we build and 2) how shall we build it by breaking the complexity into smaller and more manageable parts.

1. Start with a simple use case and implement its main success scenario.

In this way, the software design gradually evolves as you solve the problems step by step. Make sure to keep traceability between the domain model and the code. *Example: A Customer in the domain model becomes a Java class with the same name*.

2. Continue the implementation of the alternative path(s) for the use case.

In order to prevent your program from crashing if an error occurs, you should handle it gracefully by informing the user and continue if possible.

3. Pick another use case and implement it in steps as above.

If the new use case requires changes in the existing code base, it is a natural consequence of letting the design gradually grow. *Example: move redundant code to a new method*. This technique even has a name: "refactoring[2]".

4. Create a Design Class Diagram of your final code design

Hand in your solution on Fronter as a group
Solution: Java code as IntelliJ zip file AND UML artifacts
Deadline: Friday March 20 at 18.00**.**

[1] You don't have to implement all use cases. Quality over quantity!

2 Refactoring is a systematic process of improving the code without creating new functionality in order to make the code "clean" with a simple design.