Software Engineering Methods – Group 34B

Task 2: Design Patterns

Group 34B: Members:

- Maarten van der Weide
- Rithik Appachi Senthilkumar
- Stefan Creasta
- Viet Luong
- Vlad Iftimescu
- Yongcheng Huang

Scenario assigned: Rowing

Contents:

Design Pattern 1 – Builder:

- Description of design pattern implementation:

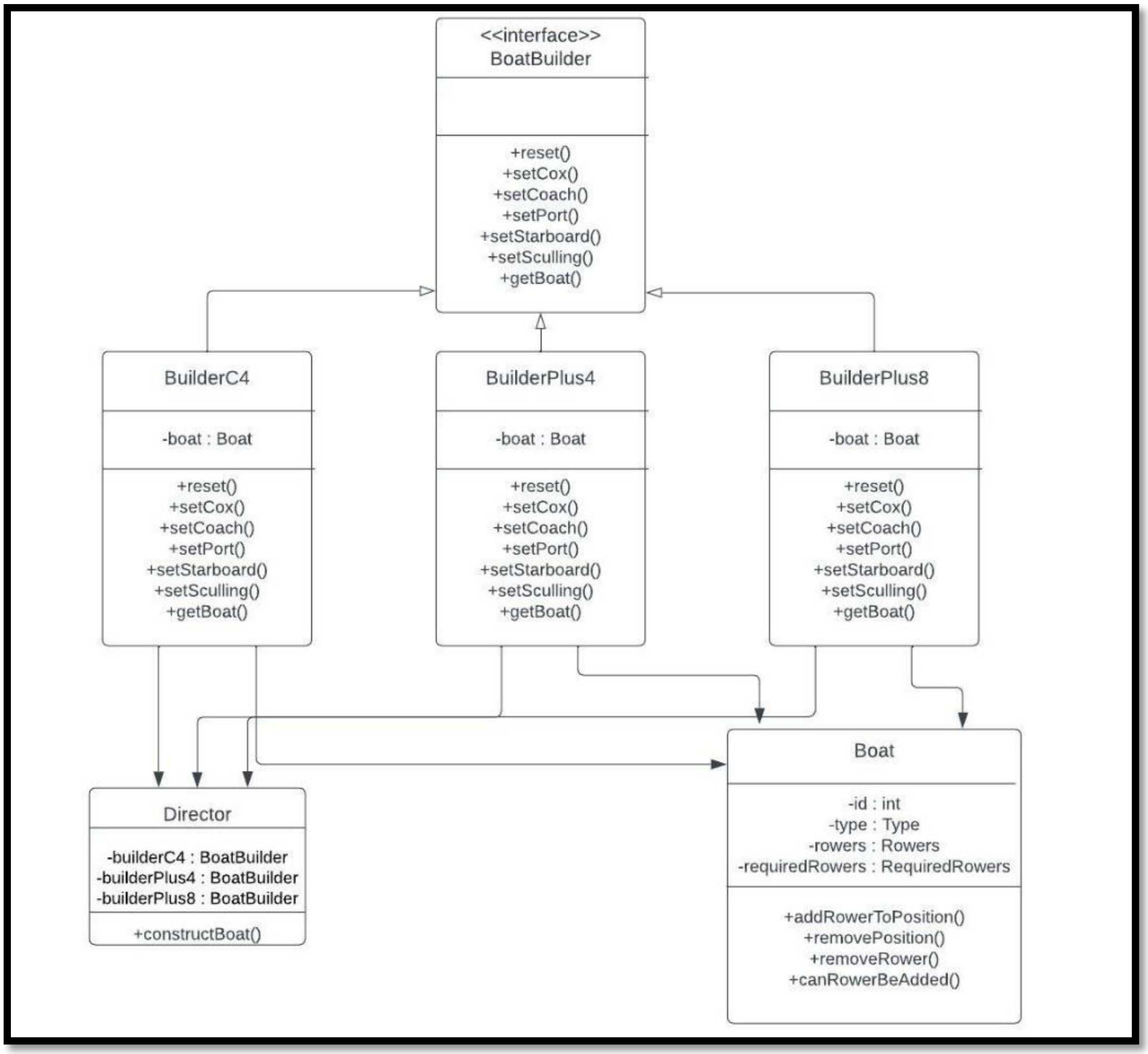
We have implemented the Builder design pattern inside the boat microservice. We made this choice, because it makes the code more flexible, and it removes the need to create a constructor with many arguments. Thus, instead of having constructors for every single type of boat with the number of required people for each position (cox, coach etc.), the builder will use setters to define the values in the amountOfPositions map field depending on the type of boat.

In the builders package there is an interface, BoatBuilder and three separate builders for each boat type, which implement this interface. Finally, the Director class actually builds a boat inside its method, by calling one of the builders, depending on the type of boat.

We had to choose for this microservice between a Factory and a Builder design pattern, as these ones are the ones which make the most sense to use in this case. The problem with the Factory design pattern is that we don’t really have a family of product objects, but a rather complex object: a boat with certain positions. Thus, building the object step by step with the help of setters is more desirable.

One of the Builder design pattern’s main advantages is the fact that it allows for variation/flexibility in a product’s internal representation. Essentially, it gives the programmer a greater extent of control over the product (in our case – the Boat) and its construction process. Moreover, it is also preferred because not every Boat is the same, and hence the builder allows for different representations for the Boats that are constructed. Having a builder design pattern for the boats also makes it scalable, which is a great advantage of this design pattern.

- UML Class Diagram of design pattern structure:



- Location of design pattern implementation:

The necessary BoatBuilder interface, Director, and the 3 specific boat builders can be found in the ‘builders’ package via the following path:

“boat-microservice/src/main/java/nl.tudelft.sem.template.boat/builders”

The Boat (and its necessary attributes) can be found via the path

“boat-microservice/src/main/java/nl.tudelft.sem.template.boat.domain”

Design Pattern 2 – Façade:

- Description of design pattern implementation:

We have implemented the Façade design pattern inside the activity microservice, for the RestService. The RestService is present in the activity microservice, and enables communication between the activity microservice and the other two, User and Boat microservices. We opted for such an implementation, mainly to be able to hide the complexities associated with the Rest Service, improve its readability and act as a convenient entry point for the complex implementations.

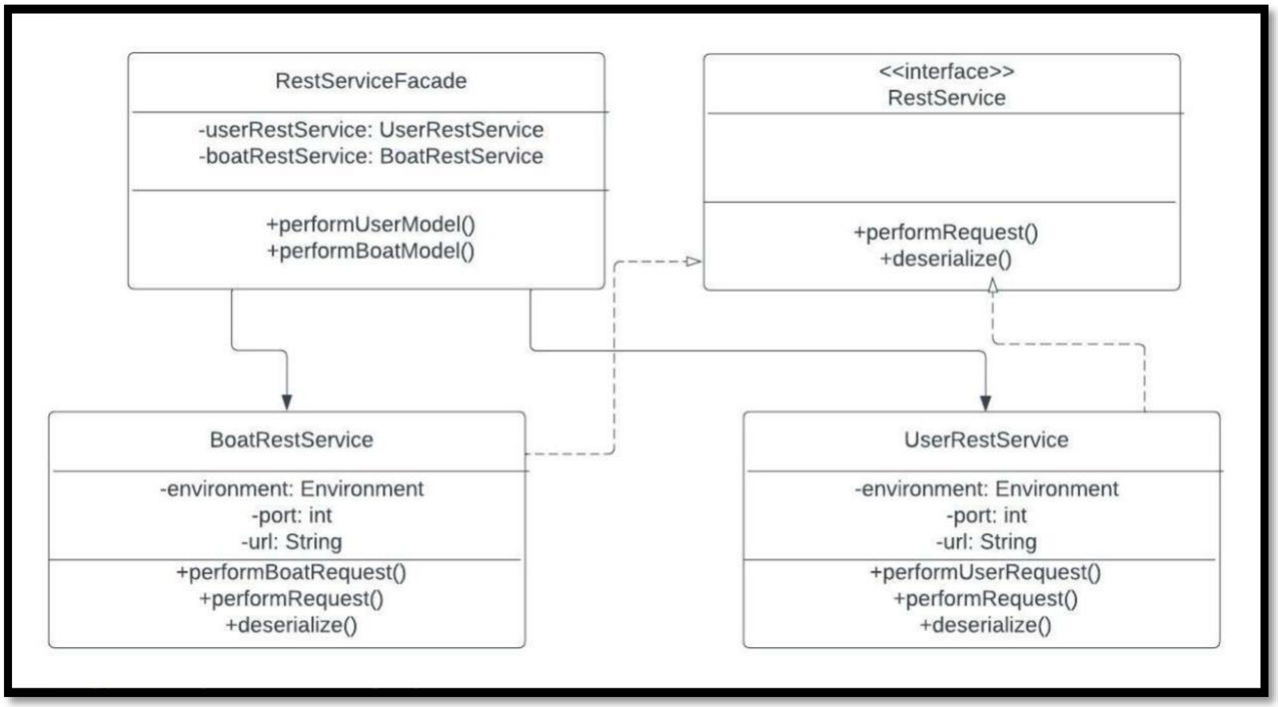
One of the main advantages associated with our implementation of the façade pattern, is the improvement in readability of the classes and relevant methods.

The façade pattern also improves the simplicity of the Rest Service structure greatly. Instead of interacting directly with the complex classes, the interactions now are masked behind a single, and simplified RestServiceFacade.

If we had used a single RestService implementation with no supporting façade pattern, said class would be monolithic and highly coupled, since the implementations for sending HTTP requests for both the user and boat microservices would all be bunched up together. The façade pattern solves this issue for us. Now, we have 2 concrete classes, one for communication with each microservice, and the RestServiceFacade consists of instances of each of the 2 classes. Now, if a method is called to communicate with the user microservice, the RestServiceFacade directs it to the appropriate Rest Service which then fulfils the communication request.

Our RestServiceFacade also acts as an entry point to the subsystem levels. Any call for the activity microservice to communicate with either the user or the boat microservices first enters the Rest Service Façade after it is, as mentioned before, appropriately redirected.

- <u>UML Class Diagram of design pattern structure:</u>



- <u>Location of design pattern implementation:</u>

The necessary RestService interface, RestServiceFacade class, and the concrete implementations UserRestService and BoatRestService (that implement the RestService interface) can be found in the following path:

“activity-microservice/src/main/java/nl.tudelft.sem.template.activity.domain/services”