Infineon Task

Task 2

1. In my understanding the term “Design Patterns” mean typical “best practice” solutions for common problems.

2.

- MVC stands for Model View Controller

- MVC pattern is used for separating the code in applications into UI, logic and the part that is controlling them.
- The View part is responsible for what the user sees, it renders the UI, we can compare it to Frontend
- The Model part handles the logic of the application that is not visible for the user, it corresponds to Backend
- The Controller part is intermediary. It is responsible for getting the request, then interacting with the Model and ten renders the necessary View. Corresponds to Middleware.

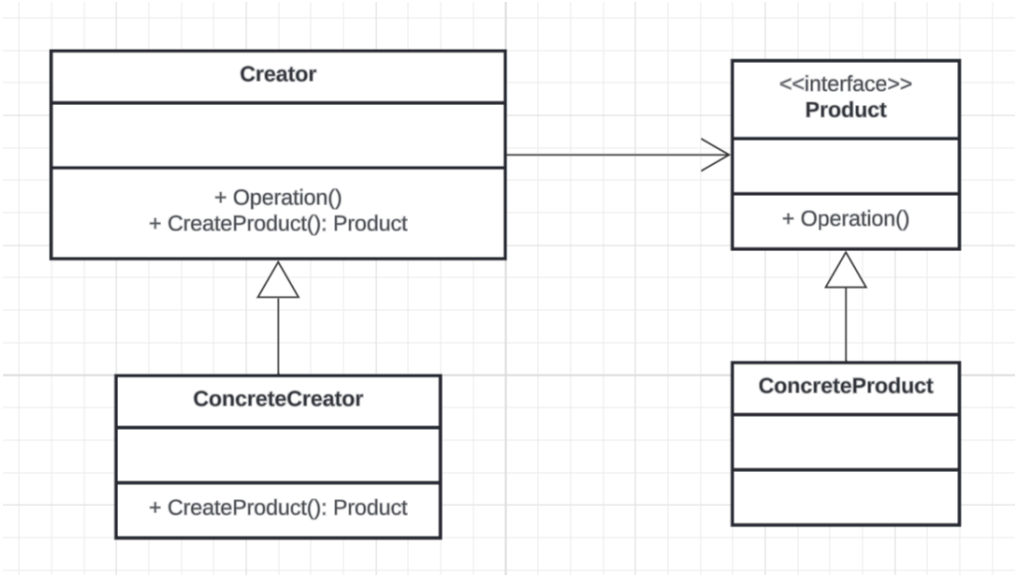
All in all the user communicates with Controller via input, then it asks the Model to do its job and based on the Model’s answer it presents the necessary UI to the user back.

- This framework is mostly used in web development as well as in desktop and mobile applications.

3.

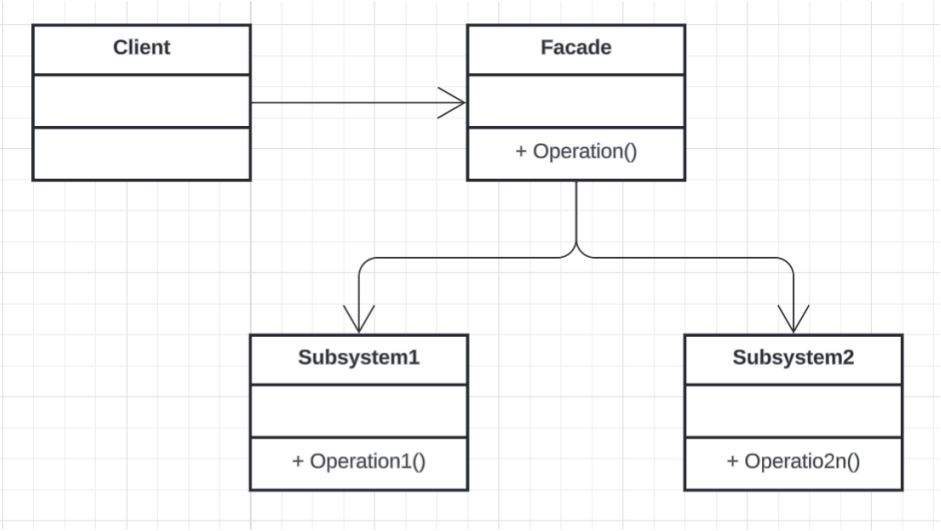
- **Factory** design pattern: provides a functionality for handling the logic of object creation. It prevents the use of ‘new’ when creating objects. Instead, it decides which class to instantiate based on the input.

I used this pattern doing the course “Software Engineering”, it helped me to simplify the program logic for object creation.



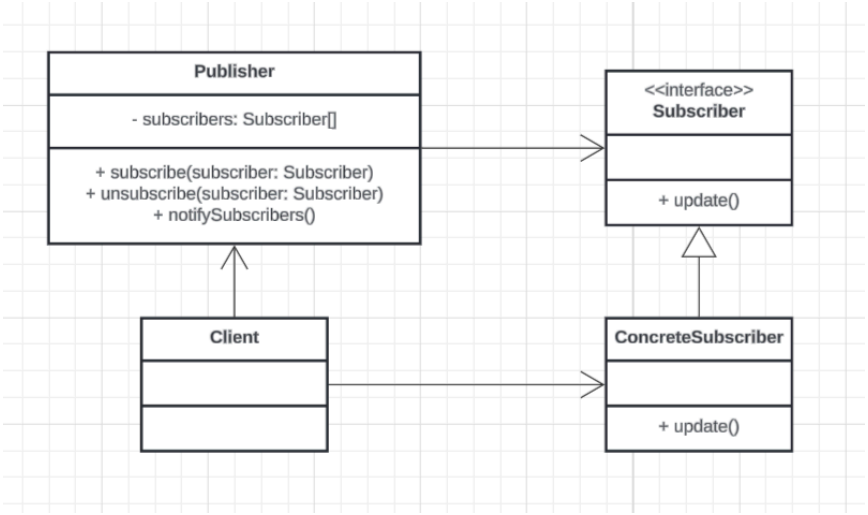
- **Facade** design pattern: defines a simple way for the client to deal with the complicated logic of the application. It helps to use features easily.

This design pattern was used when I developed some applications, for example, a video-based retrieval system. There, the user typed a query into a single interface, the Facade class organises all the hidden logic, and user gets the result.



- **Observer** design pattern: provides one-to-many dependency, so when some object changes its state or some event happens, the subscribers are notified.

Unfortunately, I do not have much experience with this pattern. I have only implemented it doing a course.



Task 3

2.

- No, because the ObjectA class is abstract.
- No, because PrintMessage method in ObjectB is private and visible only inside its class
-

- **Abstraction**: ObjectA is abstract telling all its subclasses what they must have.
- **Polymorphism**: in this example we have overloading, having two methods with the same name, but different parameters.
- **Inheritance**: we have classes that inherit properties and methods one from another.
- **Encapsulation**: we have different attributes, like public, private and protected, controlling access to the method or the variable.

Task 4

1.

- First, I would clone the repository and download all the dependencies to run the code locally. Then I am going to examine the README.md file and the documentation. If there are some tests, I would study them to get an understanding of what the code should do. When dealing with new projects, I usually concentrate on one file, then examine the next one and try to understand what they are doing and how files interact with each other. It is a good idea to commit small changes to make changes more understandable. And of course, I would use git for that.
- I am familiar with version management, so I would use feature branches and again small commits. Moreover, I think writing tests for my changes is beneficial. Finally, to ensure that everyone can understand my changes, I would update the documentation.

2.

- When the new features are added, it is important to use unit tests. I would also try to avoid smells like big class and to minimise coupling and maximise cohesion. If needed, I will refactor the code.
- During the Software Engineering course, I learned that writing understandable documentation is an important task, so I will try to answer questions like “why” and “how”, not just “what” when documenting. Also, I will comment the code so it is more clear. As for testing, I would just use various tests for the new feature, like unit or integration tests.

3.

- First, I have to identify the smell, then I will try to simplify the code, making a small simplification and then testing the solution. I will continue doing this until the smell is gone.
- If I face some smells, most probably I will have to use the Facade pattern to make an interaction with the system easier. Moreover, using the Factory pattern to simplify object creation. Also, I would probably have to use Adapter and Strategy patterns.