Technical Report II

Name: Julian David Celis Giraldo
Code: 20222020041
Workshop II

From the first version, we now proceed to develop the second version of the workshop, where we will expand the number of machines, create predefined machines, and allow the user to configure them. The class diagram according to the new requirements is as follows:

In the first version, we already had two concrete machines, so we added the six new machines required by the company, leaving us with eight concrete machines in total. Enums were added for new attributes, such as those for glasses attributes like resolution and others. Likewise, the **Builder** and **Factory** patterns were used. The **Builder** pattern is used due to the complexity of the objects involved, as they have a large number of attributes. Therefore, the **Builder** pattern was chosen to simplify the construction process. The **Factory** pattern is used to create the type of machine; once the attributes have been defined according to the client's preferences, **Factory** allows us to easily create the chosen machine for the user. In the ArcadeMachineBuilder, a method called set_increases was created, which allows me to adjust the values according to the material chosen.

The use of the **Abstract Factory** and **Builder** patterns in this context makes a lot of sense because each addresses specific design problems in our system, and together they provide a flexible and scalable solution.

Abstract Factory: For creating different types of arcade machines

The **Abstract Factory** pattern is ideal when we need to create multiple related types of objects, such as the different arcade machines. Each machine (Dance Revolution, Classical Arcade, Virtual Reality, etc.) has common attributes and behaviors (base price, memory, processors, etc.) but also specific characteristics that make them unique (for example, Dance Revolution has difficulty levels and arrows, while Virtual Reality has glasses and resolution).

Why is it useful?

- **Decouples machine creation** from the rest of the system, meaning I can add or modify machine types without changing the code that uses them.
- **Makes it easy to add new machines**. If tomorrow we need a new "Karaoke Machine," I only need to create a new concrete factory without modifying the code of the existing factories.
- **Cohesion**: Ensures that each factory creates a machine with all its defined characteristics. This prevents mixing incompatible features, such as putting virtual reality glasses on a racing machine.

How would we use it? When the client selects a type of arcade machine, like "Dance Revolution" or "Virtual Reality," I use the **Abstract Factory** pattern to create a default machine, already configured with the correct attributes.

Builder: For creating customized arcade machines

The **Builder** pattern is useful when we need to construct complex objects step by step, and in this case, it makes sense because arcade machines can be highly customizable. The client doesn't just select a machine; they can also choose the material, add video games, decide whether the games should be in high definition or not, etc. **Builder** allows us to handle all these options in a clear and organized way.

Why is it useful?

- **Allows creating the machine step by step**, which is very helpful when there are several decisions to make, such as choosing the material or adding video games.
- **More flexibility**: I don't need to create the entire machine all at once; I can add or modify things as the client requests.
- **Cleaner code**: The process of creating a customized machine remains well-structured and easy to follow.

How would we use it? First, the client selects the type of arcade machine (using the **Abstract Factory**). Then, **Builder** comes into play, allowing them to add features like the material, high-definition video games, and finally obtain a customized machine.

Why use both patterns?

The combination of both patterns is ideal because:

- **Abstract Factory** manages the creation of base or predefined machines, ensuring that each type of machine has its correct and specific configurations.
- **Builder** takes care of **customizing the machine** once it's been created, allowing the client to adjust it step by step according to their needs.

What do we gain from using them together?

1. **Extensibility**: We can add new types of arcade machines without touching the existing code, simply by creating new factories.
2. **Simple customization**: The client can flexibly customize their machine by adding or removing features without complications.
3. **Less complexity**: We keep the creation of the machines separate from their customization, making the code easier to understand and maintain.

Summary

In summary, **Abstract Factory** is responsible for creating the basic arcade machines, and **Builder** allows us to customize them flexibly. Together, they provide a modular, scalable, and maintainable design.

Note: I couldn't finish the program. I was able to add the modern machine and the patterns, but I didn't know how to relate the games. ☹