Advice Software Quality

Factory Method:

The Factory Method pattern can be compared to the Abstract Factory pattern for the following reasons:

* Centralized Object Creation: Both patterns centralize the logic for creating objects, improving code organization and maintainability.
* Consistent Interface: They ensure that products follow a consistent interface, making the system more predictable.
* Decoupling: Both patterns reduce tight coupling by abstracting the instantiation process.

However, their main functionalities differ. The Abstract Factory pattern focuses on creating families of related objects, ensuring they work together. It provides an interface for creating related or dependent objects without specifying their concrete classes. On the other hand, the Factory Method pattern defines an interface for creating an object but allows subclasses to alter the type of objects that will be created. This pattern is used to introduce variability in the object creation process without modifying existing code.

Benefits:

* Reduces tight coupling.
* Centralizes object creation logic.
* Enhances flexibility by allowing new types of objects to be added without changing existing code.

Drawbacks:

* May require multiple subclasses, which can complicate the code.

Usage in our project:

The Factory Method is used to create various elements within slides, such as bitmap items and text items. By using this pattern, it is possible to add different types of elements within the slides, and this method allows flexibility in creating them. It could also decrease resource consumption because the objects are created more efficiently.

Why we use it:

We use the Factory Method for slides to quickly add a text item or a bitmap item. This centralizes the creation logic for slide elements, making the code more organized and easier to extend with new types of items in the future.

Command Pattern:

The Command pattern is comparable to the Strategy pattern for the following reasons:

* Encapsulation of Behavior: Both patterns encapsulate behavior as objects (commands and strategies).
* Separation of Concerns: They promote separation of concerns by decoupling the client code from the implementation details.
* Flexibility and Extensibility: Both patterns allow behavior to be easily replaced or extended without modifying the client code.
* Runtime Selection of Behavior: In the Command pattern, different commands can be executed dynamically at runtime based on user input or system events. In the Strategy pattern, different algorithms can be selected dynamically at runtime based on specific conditions or requirements.
* Abstraction and Interface-based Design: Both patterns use a shared interface, allowing objects to be treated uniformly by the client code.

Despite these similarities, their primary functionalities are distinct. The Strategy pattern focuses on encapsulating algorithms as objects, allowing for the selection of different algorithms at runtime. For instance, it could define different ways a button might behave (e.g., moving to the next presentation vs. the next slide). However, the Command pattern encapsulates requests as objects, decoupling the senders and receivers of the requests. For example, it defines specific actions for buttons such as moving to the next slide or exiting the program.

Benefits:

* Promotes loose coupling between the sender and receiver of requests.
* Allows for easy addition of new commands without modifying existing code.
* Supports undo operations by storing the previous state of an object.

Drawbacks:

* Can make the code more complex, especially with many different commands and receivers.

Usage in our project:

The Command Pattern is used to implement various commands for manipulating slides, such as NextSlideCommand, PreviousSlideCommand, etc. This pattern enhances the flexibility of handling user interactions with slides and fits well with the structure of the Presentation and KeyController classes.

Why we use it:

We use the Command pattern for the presentation to execute commands for the slides, such as NextSlideCommand, PreviousSlideCommand, QuitCommand, and NewFileCommand. This allows for a clean separation of concerns and makes it easy to add new commands in the future without modifying existing code.

Singleton Pattern:

The Singleton pattern can be compared to the Static Class pattern for the following reasons:

* Single Instance: Both patterns ensure that only one instance of a class exists throughout the application.
* Global Access: They provide a global point of access to the instance, making it accessible from anywhere in the application.
* Controlled Access: Both patterns control access to the single instance to ensure consistent state and behavior.

Despite these similarities, their main functionalities are different. The Static Class pattern is essentially a class with static members, and it does not allow for lazy initialization or inheritance. On the other hand, the Singleton pattern provides a way to ensure that only one instance of a class is created, with lazy initialization and support for inheritance.

Benefits:

* Ensures a single instance of a class, providing a controlled access point.
* Supports lazy initialization, which can improve performance.
* Allows inheritance, enabling more flexible design options.

Drawbacks:

* Can introduce global state into an application, which might lead to unexpected behaviors in a multithreaded environment if not implemented correctly.

Usage in our project:

The Singleton pattern is used to manage the state of the presentation. For example, it makes sure that only one instance of the Presentation class exists, which provides a global access point for managing slides and other functionalities related to presentation.

Why we use it:

We use the Singleton pattern for managing the presentation state to make sure that there is only one instance of the Presentation class. This provides a consistent state and a single point of access for managing slides, which simplifies the code and avoids potential issues with multiple instances.