Advice

# Creational

## Factory

The factory method pattern provides an interface for objects in superclass and allows subclasses to alter the type of objects created.

Factory method will be used for the SlideItem class. Implementing this pattern will make it easier to extend the application, since it follows the Open/Closed principle. This pattern separates the construction code from the code that uses the result of the construction, so adding a new Item will only require to create a new creator subclass and override the factory method.

We chose this pattern because in the current code SlideItem objects are being created within methods with other responsibilities, so this way we separate the creation logic adhering to the Single Responsibility Principle.

# Structural

## Proxy

The proxy design pattern provides a placeholder for an object and controls access to the original objects. It also allows one to perform before or after the request gets to the original object.

The proxy pattern will be used for the BitmapItem class. We picked this pattern because in the code that was given to us when the application runs it can never access the last slide. This happens due to a mistake in the creation of a BitmapItem, trying to create an object with the type of file.

The proxy pattern will create a placeholder for the image so if the image is large or being loaded from a remote location and is taking a long time to load the placeholder will be there instead which will hopefully avoid crashes or a bugged-out page.

In conclusion the idea behind using this pattern is because the proxy works even if the service object isn’t ready or available.

## Decorator

Decorator will be used to improve on the Slide class. The decorator design pattern allows one to place an object inside special wrap objects that contain new behaviors that will then attach to the object.

Considering this, we suggest using the decorator pattern for the following reasons.

First, it enhances flexibility because it allows for the dynamic addition of further formatting and styling options such as colors, fonts, animations and more to individual slide items. This will be possible without modifying the code of the Slide class, which promotes the styling's extendibility. Flexibility also allows for scalability as new decorators can be added as needed without it affecting the rest of the code.

It also separates responsibilities between core functions and supporting features. By creating separate decorator classes, the core Slide class can focus on its primary function which is managing the slide items. This will improve modularity and clarity.

Testing is also fairly simple when it comes to decorators as they can be unit tested independently so their functionality can be verified isolated from the rest of the code. So, each additional decorator can be tested without breaking the exiting functionality.

It's also easy to maintain as it adheres to the open/closed principle which makes the codebase easy to extend and maintain without introducing bugs or causing other issues since the core class will not to be modified when enhancements are made.

# Behavioral

## Command

Command will be used for the KeyController and MenuController classes. The command pattern turns requests into standalone objects that contain all the information of the request and allows one to parameterize methods with different requests or delays. The Command pattern decouples the sender of a request (the event listener) from the receiver (the action to be executed).

In the case of KeyController and MenuController the handling of different events (keystrokes and menu selections respectively) is tightly coupled to the event listeners. Using the command pattern these events are encapsulated into separate command objects thus reducing the coupling.

It also will improve flexibility as with the command pattern, creating new commands or modifying existing commands is simple and can be done without modifying event handling logic. This makes the code easy to extend because the event listeners don't need to be modified when enhancing the rest of the code.

The command pattern also allows for undo/redo functionality. Each command can encapsulate both the action it is meant to perform and the inverse operation, which can man allow users to undo and redo actions easily. This can make the user experience more pleasant and simple.

Command objects can also be tested independently of the event handling logic. So individual behavior of commands can be observed in isolation. Additionally, you can mock command objects during testing to simulate different scenarios and verify the behavior of the event listeners.