Facade Design Pattern

Team WELM: Elijah Moreau-Arnott (emoraua), Weiyi Dai (w27dai), Matthew Hayashida (m2hayash), Lucy Yu (yf2yu)

Purpose & Intended Use Case

The Facade Design Pattern is a simplified interface to a subset of a complex existing system. The intended use case of the Facade Design Pattern is to shield the user from complex implementation details of a system, while providing the user a clear and concise set of functionalities through an interface.

Vocabulary

- Subsystem: a package or a class that provides complex functionalities
- Facade: an interface that provides a set of functionalities to the user, and abstracts away the complexities behind these functionalities which lie behind subsystems
- Client: a user who utilizes the facade

Known Consequences

Advantages

- Hides implementation details from the clients
- Simple and easy interface for the clients to use
- Weak coupling between clients and the system, so changes can be made to the system without affecting the clients

Disadvantages

- Clients cannot access underlying classes, and certain functionalities might be unavailable to clients
- It usually does not add any extra functionalities, just simplifying it for the clients to use

Non-functional Properties

- Lowers Complexity: provides a clean and simplified interface for the client to use, and reduces the complexity of having the client call methods in various subsystems
- Improves Scalability: to create additional interfaces, the maintainer of the facade can simply add new interfaces in the facade for clients to use
- Improves Usability: for clients, this pattern allows them to readily access key functionalities with minimal hassle

Similar Patterns

- Adapter: repurposes an existing interface to match what a client is expecting, whereas Facade defines a new interface for clients
- Mediator: abstracts functionalities of existing classes via a central mediator that has a multi-directional protocol and could also add functionalities. Facade abstracts

functionalities and follows a unidirectional protocol (the facade is unknown to the subsystem classes) and only specifies existing functionalities in the subsystems

Structure and Runtime Behavior

Included below are diagrams that illustrate an example using the Facade Design Pattern. This example visits the familiar scenario of interacting with a bank teller, which is the facade. A client can ask the bank teller to perform a number of actions. Depositing money, withdrawing money, and transferring money are three common actions. When the teller executes these actions, several systems are accessed and operations are performed on these systems. All of these complex operations are abstracted away from the client, as the client only needs to ask the teller to perform a general operation. It should also be noted that the facade usually only performs a subset of all the functionalities available in the subsystems that are abstracted away. In this example, there may be functionalities in the Cash System that the bank teller does not interact with for the operations offered to the client.