Establish order

Factory:

“Problem: Who should be responsible for creating objects when there are special considerations, such as complex creation logic, a desire to separate the creation responsibilities for better cohesion, etc.?

Solution: Create a Pure Fabrication object called a Factory that handles the creation.”

The Factory Method pattern allows subclasses to choose the type of object to create. It encourages loose coupling by eliminating the need to include application-specific classes in your code.

Which GRASP pattern/s is related to the factory?

Related to High Cohesion, Pure Fabrication.

Strategy

“Problem: How to design for varying, but related, algorithms or policies? How to design for the ability to change these algorithms or policies?

Solution: Define each algorithm/policy/strategy in a separate class, with a common interface.”

The strategy pattern allows you to indirectly change the behavior of an object at runtime by associating it with different child objects that can perform specific subtasks in different ways. Use this strategy when you have many similar classes that differ only in the way they perform certain behaviors.

Which GRASP pattern/s is related to the strategy?

Related to Polymorphism, Protected Variations.

Composite

“Problem: How to treat a group or composition structure of objects the same way (polymorphically) as a non-composite (atomic) object?

Solution: Define classes for composite and atomic objects so that they implement the same interface.”

Composite pattern is used when we need to treat a group of objects as a single object. Composite patterns assemble objects into tree-like structures to represent parts and entire hierarchies.

Which GRASP pattern/s is related to the composite?

Related to Polymorphism, Protected Variations.

Adapter

“Problem: How to resolve incompatible interfaces, or provide a stable interface to similar components with different interfaces?

Solution: Convert the original interface of a component into another interface, through an intermediate adapter object. “

An interface can be compatible with one of the existing objects through the adapter. The interface created by the adapter can be used by the object to safely call on the methods stored in the adapter.

Which GRASP pattern/s is related to the adapter?

Related to indirection and polymorphism.

Facade

“Problem: A common, unified interface to a disparate set of implementations or interfaces – such as within a subsystem – is required. There may be undesirable coupling to many things in the subsystem, or the implementation of the subsystem may change. What to do?

Solution: Define a single point of contact to the subsystem – a facade object that wraps the subsystem. This facade object presents a single unified interface and is responsible for collaborating with the subsystem components.”

Subsystems are wrapped by a high-level interface created by the façade to make the subsystem easier to use. The learning curve decreases to successfully leverage the subsystem.

Which GRASP pattern/s is related to the facade?

Related to indirection and protected variation.

Observer

“Problem: Different kinds of subscriber objects are interested in the state changes or events of a publisher object, and want to react in their own unique way when the publisher generates an event. Moreover, the publisher wants to maintain low coupling to the subscribers. What to do?

Solution: Define a ”subscriber”or ”listener” interface. Subscribers implement this interface. The publisher can dynamically register subscribers who are interested in an event and notify them when an event occurs.”

The Observer pattern is a software design pattern in which objects maintain a list of their dependent objects and automatically notify them of any state changes, usually by calling one of their methods.

Which GRASP pattern/s is related to the observer?

Related to Polymorphism and protected variation.

2. Problem scenarios

2.1 Scenario 1

The class board is the most compatible when it comes to save/load a game to/from hard drive. The board will act as the information expert because the class knows everything about the board. Adding this functionality to the board, will result in low coupling and violation of the high cohesion pattern. The reason why the high cohesion pattern is violated, is because the main purpose of the class board is to only generate the board, and with adding functionality violates high cohesion. To keep cohesion high is by adding a subclass to the class board that has the purpose of loading/saving the board.

2.2 Scenario 2

The result is to send the same error message to different users depending on the operative systems, for example a Windows user and a Linux user should receive the same error message.

To achieve this, we need the help of polymorphism, the protected variation GRASP patterns, and the strategy GoF pattern. Polymorphism is the pattern connecting all the classes together while strategy is the interface that receives and sends the error message depending on the operative systems. Protected variations pattern, helps the interface to identify the operative system used by the user.

The best solution out of the patterns mentioned above is strategy. When we have multiple algorithms used for a specific task, we can connect them to an interface to make the code more flexible.