Behavioral Design Patterns

Parinya Ekparinya

Parinya.Ek@kmitl.ac.th

Acknowledgement

The content of the following slides are partially based on the listed material as follows

- Object-Oriented Patterns & Frameworks by Dr. Douglas C. Schmidt
- https://en.wikipedia.org/wiki/Strategy_pattern
- https://en.wikipedia.org/wiki/Observer_pattern
- https://en.wikipedia.org/wiki/Command_pattern
- https://en.wikipedia.org/wiki/Template_method_pattern
- https://en.wikipedia.org/wiki/Iterator_pattern
- https://en.wikipedia.org/wiki/Null_Object_pattern

WARNING:

Overuse of design patterns can lead to code that is downright over-engineered. Always go with the simplest solution that does the job and introduce patterns only where the need emerges.

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2008). *Head first design patterns*. " O'Reilly Media, Inc."

Behavioral Design Patterns

- Strategy

- Observer

- Command

- Template Method

- Iterator

- Null Object

Strategy

Strategy: Problem

- How to independently vary algorithm from clients that use it?
- How to defer the decision about which algorithm to use until runtime?

Applicability

- when an object should be configurable with one of many algorithms,
- and all algorithms can be encapsulated,
- and one interface covers all encapsulations

Strategy: Solution

Intent

- Define a family of algorithms, encapsulate each one, & make them interchangeable to let clients & algorithms vary independently.

What solution does the Strategy design pattern describe?

- Define a family of algorithms, encapsulate each one, and make them interchangeable.
- Strategy lets the algorithm vary independently from clients that use it.
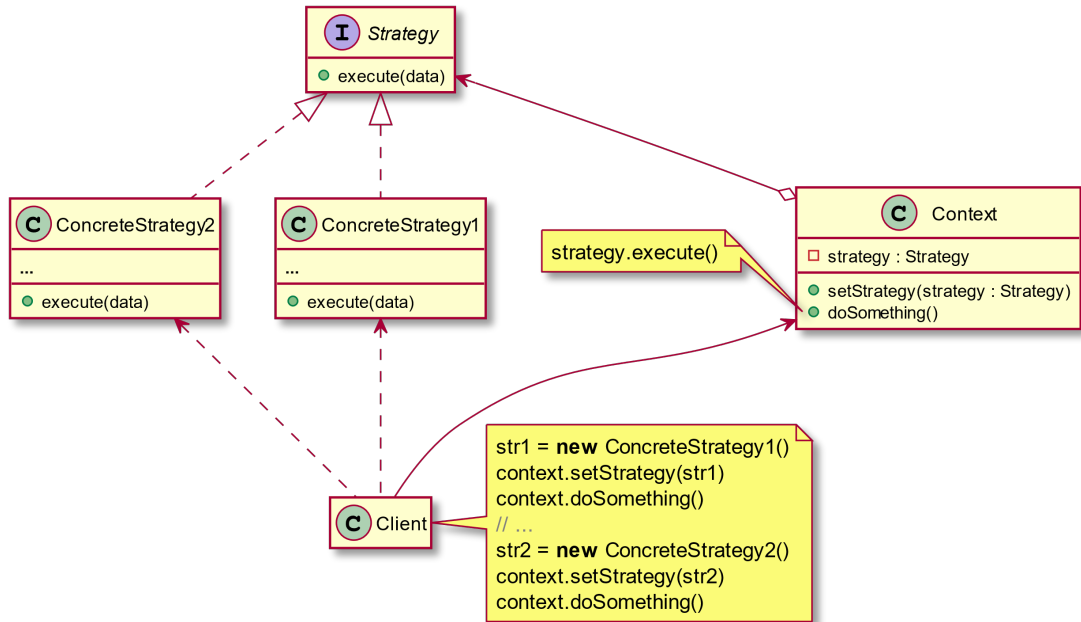
Strategy: Consequences

+ Greater flexibility, reuse

+ Can change algorithms dynamically.

- Strategy creation & communication overhead.

- Inflexible Strategy interface.

- Semantic incompatibility of multiple strategies used together.

Strategy: Examples

- [Strategy (refactoring.guru)](#)
- [Strategy pattern – Wikipedia](#)
- [Strategy Design Pattern in Java - Example Tutorial – JournalDev](#)

Observer

Also known as:
Event-Subscriber, Listener

Observer: Problem

- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state, an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

Applicability

- An abstraction has two aspects, one dependent on the other.
- A change to one object requires changing untold others.
- An object should notify unknown other objects.

Observer: Solution

Intent

- Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated.

What solution does the Observer design pattern describe?

- Define Subject and Observer objects.

- So that when a subject changes state, all registered observers are notified and updated automatically (and probably asynchronously).
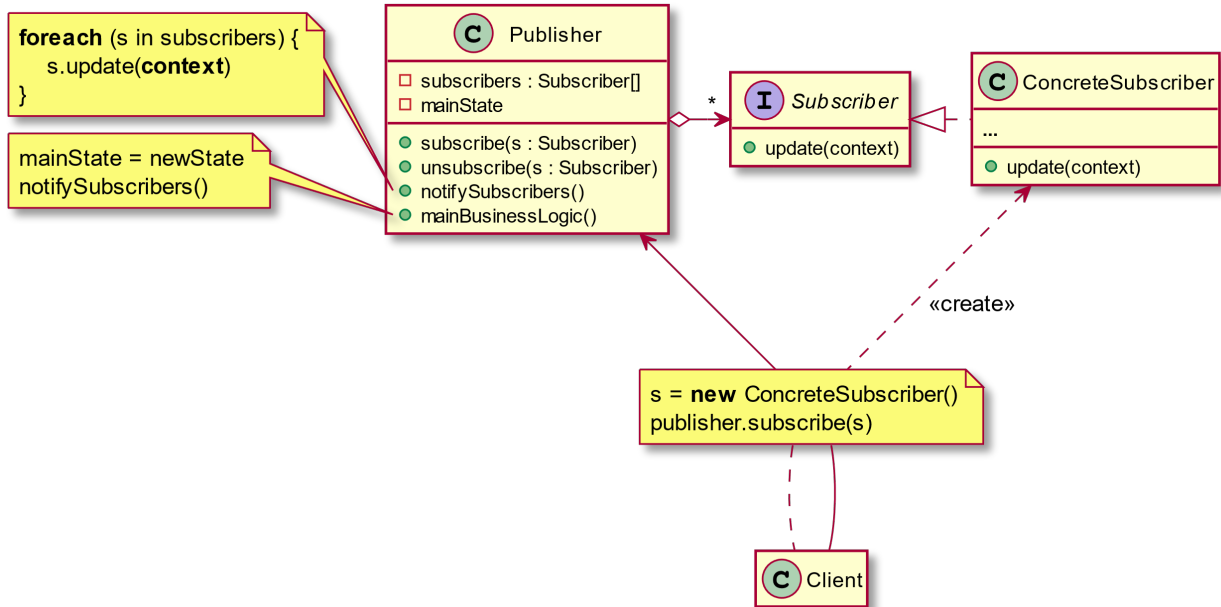
Observer: Consequences

+ Modularity: subject & observers may vary independently.
+ Extensibility: can define & add any number of observers.
+ Customizability: different observers offer different views of subject.
− Unexpected updates: observers don't know about each other.
− Update overhead: might need hints or filtering.

Observer Examples

- Observer (refactoring.guru)
- Observer pattern – Wikipedia
- Observer Design Pattern in Java – JournalDev

Command

Also known as:

Action, Transaction

Command Problem

- Coupling the invoker of a request to a particular request should be avoided. That is, hard-wired requests should be avoided.
- It should be possible to configure an object (that invokes a request) with a request

Applicability
- To parameterize objects with an action to perform
- To specify, queue, & execute requests at different times
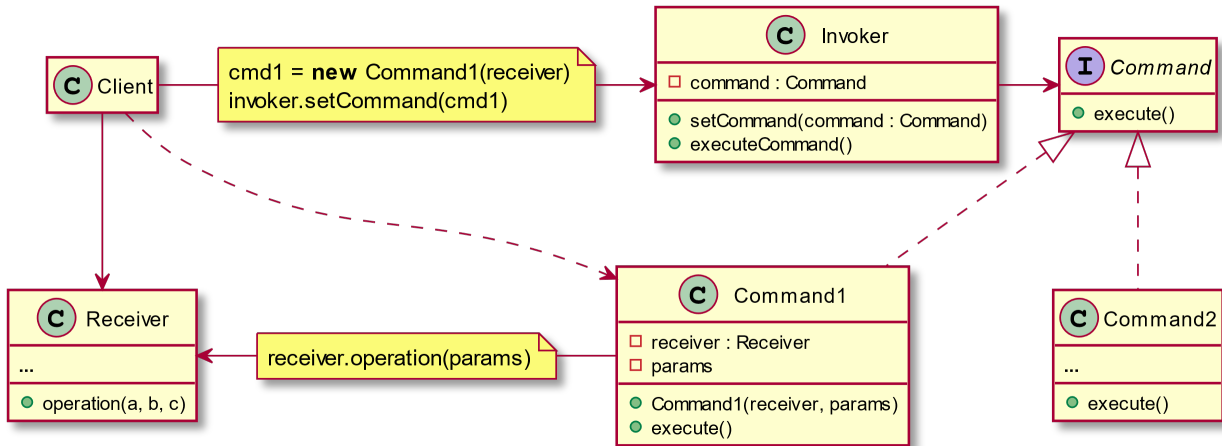- For multilevel undo/redo.

Command: Solution

Intent

- Encapsulate the request for a service.

Using the command design pattern describes the following solution:

- Define separate (command) objects that encapsulate a request.
- A class delegates a request to a command object instead of implementing a particular request directly.

List of Commands = Execution History

Command: Consequences

+ Abstracts executer of a service.

+ Supports arbitrary-level undo-redo.

+ Composition yields macro-commands.

− Might result in lots of trivial command subclasses.

− Excessive memory may be needed to support undo/redo operations.

Command: Examples

- Command (refactoring guru)

- Command pattern – Wikipedia

- Command Design Pattern – JournalDev

Template
Method

Template Method: Problem

- How to define the overall structure of the operation in a base class, but allow subclasses to refine, or redefine, certain steps?

Applicability

- Implement invariant aspects of an algorithm once & let subclasses define variant parts.
- Localize common behavior in a class to increase code reuse.
- Control subclass extensions.

Template Method: Solution

Intent

- Provide a skeleton of an algorithm in a method, deferring some steps to subclasses

This pattern has two main parts:

- The "template method" is implemented as a method in a base class (usually an abstract class). This method contains code for the parts of the overall algorithm that are invariant. The template ensures that the overarching algorithm is always followed.

- Subclasses of the base class "fill in" the empty or "variant" parts of the "template" with specific algorithms that vary from one subclass to another.
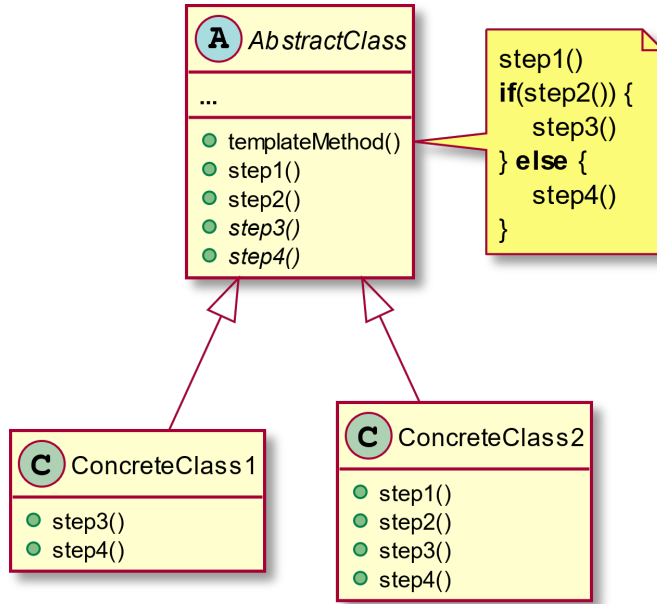
Template Method Consequences

- Leads to inversion of control ("Hollywood principle": don't call us – we'll call you).
+ Promotes code reuse.
+ Lets you enforce overriding rules.
– Must subclass to specialize behavior (cf. Strategy pattern).

Template Method: Examples

- [Template Method (refactoring.guru)](#)
- [Template method pattern – Wikipedia](#)
- [Template Method Design Pattern in Java – JournalDev](#)

Iterator

Iterator: Problem

- The elements of an aggregate object should be accessed and traversed without exposing its representation (data structures).
- New traversal operations should be defined for an aggregate object without changing its interface.

Applicability

- Require multiple traversal algorithms over an aggregate
- Require a uniform traversal interface over different aggregates
- When aggregate classes & traversal algorithm must vary independently

Iterator: Solution

Intent

- Access elements of an aggregate (container) without exposing its representation

What solution does the Iterator design pattern describe?

- Define a separate (iterator) object that encapsulates accessing and traversing an aggregate object.

- Clients use an iterator to access and traverse an aggregate without knowing its representation (data structures).
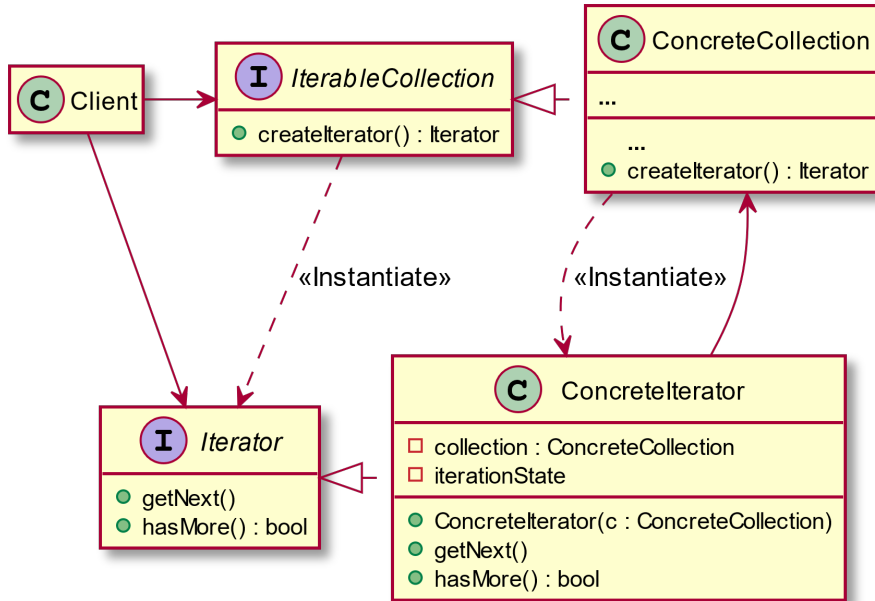
Iterator: Consequences

+ Flexibility: aggregate & traversal are independent.

+ Support multiple iterators & multiple traversal algorithms.

– Incur additional communication overhead between iterator & aggregate. This is particularly problematic for iterators in concurrent or distributed systems.

Iterator: Examples

- Iterator (refactoring.guru)
- Iterator pattern – Wikipedia
- Iterator Design Pattern in Java – JournalDev

Null Object
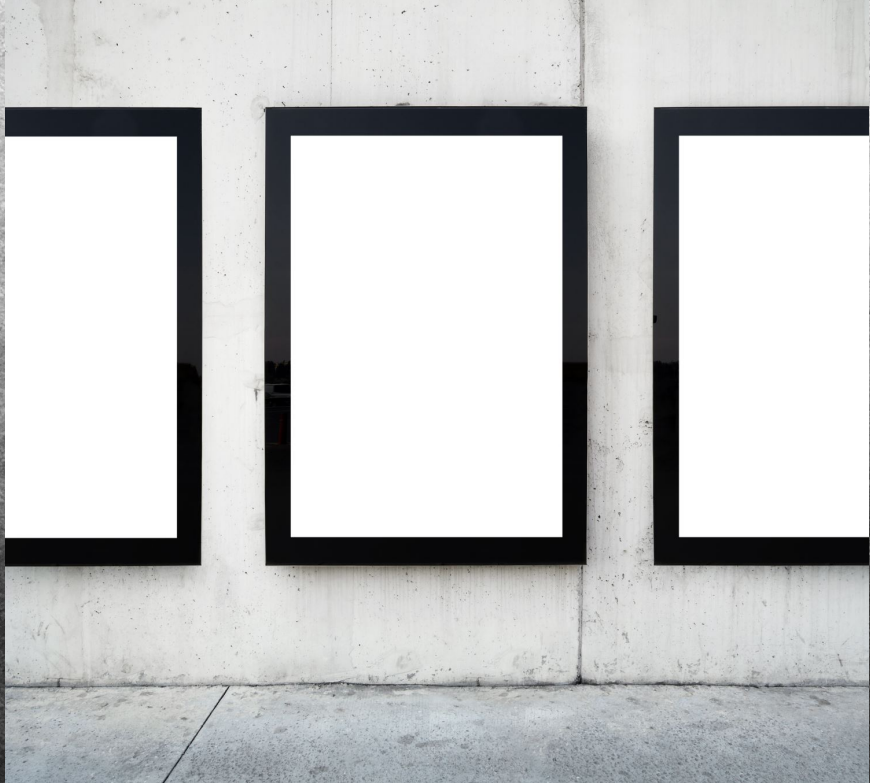
Null Object Problem

- In most object-oriented languages, references may be null.
- These references need to be checked to ensure they are not null before invoking any methods, because methods typically cannot be invoked on null references.
- How can the absence of an object — the presence of a null reference — be treated transparently?

Applicability

- An object requires a collaborator.
- Some collaborator instances should do nothing.
- You want to abstract the handling of null away from the client.

Null Object Solution

Intent

- Provide an object as a surrogate for the lack of an object of a given type.

What solution does the Null Object design pattern describes?

- The uses of a Null Object, an object with no referenced value or with defined neutral ("null") behavior.

- The behavior (or lack thereof) of such objects.
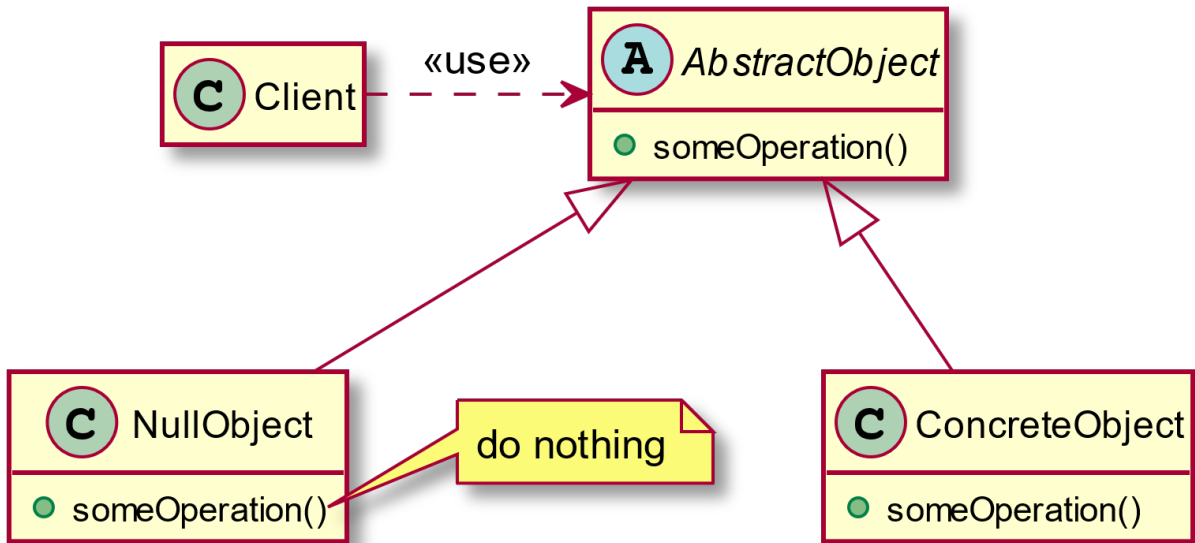
Null Object: Consequences

Simplifies client code, because it avoids having to write testing code which handles the null collaborator specially.

Can be difficult to implement if various clients do not agree on how the null object should do nothing as when your AbstractObject interface is not well defined.

Null Object Examples

- Null object pattern – Wikipedia
- Null Object Design Pattern (sourcemaking.com)