The Strategy and The Iterator Design Patterns

Hui Chen [a]

[a]CUNY Brooklyn College, Brooklyn, NY, USA

April 22, 2025

Outline

Outline

Software Design

- Design starts mostly from/with requirements – evolving mostly from functionalities and other non-functional characteristics
    - In the waterfall model Design generally occurs after Requirements
    - In agile, design is performed during in each iteration
- To answer: How is the software solution going to be structured?
    - What are the main components – (functional composition) often directly from requirements' functionalities (e.g., use cases, user stories, scenarios)
    - How are these components related? – Possibly re-organize the components (composition/decomposition)
- Two main levels of design:
    - Architectural (high level) design
    - Detailed design
    - Different design concerns at different abstraction levels (e.g. classes vs. modules vs. entire system)
- How should we depict design – what notation/language?

Review: High-level and Low-level Designs

Architectural design (high-level design) patterns and styles

- MVC, Layered, Pipeline, Client-Server, SOA, . . .

Detailed design (low-level design)

- Functional decomposition, database design, Object-Oriented design, user-interface design, . . .
- Object-Oriented Design and UML – focused on modeling
- To discuss more about Object-Oriented design

Outline

Strategy Design Pattern

Source: Module 2 by Martin Robillard

- the Review `Comparator` interface
- the Strategy Design Pattern

Java Interface Revisited

Let's consider Java interface `Comparator`

- How is it defined?
- How can it be used?

The Comparator Interface

Let's consider Java interface Comparator

- How is it defined?

```
interface Comparator<T> {
  int compare(T obj1, T obj2)
}
```

- How can it be used?

```
sort(List<T> list, Comparator<? super T> c)
```

The Comparator Interface

Let's consider Java interface Comparator

- How is it defined?

```
interface Comparator<T> {
  int compare(T obj1, T obj2)
}
```

- "*Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.*"
- How can it be used?

```
sort(List<T> list, Comparator<? super T> c)
```

Sorting often requires a comparator specific for a type – e.g. sorting instances of type Card, requires a Comparator for playing cards

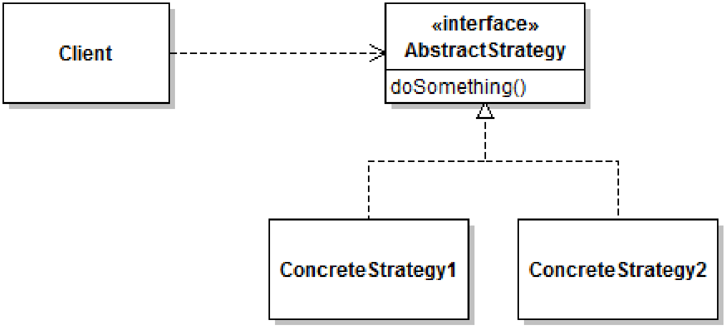
How is it designed?

How does the design come to be?

`Comparator` epitomizes a design pattern, called the Strategy design pattern.

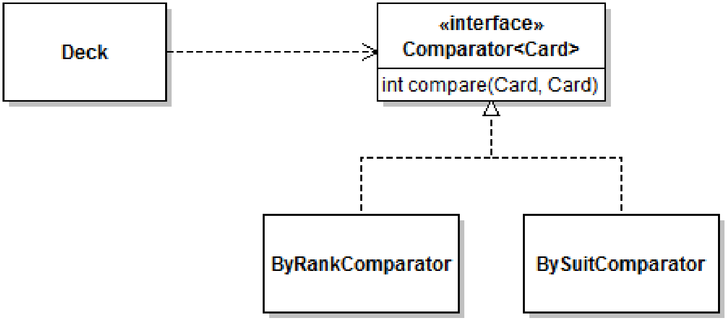
The Stategy Design Pattern

"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithms vary independently of clients that use it." – from the *Gang of Four Book*



The Strategy Design Pattern: Example

Consider that we need to sort a deck of cards



Let's discuss,

1. Does this design have Extensibility?
2. Does this design have Loose Coupling?

Outline

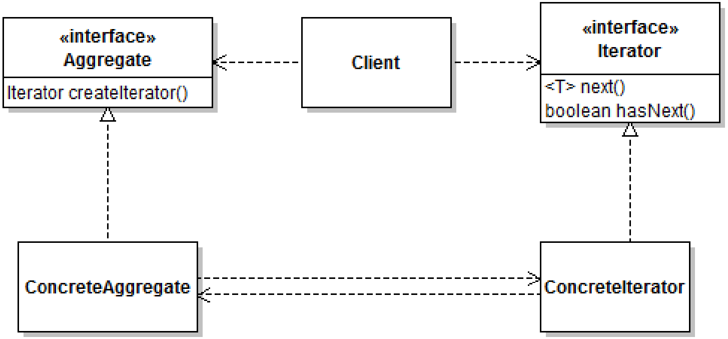
Recap: Violations of Class Design Guidelines

```java
public class Deck
{
   // violates 1. public -> no door to guide the data field
   public Stack<Card> aCards = new Stack<>();

   // violates 4. return reference to a class variable -> font door is open
   public Stack<Card> getCards()
   { return aCards; }

   /* violates 2 and 3. set a reference to a class variable; but caller
      keeps a reference -- back door open because caller has a reference
      to containing object */
   public void setStack(Stack<Card> pCards)
   { aCards = pCards; }

   /* violates 3. set a reference to a class variable; but caller
      keeps a reference -- back door open because caller has a reference
      to containing object */
   public void applyAll( List<Stack<Card>> pTaskList )
   { pTaskList.add(aCards); }
}
```

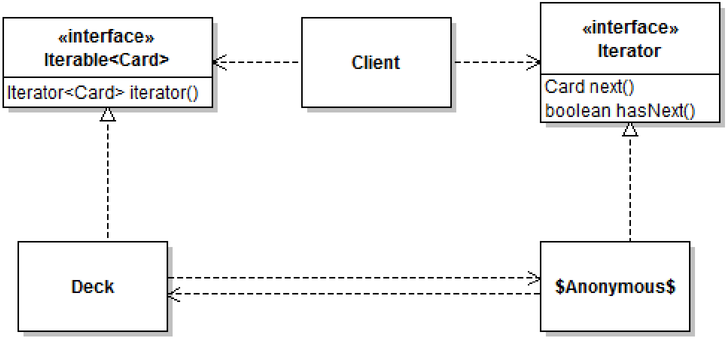
Perhaps, the Deck class was ill-conceptualized ... what we really want to iterate over the deck of cards, then ...

The Iterator Design Pattern

“Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation” – from the *Gang of Four book*



The Iterator Design Pattern: Example



With this design, we do not expose its underlying representation of the "state".

Summary and Questions?

- The Strategy design pattern
- The Iterator design pattern
- Questions?

Outline

"Introduction to Software Design with Java" by Martin P. Robillard
"Engineering Software as a Service" by Armando Fox and David Patterson (2nd Edition)
"Essentials of Software Engineering" by Frank Tsui, Orlando Karam, and Barbara Bernal(4th Edition)