

The Strategy, Iterator, and Singleton Design Patterns

Hui Chen ^a

^aCUNY Brooklyn College, Brooklyn, NY, USA

April 26, 2022

Outline

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern
- 6 References

Outline

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern
- 6 References

Project Meeting

Before next project iteration, each group should schedule a meeting with me in this or the next week – more scheduling details will be on Blackboard.

Agenda and Objectives

- ▶ Plan for next iteration
- ▶ Discuss group and individual progress
- ▶ Identify gaps and improvements
- ▶ Any issues you may have regarding the class

Outline

- 1 Project Meeting
- 2 Background**
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern
- 6 References

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

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern**
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern
- 6 References

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?

```
1 interface Comparator<T> {  
2     int compare(T obj1, T obj2)  
3 }
```

- ▶ How can it be used?

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

The Comparator Interface

Let's consider Java interface Comparator

▶ How is it defined?

```
1 interface Comparator<T> {
2     int compare(T obj1, T obj2)
3 }
4
5 ““{\em Compares its two arguments for order. Returns a
        negative integer, zero, or a
6 positive integer as the first argument is less than,
        equal to, or greater than
7 the second.}””
```

▶ How can it be used?

```
1 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

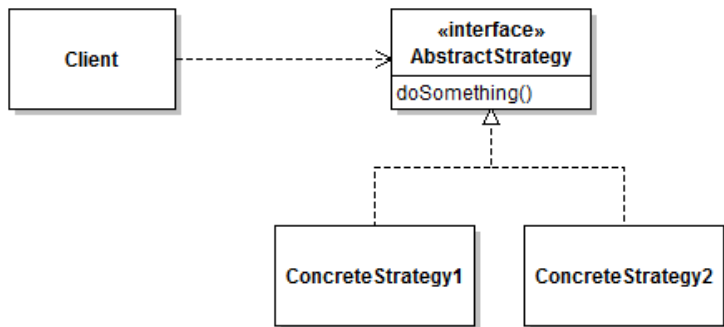
How is it designed?

How does the design come to be?

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

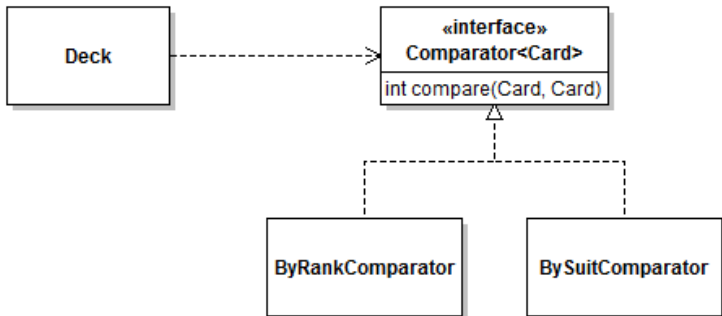
The Strategy Design Pattern

“Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithms vary independently from 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

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern**
- 5 Singleton Pattern
- 6 References

Recap: Violations of Class Design Guidelines

```

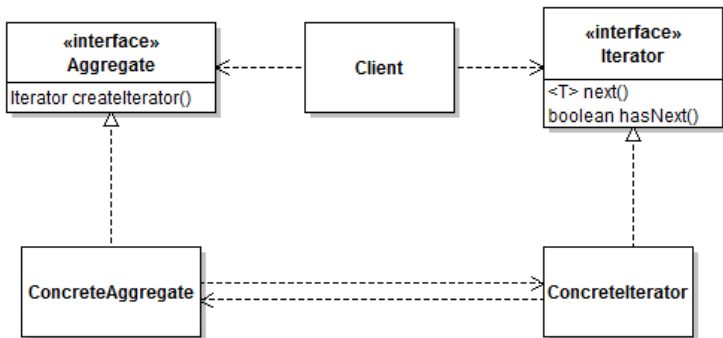
1 public class Deck
2 {
3     // violates 1. public -> no door to guide the data field
4     public Stack<Card> aCards = new Stack<>();
5
6     // violates 4. return reference to a class variable -> front door
       is open
7     public Stack<Card> getCards()
8     { return aCards; }
9
10    /* violates 2 and 3. set a reference to a class variable; but
       caller
11       keeps a reference -- back door open because caller has a
       reference
12       to containing object */
13    public void setStack(Stack<Card> pCards)
14    { aCards = pCards; }
15
16    /* violates 3. set a reference to a class variable; but caller
       keeps a reference -- back door open because caller has a
17       reference
18       to containing object */
19    public void applyAll( List<Stack<Card>> pTaskList )
20    { pTaskList.add(aCards); }
21 }

```

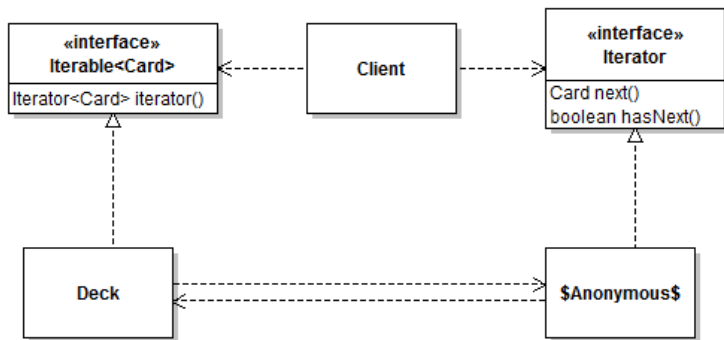
Perhaps, the Deck class was ill-conceptualized ... what we really want to iterate over the deck of cards, then ...

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”.

Outline

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern**
- 6 References

The Singleton Design Pattern: Motivation

Singleton: Ensuring there's only one of something

- ▶ Technically, a class that provides only 1 instance, which anyone can access

A static class? But it would be nicer if we can instantiate something – use the singleton pattern.

- ▶ It provides a principled way to ensure that there is only one instance of a given class as any point in the execution of a program.
- ▶ It is useful to simplify the access to stateful objects that typically assume the role of a *controller* of some sort.
 - ▶ e.g., controller in MVC?

The Singleton Design Pattern: Implementation

- ▶ A private constructor for the Singleton, so clients cannot create duplicate objects;
- ▶ A static final field keeping a reference to the single instance of the singleton object.
- ▶ A static accessor method, usually called `instance()`, that returns the unique instance of the Singleton.

The Singleton Design Pattern: Example

```
1 public class Controller {
2     // static final field referencing to an instance of this
      class
3     private final static Controller _instance = new Controller()
      ;
4
5     // static accessor method
6     public static Controller instance() {
7         return _instance;
8     }
9
10    // constructor is private
11    private Controller() {
12        // Initialize members here, like various views.
13    }
14 }
```

Summary and Questions?

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

Outline

- 1 Project Meeting
- 2 Background
- 3 Strategy Pattern
 - Recap: Comparator
 - The Strategy Design Pattern
- 4 Iterator Pattern
- 5 Singleton Pattern
- 6 References

- “[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)