The Singleton Design Pattern

Hui Chen [a]

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

April 10, 2025

Outline

1. Background
2. Singleton Pattern
3. References

Outline

1 Background

2 Singleton Pattern

3 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 Background

2 Singleton Pattern

3 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., contoller 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 }
```

Singleton vs. Static Class

Instead of a singleton, can we just use a class with static members (variables and methods)?

Singleton vs. Static Class

- A singleton is a class that can be instantiated, but only once vs. a static class cannot be instantiated at all
- A singleton can be subclassed, while a static class cannot
- A singleton can be passed around like any other object, while a static class cannot
- A singleton can implement interface, but the static class cannot

Summary and Questions?

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

Outline

1 Background

2 Singleton Pattern

3 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)