ITSS SOFTWARE DEVELOPMENT

12. DESIGN PATTERNS

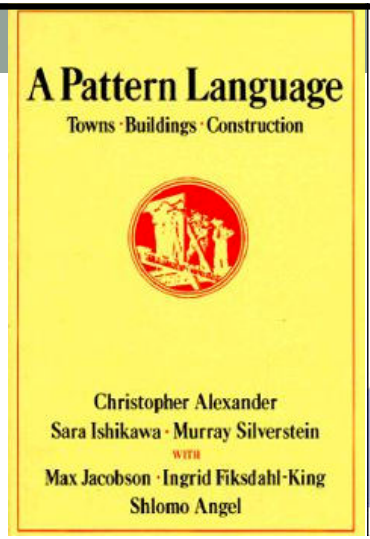
Nguyen Thi Thu Trang

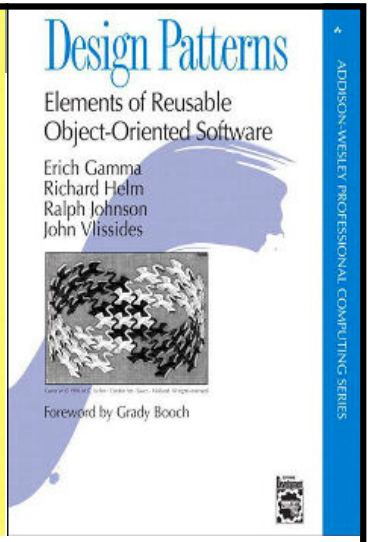
trangntt@soict.hust.edu.vn

Content

1. What are design patterns?
2. How to describe a design pattern?
3. Design pattern categories

Design Patterns





- Published in 1994
- "Each pattern describes a **problem** which occurs over and over **again** in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution **a million times over**, without ever doing it the same way twice"
  - Christopher Alexander
- Today's amazon.com stats

Amazon Best Sellers Rank: #2,069 in Books (See Top 100 in Books)
#1 in Books > Computers & Internet > Computer Science > Software Engineering > **Design Tools & Techniques**
#1 in Books > Computers & Internet > Programming > Software Design, Testing & Engineering > **Software Reuse**
#3 in Books > Nonfiction > Foreign Language Nonfiction > **French**

What and why design patterns?

- A standard solution to a common programming problem
  - a design or implementation structure that achieves a particular purpose
  - a high-level programming idiom
- A technique for making code more flexible
  - reduce coupling among program components
- Short-hand for describing program design
  - a description of connections among program components (static structure)
  - the shape of a heap snapshot or object model (dynamic behaviour)

Whence design patterns?

- The Gang of Four (GoF)
  - Gamma, Helm, Johnson, Vlissides
- Each an aggressive and thoughtful programmer
- Empiricists, not theoreticians
- Found they shared a number of “tricks” and decided to codify them – a key rule was that nothing could become a pattern unless they could identify at least three real examples

An example of a GoF pattern

- Given a class A, what if you want to guarantee that there is precisely one instance of A in your program? And you want that instance globally available?
  - First, why might you want this?
  - Second, how might you achieve this?

Implementing Singleton

- Make constructor(s) **`private`** so that they can not be called from outside by clients.
- Declare a single **`private static`** instance of the class.
- Write a public **`getInstance()`** or similar method that allows access to the single instance.
  - May need to protect / synchronize this method to ensure that it will work in a multi-threaded program.

Several solutions

```
public class Singleton {
  private static final Singleton instance;
  // Private constructor prevents instantiation from other classes
  private Singleton() {
    instance = new Singleton();
  }
  public static Singleton getInstance() {
    return instance;
  }
}
```

Eager allocation of instance

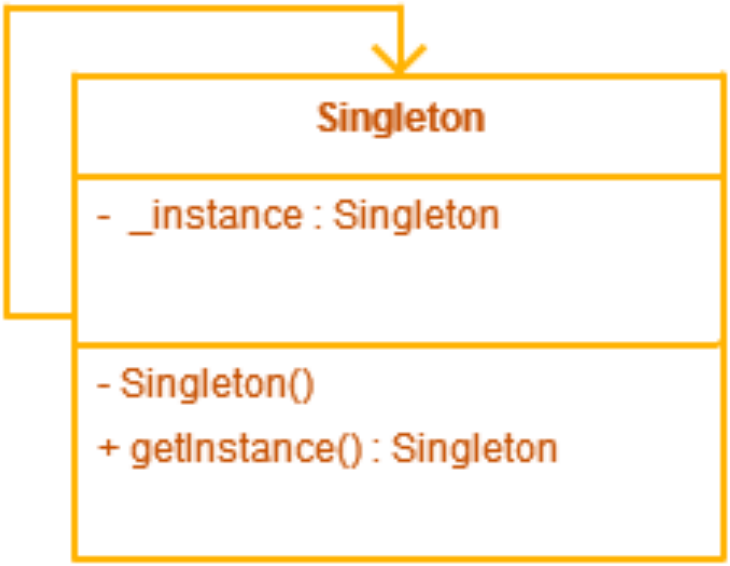
```
public class Singleton {
  private static Singleton _instance;
  private Singleton() { }
  public static synchronized Singleton getInstance() {
    if (null == _instance) {
      _instance = new Singleton();
    } return _instance;
  }
}
```

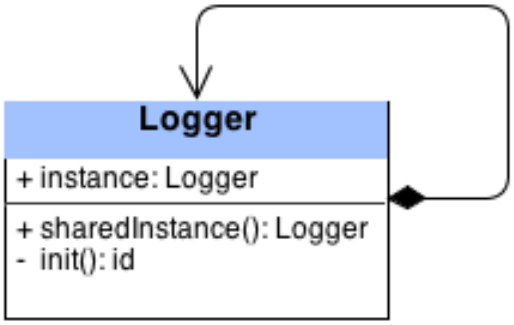
Lazy allocation of instance

Possible reasons for Singleton

- One **`RandomNumber`** generator
- One **`Restaurant`**, one **`ShoppingCart`**
- One **`KeyboardReader`**, etc…
- Make it easier to ensure some key invariants
- Make it easier to control when that single instance is created – can be important for large objects
- …

Singleton design patterns – diagram





Content

1. What are design patterns?
2. How to describe a design pattern?
3. Design pattern categories

Main elements of a design pattern

- *Pattern Name:*
  - A common name to talk about
- *Problem:*
  - Context: when to apply the pattern
  - May include a list of conditions for applying the pattern
- *Solution:*
  - Abstract description of a design problem and how a general arrangement of elements solves it
  - Elements making up the design, their relationships/responsibilities and collaborations
  - Like a template, language-neutral
- *Consequences:*
  - Results and tradeoff of applying patterns
  - Impacts on system's flexibility, extensibility or portability

Describing a pattern

1. Name and classification
2. Also Known As
3. Motivation
4. Applicability
5. Structure
6. Participants
7. Collaboration
8. Consequences
9. Implementations
10. Sample Code
11. Known Uses
12. Related Patterns

Describing for Singleton pattern?

Singleton pattern

- **Singleton**: An object that is the only object of its type
  (one of the most known / popular design patterns)
  - Ensuring that a class has at most one instance.
  - Providing a global access point to that instance.
    - e.g. Provide an accessory method that allows users to see the instance.

- *Advantages:*
  - Takes responsibility of managing that instance away from the programmer (illegal to construct more instances).
  - Saves memory.
  - Avoids bugs arising from multiple instances.

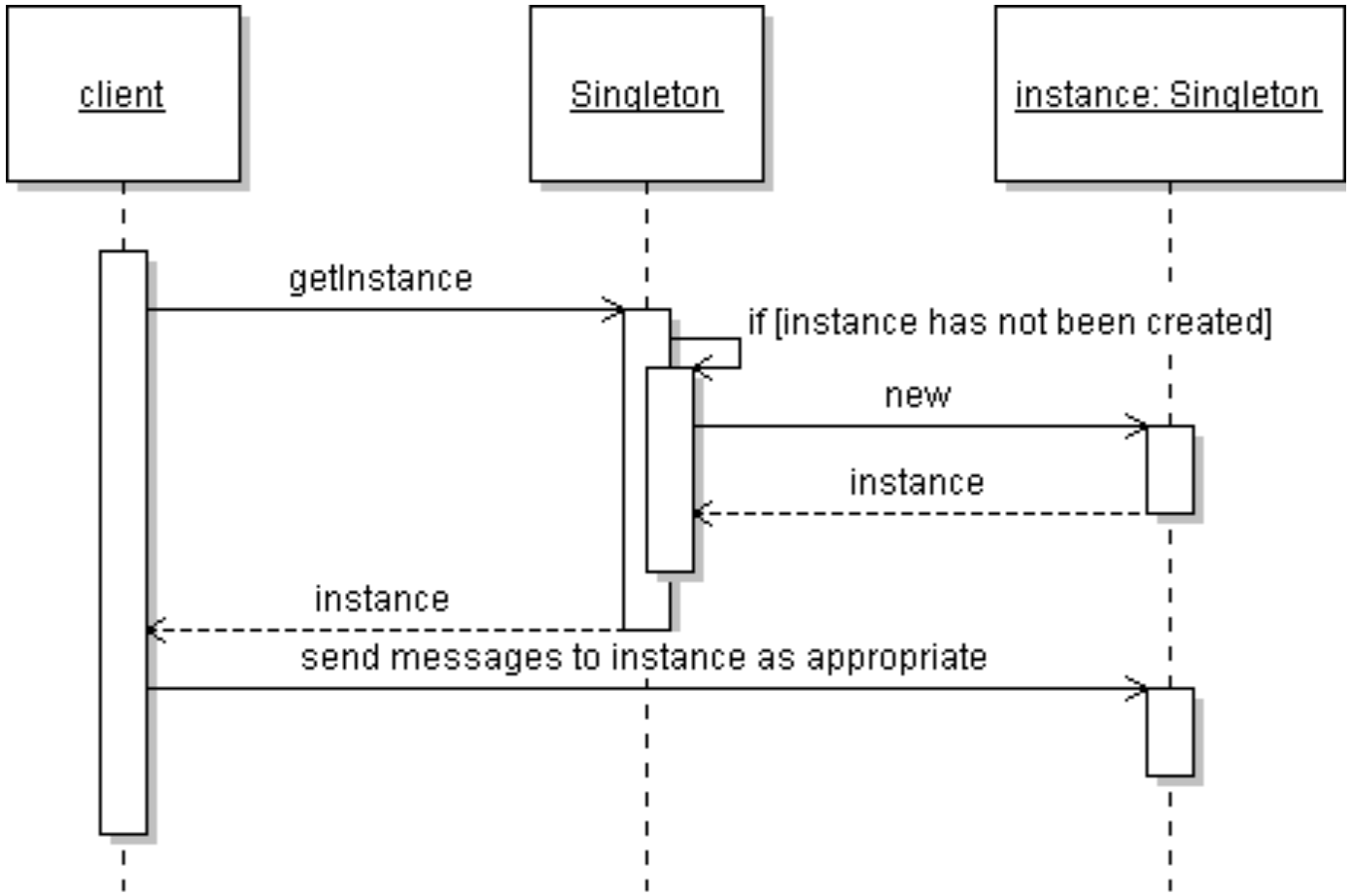
Restricting objects

- One way to avoid creating objects: use static methods
  - Examples: `Math`, `System`
  - Is this a good alternative choice? Why or why not?

- *Disadvantage*: Lacks flexibility
  - Static methods can't be passed as an argument, nor returned.

- *Disadvantage*: Cannot be extended
  - Example: Static methods can't be sub-classed and overridden like an object's methods could be.

Singleton sequence diagram



Singleton example

- Class **`RandomGenerator`** generates random numbers.

```
public class RandomGenerator {
    private static final RandomGenerator gen =
                new RandomGenerator();

    public static RandomGenerator getInstance() {
        return gen;
    }

    private RandomGenerator() {}

    ...
}
```

Lazy initialization

- Can wait until client asks for the instance to create

```
public class RandomGenerator {
    private static RandomGenerator gen = null;

    public static RandomGenerator getInstance() {
        if (gen == null) {
            gen = new RandomGenerator();
        }
        return gen;
    }

    private RandomGenerator() {}

    ...
}
```

Singleton Comparator

- Comparators make great singletons because they have no state:

```
public class LengthComparator
        implements Comparator<String> {
    private static LengthComparator comp = null;

    public static LengthComparator getInstance() {
        if (comp == null) {
            comp = new LengthComparator();
        }
        return comp;
    }

    private LengthComparator() {}

    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

Content

1. What are design patterns?
2. How to describe a design pattern?
3. Design pattern categories

GoF patterns: three categories

- *Creational Patterns* – these abstract the object-instantiation process
  - **`Factory Method, Abstract Factory, Singleton, Builder, Prototype`**
- *Structural Patterns* – these abstract how objects/classes can be combined
  - **`Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy`**
- *Behavioral Patterns* – these abstract communication between objects
  - **`Command, Interpreter, Iterator, Mediator, Observer, State, Strategy, Chain of Responsibility, Visitor, Template Method`**

Design Patterns classification

Purpose / Scope	Creation	Structure	Behaviour
Class	*Factory method*	Adapter (class)	Interpreter, Template Method
Object	*Abstract Factory*	Adapter (object)	Chain of Responsibility
	Builder	Bridge	Command
	Prototype	Composite	Iterator
	Singleton	Decorator	Mediator
		Façade	Memento
		Flyweight	Observer
		Proxy	State, Strategy, Visitor

Presentation of a Design Pattern (DP)

- Name – Alias: Tên, tên gọi khác
- Classification: Phân loại
- Intent: Mục đích
- Motivation: Khi nào cần sử dụng mẫu này
  - Bài toán đặt ra
  - Giải pháp nếu không dùng DP (nếu có)
- Solution: Giải pháp khi dùng DP (ví dụ và tổng quát)
  - Biểu đồ lớp / Biểu đồ tương tác
  - Mã nguồn minh hoạ
- Pros and cons
  - Phân tích ưu nhược điểm khi sử dụng DP này
- Applicability
  - Các ví dụ ứng dụng trong thực tế, đặc biệt những ví dụ phổ biến