Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

Introduction to Design Patterns

Introduction to Design Patterns

- There are many ways to deal with **software design problems**, but more **flexible** or **reusable** solutions are preferred in the industry.

- One of these preferred methods is that of **design patterns**.

Introduction to Design Patterns (contd.)

- A design pattern is a **practical proven solution** to a recurring design problem.
- It **allows you to use previously outlined solutions that expert developers have often used to solve a software problem**, so that you do not need to build a solution from the basics of object-oriented programming principles every time.
- These solutions are **not just theoretical**—they are **actual solutions used in the industry**.
- Design patterns may also be **used to help fix tangled, structureless software code**, also known as "spaghetti code."

Introduction to Design Patterns (contd.)

- It is not always obvious which design pattern to use to solve a software design problem, especially as many design patterns exist.

- **Practice and experience** will make you better at selecting which design patterns to use for different problems. Some people even become **design experts** who know the design patterns to use in solving particular software design problems!

Introduction to Design Patterns (contd.)

- It is important to understand that design patterns are **not just a concrete set of source code that you memorize and put into your software**, like a Java library or framework.
- Instead, **design patterns are more conceptual**. They are knowledge that you can apply within your software design, to guide its structure, and make it more flexible and reusable.
- Design patterns **help software developers** so that they have a guide to help them solve design problems the way an expert might, so not everything needs to be built from scratch.
- Design patterns **serve almost like a coach to help developers** reach their full potential!

Introduction to Design Patterns (contd.)

- A **strong advantage** of design patterns is that they have already been **proven by experts**.
  - This means that you do not need to go through the trials they have, and you go straight to creating better written software.

- Another **advantage** of design patterns is that they **help create a design vocabulary**.
  - This means that design patterns provide a simplified means of discussing design solutions, so that they do not need to explained over and over.

    For example,

    design patterns might give **patterns names**, making it easier to discuss them. This saves time, and ensures that everyone is referring to the same pattern. This leaves less room for misunderstanding

Gang of Four's Pattern Catalogue

- The four authors of the famous book **Design Patterns: Elements of Reusable Object-Oriented Software**—Gamma, Helm, Johnson, and Vlissides—have been collectively nicknamed the **Gang of Four**.
- These authors wrote their book based on their own experiences as developers. When each had developed programs and graphical applications, they had discovered patterns emerging in their design solutions.
- They formalized those patterns into a reference, which became their book.
- The patterns developed by the Gang of Four were organized to be readable, and often named after their purpose.
- The grouping of patterns together forms a catalog, otherwise known as the **Gang of Four's design pattern catalog**.

Categories of Patterns

- The Gang of Four's pattern catalog contains twenty-three (23) patterns.
- These patterns can be sorted into three different categories:
  - Creational patterns,
  - Structural patterns, and
  - Behavioral patterns
- These categories were used to simply organize and characterize the patterns in their book.

Creational Patterns

- Creational patterns deal with the **creation or cloning new objects**.
- Cloning an object occurs when you are creating an object that is similar to an existing one, and instead of instantiating a new object, you clone existing objects instead of instantiating them.
- The different ways of creating objects will greatly influence how a problem is solved. Different languages therefore impact what patterns are possible to use.

Structural Patterns

- Structural patterns describe **how objects are connected to each other**. These patterns relate to the design principles of **decomposition** and **generalization**.

- Structural patterns use these relationships and describe how they should work to achieve a particular design goal. Each structural pattern **determines the various suitable relationships among the objects**.

- A good metaphor for considering structural patterns is that of pairing different kinds of foods together: flavor determines what ingredients can be mixed together to form a suitable relationship.

Behavioral Patterns

- Behavioral patterns focus on **how objects distribute work**, and describe **how each object does a single cohesive function**.

- Behavioral patterns also focus on **how independent objects work towards a common goal**.

- A good metaphor for considering behavioral patterns is that of a race car pit crew at a track. Every member of the crew has a role, but together they work as a team to achieve a common goal. Similarly, a behavioral pattern lays out the overall goal and purpose for each object.

Gang of Four's Pattern Catalog

Creational	Structural	Behavioral
Abstract Factory	Adapter	Chain of Responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

Gang of Four's Pattern Catalog

Creational	Structural	Behavioral
Abstract Factory	Adapter	Chain of Responsibility
Builder	Brideg	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

Singleton Pattern

Singleton Pattern: Intent

- Ensure a class only has one instance, and provide a global point of access to it.

Singleton Pattern: Applicability

- Use the Singleton pattern when
  - there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.

Singleton Pattern

- A singleton is **a creational pattern**, which describes **a way to create an object**. It is a powerful technique, but also one of the simplest examples of a design pattern.

- In a singleton design pattern only has one object of a class.

- Another **goal** of singleton design pattern is that **the single object is globally accessible within the program**.

Singleton Pattern (contd.)

- In order to implement a singleton design pattern, best practice is to build the **one and only one** goal into the class itself, so that creating another instance of a Singleton class is not even possible.

- Let us examine this in code.

- If a class has a `public` constructor, an object of this class can be instantiated at any time.

```
//class NotSingleton
public class NotSingleton
{
    //public constructor
    public NotSingleton()
    {

    }
}
```

Singleton Pattern (contd.)

```
public class ExampleSingleton
{
// lazy construction
// the class variable is null if no instance is instantiated
    private static ExampleSingleton uniqueInstance = null;

    private ExampleSingleton()
    {
    }
    // lazy construction of the instance
    public static ExampleSingleton getInstance()
    {
        if (uniqueInstance == null)
        {
            uniqueInstance = new ExampleSingleton();
        }
        return uniqueInstance;
    }
}
```

Singleton Pattern (contd.)

- In the example, **the regular constructor is hidden**.
- Other classes are forced to call the public `getInstance` method. This puts in place basic gatekeeping, and **ensures that only one object of this class is created**.
- The same method can be used to globally reference the single object if it has already been created.

Singleton Pattern (contd.)

- An **advantage** of this version of a Singleton class is **lazy creation**.
  - Lazy creation means that the object is not created until it is truly needed. This is helpful, especially if the object is large. As the object is not created until the `getInstance` method is called, the program is more efficient.

- There are **trade-offs** to the Singleton design principle.
  - If there are multiple computing threads running, there could be issues caused by the threads trying to access the shared single object.

Singleton Pattern (contd.)

- In real use, there may be **variations of how Singleton is realized**, as design patterns are **defined by purpose and not exact code**.
- The intent of a Singleton pattern is **to provide global access to a class that is restricted to one instance**.
- In general, **this is achieved by having a private constructor**, with a `public` method that instantiates the class "if" it is not already instantiated.

It's Quiz Time

1. ..... patterns describe how objects are connected to each other. These patterns relate to the design principles of decomposition and generalization.
   1.1 Creational
   1.2 Structural
   1.3 Behavioral

2. A goal of singleton design pattern is that the single object is globally accessible within the program. (True or False)

3. If a class has a `public` constructor, an object of this class cannot be instantiated at any time. (True or False)