Software Architecture & Design SEC3071

Lecture No. 32

Muhammad Shahid
Department of Computer Science
National Textile University

shahid.abdullah@hotmail.com

Last Lecture Review

- Introduction to Design Patterns
- Brief History of Design Patterns
- What are Patterns?
- What are Design Patterns?
- Essence of Design Patterns
- Design Patterns Space
- How Patterns are Described

Agenda – What will you Learn Today?

Creational Design Patterns

Singleton Design Pattern

Creational Design Patterns

Design Patterns Space

- **Creational Patterns**
  - Deal with initializing and configuring of classes and objects
- **Structural Patterns**
  - Deal with decoupling interface and implementation of classes and objects
- **Behavioral Patterns**
  - Deal with dynamic interactions among societies of classes and objects

Design Patterns Space

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

Design Patterns Space – Scope

- **Class**: Defined through inheritance between classes

- **Object**: Dynamically, defined through associations between objects

How Patterns are Described

- The GoF book describes a pattern using the following four attributes:

1. **Name**: The name to describes the pattern, its solutions and consequences in a word or two
2. **Problem**: Describes when to apply the pattern
3. **Solution**: Describes the elements that make up the design, their relationships, responsibilities, and collaborations
4. **Consequences**: The results and trade-offs in applying the pattern

Creational Design Patterns

- Deal with one of the most commonly performed tasks in an OO application, the **creation of objects**
- Support a **uniform**, **simple**, and **controlled** mechanism to create objects
- Allow the encapsulation of the details about what classes are instantiated and how these instances are created
- Encourage the use of interfaces, which reduces coupling

Singleton Design Pattern

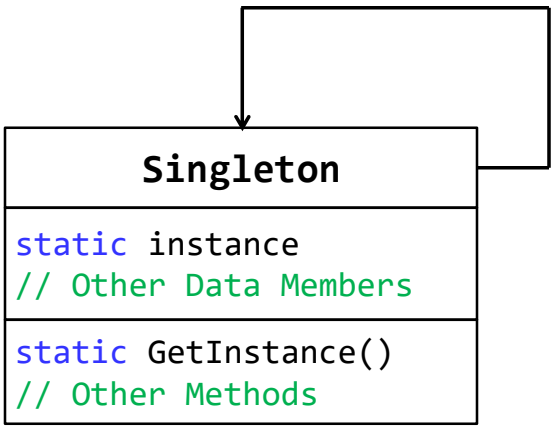
Singleton Design Pattern

- Sometimes, there may be a need to have one and only one instance of a given class during the lifetime of an application
- In environments where resources are rare, you may need to restrict how many counted instances of a class are created
- **Examples**: database connection, registry values etc.

Singleton Design Pattern - Definition

- **Singleton Design Pattern** ensures that there is only one instance of a class and provides global point of access to it.

Singleton Pattern – Class Diagram



Code Implementation with Changes

```
class Singleton
{
    private static Singleton instance;
    private Singleton() {
        Console.WriteLine("Created Successfully!!!");
    }
    public static Singleton GetInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        else {
            Console.WriteLine("Not allowed!!!");
        }
        return instance;
    }
} // End of Singleton class
```


Singleton Design Pattern

```
static void Main(string[] args)
{
      Singleton.GetInstance();

}
```

```
Created Successfully!!!
```


Singleton Design Pattern

```
static void Main(string[] args)
{
      Singleton.GetInstance();
      Singleton.GetInstance();
}
```

```
Created Successfully!!!
Not Allowed!!!
```


Singleton in Multithreaded Environment

Singleton in Multithreaded Environment

- There is a problem with the code above in a multithreaded environment
- Two threads might get a hold of two different instances if they enter at same time in the `GetInstance()` method

Singleton in Multithreaded Environment

```
public class Singleton {
        private static Singleton instance;
        private static readonly object obj = new object();

        private Singleton() { // Some code}

        public static Singleton GetInstance {
                lock (obj) {
                        if (instance == null) {
                                instance = new Singleton();
                        }
                return instance;
                }
        }
}
```

Singleton in Multithreaded Environment

```
public class Singleton {
        private static Singleton instance;
        private static readonly object obj = new object();

        private Singleton() { // Some code}

        public static Singleton GetInstance {
                lock (obj) {
                        if (instance == null) {
                                instance = new Singleton();
                        }
                return instance;
                }
        }
}
```

Problem with Solution

- Synchronization decreases performance by factor of 100 !!!
- This solution fixes our problem but it is very expensive

Problem with Solution

Double-Checked Locking

- With this solution, we first check to see if an instance is created and only then we synchronize
- This way we only synchronize the first call and there's no performance issue anymore

Problem with Solution

```
public class Singleton {
    private static Singleton instance;
    private static readonly object obj = new object();

    private Singleton() { // any code }

    public static Singleton GetInstance {
            if (instance == null) {
                lock (obj) {
                    if (instance == null) {
                        instance = new Singleton();
                    }
                }
            }
            return instance;
    }
}
```

Problem with Solution

```
public class Singleton {
    private static Singleton instance;
    private static readonly object obj = new object();

    private Singleton() { // any code }

    public static Singleton GetInstance {
            if (instance == null) {
                lock (obj) {
                    if (instance == null) {
                        instance = new Singleton();
                    }
                }
            }
            return instance;
    }
}
```

Singleton Pattern – Applicability

- There must be exactly one instance of a class, and it must be accessible to clients from a well-known access points
- When the sole instance should be extensible by sub classing and clients should be able to use an extended instance without modifying the underlying code

Recap

- Creational Design Patterns
- Singleton Design Pattern
- Possible Implementations
  - Using Global Variables
  - Problem in the Approach
  - Solution
- Singleton Design Pattern
  - Definition
  - Class Diagram
  - Code Implementation
- Singleton in Multithreaded Environment
- Singleton Pattern – Applicability

Recap

Creational Design Patterns

Singleton Design Pattern

Questions