Software Engineering and Information System

Lecture 11:

Solid Principles

Md. Al-Hasan

Lecturer

Department of Computer Science & Engineering (CSE)

Bangladesh Army University of Science & Technology (BAUST)

Overview

In this class we will discuss

- ✓SOLID Acronym and Introduction
- ✓SOLID design principles
- ✓Why SOLID

Motivation behind the usage of SOLID Principles

- In any enterprise software application development when we design and develop software systems, we need to account the below factors during the development cycle.

- **Maintainability :** Maintainable systems are very important to the organizations.
- **Testability :** Test driven development (TDD) is required when we design and develop large scale systems
- **Flexibility and Extensibility :** Flexibility and extensibility is a very much desirable factor of enterprise applications. Hence we should design the application to make it flexible so that it can be adapt to work in different ways and extensible so that we can add new features easily.
- **Parallel Development :** It is one of the key features in the application development as it is not practical to have the entire development team working simultaneously on the same feature or component.
- **Loose Coupling :** We can address many of the requirements listed above by ensuring that our design results in an application that loosely couples many parts that makes up the application.

SOLID Principles and Design Patterns plays a key role in achieving all of the above points.

SOLID Introduction

In object-oriented computer programming, the term **SOLID** is a mnemonic acronym for five design principles intended to make software designs more understandable, flexible and maintainable.

- **SOLID** principles are the design principles that enable us manage most of the software design problems
- The principles are a subset of many principles promoted by Robert C. Martin
- The SOLID acronym was first introduced by Michael Feathers

SOLID Acronym

- **S** : **S**ingle Responsibility Principle (SRP)
- **O** : **O**pen closed Principle (OSP)
- **L** : **L**iskov substitution Principle (LSP)
- **I** : **I**nterface Segregation Principle (ISP)
- **D** : **D**ependency Inversion Principle (DIP)

Single Responsibility Principle

- Robert C. Martin expresses the principle as, "A class should have only one reason to change”

- Every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class

Liskov Substitution Principle

- Introduced by Barbara Liskov state that “objects in a program should be replaceable with instances of their sub-types without altering the correctness of that program”
- If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module
- We can also state that Derived types must be substitutable for their base types

Open/Closed Principle

- “Software entities should be open for extension, but closed for modification”
- The design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code
- The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged

Interface Segregation Principle

- “Many client-specific interfaces are better than one general-purpose interface”
- We should not enforce clients to implement interfaces that they don't use. Instead of creating one big interface we can break down it to smaller interfaces

Dependency Inversion Principle

- One should “depend upon abstractions, [not] concretions“
- Abstractions should not depend on the details whereas the details should depend on abstractions
- High-level modules should not depend on low level modules

SOLID Principles helps to

- Achieve reduction in complexity of code
- Increase readability, extensibility and maintenance
- Reduce error and implement Reusability
- Achieve Better testability
- Reduce tight coupling

Design Pattern Vs. Solid Principle

In the software engineering, design principle and design pattern **are not the same**.

- **Design Principle:** It provides high level guidelines to design better software applications. Design principles do not provide implementation and not bound to any programming language. E.g. SOLID (SRP, OCP, LSP, ISP, DIP) principles.

- **Design Pattern:** It provides low level solution (implementation) for the commonly occurring object oriented problem. In another word, design pattern suggest specific implementation for the specific object oriented programming problem.
  For example, if you want create a class that can only have one object at a time then you can use Singleton design pattern which suggests the best way to create a class that can only have one object.

A successful application development depends on

- **Architecture**: choosing an architecture is the first step in designing application based on the requirements. **Example :** MVC, WEBAPI, MVVM etc.

- **Design Principles**: Application development process need to follow the design principles

- **Design Patterns**: We need to choose correct design patterns to build the