Software Engineering Fundamentals (SEF): MS.NET

Coding Standards

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Presentation overview

a) SOLID principles
b) KISS, DRY, YAGNI principles
c) Code readability

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

SOLID principles

a) Defined by Robert C. Martin (not all elaborated by him)
b) Acronym by Michael Feathers
c) Maintainability, extensibility, robustness

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Single Responsibility Principle

SRP: A class should have only one reason to change.

Advantages?

a) Small and simple to understand classes
b) Easy to test
c) Easy to switch implementations

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Open Closed Principle

OCP: A class should be open for extension, but closed for modification.

Advantages?

a) Minimize risk of introducing bugs into existing functionality

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle

LSP: Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T.

LSP: Derived class should not break client code when used in place of base class.

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle

a) Preconditions cannot be strengthened in a subtype
b) Postconditions cannot be weakened in a subtype
c) Invariants of the supertype must be preserved in a subtype

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle

a) Contravariance of method arguments in the subtype
b) Covariance of return types in the subtype

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle (contravariance)

```
6 references
public class Document
{
}

1 reference
public class CurriculumVitae : Document
{
}

3 references
public interface IDocumentEqualityComparer<in T> where T : Document
{
    1 reference
    bool AreEqual(T document1, T document2);
}

2 references
public class DocumentEqualityComparer : IDocumentEqualityComparer<Document>
{
    1 reference
    public bool AreEqual(Document document1, Document document2)...
}

0 references
public class ContravarianceExample
{
    0 references
    public void Show()
    {
        IDocumentEqualityComparer<Document> documentEqualityComparer = new DocumentEqualityComparer();
        IDocumentEqualityComparer<CurriculumVitae> cvEqualityComparer = new DocumentEqualityComparer();
    }
}
```

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle (covariance)

```
3 references
public class Document
{
}

3 references
public class CurriculumVitae : Document
{
}

3 references
public interface IDocumentParser<out T> where T : Document
{
    1 reference
    T Parse(string text);
}

2 references
public class CurriculumVitaeParser : IDocumentParser<CurriculumVitae>
{
    1 reference
    public CurriculumVitae Parse(string text)...
}

0 references
public class CovarianceExample
{
    0 references
    public void Show()
    {
        IDocumentParser<CurriculumVitae> cvParser = new CurriculumVitaeParser();
        IDocumentParser<Document> documentParser = new CurriculumVitaeParser();
    }
}
```

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle

a) No new exceptions should be thrown by methods of the subtype

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Liskov Substitution Principle

Advantages?

a) Imagine big and complex system like Windows OS. You extend a class which is used in tens of other classes. By adhering to LSP risk of breaking whole system is minimized.

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Interface Segregation Principle

ISP: Client should not be forced to depend on methods it does not use.

Advantages?

a) Implementer is not pushed to implement methods that it doesn't need
b) Client has no temptation to use more than it needs

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Dependency Inversion Principle

DIP: High-level modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details. Details should depend on abstractions.

Advantages?

a) Easy to switch implementations

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Other principles

a) KISS: Keep It Simple, Stupid
b) DRY: Don’t Repeat Yourself
c) YAGNI: You Ain’t Gonna Need It

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Code readability

a) Clear names
b) Avoid comments
c) Formatting

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Tools

a) StyleCop
b) JetBrains R#

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Summary

Keep Your POOP SOLID and DRY

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

References

a) SOLID in C# by Chris Klug (TechEd North America 2014)

b) Adaptive Code via C#: Agile coding with design patterns and SOLID principles by Gary McLean Hall book

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.

Code examples

a) https://github.com/maksims-ahadovs/SOLID

© Accenture. By Maksims Ahadovs, Dmitri Kartashov-Gawski & Artem Slobolinskyi.

Questions?

I hope no =(

© Accenture. By Maksims Ahadovs & Dmitri Kartashov-Gawski.