

# Fundamentals: The Liskov Substitution Principle

Steve Smith

<http://pluralsight.com/>



# Outline

- LSP Defined
- The Problem
- An Example
- Refactoring to Apply LSP
- Related Fundamentals

# LSP: The Liskov Substitution Principle

*The Liskov Substitution Principle states that Subtypes must be substitutable for their base types.*

Agile Principles, Patterns, and Practices in C#

Named for Barbara Liskov, who first described the principle in 1988.



# LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

# Substitutability

**Child classes must not:**

- 1) Remove base class behavior**
- 2) Violate base class invariants**

**And in general must not require calling code to know they are different from their base type.**

# Inheritance and the IS-A Relationship

Naïve OOP teaches use of IS-A to describe child classes' relationship to base classes

LSP suggests that IS-A should be replaced with  
IS-SUBSTITUTABLE-FOR

# Invariants

- Consist of reasonable assumptions of behavior by clients
- Can be expressed as preconditions and postconditions for methods
- Frequently, unit tests are used to specify expected behavior of a method or class
- *Design By Contract* is a technique that makes defining these pre- and post-conditions explicit within code itself.
- To follow LSP, derived classes must not violate any constraints defined (or assumed by clients) on the base classes

# Demo

Violating LSP using Shapes





# The Problem

- Non-substitutable code breaks polymorphism
- Client code expects child classes to work in place of their base classes
- “Fixing” substitutability problems by adding if-then or switch statements quickly becomes a maintenance nightmare (and violates OCP)

# LSP Violation “Smells”

```
foreach (var emp in Employees)
{
    if(emp is Manager)
    {
        _printer.PrintManager(emp as Manager);
    }
    else
    {
        _printer.PrintEmployee(emp);
    }
}
```

# LSP Violation “Smells”

```
public abstract class Base
{
    public abstract void Method1();
    public abstract void Method2();
}

public class Child : Base
{
    public override void Method1()
    {
        throw new NotImplementedException();
    }
    public override void Method2()
    {
        // do stuff
    }
}
```

Follow ISP!

*Use small interfaces so you don't require classes to implement more than they need!*

# Demo

Refactoring to a Better Design



# When do we fix LSP?

- If you notice obvious smells like those shown
- If you find yourself being bitten by the OCP violations LSP invariably causes

# LSP Tips

- **“Tell, Don’t Ask”**
  - Don’t interrogate objects for their internals – move behavior to the object
  - Tell the object what you want it to do
- **Consider Refactoring to a new Base Class**
  - Given two classes that share a lot of behavior but are not substitutable...
  - Create a third class that both can derive from
  - Ensure substitutability is retained between each class and the new base

# Summary

- Conformance to LSP allows for proper use of polymorphism and produces more maintainable code
- Remember IS-SUBSTITUTABLE-FOR instead of IS-A
- **Related Fundamentals:**
  - Polymorphism
  - Inheritance
  - Interface Segregation Principle
  - Open / Closed Principle
- **Recommended Reading:**
  - Agile Principles, Patterns, and Practices by Robert C. Martin and Micah Martin [<http://amzn.to/agilepppcsharp>]

# Credits

- **Images Used Under License**

- <http://www.lostechies.com/blogs/derickbailey/archive/2009/02/11/solid-development-principles-in-motivational-pictures.aspx>



For more in-depth **online** developer **training** visit



**on-demand** content from authors you **trust**