SOLID

Principles for Maintainability of Code

Peter Levinsky – IT Roskilde

03.10.2024

S O L I D

- **S** Single Responsibility
- **O** Open / Closed
- **L** Liskov Substitution
- **I** Interface Segregation
- **D** Dependency Injection/Inversion

S O L I D

- **S** Single Responsibility
- **O** Open / Closed
- **L** Liskov Substitution
- **I** Interface Segregation
- **D** Dependency Injection/Inversion

The Liskov Substitution Principle

- Definition

If the class S is a subtype of the class T,
then objects of type T may be replaced with objects of type S,
without breaking the program

The Liskov Substitution Principle

- Principle relating to how to create inheritance hierarchies

- Ensures that a client can use subclasses of provided classes **without** changing the expected behaviour

The Liskov Substitution Principle

```
public class T {
   public override void SayHello(string name)
   {
      Console.WriteLine($“Hello {name}");
   }
}

public class S : T {
   public override void SayHello(string name)
   {
      Console.WriteLine($"Hola {name}");
   }
}
```

The Liskov Substitution Principle

```
public class Client
{
    public void DoSomething(T obj)
    {
        obj.SayHello("Alex");
        obj.SayHello("Betty");
    }
}

Client aClient = new Client();
aClient.DoSomething(new T());
aClient.DoSomething(new S()); // Have I broken something…?
```

The Liskov Substitution Principle

```
// What does this interface do…?
public interface IT
{
   void SayHello(string name);
}
```

The Liskov Substitution Principle

```
public interface IT
{
    /// <summary>
    /// Contract: Invoking this method should print a
    /// message on the screen.
    /// The message should
    ///   1) Have a polite greeting nature.
    ///   2) Use the name provided in the argument.
    ///   3) Be in English.
    /// No side effect should occur by calling this method.
    /// </summary>
    void SayHello(string name);
}
```

The Liskov Substitution Principle

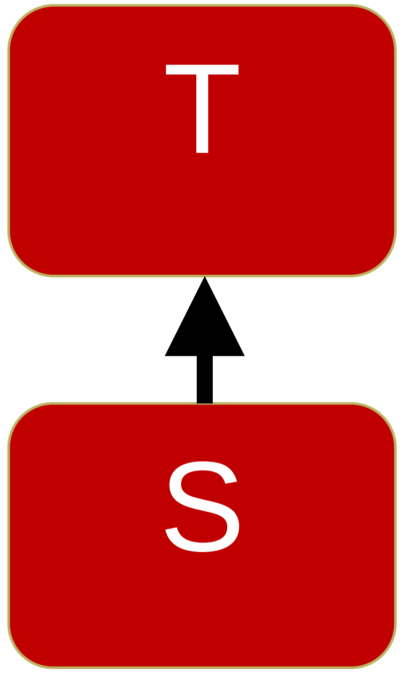
```
public class CheckedGreeting : T
{
   public override void SayHello(string name)
   {
      if (name.Length < 3)
      {
         throw new ArgumentException("Name too short!");
      }

      base.SayHello(name);
   }
}
```

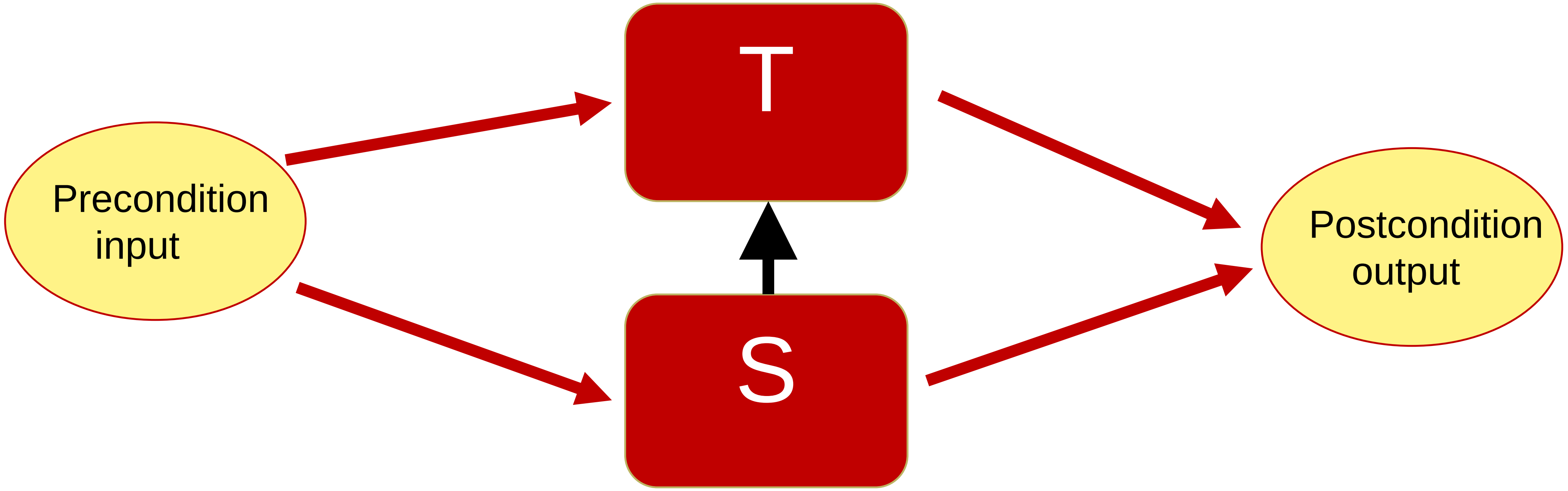
The Liskov Substitution Principle

- **More detailed definition:**

- *If the class **S** is a subtype of the class **T**, then it must always hold that*
  - ***Preconditions** in **T** are never strengthened by **S**.*
  - ***Postconditions** in **T** are never weakened by **S**.*
  - ***Invariants** in **T** must be preserved by **S**.*



The Liskov Substitution Principle



The Liskov Substitution Principle

```
// Precondition was strengthened...
public class CheckedGreeting : T
{
    public override void SayHello(string name)
    {
        if (name.Length < 3)
        {
            throw new ArgumentException("Name too short!");
        }

        base.SayHello(name);
    }
}
```

Precondition

The Liskov Substitution Principle

```
public class Client
{
   public void DoSomething(T obj)
   {
      obj.SayHello("Alex");
      obj.SayHello("Bo");
   }
}

Client aClient = new Client();
aClient.DoSomething(new T()); // OK
aClient.DoSomething(new CheckedGreeting()); // Oops...
```

The Liskov Substitution Principle

Improvement

– for precondition

```
public class Name
{
   public string Value { get; }

   public Name(string value)
   {
      if (value.Length < 3) { throw new …}
      Value = value;
   }
}
```

The Liskov Substitution Principle

Input restricted

```
public class Greeting : IGreeting
{
    public void SayHello(Name name)
    {
        Console.WriteLine($"Hello {name.Value}");
    }
}
```

The Liskov Substitution Principle

Improvement
– for postcondition

```
public class Salery
{
   public int Value { get; }

   public Salery(int value)
   {
      if (value < 10000 || 1000000 < value)
        { throw new …}
      Value = value;
   }
}
```

The Liskov Substitution Principle

```
public interface IEmployee
{
    /// <summary>
    /// Contract: the yearly salary returned must
    /// be a value between 10,000 and 1,000,000
    /// </summary>
    Salery GetYearlySalary();
}
```

output restricted

Exercises

- **SOLID 1,2,3**
- **Mandatory Assignment**