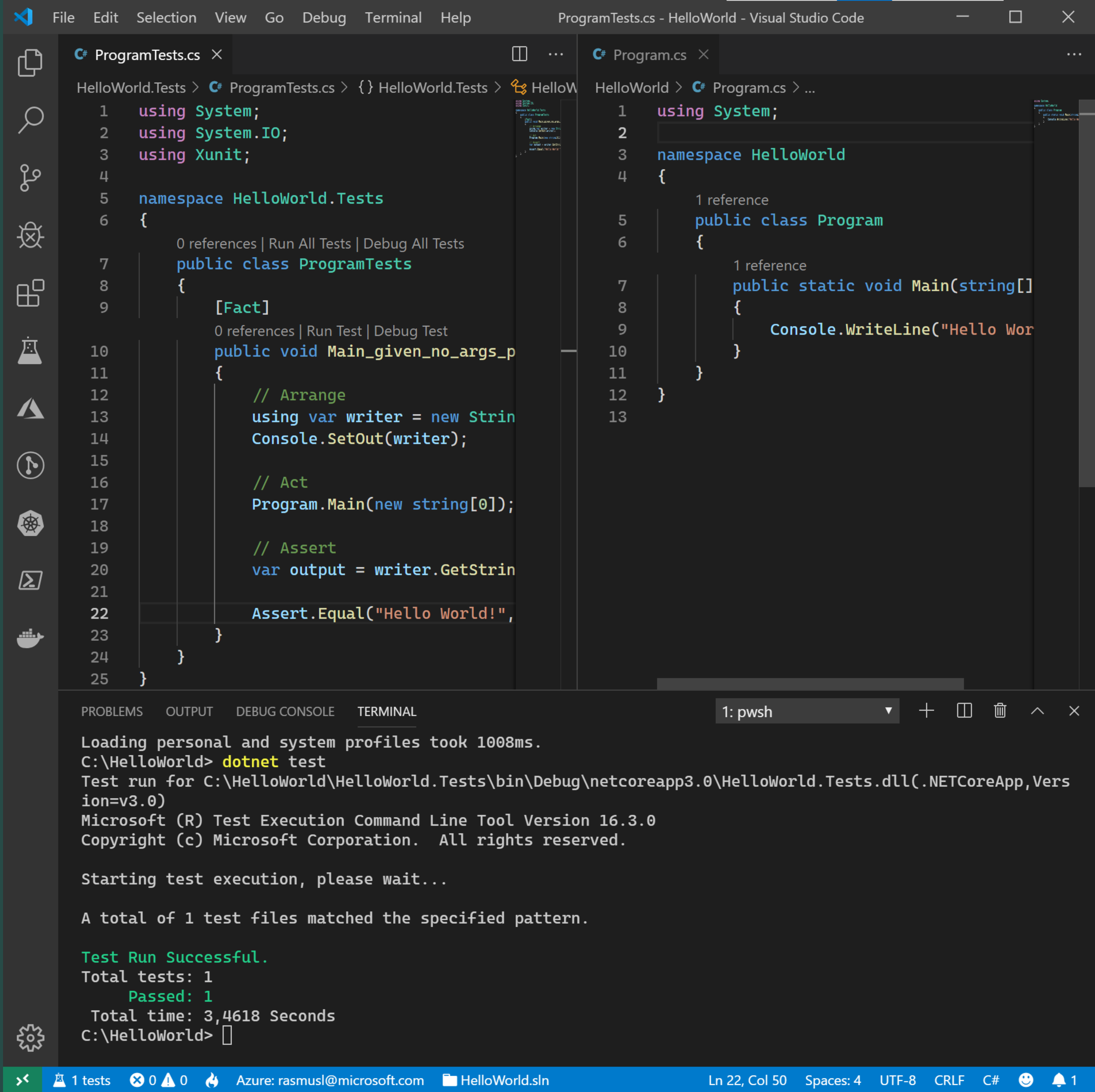
C# Design Patterns in Practice

Rasmus Lystrøm
Associate Professor
ITU
rnie@itu.dk


File Edit Selection View Go Debug Terminal Help
ProgramTests.cs - HelloWorld - Visual Studio Code
ProgramTests.cs
Program.cs
HelloWorld.Tests > ProgramTests.cs > {} HelloWorld.Tests > HelloW
HelloWorld > Program.cs > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: pwsh
1 tests
Azure: rasmusl@microsoft.com
HelloWorld.sln
Ln 22, Col 50 Spaces: 4 UTF-8 CRLF C#

```csharp
using System;
using System.IO;
using Xunit;

namespace HelloWorld.Tests
{
    0 references | Run All Tests | Debug All Tests
    public class ProgramTests
    {
        [Fact]
        0 references | Run Test | Debug Test
        public void Main_given_no_args_p
        {
            // Arrange
            using var writer = new Strin
            Console.SetOut(writer);

            // Act
            Program.Main(new string[0]);

            // Assert
            var output = writer.GetStrin

            Assert.Equal("Hello World!",
        }
    }
}
```

```csharp
using System;

namespace HelloWorld
{
    1 reference
    public class Program
    {
        1 reference
        public static void Main(string[]
        {
            Console.WriteLine("Hello Wor
        }
    }
}
```

```
Loading personal and system profiles took 1008ms.
C:\HelloWorld> dotnet test
Test run for C:\HelloWorld\HelloWorld.Tests\bin\Debug\netcoreapp3.0\HelloWorld.Tests.dll(.NETCoreApp,Vers
ion=v3.0)
Microsoft (R) Test Execution Command Line Tool Version 16.3.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...

A total of 1 test files matched the specified pattern.

Test Run Successful.
Total tests: 1
     Passed: 1
 Total time: 3,4618 Seconds
C:\HelloWorld>
```

Agenda

Gilded Rose recap

Design Patterns

Gilded Rose

Demo

Code Metrics (Visual Studio Proper)

Maintainability Index

Between 0 and 100. Higher is better. Aim for higher than 20

Cyclomatic Complexity

Lower is better. Split methods with complexity > 10

Depth of Inheritance

Between 1 and infinity

Lower is better, but sometimes inheritance is good

Class Coupling

Lower is better. Aim for max 9

Lines of Code

https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values

Code Metrics - Original

	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Program	57	3	1	5	49
GildedRose	54	20	1	2	85
Item	100	6	1	0	6
Total	**70**	**29**	**1**	**5**	**151**

Approach

Understand the task at hand – inspect the code

Write tests to ensure the program works to specification

Refactor, refactor, refactor

Extract methods

Implement *Conjured*

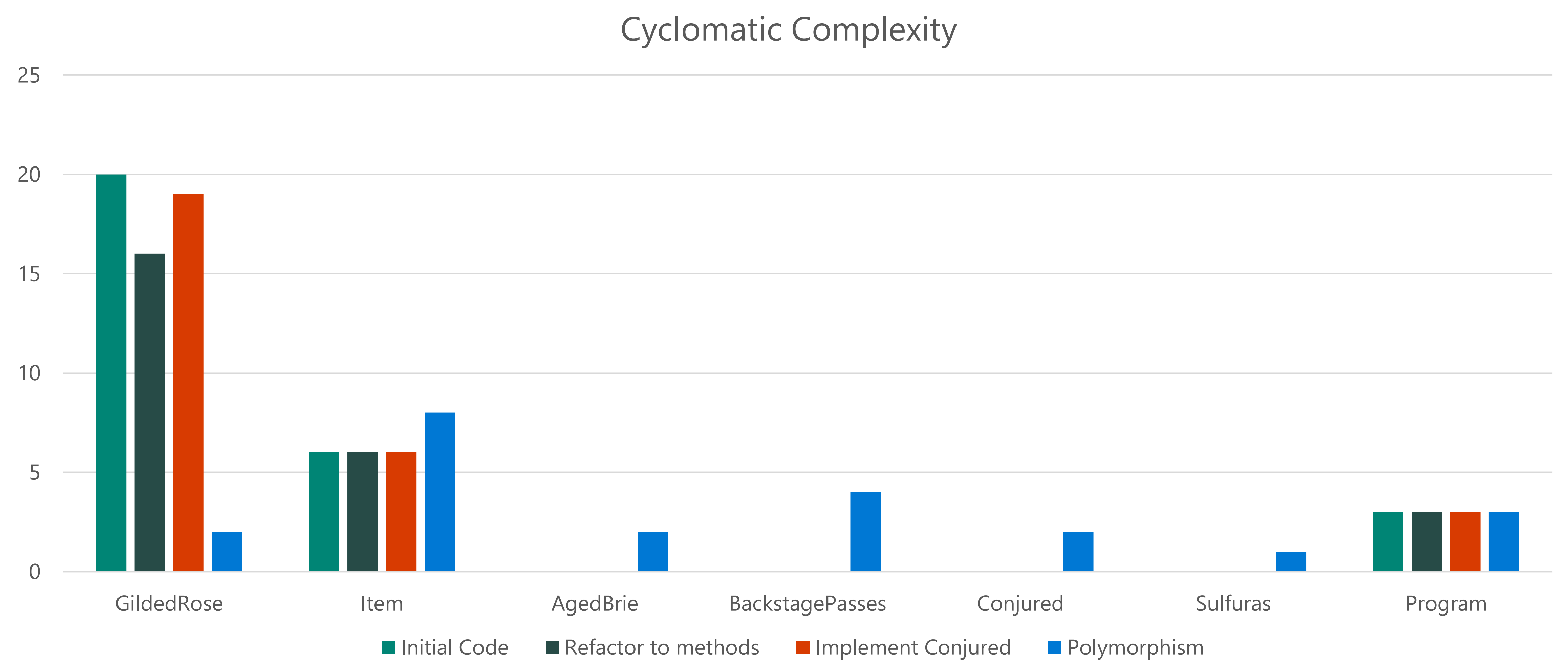
Refactor, refactor, refactor

Introduce polymorphism

Code Metrics – Polymorphed

	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
Program	57	3	1	8	33
GildedRose	93	2	1	5	14
Item	91	8	1	1	19
AgedBrie	71	2	2	2	15
BackstageP.	64	4	2	2	23
Conjured	71	2	2	2	15
Sulfuras	100	1	2	1	6
Total	**78**	**22**	**2**	**13**	**152**

Cyclomatic Complexity



Design Patterns in Practice

Design Patterns

IoC Container

Builder

Adapter

Factory Method

Singleton

Iterator

Façade

Chain of Responsibility

Strategy

Bridge

Saved for later:

Command (MVVM)

Observer (MVVM)

Proxy (Web API)

IoC Container

Tool to facilitate dependency injection.

Using a factory to either manually or automatically create types at runtime.

Various implementations:

- Microsoft.Extensions.DependencyInjection
- Ninject
- Unity
- AutoFac
- StructureMap

IoC Container II

Lifetime:

Transient (every time)

Scoped (once per request)

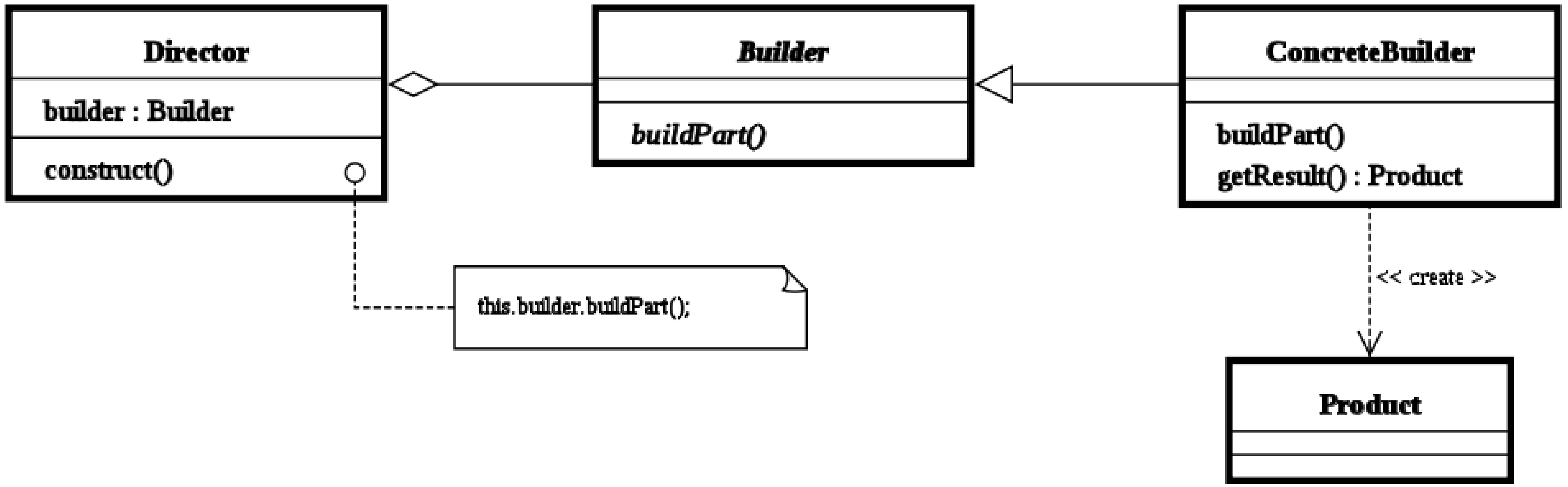
Singleton (once)

Builder

Separate the construction of a complex object from its representation

serviceCollection.AddScoped<,>();

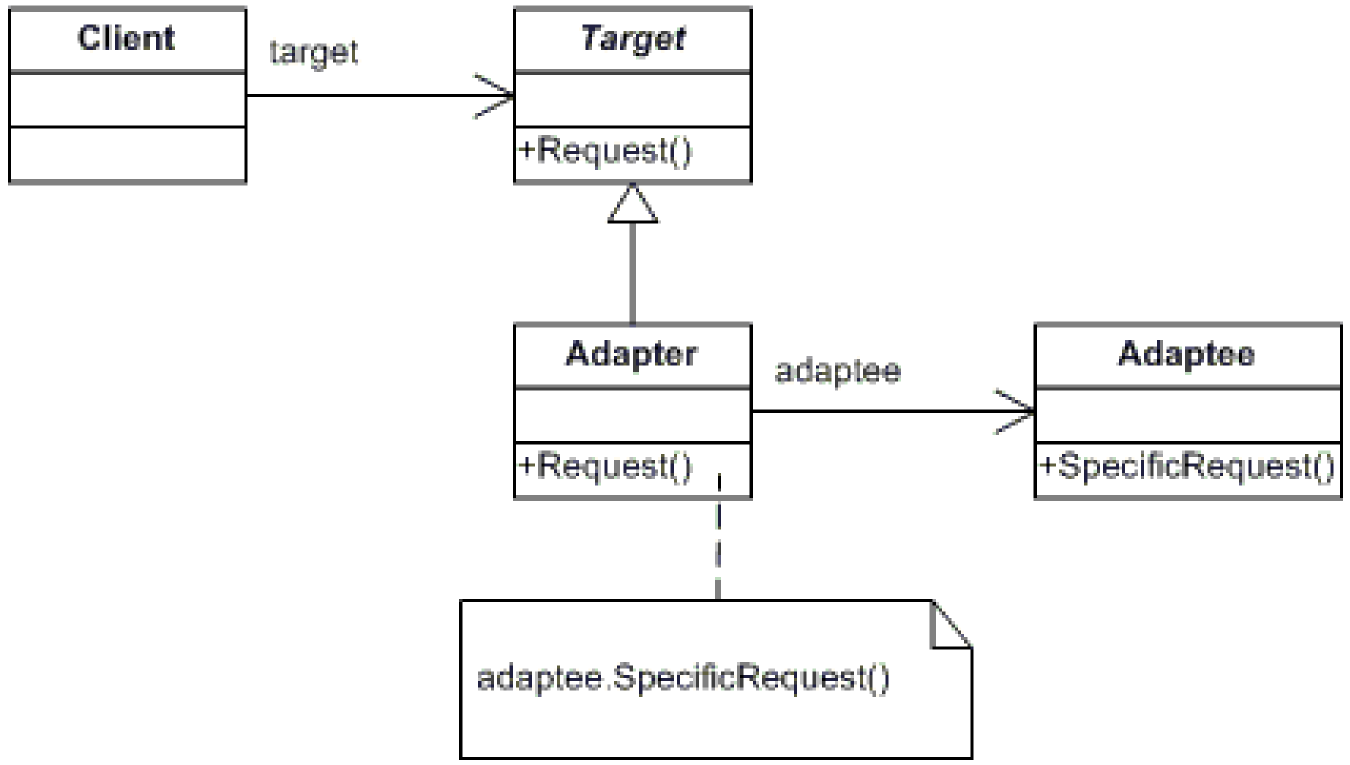
serviceCollection.BuildServiceProvider();



Adapter aka Wrapper

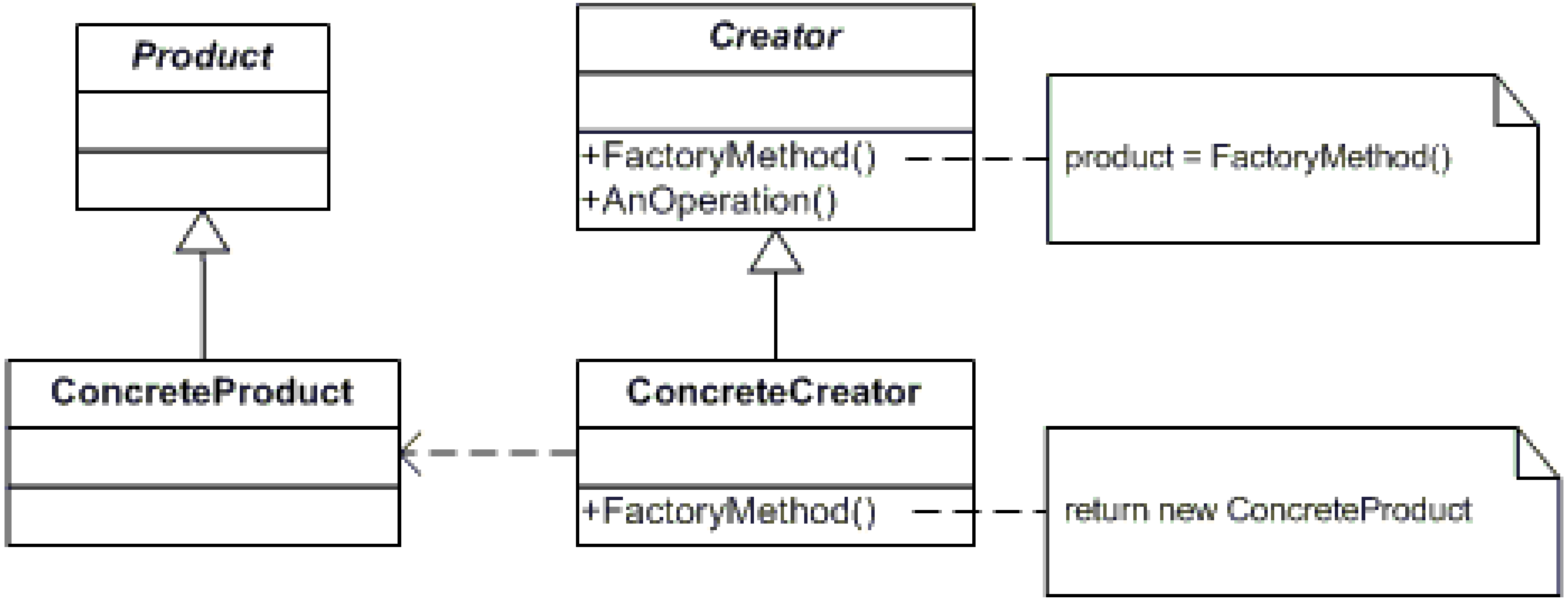
Unmodifiable implementation which does not match the interface you need.

Static or sealed class or class in another assembly.



Factory Method

A method which can creates instances of a given type.



Singleton

Only ever one single instance of a given type.

Considered an anti-pattern by many, it:

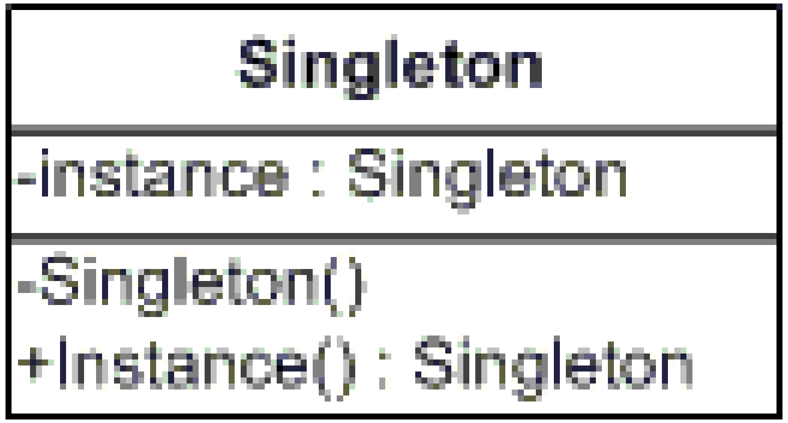
- is overused
- introduces unnecessary restrictions in situations where a sole instance of a class is not actually required
- introduces global state into an application

Singleton II

Use carefully

Implement using an interface

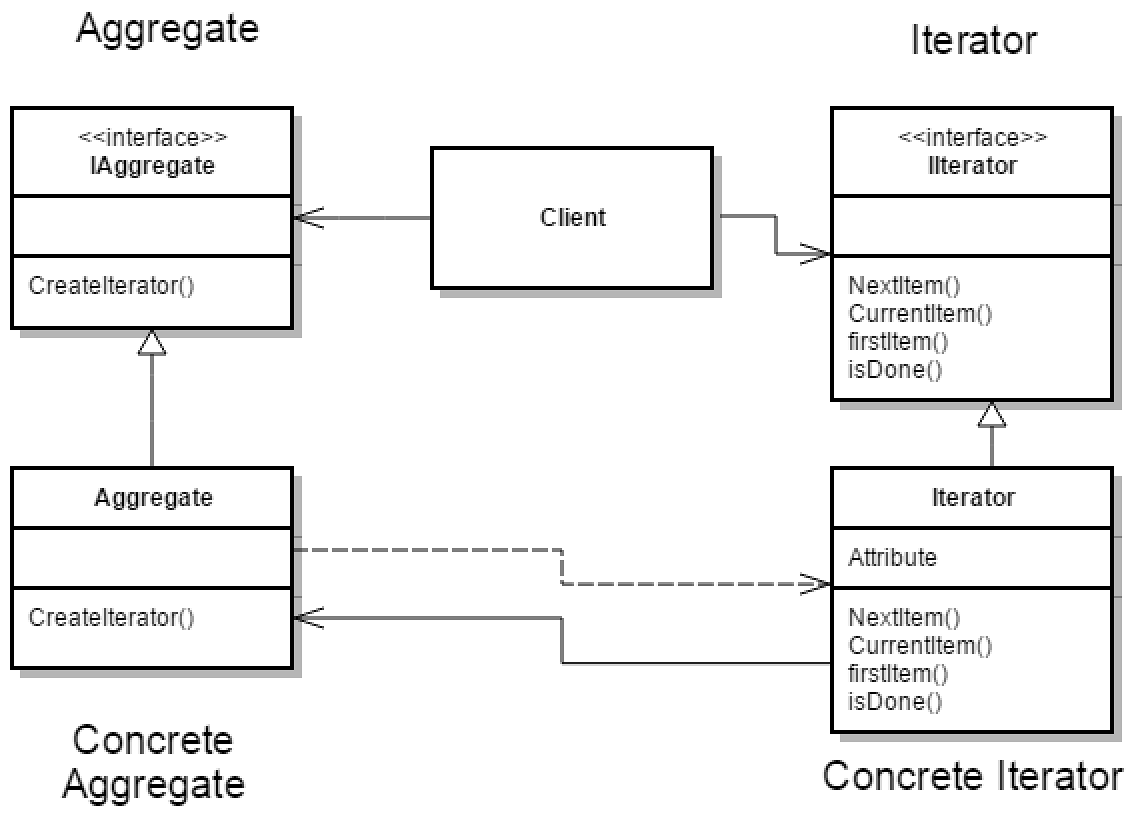
Use an IoC container



Iterator

Provide a way to access the elements of an aggregate object (collection) sequentially without exposing the underlying representation

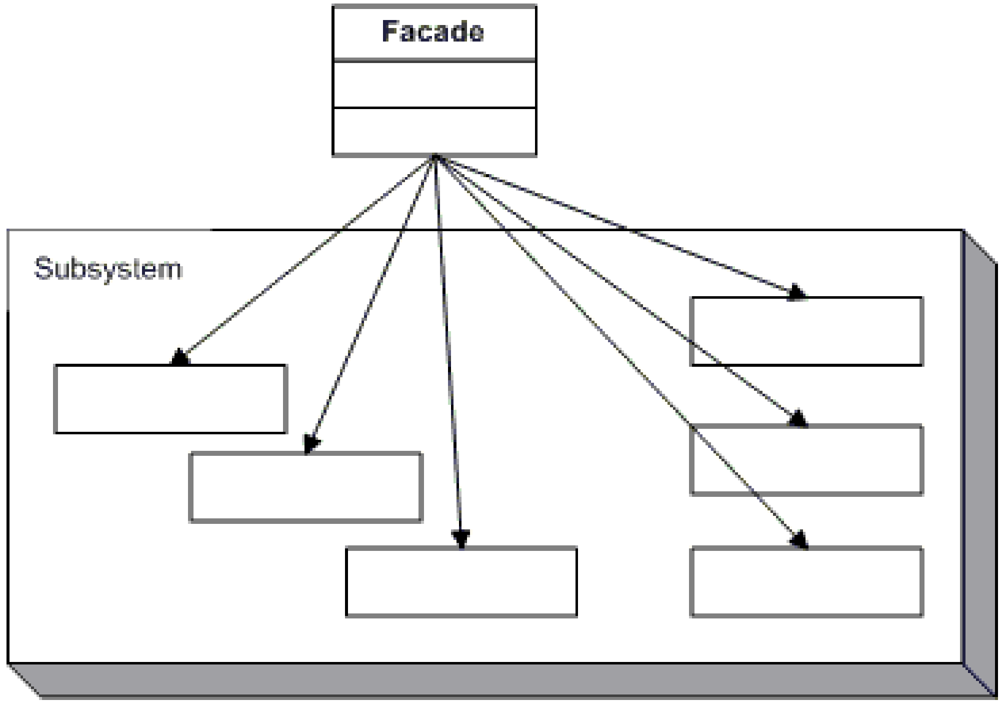
```
IEnumerable
IEnumerable<T>
IEnumerator
IEnumerator<T>
foreach...
```



Façade

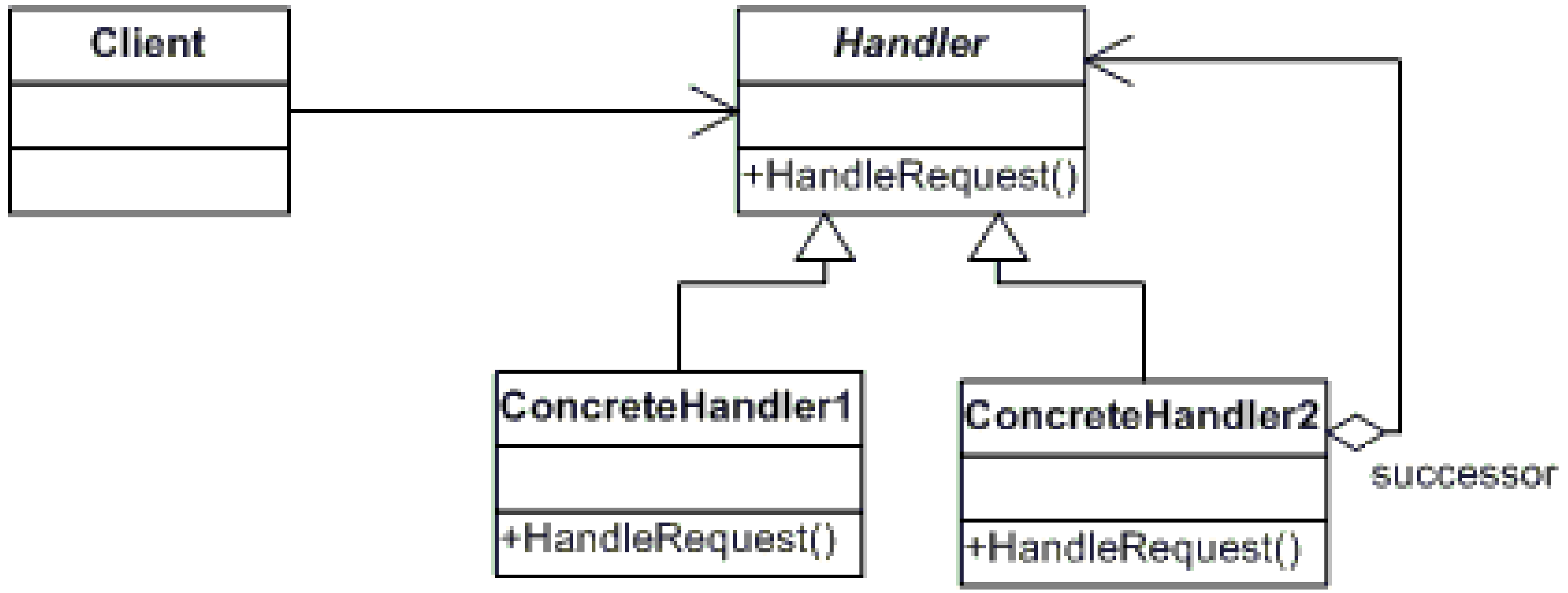
Simplify the use of a system

Provide a unified interfaces for a group of “dispersed” functionalities from a multitude of interfaces/classes



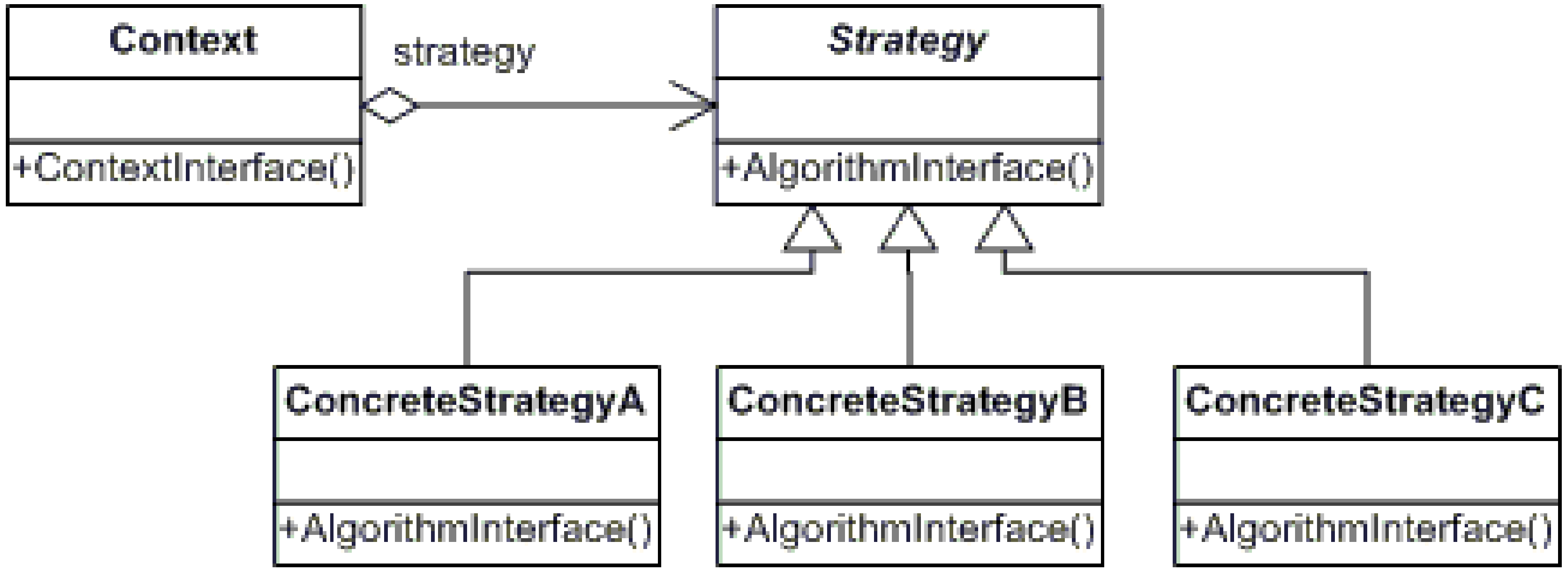
Chain of Responsibility

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



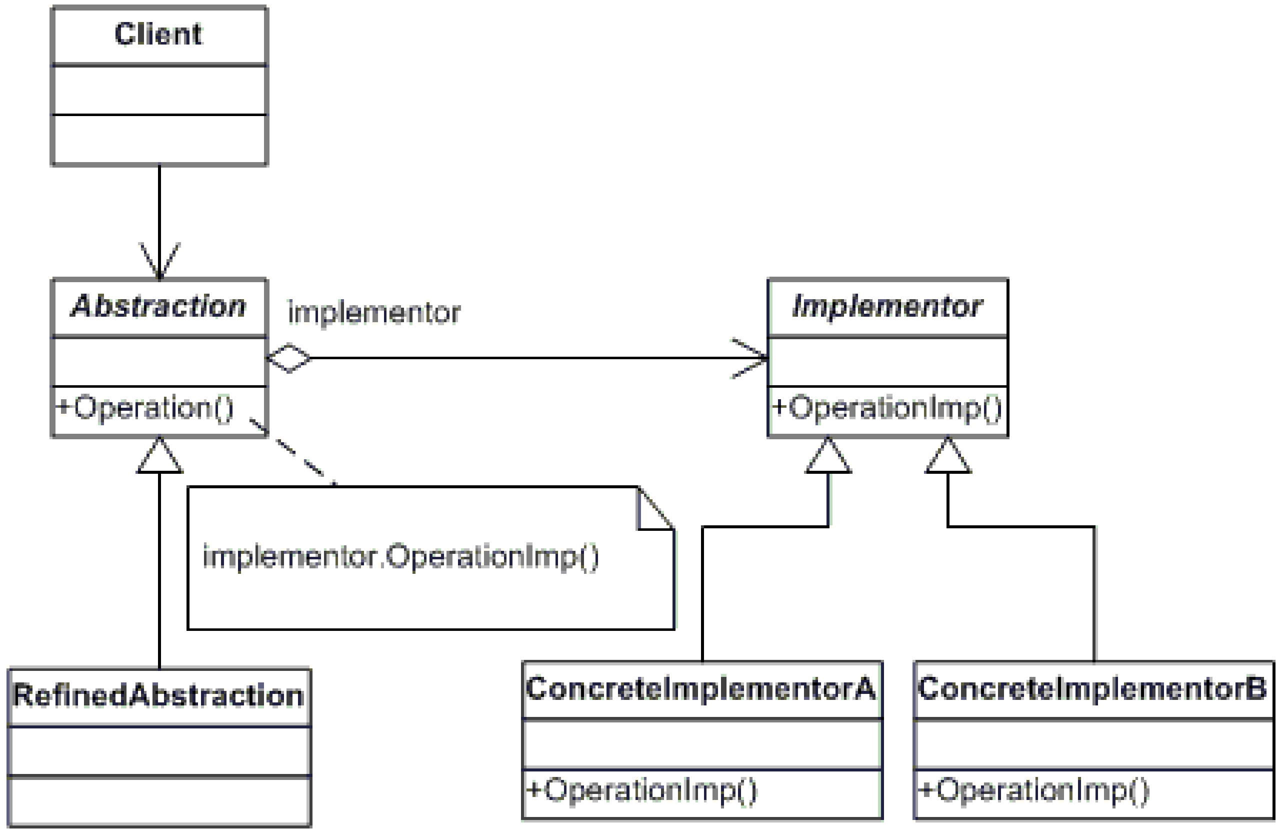
Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.



Bridge

Decouple an abstraction from its implementation so that the two can vary independently.


Client
Abstraction
+Operation()
implementor
Implementor
+OperationImp()
implementor.OperationImp()
RefinedAbstraction
ConcreteImplementorA
+OperationImp()
ConcreteImplementorB
+OperationImp()