
Spring.NET 프레임워크 활용 가이드

Version 0.2

(주)엔소아 컨설팅
대표 컨설턴트
전병선

Spring.NET 프레임워크 활용 가이드

- 애플리케이션 프레임워크
- Spring.NET 프레임워크
- IoC 컨테이너와 DI
- AOP (Aspect-Oriented Programming)

애플리케이션 프레임워크

- 프레임워크(framework)
 - 틀 구조, 뼈대, 골격, 구조, 구성
- 애플리케이션 프레임워크(application framework)
 - 애플리케이션을 구축하고 설계할 수 있도록 도와주는 틀 구조
 - 프레임워크는 소프트웨어의 특정한 클래스에 대하여 재사용할 수 있는 설계로 구성된 관련된 클래스들의 집합이다. 프레임워크는 설계를 추상적인 클래스로 분리하고 그들의 책임과 협동 관계를 정의함으로써 아키텍처적인 가이드를 제공한다. 여러분은 프레임워크로부터 추상적인 클래스를 서브클래싱하여 애플리케이션에 특정한 서브 클래스를 생성함으로써 특정한 애플리케이션에 대하여 프레임워크를 커스터마이징한다
 - GoF(gang of four)의 Design Patterns

애플리케이션 프레임워크

- 공통 라이브러리 != 프레임워크
 - 제어의 역흐름(IoC, inversion of control)
 - 할리우드 원칙(Hollywood Principle) :
“나를 부르지 마라. 내가 너를 부를 것이다”
“Don't call me, I'll call you.”
 - 프레임워크 코드가 전체 애플리케이션의 처리 흐름을 제어하며, 특정한 이벤트가 발생할 때 다형성(polymorphism)을 통해 애플리케이션이 확장한 메서드를 호출함으로써 제어가 프레임워크로부터 애플리케이션으로 거꾸로 흐르게 한다.

애플리케이션 프레임워크

- 프레임워크의 이점

- 모듈화(modularity)

- 프레임워크는 구현을 인터페이스 뒤에 감추는 캡슐화를 통해서 모듈화를 강화한다.
 - 프레임워크의 모듈화는 설계와 구현의 변경에 따르는 영향을 국소화함으로써 손쉽게 소프트웨어의 품질을 향상시킬 수 있게 한다

- 재사용성(reusability)

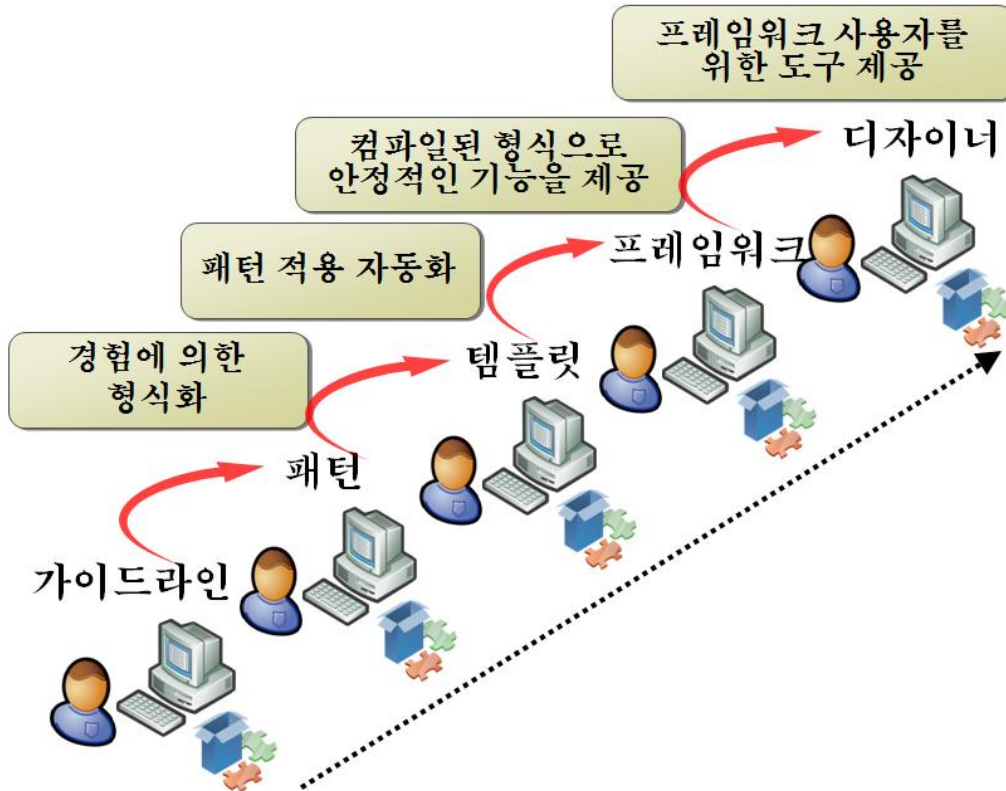
- 프레임워크가 제공하는 인터페이스는 여러 애플리케이션에서 반복적으로 사용할 수 있는 일반적인 컴포넌트를 정의할 수 있게 함으로써 재사용성을 높여준다.
 - 프레임워크 재사용성은 도메인 지식과 경험있는 개발자들의 이전의 노력을 활용하여, 애플리케이션의 요구사항과 소프트웨어 설계에 대한 공통의 솔루션을 반복적으로 재개발하고 그에 대해 유효성을 다시 확인하는 작업을 피할 수 있게 한다.
 - 프레임워크 컴포넌트를 재사용하는 것은 소프트웨어의 품질, 성능, 신뢰성, 상호운용성을 향상시킬 뿐만 아니라, 프로그래머의 생산성을 상당히 높여준다.

- 확장성(extensibility)

- 프레임워크는 다형성(polymorphism)을 통해 애플리케이션이 프레임워크의 인터페이스를 확장할 수 있게 한다.
 - 프레임워크 확장성은 새로운 애플리케이션 서비스와 특성을 커스터마이징하는 것을 보장하는데 있어서 필수적인 사항이며, 또한 프레임워크를 애플리케이션의 가변성으로부터 분리함으로써 재사용성의 이점을 얻게 한다.

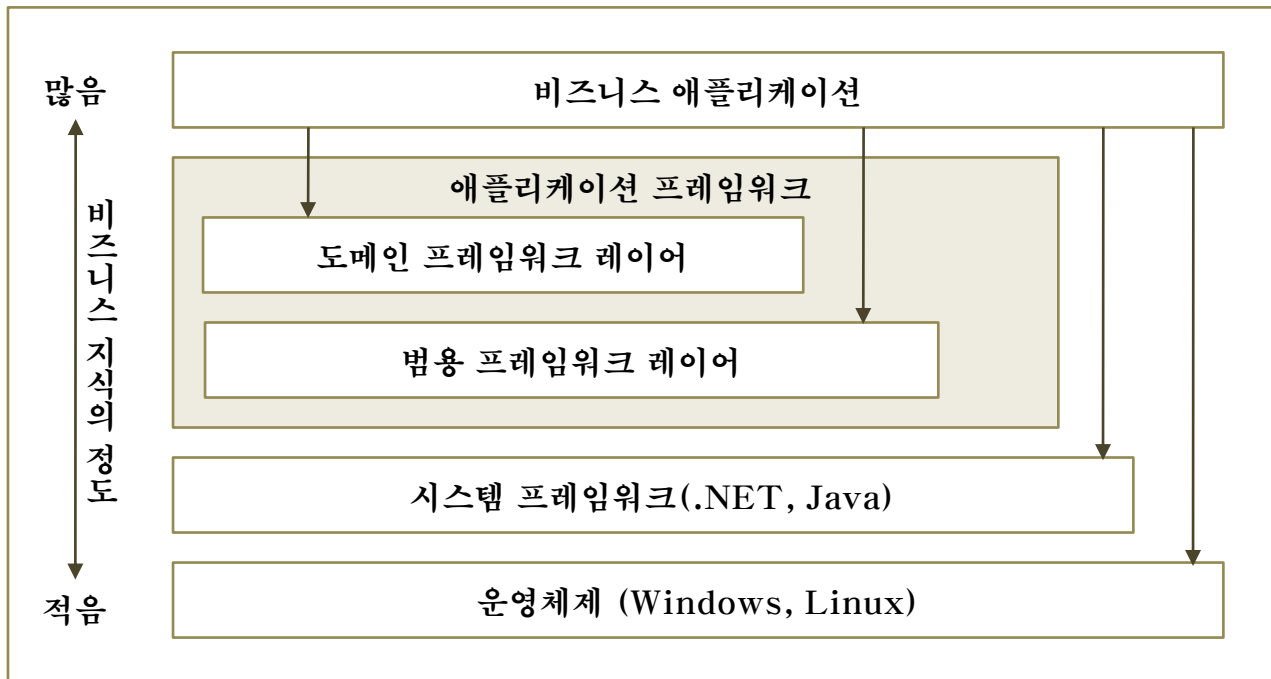
애플리케이션 프레임워크

- 프레임워크 개념의 진화 과정



애플리케이션 프레임워크

- 프레임워크 레이어(framework layer)



애플리케이션 프레임워크

- 프레임워크 레이어(framework layer)
 - 시스템 프레임워크(system framework)
 - 파운데이션 프레임워크(foundation framework)
 - 애플리케이션과 애플리케이션 프레임워크에 프로그래밍 모델을 제공하는 프레임워크
 - OMG의 CORBA
 - Sun Microsystems의 J2EE
 - Microsoft의 .NET 프레임워크나 COM/DCOM
 - 도메인 프레임워크(domain-specific framework)
 - 특정한 비즈니스 도메인을 대상으로 하는 프레임워크 컴포넌트로 구성된다
 - 특정한 비즈니스 도메인의 모든 애플리케이션에 공통된 비즈니스 지식을 구현한다
 - 범용 프레임워크(cross-domain framework)
 - 비즈니스 도메인에 관계없이 대부분의 애플리케이션에서 공통적으로 발견될 수 있는 컴포넌트와 서비스로 구성된다
 - Struts, Spring, Spring.NET, Microsoft의 Enterprise Library, Application Block

애플리케이션 프레임워크

- 공통 스팟과 핫 스팟

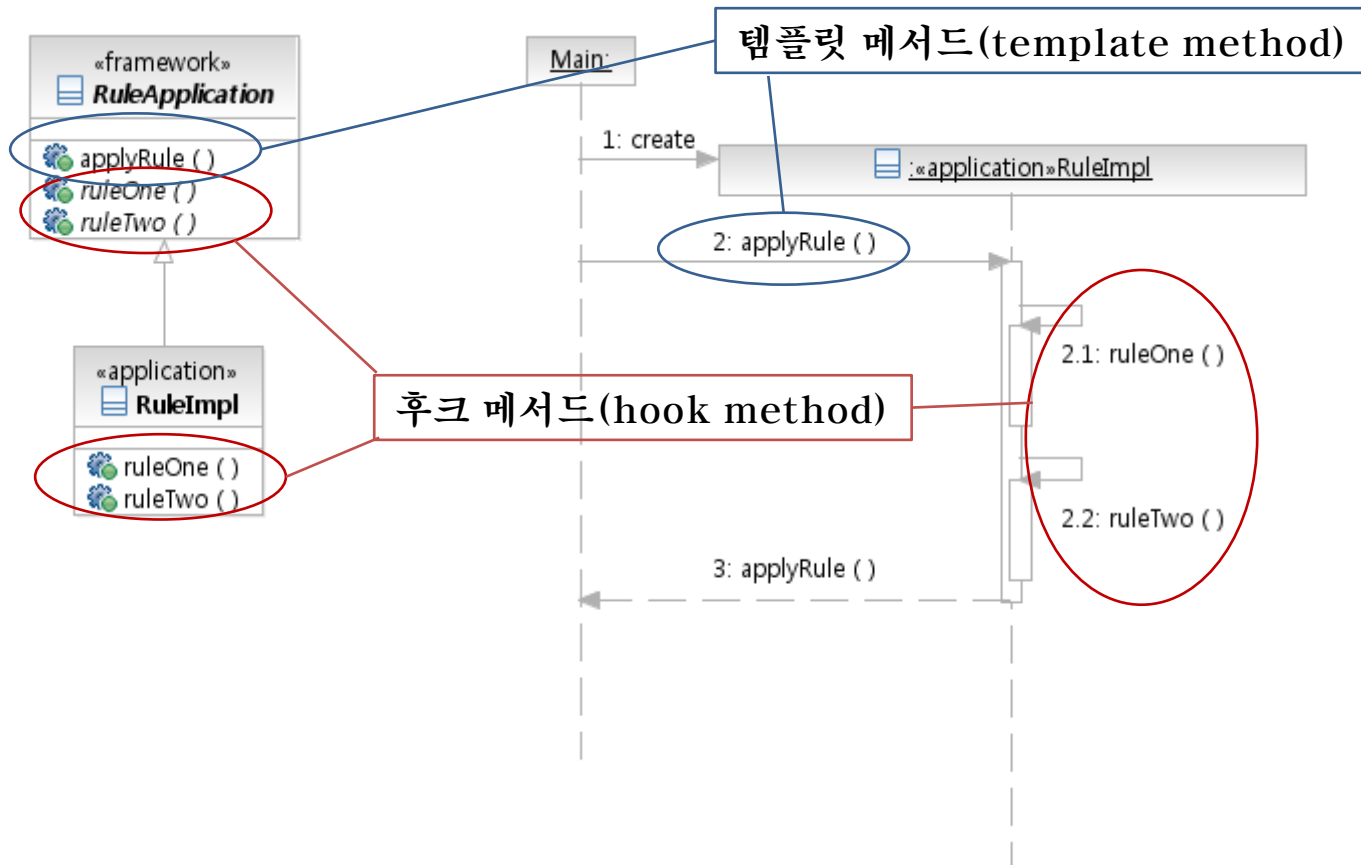
- 프레임워크 개발은 대상 애플리케이션에서 공통성(commonality)과 가변성(variability)를 분석하는 것에서부터 시작한다
- 공통 스팟(common spot)
 - 애플리케이션에서 공통성이 반복해서 나타나는 위치
 - 공통 스팟의 공통적인 부분은 프레임워크 컴포넌트 안에 구현한다
- 핫 스팟(hot spot)
 - 애플리케이션에서 가변적인 위치
 - 애플리케이션 마다 가변적이어서 프레임워크를 커스터마이징 할 수 있는 위치
 - 핫 스팟은 나중에 애플리케이션이 커스터마이징하여 채워넣을 수 있도록 비워둔다. 각 애플리케이션은 프레임워크에 있는 핫 스팟 위치에 고유한 기능을 구현하여 끼워넣음으로써 프레임워크가 각 애플리케이션마다 서로 다르게 행위를 하게 한다

애플리케이션 프레임워크

- 프레임워크 구현 기법
 - 화이트박스 프레임워크(white-box framework)
 - 블랙박스 프레임워크(black-box framework)
 - 그레이박스 프레임워크(gray-box framework)

애플리케이션 프레임워크

- 화이트박스 프레임워크(white-box framework)
 - 추상 클래스(abstract class)로 구성



애플리케이션 프레임워크

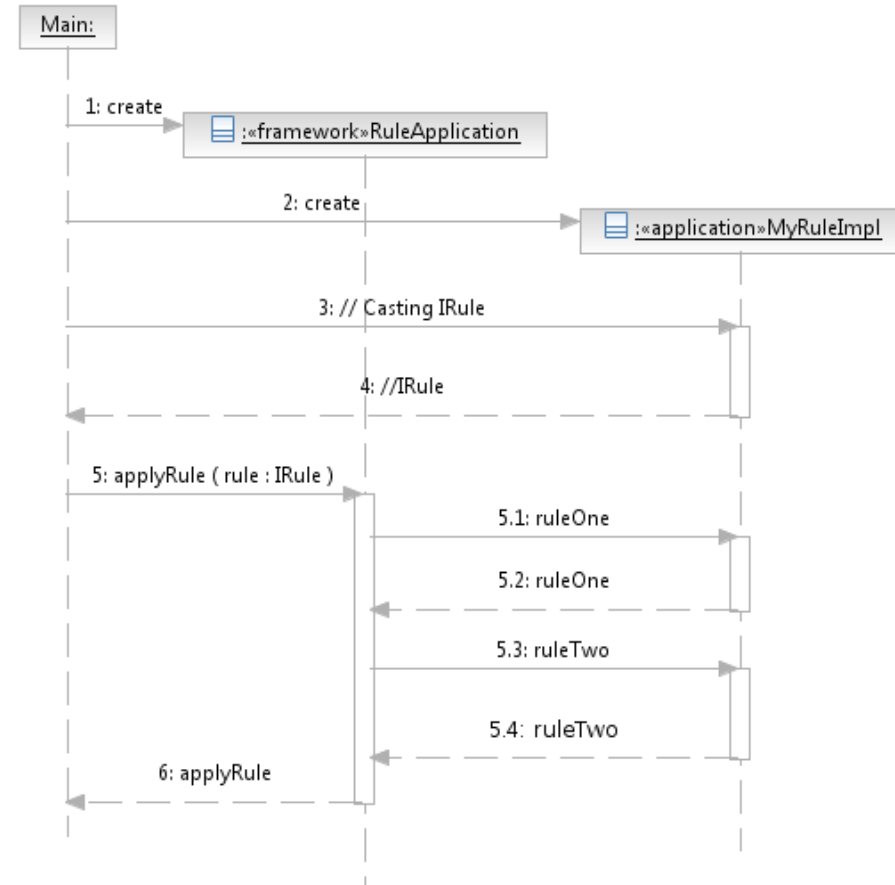
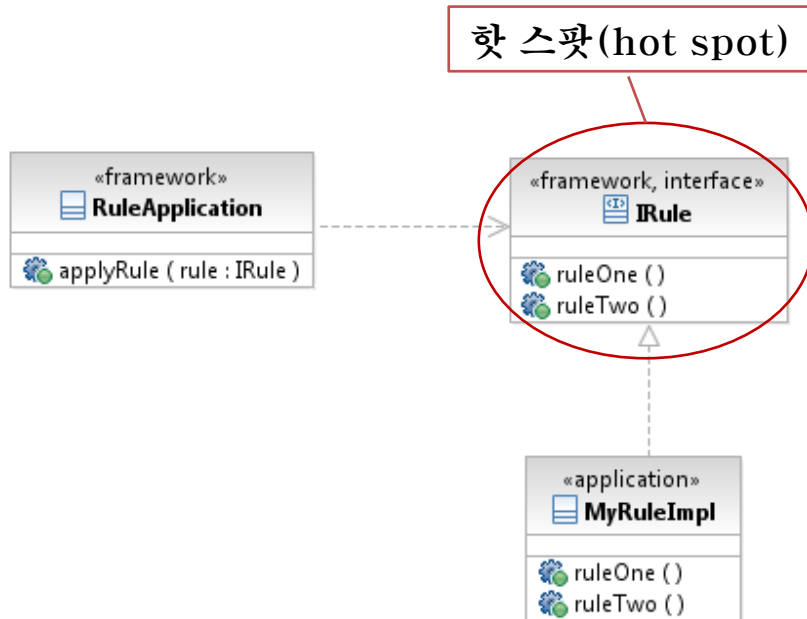
- 화이트박스 프레임워크(white-box framework)
 - 후크 메서드(hook method)
 - 애플리케이션 고유의 로직을 채워 넣은 핫 스팟으로서의 역할을 한다
 - 가상 함수(virtual function)로 구현한다
 - 템플릿 메서드(template method)
 - 템플릿 메서드는 특정한 오퍼레이션의 프로세스 흐름을 기술하며 공통 스팟으로서의 역할을 한다

애플리케이션 프레임워크

- 화이트박스 프레임워크(white-box framework)
 - 장점
 - 구현하기 쉽다
 - 단점
 - 템플릿 메서드에 프로세스의 흐름이 고정되어 있어 유연성이 부족하다
 - 개발자가 프레임워크 내부의 세부 사항을 잘 알고 있어야 후크 메서드를 구현할 수 있다

애플리케이션 프레임워크

- 블랙박스 프레임워크(black-box framework)
 - 인터페이스(interface)로 구성

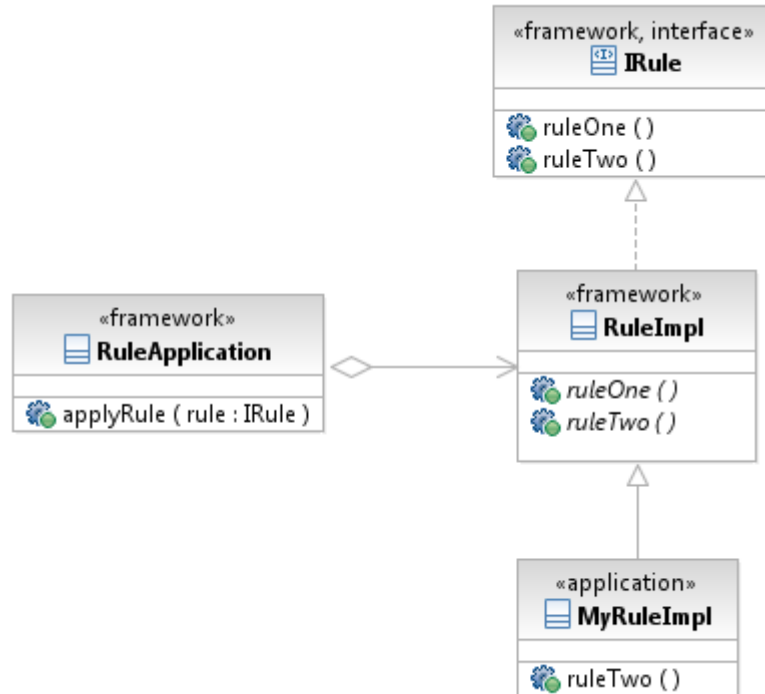


애플리케이션 프레임워크

- 블랙박스 프레임워크(black-box framework)
 - 장점
 - 내부 구현을 알 필요가 없다
 - 더 큰 유연성을 제공할 수 있다
 - 단점
 - 구현하기 어렵다
 - 유연성을 확보하기 위해서는 애플리케이션이 템플릿 메서드를 구현해야 하는 경우도 있다

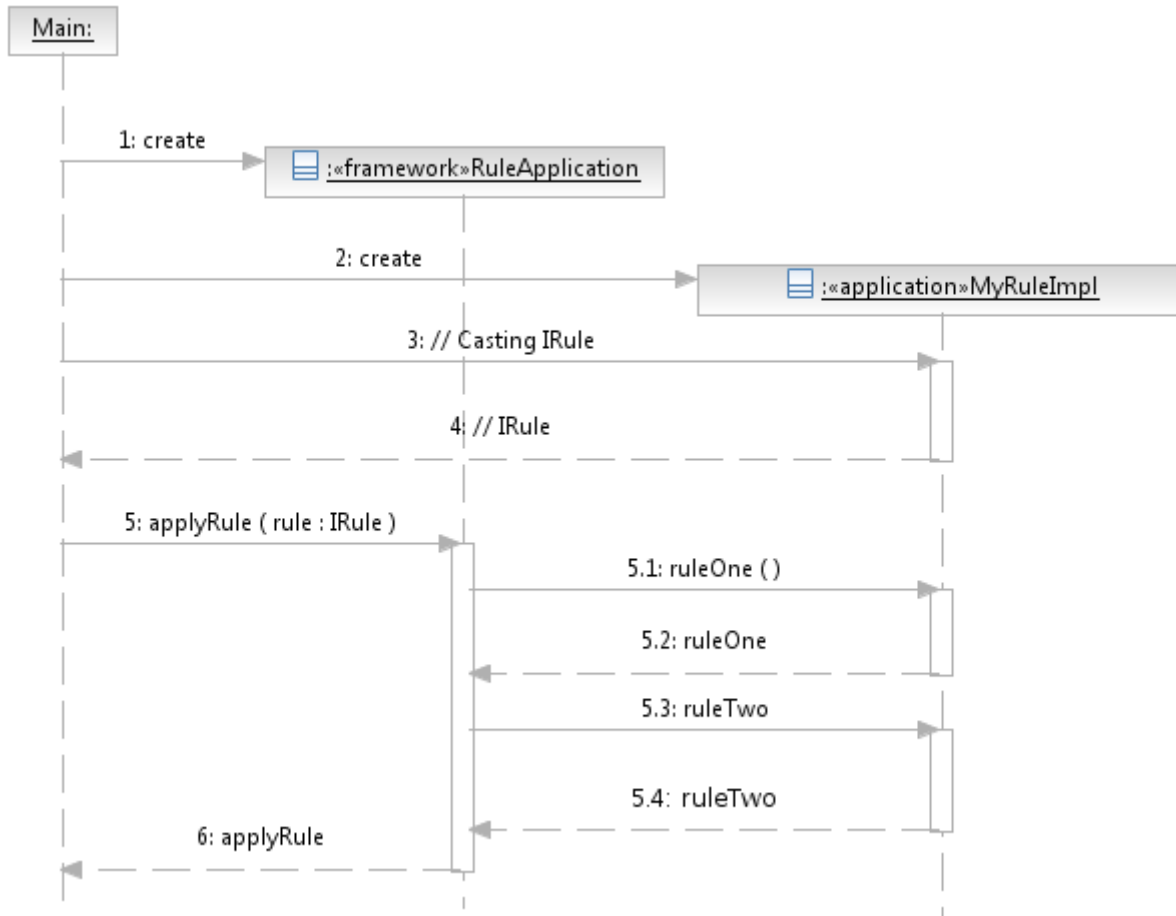
애플리케이션 프레임워크

- 그레이박스 프레임워크(gray-box framework)
 - 화이트박스 프레임워크와 블랙박스 프레임워크의 장단점을 혼합



애플리케이션 프레임워크

- 그레이박스 프레임워크(gray-box framework)



Spring.NET 프레임워크 활용 가이드

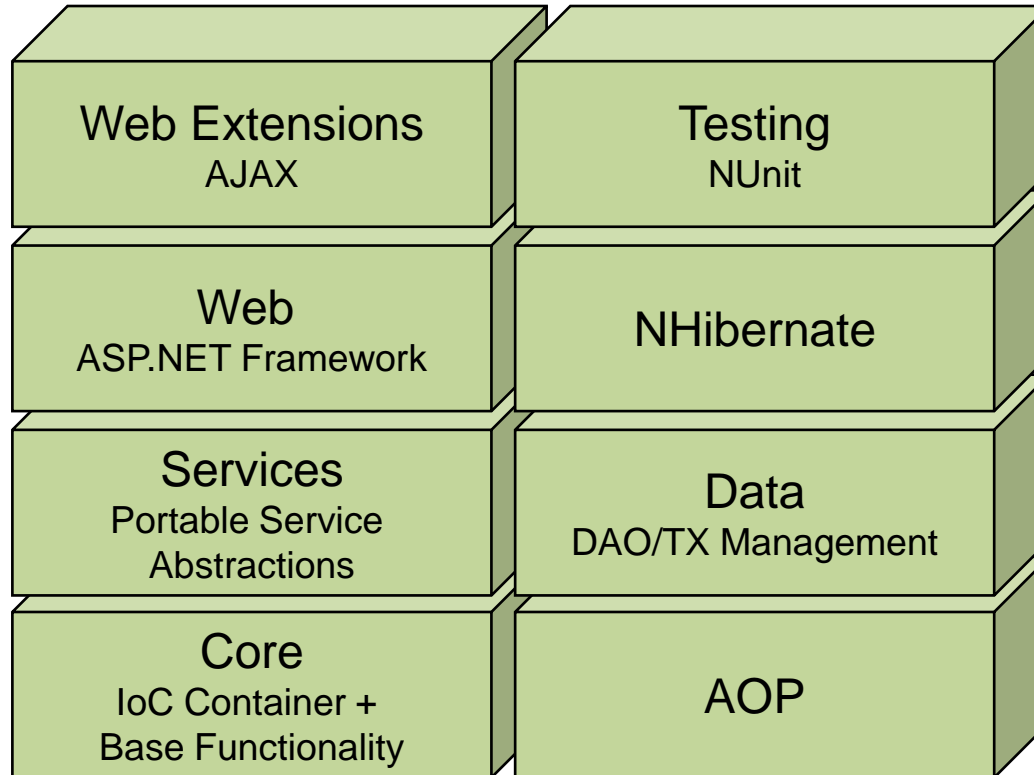
- 애플리케이션 프레임워크
- Spring.NET 프레임워크
- IoC 컨테이너와 DI

Spring.NET 프레임워크

- Spring.NET
 - 객체의 라이프사이클을 관리하기 위하여 Dependency Injection(DI)를 사용하는 경량(lightweight) 컨테이너
 - 엔터프라이즈 급 .NET 애플리케이션 개발을 위한 애플리케이션 프레임워크
 - J2EE 애플리케이션 개발을 위해 만든 자바 진영의 Spring 프레임워크를 닷넷 환경에 맞도록 포팅하되 ASP.NET, Persistence, .NET Remoting 등의 닷넷 기술을 적극 활용함

Spring.NET 프레임워크

- Spring.NET 프레임워크의 주요 모듈



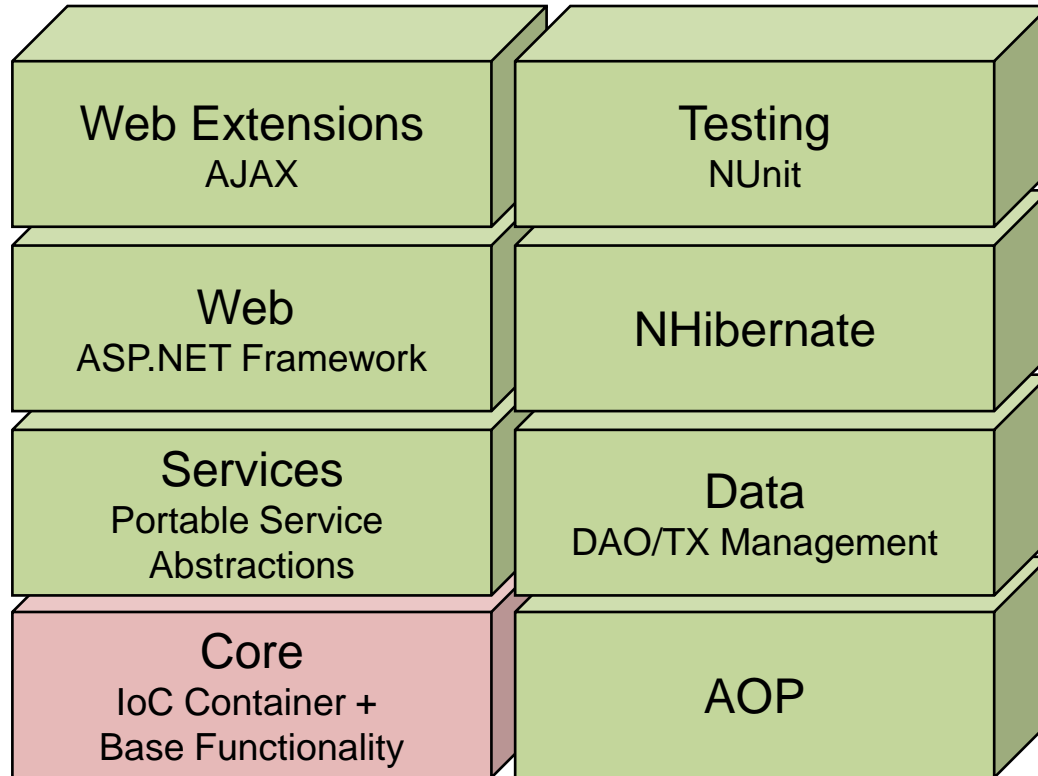
Spring.NET 프레임워크

- Spring.NET 프레임워크가 제공하는 기능들
 - DI(dependency injection)
 - AOP(aspect-oriented programming)
 - 데이터 액세스 지원 : ORM(object-relation mapping) 프레임워크인 NHibernate 지원을 포함
 - 웹 프레임워크
 - 통합 테스트
 - 동적 리플렉션(dynamic reflection)
 - 스레딩(threading) 및 동시성(concurrency)
 - 객체 풀링(object pooling)
 - 표현식 평가(expression evaluation)
 - 서비스
 - 기타...

Spring.NET 프레임워크

- Spring.NET 프레임워크의 장점
 - 모듈화 및 이해하기 쉬움
 - 코드를 테스트하기 쉬움
 - 일관성 있는 데이터 액세스 프레임워크를 제공함
 - 일관성 있는 구성(configuration)
 - 확장하기 쉬움
 - 처음부터 하지 않아도 됨

IoC 컨테이너와 DI



IoC 컨테이너와 DI

- 다음 코드를 생각해 보자

```
public class A {  
    public B cb; // A는 B에 종속된다  
    public A() { cb = get_B_Object(...); }  
    ...  
}  
public class B { ... }  
public class Container {  
    public A createA() {  
        return new A();  
    }  
}
```

- 문제점
 - A는 B 인스턴스를 생성하는 방법에 종속된다

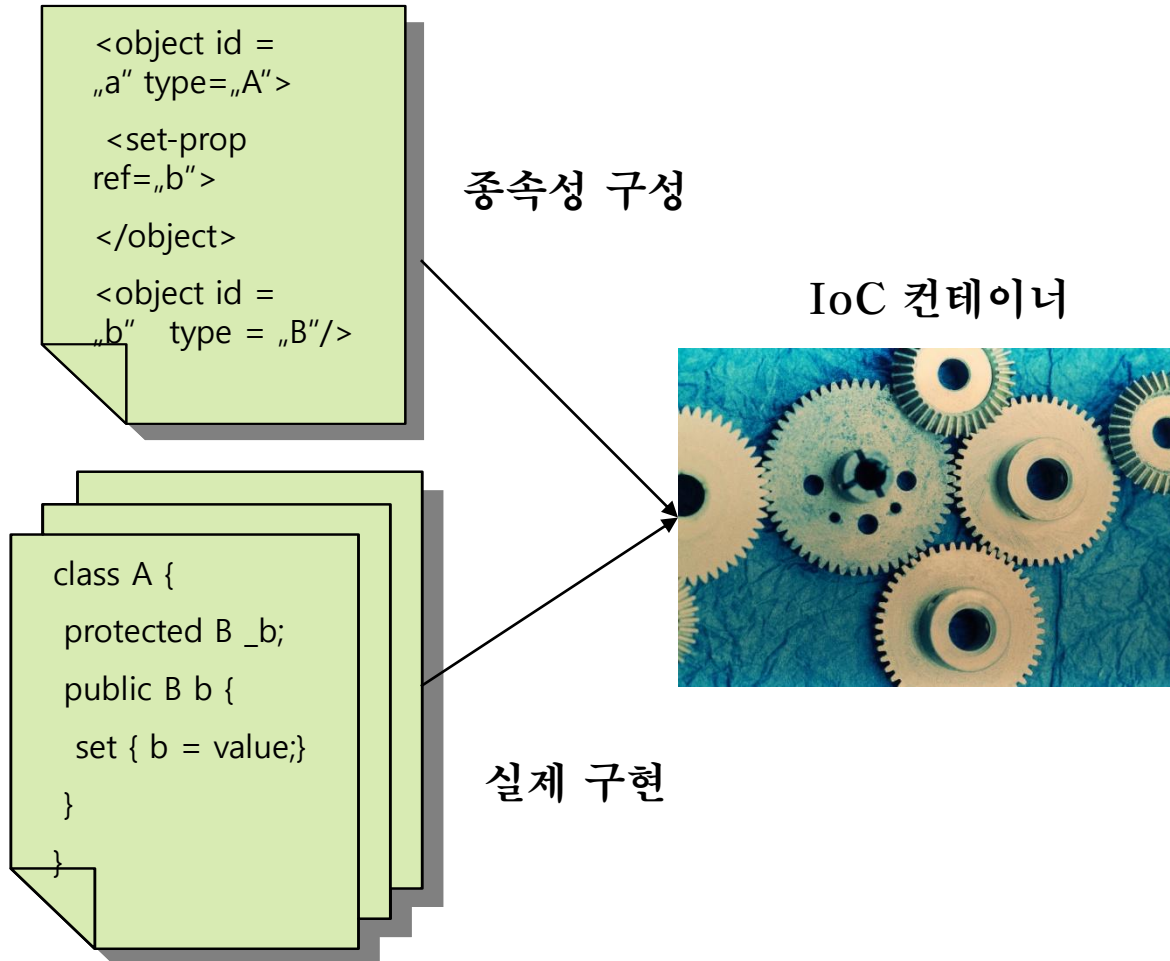
IoC 컨테이너와 DI

- 종속성 제거

```
public class A {  
    protected B b1; // A는 B에 종속된다  
    protected B b2;  
    public A(B b1) { this.b1 = b1; } // dependency injection  
    public B B2 { set { this.b2 = value; } } // dependency injection  
    ...  
}  
public class B { ... }  
public class IoCContainer {  
    public A createA() {  
        B b1 = get_B_Object(...);  
        A a = new A(b1); // constructor injection  
        B b2 = get_B_Object(...);  
        a.B2 = b2; // setter aka property injection  
        return a;  
    }  
}
```

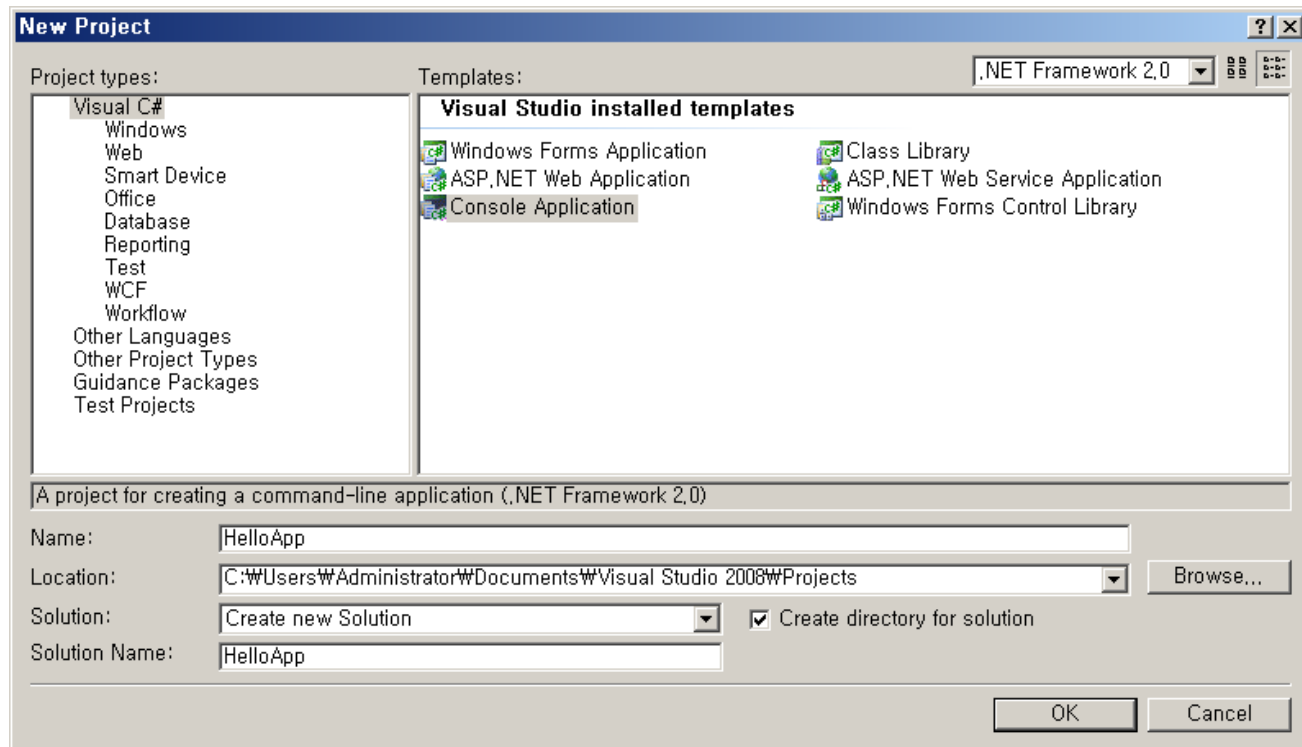
IoC 컨테이너와 DI

- DI(Dependency Injection)



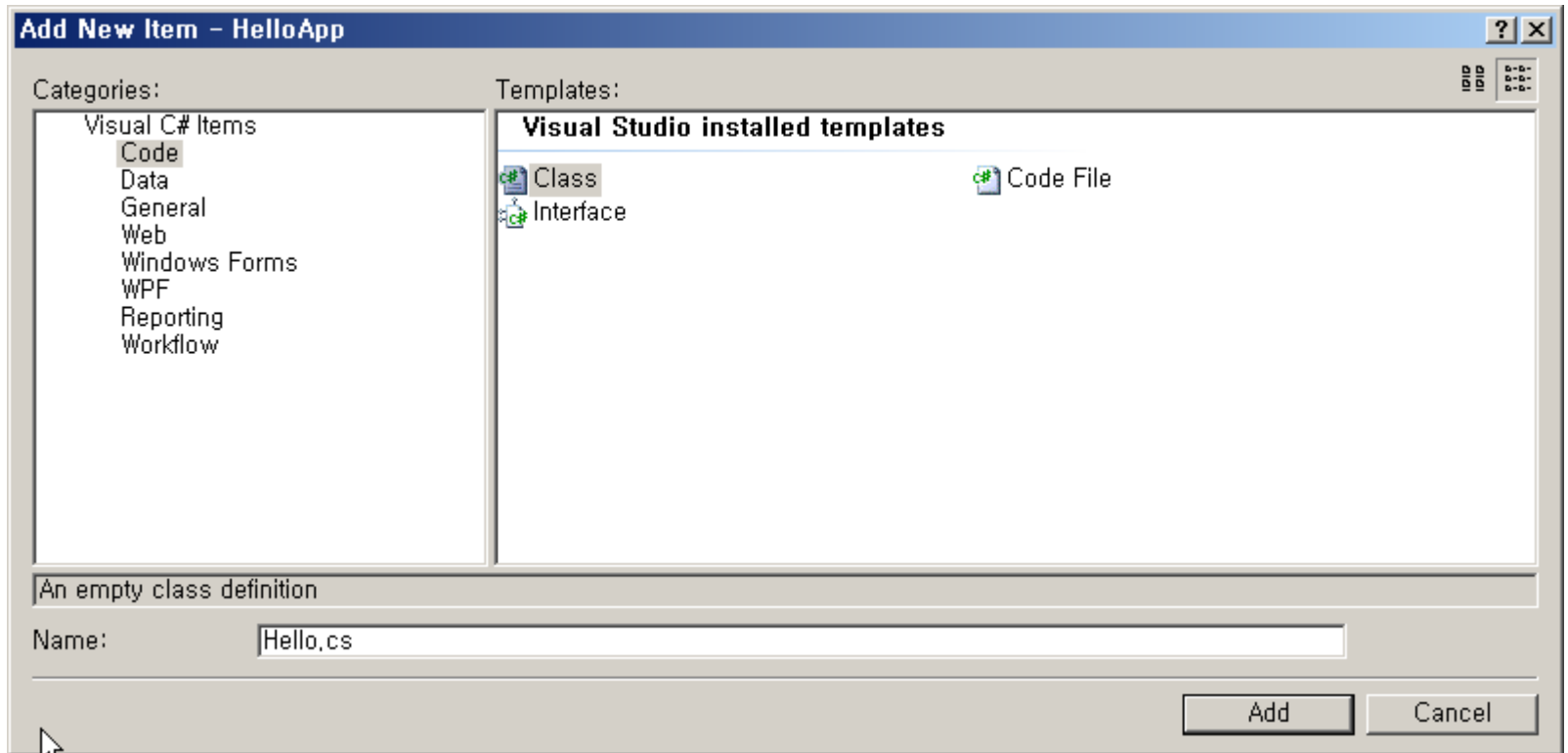
첫번째 Spring.NET 애플리케이션

- 프로젝트 생성
 - HelloApp 콘솔 애플리케이션



첫번째 Spring.NET 애플리케이션

- Hello 클래스 생성



첫번째 Spring.NET 애플리케이션

- Hello 클래스 구현

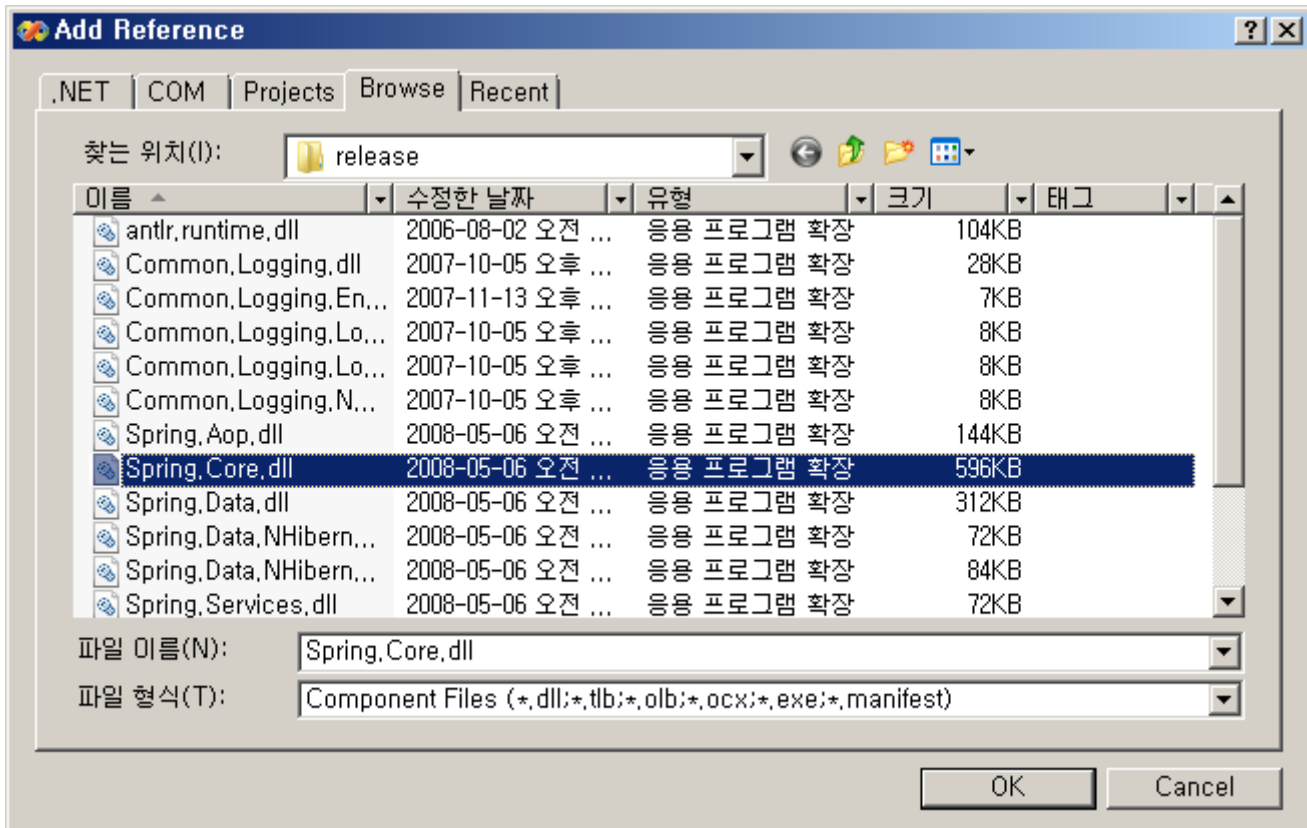
```
// 파일명: Hello.cs

using System;
using System.Collections.Generic;
using System.Text;

namespace HelloApp
{
    class Hello
    {
        public string sayHello(string name)
        {
            return "안녕하세요? " + name + "씨!";
        }
    }
}
```

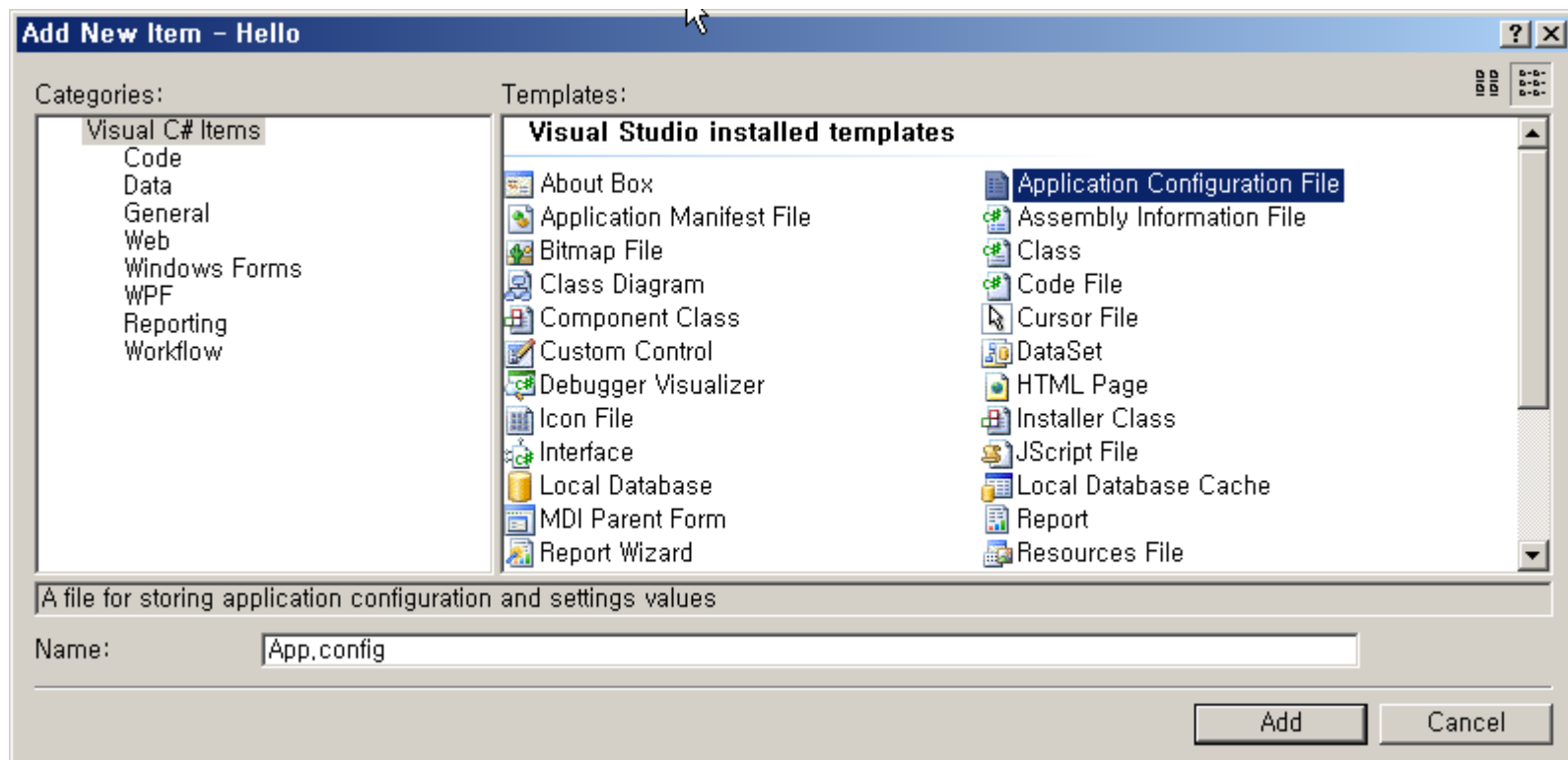
첫번째 Spring.NET 애플리케이션

- Spring.NET 프레임워크 참조
 - Spring.core.dll



첫 번째 Spring.NET 애플리케이션

- 애플리케이션 구성 파일 생성



첫번째 Spring.NET 애플리케이션

- 구성 설정

// 파일명: App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core"/>
      <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
    </sectionGroup>
  </configSections>
  <spring>
    <context>
      <!-- using section in App.config -->
      <resource uri="config://spring/objects"/>
    </context>
    <objects xmlns="http://www.springframework.net" >
      <description>첫번째 Spring.NET 애플리케이션</description>
      <object id="MyHello"
        type="HelloApp.Hello, HelloApp">
      </object>
    </objects>
  </spring>
</configuration>
```


첫번째 Spring.NET 애플리케이션

- Hello 클래스 사용

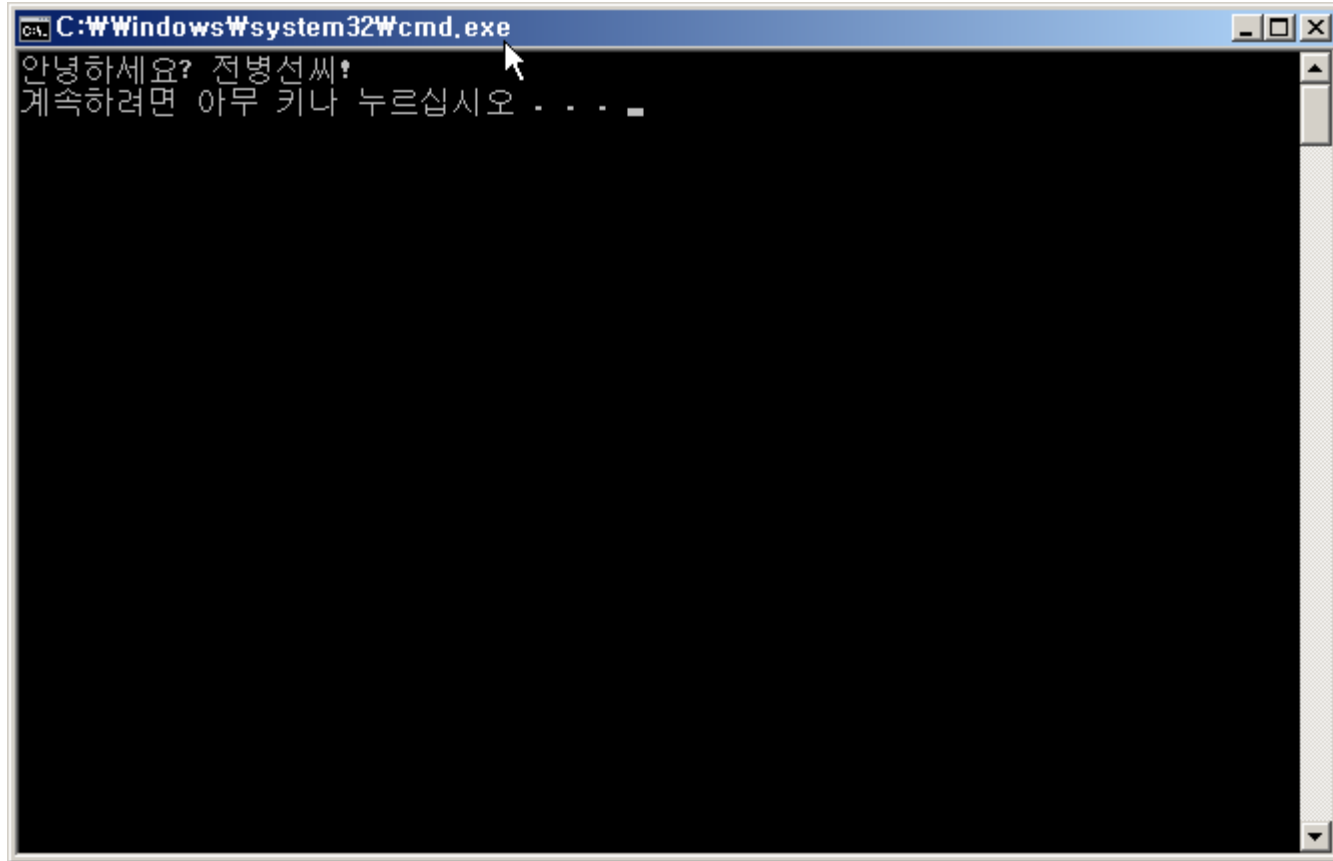
```
// 파일명: Program.cs

using System;
using System.Collections.Generic;
using System.Text;
using Spring.Context;
using Spring.Context.Support;

namespace HelloApp {
    class Program {
        static void Main(string[] args) {
            IApplicationContext ctx = ContextRegistry.GetContext();
            Hello hello = (Hello)ctx.GetObject("MyHello");
            string name = "전병선";
            string result = hello.sayHello(name);
            Console.WriteLine(result);
        }
    }
}
```

첫번째 Spring.NET 애플리케이션

- HelloApp 애플리케이션 실행



첫번째 Spring.NET 애플리케이션

• 구성과 코드

- MyHello 객체 정의

// 파일명: App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextSection" />
      <section name="objects" type="Spring.Context.Support.ObjectsSection" />
    </sectionGroup>
  </configSections>
  <spring>
    <context>
      <!-- using section in App.config -->
      <resource uri="config://spring/objects"/>
    </context>
    <objects xmlns="http://www.springframework.net" >
      <description>첫번째 Spring.NET 애플리케이션</description>
      <object id="MyHello"
        type="HelloApp.Hello, HelloApp">
      </object>
    </objects>
  </spring>
</configuration>
```

// 파일명: Hello.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloApp
{
  class Hello
  {
    public string sayHello(string name)
    {
      return "안녕하세요? " + name + "씨!";
    }
  }
}
```

Configuration: N/A Platform: x86

Assembly name: HelloApp

Target Framework: .NET Framework 2.0

Startup object: (Not set)

첫번째 Spring.NET 애플리케이션

• 구성과 코드

- MyHello 객체 사용

// 파일명: App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Core.ContextConfig, Spring.Core" />
      <section name="objects" type="Spring.Core.ObjectConfig, Spring.Core" />
    </sectionGroup>
  </configSections>
  <spring>
    <context>
      <!-- using section in App.config -->
      <resource uri="config://spring/objects"/>
    </context>
    <objects xmlns="http://www.springframework.org/schema/beans" >
      <description>첫번째 Spring.NET 애플리케이션</description>
      <object id="MyHello" type="HelloApp.Hello, HelloApp">
      </object>
    </objects>
  </spring>
</configuration>
```

// 파일명: Program.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using Spring.Context;
using Spring.Context.Support;

namespace HelloApp {
  class Program {
    static void Main(string[] args) {
      IApplicationContext ctx = ContextRegistry.GetContext();
      Hello hello = (Hello)ctx.GetObject("MyHello");
      string name = "전병선";
      string result = hello.sayHello(name);
      Console.WriteLine(result);
    }
  }
}
```

첫번째 Spring.NET 애플리케이션

- IHello 인터페이스 도입
 - Hello 클래스와 Program 클래스 사이의 종속성 제거

```
// 파일명: IHello.cs

using System;
using System.Collections.Generic;
using System.Text;

namespace HelloApp
{
    interface IHello
    {
        string sayHello(string name);
    }
}
```

첫번째 Spring.NET 애플리케이션

- IHello 인터페이스 도입
 - Hello 클래스와 Program 클래스 코드 변경

// 파일명: Hello.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace HelloApp
{
    class Hello : IHello
    {
        public string sayHello(string name)
        {
            return "안녕하세요? " + name + "씨!";
        }
    }
}
```

// 파일명: Program.cs

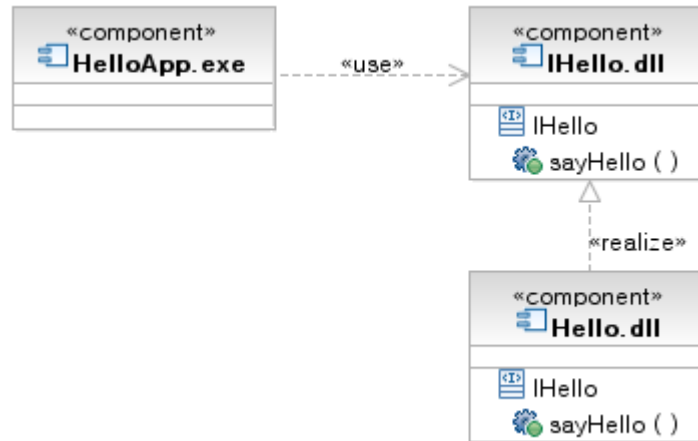
```
using System;
using System.Collections.Generic;
using System.Text;
using Spring.Context;
using Spring.Context.Support;

namespace HelloApp {
    class Program { {
        static void Main(string[] args) {
            ApplicationContext ctx = ContextRegistry.GetContext();
            IHello hello = (IHello)ctx.GetObject("MyHello");
            string name = "전병선";
            string result = hello.sayHello(name);
            Console.WriteLine(result);
        }
    }
}
```

첫번째 Spring.NET 애플리케이션

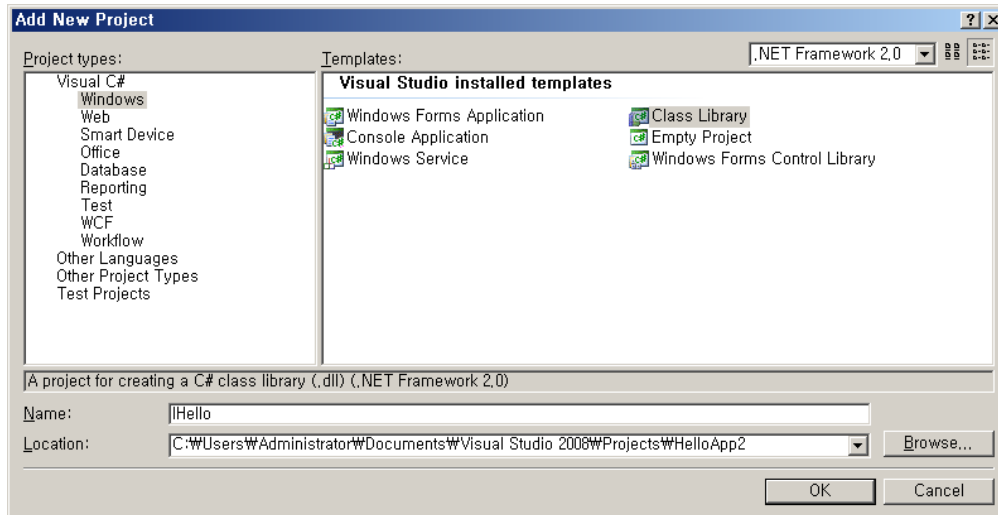
- Hello 컴포넌트 도입

- Hello 클래스의 기능을 HelloApp 애플리케이션으로부터 분리하여 별도의 라이브러리로 구현
- HelloApp 애플리케이션과 Hello 컴포넌트 사이의 종속성 제거



첫번째 Spring.NET 애플리케이션

- IHello 인터페이스 컴포넌트
 - IHello 인터페이스 정의



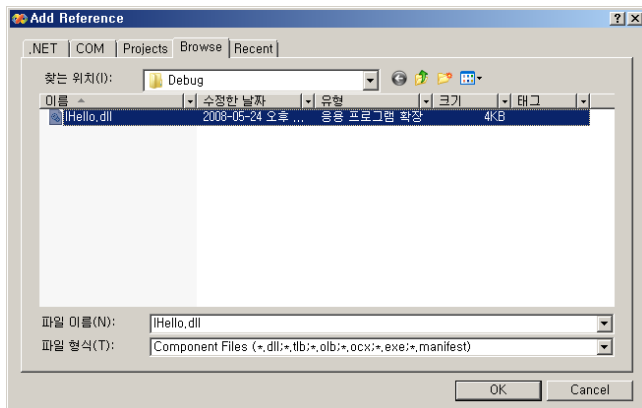
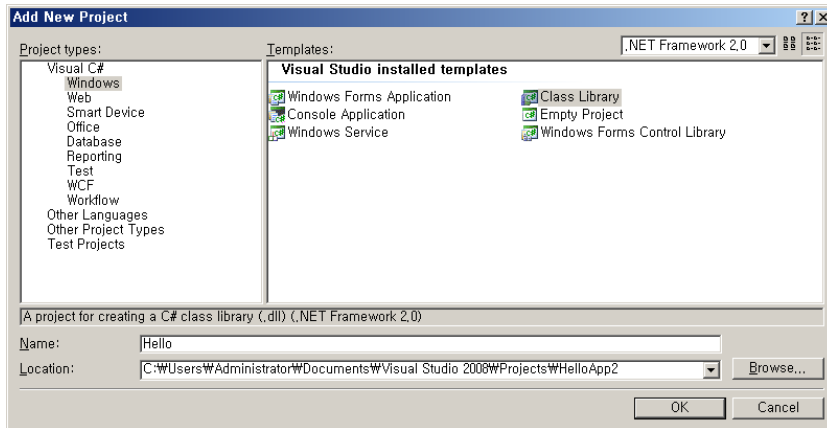
// 파일명: IHello.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Hello
{
    interface IHello
    {
        string sayHello(string name);
    }
}
```


첫번째 Spring.NET 애플리케이션

- Hello 구현 컴포넌트
 - IHello 인터페이스 실현



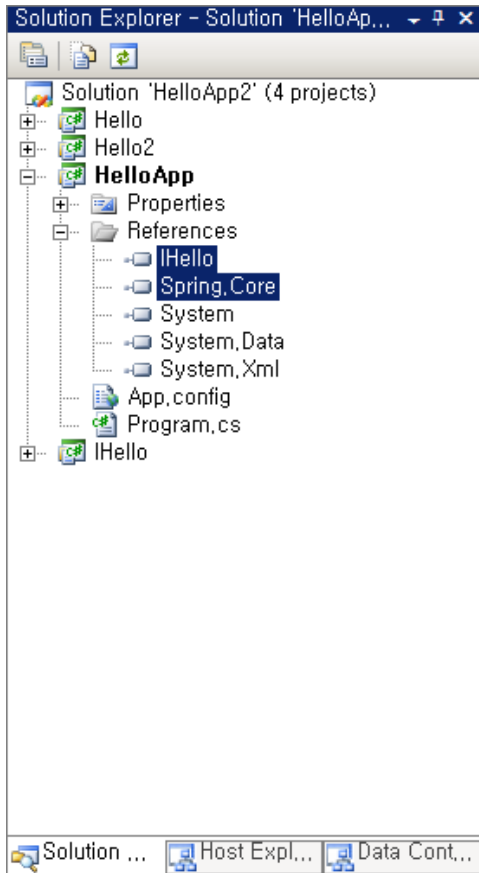
// 파일명: Hello.cs

```
using System;  
using System.Collections.Generic;  
using System.Text;
```

```
namespace Hello  
{  
    class Hello : IHello  
    {  
        public string sayHello(string name)  
        {  
            return "안녕하세요? " + name + "씨!";  
        }  
    }  
}
```

첫번째 Spring.NET 애플리케이션

- HelloApp 애플리케이션
 - IHello 인터페이스를 통해 Hello 컴포넌트 사용



```
// 파일명: Program.cs
using System;
using System.Collections.Generic;
using System.Text;
using Spring.Context;
using Spring.Context.Support;
using Hello;

namespace HelloApp {
    class Program {
        static void Main(string[] args) {
            ApplicationContext ctx = ContextRegistry.GetContext();
            IHello hello = (IHello)ctx.GetObject("MyHello");
            string name = "전병선";
            string result = hello.sayHello(name);
            Console.WriteLine(result);
        }
    }
}
```

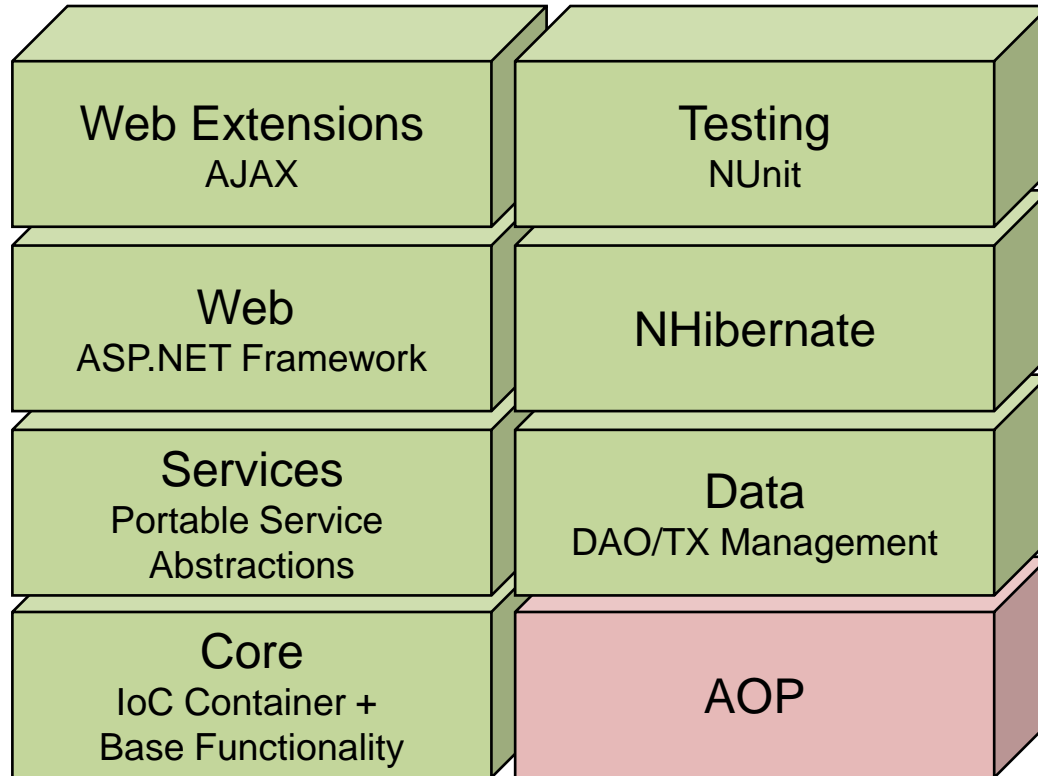
첫번째 Spring.NET 애플리케이션

- HelloApp 애플리케이션의 Hello 컴포넌트 참조

// 파일명: App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="spring">
      <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core"/>
      <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
    </sectionGroup>
  </configSections>
  <spring>
    <context>
      <!-- using section in App.config -->
      <resource uri="config://spring/objects"/>
    </context>
    <objects xmlns="http://www.springframework.net" >
      <description>첫번째 Spring.NET 애플리케이션</description>
      <object id="MyHello"
        type="Hello.Hello, Hello">
      </object>
    </objects>
  </spring>
</configuration>
```

AOP (Aspect-Oriented Programming)

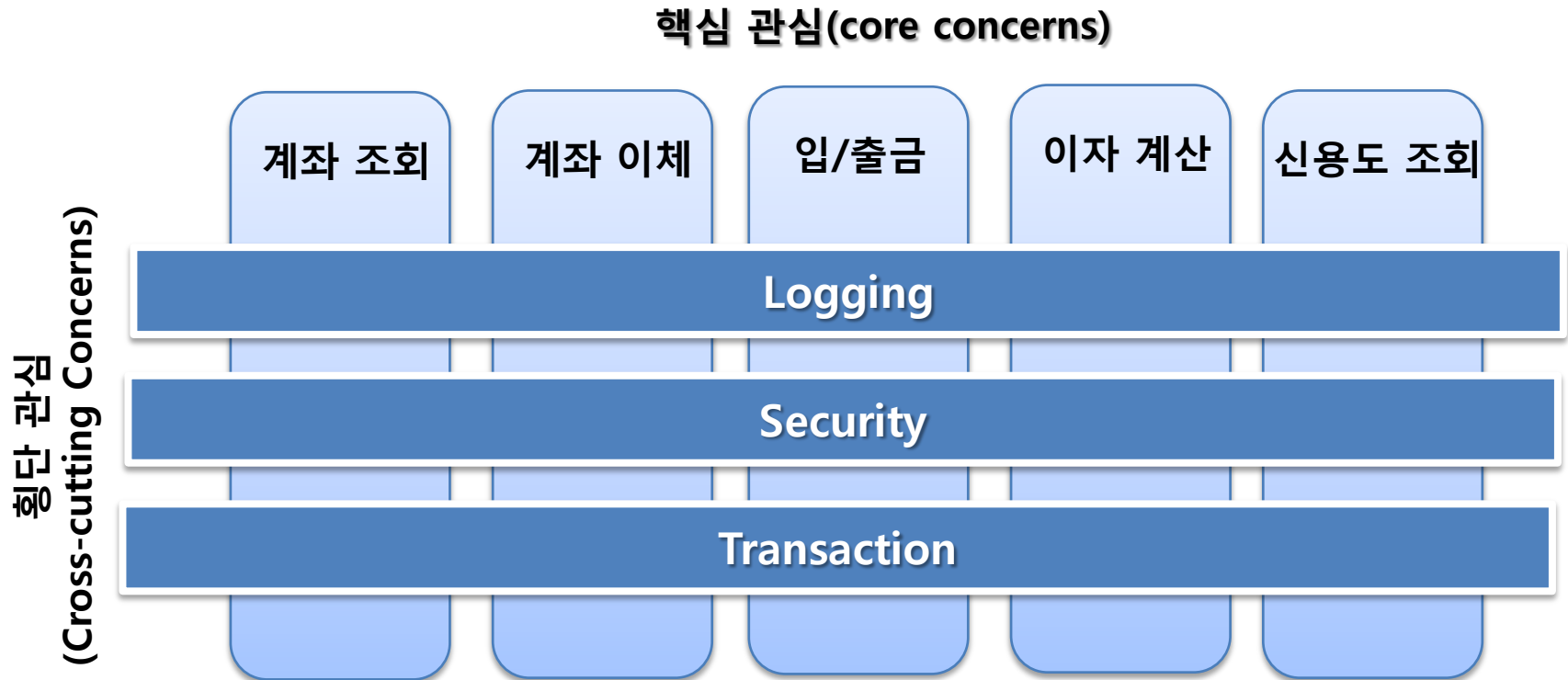


AOP (Aspect-Oriented Programming)

- 시스템 설계/개발 중에 늘 만나게 되는 문제들...
 - 로깅 (Logging)
 - 보안/인증 (security/authentication)
 - 트랜잭션(transaction)
 - 리소스 풀링(resource pooling)
 - 에러 검사(error checking)
 - 정책 적용(policy enforcement)
 - 멀티 쓰레드 안전 관리(multithread safety)
 - 데이터 퍼시스턴스 (data persistence)
 - ...
- 객체지향 설계에 의해 핵심 요구사항과 기능들을 클래스/컴포넌트들에 잘 모듈화하였지만 여전히 공통적으로 나타나는 문제들을 어떻게 처리할 것인가?

AOP (Aspect-Oriented Programming)

- OOP로 횡단 관심사들을 해결할 수 있을까?
- AOP는 OOP를 대체할 수 있을까?



AOP (Aspect-Oriented Programming)

- 관점 지향 프로그래밍이란?
 - 관심의 분리(Separation of Concern)를 통해 문제 영역(problem domain)을 핵심 관심사(core concerns)와 횡단 관심사(cross-cutting concerns)의 독립적 모듈로 분해하는 프로그래밍 패러다임.
- 장점
 - 관심의 분리(Separation of Concern)를 이끌어 낸다.
 - 비즈니스 로직에 대한 이해도를 높일 수 있다.
 - 생산성을 향상시킨다.
 - 비즈니스 코드와 횡단 관심사들간의 결합을 없앤다.
 - 비즈니스 코드의 재사용성을 높인다.
 - 확장이 용이하다.

AOP (Aspect-Oriented Programming)

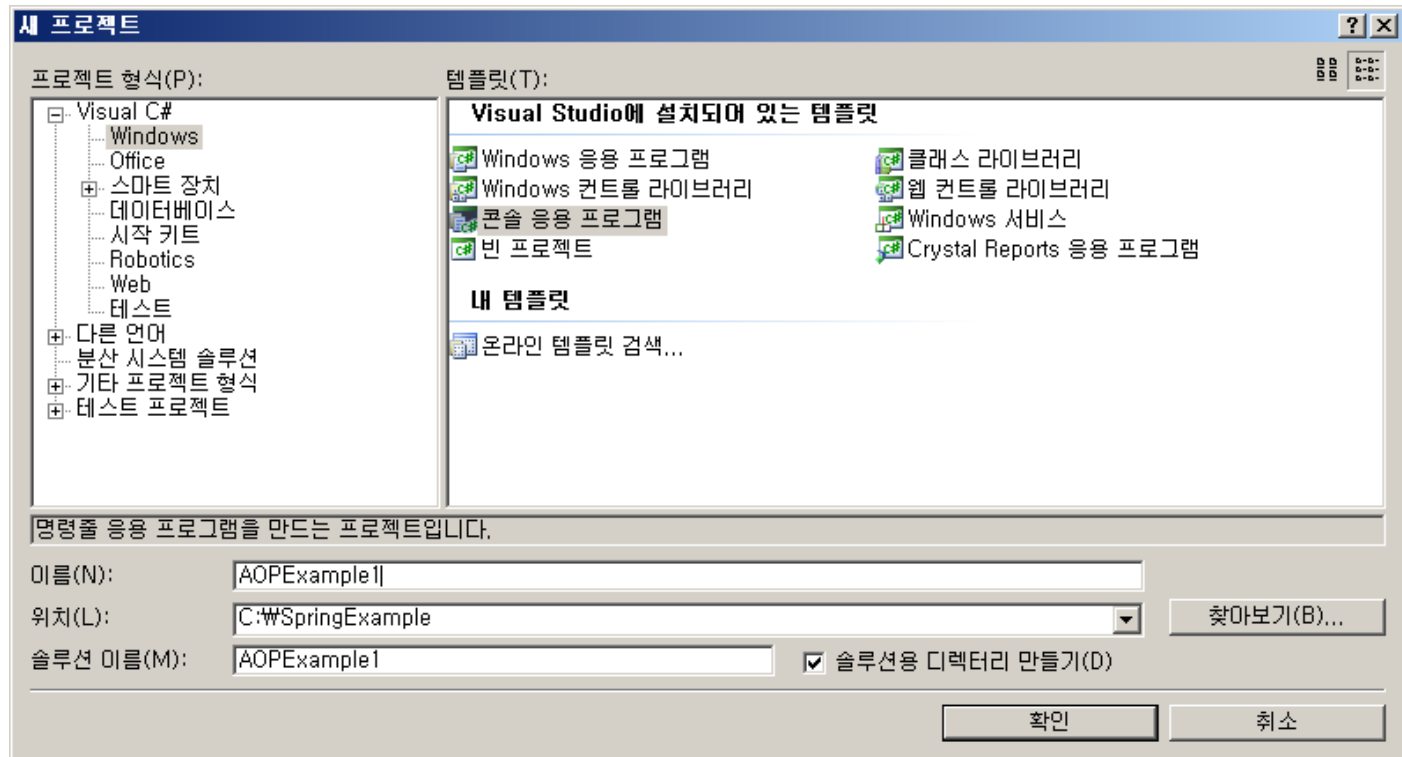
- AOP 기본 용어

- cross-cutting concerns
 - 보안,로그,인증과 같이 시스템에 전반적으로 산재된 기능들
- Aspect
 - 포인트컷(어디에서)과 어드바이스(무엇을 할 것인지)를 합쳐놓은 것.
- Joinpoint
 - 횡단 관심 모듈의 기능이 삽입되어 동작할 수 있는 실행 가능한 특정위치.
- Advice/interceptor
 - 각 조인포인트에 삽입되어져 동작할 수 있는 코드
- Pointcut
 - 어떤 클래스의 어느 조인포인트를 사용할 것인지를 결정하는 선택 기능
- Weaving / 크로스컷팅
 - 포인트컷에 의해서 결정된 조인포인트에 지정된 어드바이스를 삽입하는 과정

참고 : <http://www.zdnet.co.kr/builder/dev/java/0,39031622,39147106,00.htm>

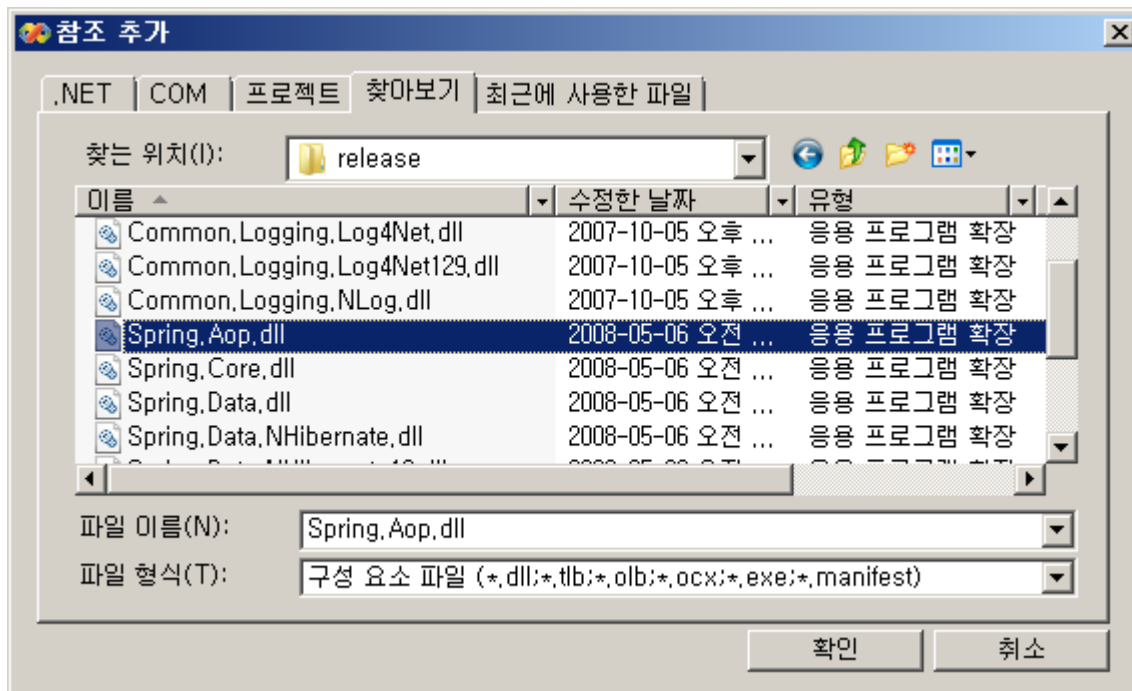
두번째 Spring.NET 애플리케이션

- 프로젝트 생성
 - AOPEXample1 콘솔 애플리케이션



두번째 Spring.NET 애플리케이션

- 참조 추가
 - Spring.Aop.dll



두번째 Spring.NET 애플리케이션

- 핵심 관심사를 수행하는 컴포넌트를 위한 인터페이스 생성
 - ICommand.cs

```
// ICommand.cs 파일

using System;
using System.Collections.Generic;
using System.Text;

namespace Spring.AOPEXample.Command
{
    public interface ICommand
    {
        object Execute(object context);
    }
}
```

두번째 Spring.NET 애플리케이션

- 핵심 관심사를 수행하는 컴포넌트를 위한 클래스 생성
 - ServiceCommand.cs

```
// ServiceComponent.cs 파일

using System;
using System.Collections.Generic;
using System.Text;

namespace Spring.AOPExample.Command
{
    // advised object
    public class ServiceCommand : ICommand
    {
        #region ICommand Members

        public object Execute(object context)
        {
            Console.WriteLine("\tService implementation : [{0}]", context);
            return null;
        }

        #endregion
    }
}
```

두번째 Spring.NET 애플리케이션

- 횡단 관심사를 처리하는 Advice 생성 (around advice)
 - ConsoleLoggingAroudAdvice.cs

```
// ConsoleLoggingAroudAdvice.cs 파일

using System;
using System.Collections.Generic;
using System.Text;

using AopAlliance.Intercept;

namespace Spring.AOPExample.Advice
{
    // around advice
    public class ConsoleLoggingAroudAdvice : IMethodInterceptor
    {
        #region IMethodInterceptor Members
        public object Invoke(IMethodInvocation invocation)
        {
            Console.WriteLine("Advice executing; calling the advised method...");
            object returnValue = invocation.Proceed();
            Console.WriteLine("Advice executed; advised method returned " + returnValue);
            return returnValue;
        }
        #endregion
    }
}
```

두번째 Spring.NET 애플리케이션

- Weaving 작업

```
// Program.cs

using System;
using System.Collections.Generic;
using System.Text;
using Spring.Aop.Framework;

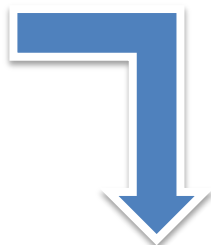
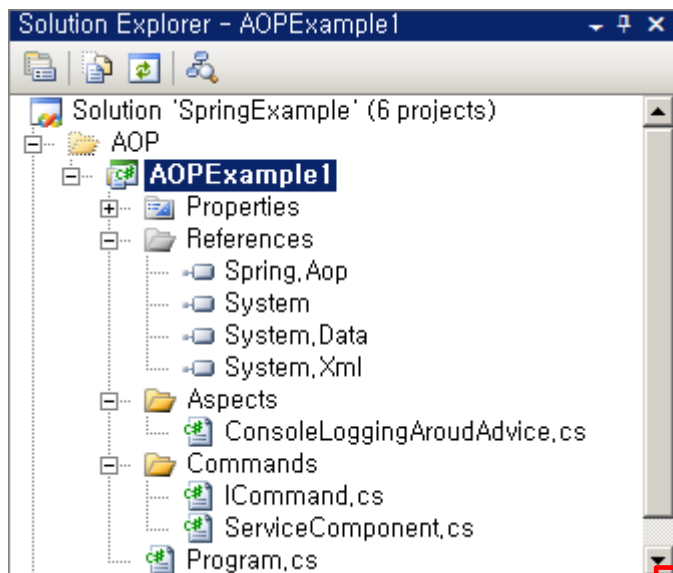
using Spring.AOPExample.Advice;
using Spring.AOPExample.Command;

namespace Spring.AOPExample.Proxy
{
    class Program
    {
        static void Main(string[] args)
        {
            ProxyFactory factory = new ProxyFactory(new ServiceCommand());
            factory.AddAdvice(new ConsoleLoggingAroudAdvice());
            ICommand command = (ICommand)factory.GetProxy();
            command.Execute("This is the argument");

            Console.ReadLine();
        }
    }
}
```

두번째 Spring.NET 애플리케이션

- 실행 결과



```
file:///C:/Temp/Share/Spring/HelloApp2/AOPExample1/bin/Deb...  
Advice executing; calling the advised method...  
    Service implementation : [This is the argument]  
Advice executed; advised method returned
```

두번째 Spring.NET 애플리케이션

- Configuration 사용하기 (App.config 파일)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>...

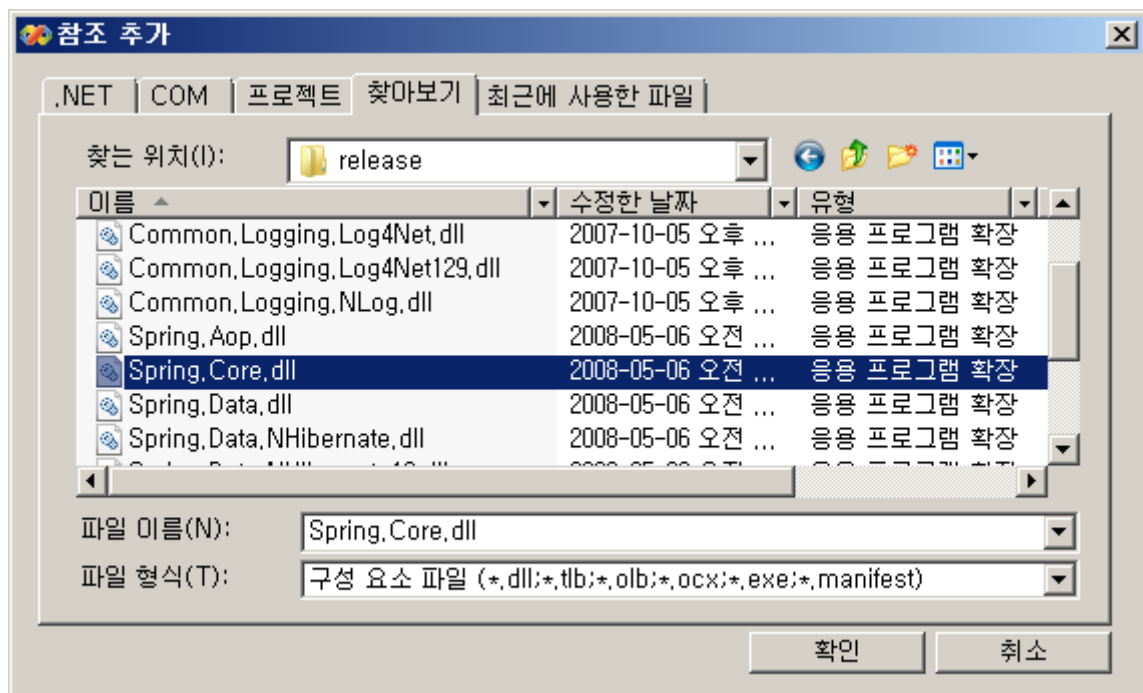
  <spring>
    <context>
      <resource uri="config://spring/objects" />
    </context>

    <objects xmlns="http://www.springframework.net">
      <description>AOP 테스트를 위한 두번째 Spring.NET 애플리케이션</description>
      <object id="ConsoleLoggingAroudAdvice"
        type="Spring.AOPExample.Advice.ConsoleLoggingAroudAdvice" />

      <object id="myServiceCommand" type="Spring.Aop.Framework.ProxyFactoryObject">
        <property name="target">
          <object id="myServiceObjectTarget"
            type="Spring.AOPExample.Command.ServiceCommand" />
        </property>
        <property name="InterceptorNames">
          <list>
            <value>ConsoleLoggingAroudAdvice</value>
          </list>
        </property>
      </object>
    </objects>
  </spring>
</configuration>
```


두번째 Spring.NET 애플리케이션

- Configuration 이용하기
 - 참조 추가 : Spring.Core.dll



두번째 Spring.NET 애플리케이션

- Configuration 이용하기
 - ProxyFactory 대신 context 사용하기

```
// Program.cs

using System;
using System.Collections.Generic;
using System.Text;

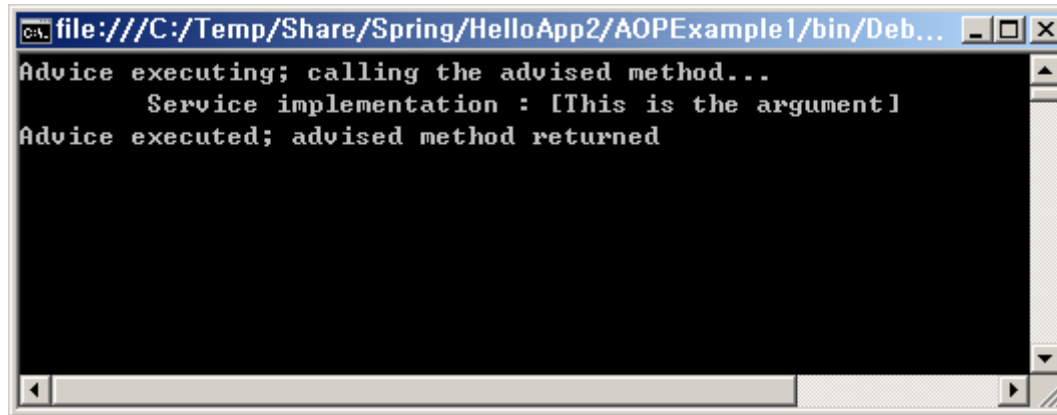
using Spring.Context;
using Spring.Context.Support;

using Spring.AOPExample.Advice;
using Spring.AOPExample.Command;

namespace Spring.AOPExample.Proxy
{
    class Program
    {
        static void Main(string[] args)
        {
            IApplicationContext ctx = ContextRegistry.GetContext();
            ICommand command = (ICommand)ctx["myServiceCommand"];
            command.Execute(null);
            Console.ReadLine();
        }
    }
}
```

두번째 Spring.NET 애플리케이션

- 실행 결과



```
file:///C:/Temp/Share/Spring/HelloApp2/AOPExample1/bin/Deb...
Advice executing; calling the advised method...
    Service implementation : [This is the argument]
Advice executed; advised method returned
```

두번째 Spring.NET 애플리케이션

• Configuration 분석

1. 적용할 Advice를 ConsoleLoggingAroundAdvice라는 이름의 객체로 등록.
2. Advice를 적용할 대상은 ServiceCommand 타입이며 myServiceObjectTarget이라는 객체 이름으로 myServiceCommand 객체에 등록된다.
3. 등록된 Advice는 myServiceCommand 객체의 타겟에 대하여 interceptor로서 동작한다.

```
<objects xmlns="http://www.springframework.net">
  <description>AOP 테스트를 위한 두번째 Spring.NET 애플리케이션</description>
  <object id="ConsoleLoggingAroundAdvice"
    type="Spring.AOPExample.Advice.ConsoleLoggingAroundAdvice" />

  <object id="myServiceCommand" type="Spring.Aop.Framework.ProxyFactoryObject">
    <property name="target">
      <object id="myServiceObjectTarget"
        type="Spring.AOPExample.Command.ServiceCommand" />
    </property>
    <property name="InterceptorNames">
      <list>
        <value>ConsoleLoggingAroundAdvice</value>
      </list>
    </property>
  </object>
</objects>
```

두번째 Spring.NET 애플리케이션

- Configuration 분석

```
class Program
{
    static void Main(string[] args)
    {
        IApplicationContext ctx = ContextRegistry.GetContext();
        ICommand command = (ICommand)ctx["myServiceCommand"];
        command.Execute(null);
        Console.ReadLine();
    }
}
```

```
<objects xmlns="http://www.springframework.org/schema/beans" >
  <description>AOP 테스트를 위한 두번째 </description>
  <object id="ConsoleLoggingAroundAdvice"
    type="Spring.AOPExample.Advice.ConsoleLoggingAroundAdvice" />

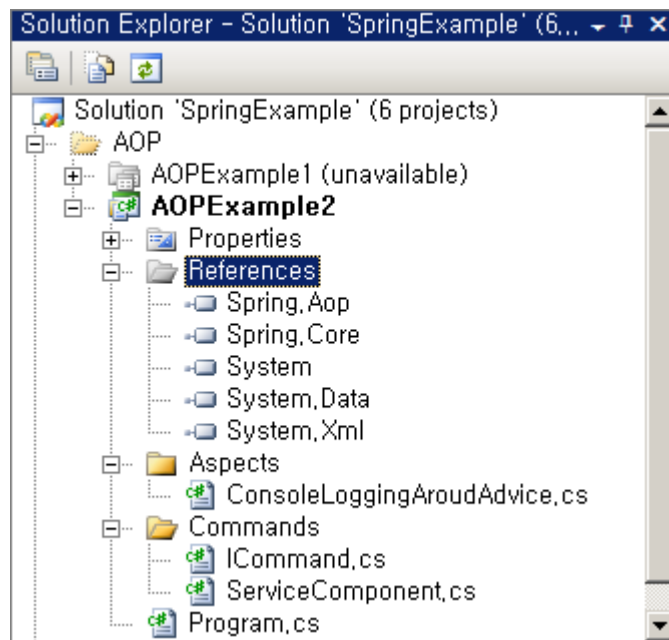
  <object id="myServiceCommand" type="Spring.Aop.Framework.ProxyFactoryObject">
    <property name="target">
      <object id="myServiceObjectTarget"
        type="Spring.AOPExample.Command.ServiceCommand" />
    </property>
    <property name="InterceptorNames">
      <list>
        <value>ConsoleLoggingAroundAdvice</value>
      </list>
    </property>
  </object>
</objects>
```

두번째 Spring.NET 애플리케이션

- 특정 메서드에만 Advice를 적용하려면?
- 특정 문자열로 시작하는 메서드들에만 Advice를 적용하려면?
 - 예) LogXXX(), LogYYY(), LogZZZ() 메서드에서는 로그를 남기고 싶다.
- → IPointcut 인터페이스 활용

두번째 Spring.NET 애플리케이션

- AOPExample2 프로젝트
 1. C# Console Application 생성
 2. AOPExample1 프로젝트에서 파일 복사
 - ConsoleLoggingAroundAdvice.cs
 - ICommand.cs
 - ServiceComponent.cs
 3. 참조 추가
 - Spring.Aop.dll
 - Spring.Core.dll



두번째 Spring.NET 애플리케이션

- AOPEXample2 프로젝트
 - ICommand 인터페이스에 메서드 추가

```
// ICommand.cs 파일

using System;
using System.Collections.Generic;
using System.Text;

namespace Spring.AOPEXample.Command
{
    public interface ICommand
    {
        object Execute(object context);
        void DoExecute(); // 새로 추가된 메서드
    }
}
```


두번째 Spring.NET 애플리케이션

- AOPEXample2 프로젝트
 - ServiceComponent 클래스 수정

```
// ServiceComponent.cs 파일

using System;
using System.Collections.Generic;
using System.Text;

namespace Spring.AOPEXample.Command
{
    // advised object
    public class ServiceCommand : ICommand
    {
        #region ICommand Members
        public object Execute(object context) {...}
        // 새로 추가된 메서드 구현
        public void DoExecute()
        {
            Console.WriteLine("\tService implementation : DoExecute()...");
        }
        #endregion
    }
}
```

두번째 Spring.NET 애플리케이션

- AOPEXample2 프로젝트

- ProxyFactory에 Advice 적용하기 : “Do”로 시작하는 메서드에만..

```
// Program.cs 파일
using System;
using System.Collections.Generic;
using System.Text;

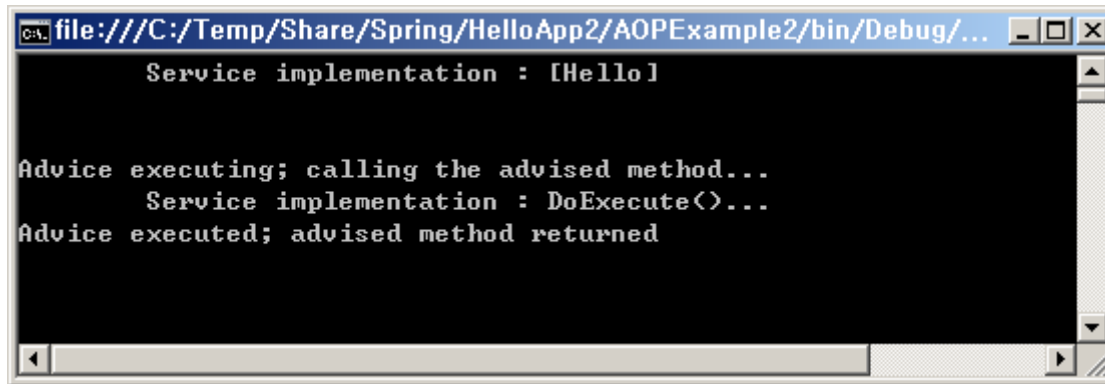
using Spring.Aop.Framework;
using Spring.Aop.Support;
using Spring.AOPEXample.Command;
using Spring.AOPEXample.Advice;

namespace AOPEXample2
{
    class Program
    {
        static void Main(string[] args)
        {
            ProxyFactory factory = new ProxyFactory(new ServiceCommand());
            factory.AddAdvisor(new DefaultPointcutAdvisor(
                new SdkRegularExpressionMethodPointcut("Do"),
                new ConsoleLoggingAroudAdvice()));

            ICommand command = (ICommand)factory.GetProxy();
            command.Execute("Hello"); // Advice 대상이 아님
            Console.WriteLine("\n");
            command.DoExecute();      // "Do"로 시작하는 메서드는 Advice 대상.
            Console.ReadLine();
        }
    }
}
```

두번째 Spring.NET 애플리케이션

- AOPExample2 프로젝트
 - 실행 결과



```
file:///C:/Temp/Share/Spring/HelloApp2/AOPExample2/bin/Debug/...  
Service implementation : [Hello]  
  
Advice executing; calling the advised method...  
    Service implementation : DoExecute()...  
Advice executed; advised method returned
```

- Execute()메서드 : Advice 적용되지 않았음
- DoExecute()메서드 : Advice가 적용되었음.

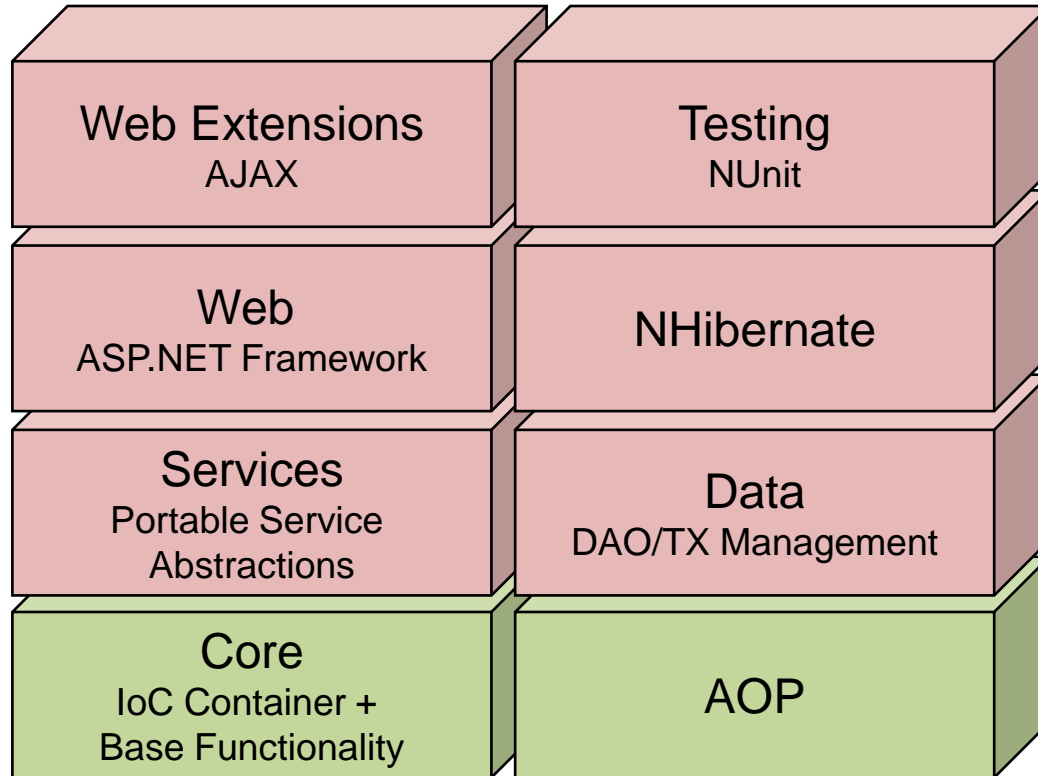
두번째 Spring.NET 애플리케이션

- AOPExample2 프로젝트
 - 구성 설정 이용하여 실행

```
<objects xmlns="http://www.springframework.net">
  <description>AOP 테스트를 위한 두번째 Spring.NET 애플리케이션</description>
  <object id="cLogAroudAdvice"
    type="Spring.Aop.Support.RegularExpressionMethodPointcutAdvisor">
    <property name="pattern" value="Do"/>
    <property name="advice">
      <object type="Spring.AOPExample.Advice.ConsoleLoggingAroudAdvice"/>
    </property>
  </object>

  <object id="myServiceCommand" type="Spring.Aop.Framework.ProxyFactoryObject">
    <property name="target">
      <object id="myServiceObjectTarget"
        type="Spring.AOPExample.Command.ServiceCommand" />
    </property>
    <property name="InterceptorNames">
      <list>
        <value>cLogAroudAdvice</value>
      </list>
    </property>
  </object>
</objects>
```

To be continued...



감사합니다!

전병선(田炳善)

- (주)엔소아컨설팅그룹 대표이사
- IT 아키텍트/컨설턴트
- bsjun@ensoa.co.kr
- 010-2059-4845

