스프링 프레임 워크

김은영/ rola70@nate.com
010-2559-8279

- Spring 프레임워크의 소개
- Spring 프레임워크 설치
- Spring 프레임워크의 모듈 구성
- DI(Dependency Injection)의 이해
- AOP(Aspect Object Programming)의 이해

스프링 프레임워크란?

- Rod Johnson : EJB를 사용하지 않고 엔터프라이즈 어플리케이션을 개발하는 방법을 소개
  - > 스프링의 모태가 됨.

- 단순한 웹 어플리케이션 구축에서 금융시스템과 같은 복잡한 엔터프라이즈 어플리케이션 까지 사용범위가 확대됨

스프링의 특징

- 어플리케이션 프레임워크
- 경량(Lightweight) 컨테이너
- Dependency Injection(DI)
- Aspect Oriented Programming(AOP)

낮은 결합도 (loosely coupled)

- 하나의 오브젝트가 변경이 일어 날 때 관계를 맺고 있는 다른 오브젝트에게 변화를 요구하는 정도->결합도
- 하나의 변경이 발생할 때 마다 마치 파문이 이는 것처럼 다른 여타 모듈과 객체로 변경에 대한 요구가 전파되지 않는 상태
- 결합도가 높아지면 변경에 따른 작업량이 많아 지고, 변경으로 인해 버그가 발생할 가능성이 높아진다.

제어의 역전
IoC(Inversion of Control)

- 오브젝트가 자신이 사용할 오브젝트를 스스로 선택하지 않는다. 당연히 생성하지도 않는다.
- 또, 자신도 어떻게 만들어지고 어디서 사용되는지 알 수 없다.
- 모든 제어 권한을 자신이 아닌 다른 대상에게 위임하기 때문이다.
- 템플릿 메소드는 제어의 역전이라는 개념을 활용해 문제를 해결하는 디자인 패턴이다.
- 프레임워크도 제어의 역전 개념이 적용된 대표적인 기술이다.

- 라이브러리 : 라이브러리를 사용하는 어플리케이션 코드는 어플리케이션 흐름을 직접 제어한다. 동작하는 중에 필요한 기능이 있을 때 능동적으로 라이브러리를 사용한다.
- 프레임워크 : 어플리케이션 코드가 프레임워크에 의해 사용한다. 프레임워크 위에 개발한 클래스를 등록해두고, 프레임워크가 주도하는 중에 개발자가 만든 애플리케이션 코드를 사용하도록 만드는 방식이다.
- 애플리케이션 코드는 프레임워크가 짜 놓은 틀에서 수동적으로 동작해야 한다.

의존관계 주입(DI)

- 몇몇사람의 제안으로 스프링이 제공하는 IoC 방식을 핵심을 짚어주는 의존관계주입 (Dependency Injection)이라는 좀 더 의도가 명확히 드러나는 이름을 사용하기 시작함.

- 오브젝트 레퍼런스를 외부로부터 제공(주입) 받고 이를 통해 여타 오브젝트와 다이내믹하게 의존관계가 만들어지는 것이 핵심이다.

AOP

- OOP : 복잡해져 가는 애플리케이션의 요구조건과 기술적인 난해함을 모두 해결하는데 한계가 있음.

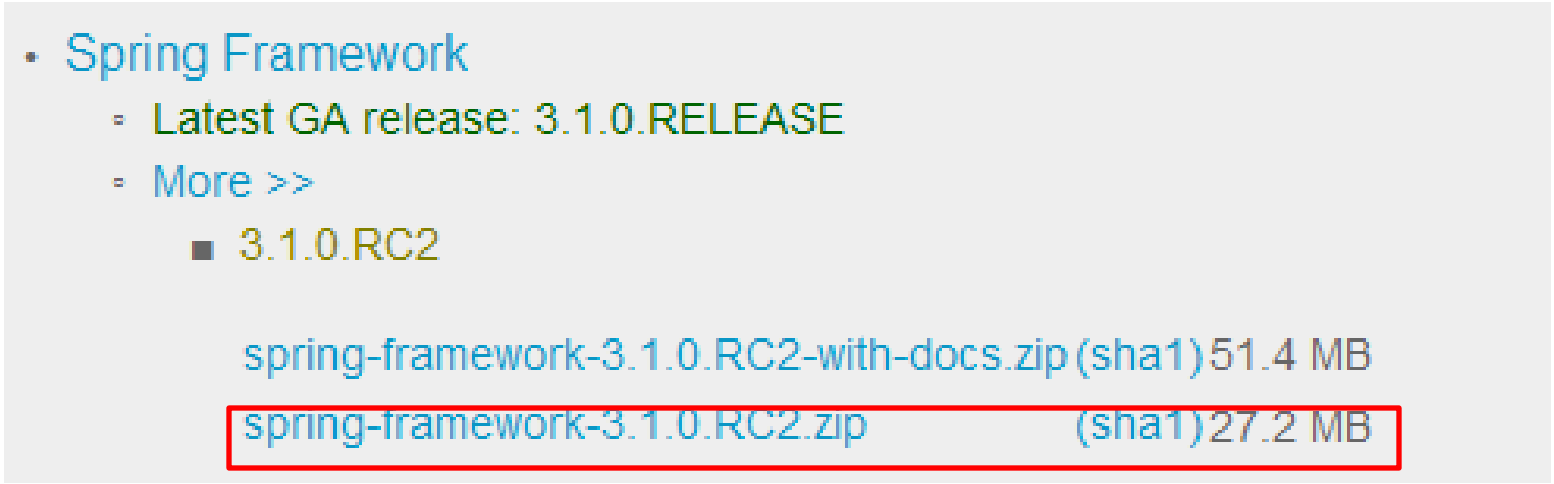
- AOP : 객체지향 기술의 한계와 단점을 극복하도록 도와주는 보조적인 프로그래밍 기술

스프링의 장점

- 필요한 인스턴스를 스프링에서 미리 생성해 준다.
- 클래스 사이의 결합(loosely coupled)을 느슨하게 할 수 있어 클래스 간의 의존 관계가 약해진다.

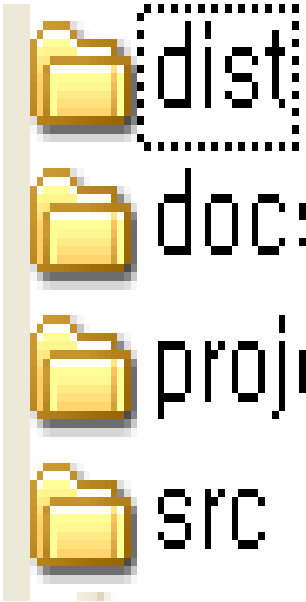
스프링의 환경설정

- http://www.springframework.org/download



- http://springide.org/updatesite
- http://dist.springframework.org/release/IDE

폴더 구성



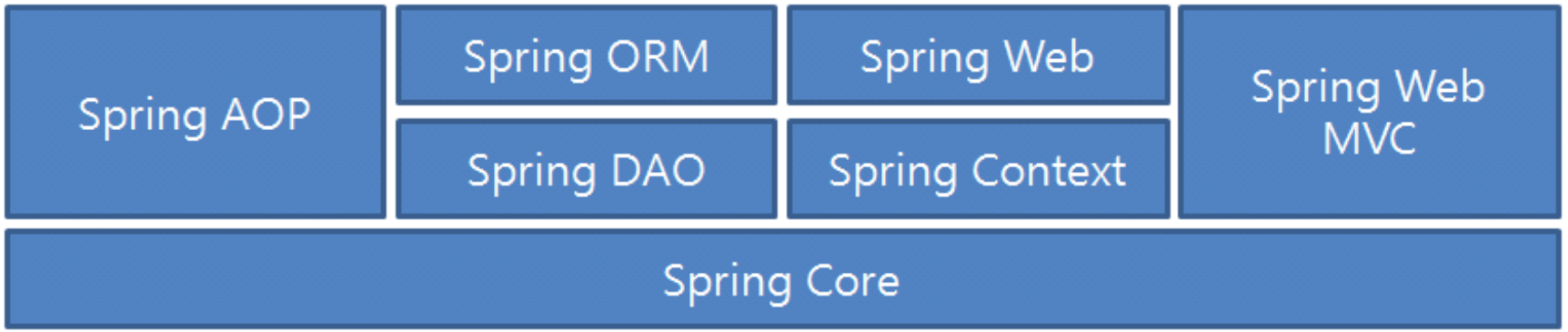
각 모듈별 jar파일 포함

API 문서

모듈별 소스코드 및 빌드관련 파일

모듈별 소스jar

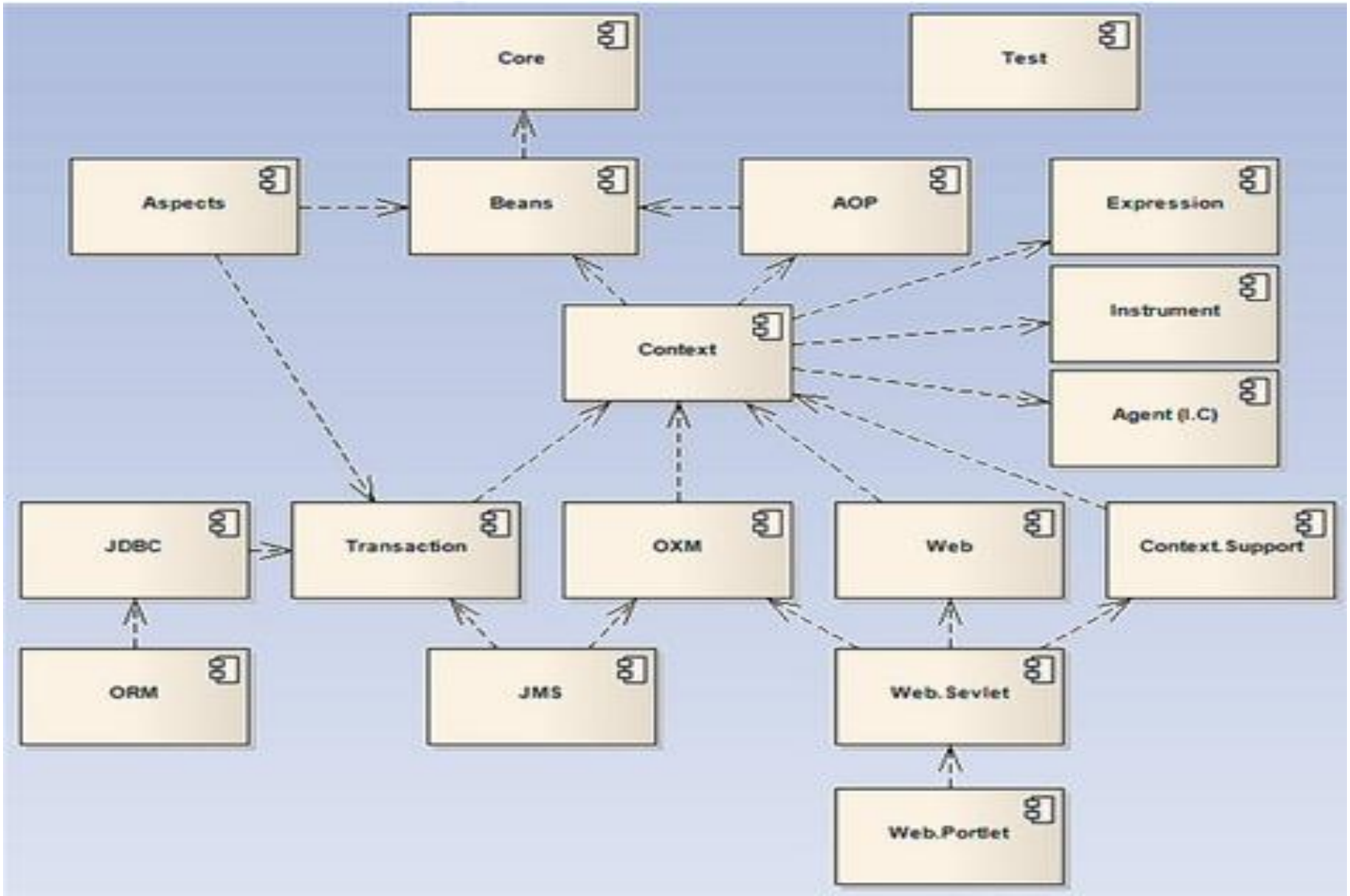
스프링을 구성하는 모듈


Spring AOP
Spring ORM
Spring Web
Spring Web MVC
Spring DAO
Spring Context
Spring Core

스프링을 구성하는 모듈


org.springframework.aop-3.1.0.RELEASE
org.springframework.asm-3.1.0.RELEASE
org.springframework.aspects-3.1.0.RELEASE
org.springframework.beans-3.1.0.RELEASE
org.springframework.context-3.1.0.RELEASE
org.springframework.context.support-3.1.0.RELEASE
org.springframework.core-3.1.0.RELEASE
org.springframework.expression-3.1.0.RELEASE
org.springframework.instrument-3.1.0.RELEASE
org.springframework.instrument.tomcat-3.1.0.RELE...
org.springframework.jdbc-3.1.0.RELEASE
org.springframework.jms-3.1.0.RELEASE
org.springframework.orm-3.1.0.RELEASE
org.springframework.oxm-3.1.0.RELEASE
org.springframework.test-3.1.0.RELEASE
org.springframework.transaction-3.1.0.RELEASE
org.springframework.web-3.1.0.RELEASE
org.springframework.web.portlet-3.1.0.RELEASE
org.springframework.web.servlet-3.1.0.RELEASE
org.springframework.web.struts-3.1.0.RELEASE

스프링 모듈의 의존 관계


Core
Test
Aspects
Beans
AOP
Expression
Instrument
Context
Agent (I.C)
JDBC
Transaction
OXM
Web
Context.Support
ORM
JMS
Web.Sevlet
Web.Portlet

필수 라이브러리

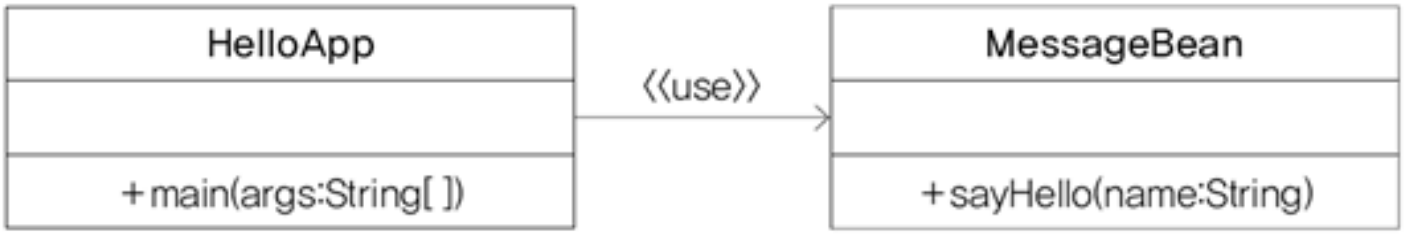
'spring-framework-3.1.0.RELEASE₩projects₩spring-build₩lib₩ivy

commons-logging
ALZip JAR File
60KB

DI(Dependency Injection)

- 스프링 컨테이너가 지원하는 핵심 개념 중의 하나
- 객체간의 의존 관계를 객체 자신이 아닌 외부의 조립기가 수행 해 준다는 개념.

첫 번째 예제는 단순한 샘플 어플리케이션으로 sample1 패키지에 MessageBean과 HelloApp라는 2개의 클래스가 있다.

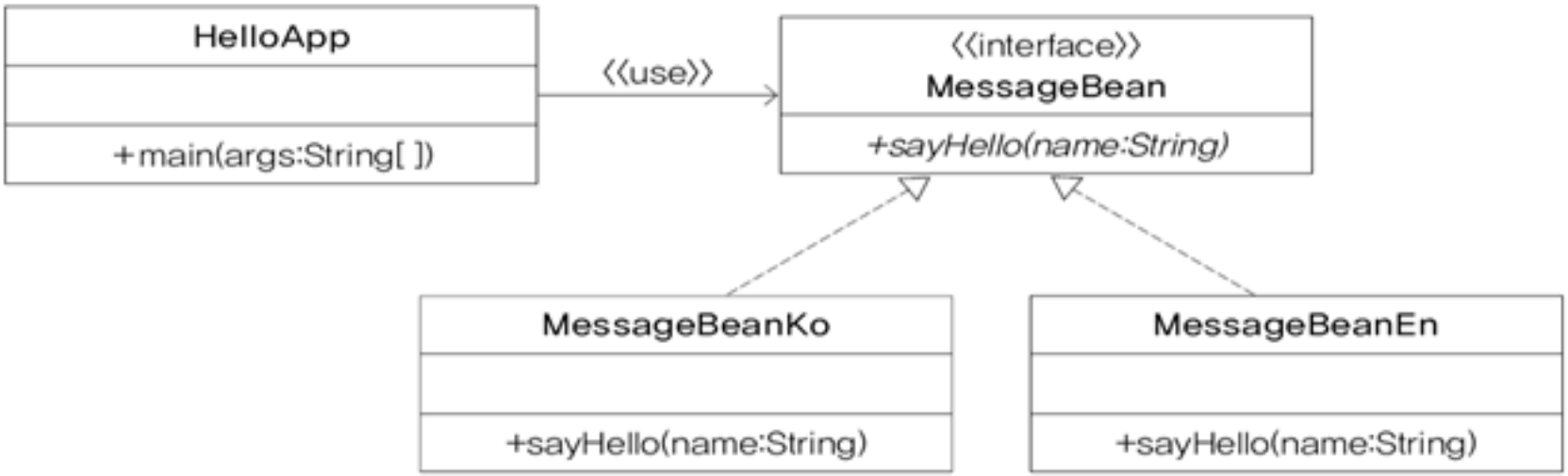


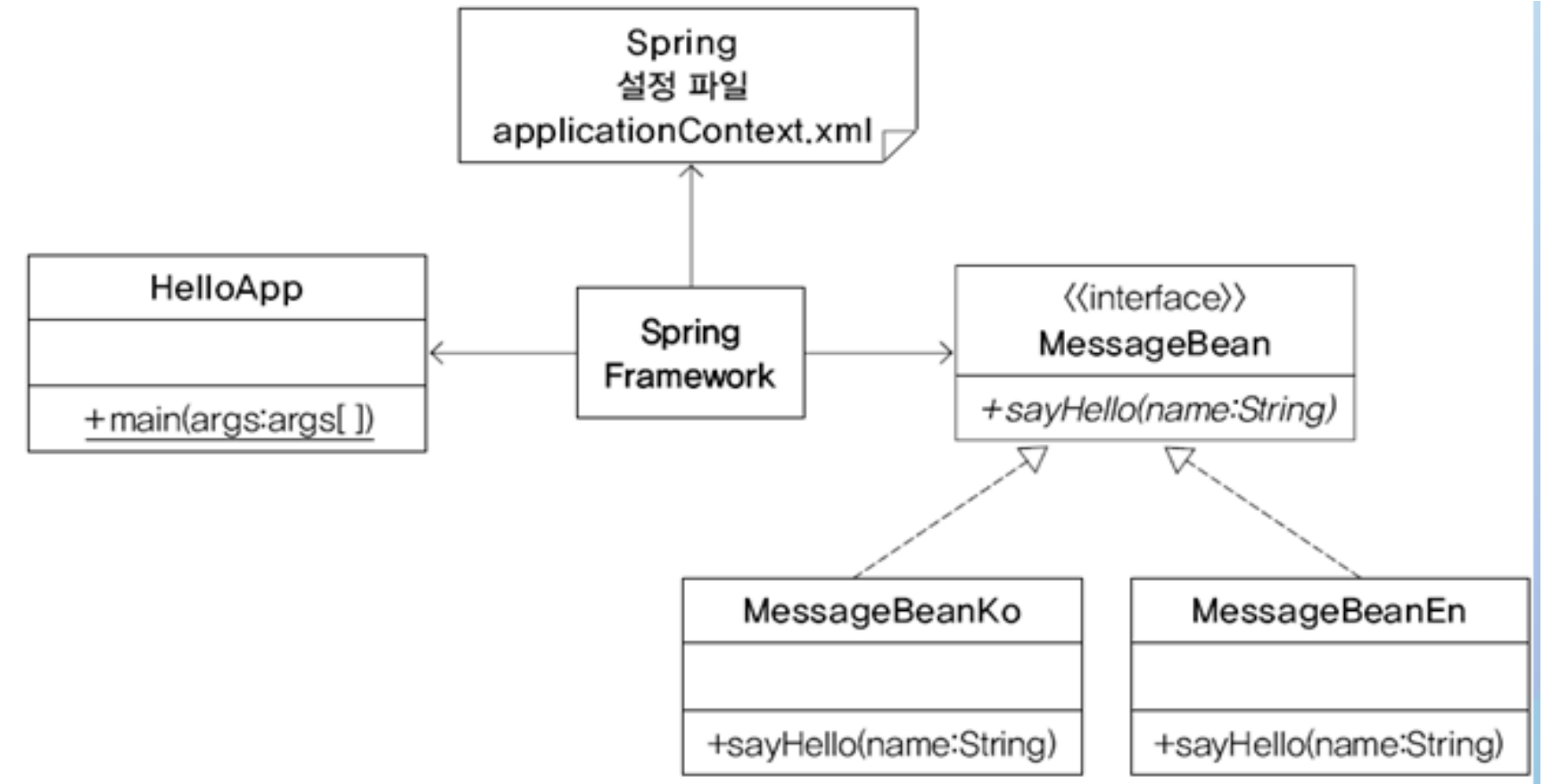
▲ sample1 클래스 그림

MessageBean 클래스는 멤버로 sayHello() 메소드를 한 개 가지며, 이 메소드는 '이름'을 String형 인수로 받아서 화면에 'Hello, ○○!' 라고 출력한다.

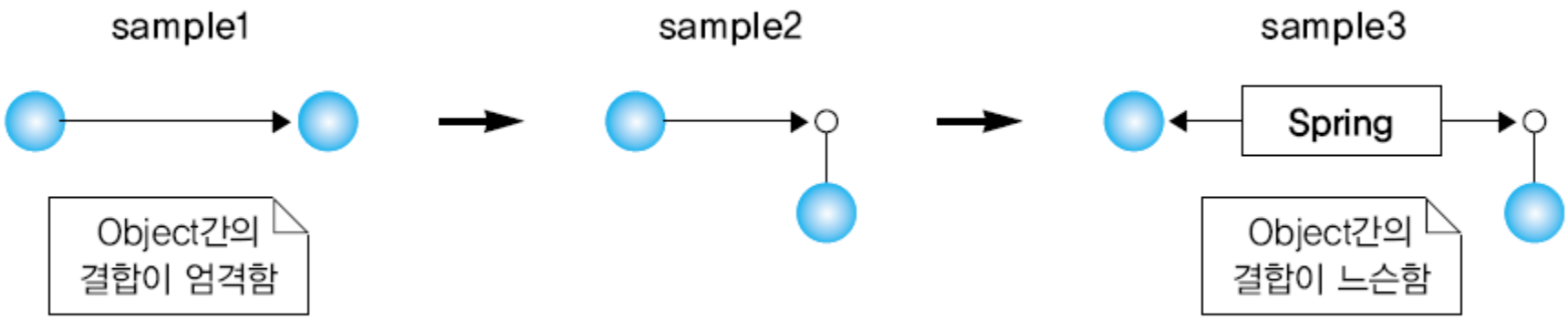
HelloApp 클래스는 main() 메소드 안에서 MessageBean의 인스턴스를 생성하여 sayHello() 메소드를 호출할 뿐이다.

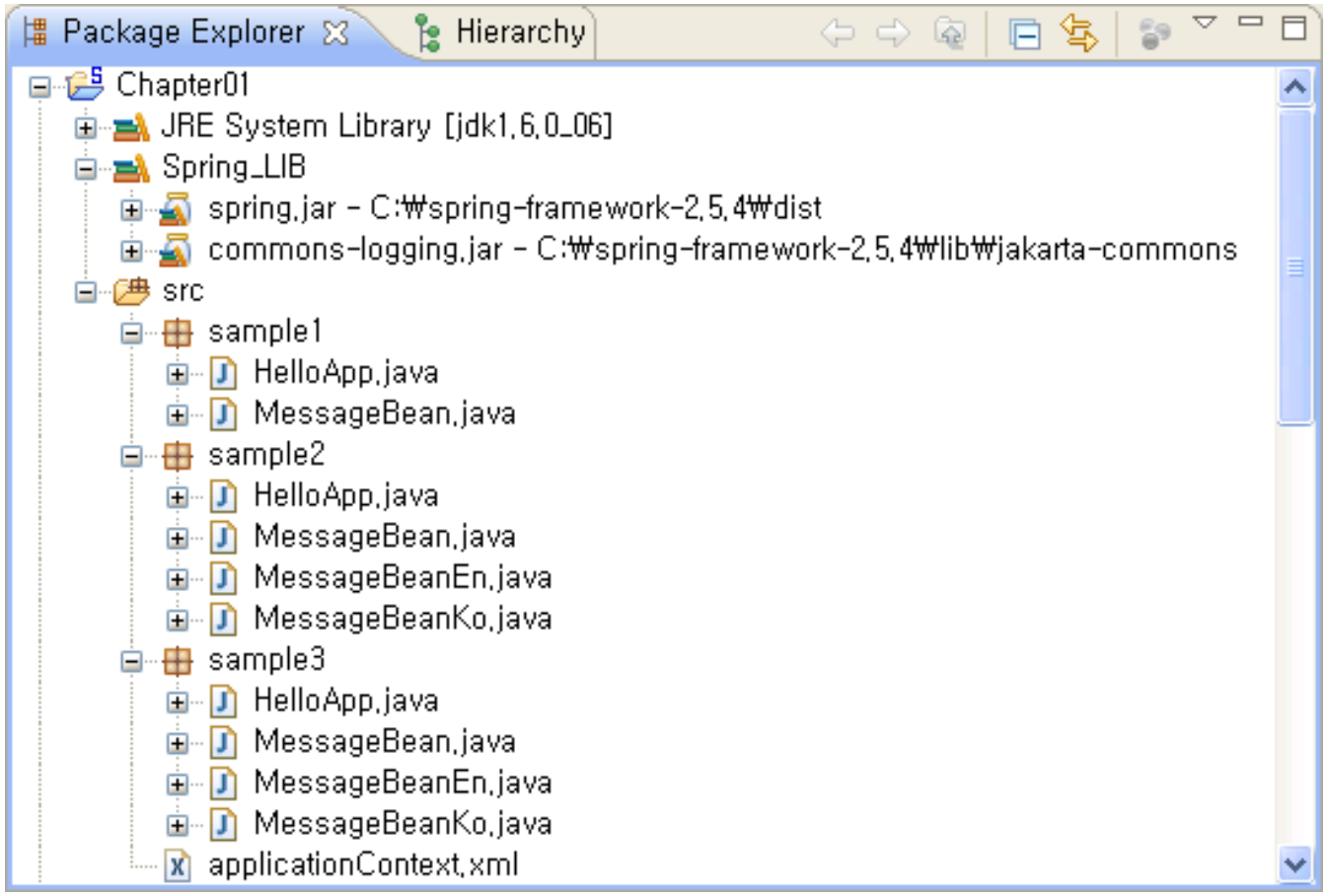
두 번째 샘플 어플리케이션은 인터페이스를 이용하여 클래스 사이의 의존 관계를 약하게 설정하고 있다. sample2 패키지에 MessageBean 인터페이스를 마련하고, 이 인터페이스를 구현한 두 개의 클래스(MessageBeanEn과 MessageBeanKo)를 작성하였다.





▲ sample3 클래스 그림(스프링 사용)


sample1
sample2
sample3
Spring
Object간의
결합이 엄격함
Object간의
결합이 느슨함


Package Explorer
Hierarchy
Chapter01
JRE System Library [jdk1.6.0_06]
Spring_LIB
spring.jar - C:₩spring-framework-2.5.4₩dist
commons-logging.jar - C:₩spring-framework-2.5.4₩lib₩jakarta-commons
src
sample1
HelloApp.java
MessageBean.java
sample2
HelloApp.java
MessageBean.java
MessageBeanEn.java
MessageBeanKo.java
sample3
HelloApp.java
MessageBean.java
MessageBeanEn.java
MessageBeanKo.java
applicationContext.xml

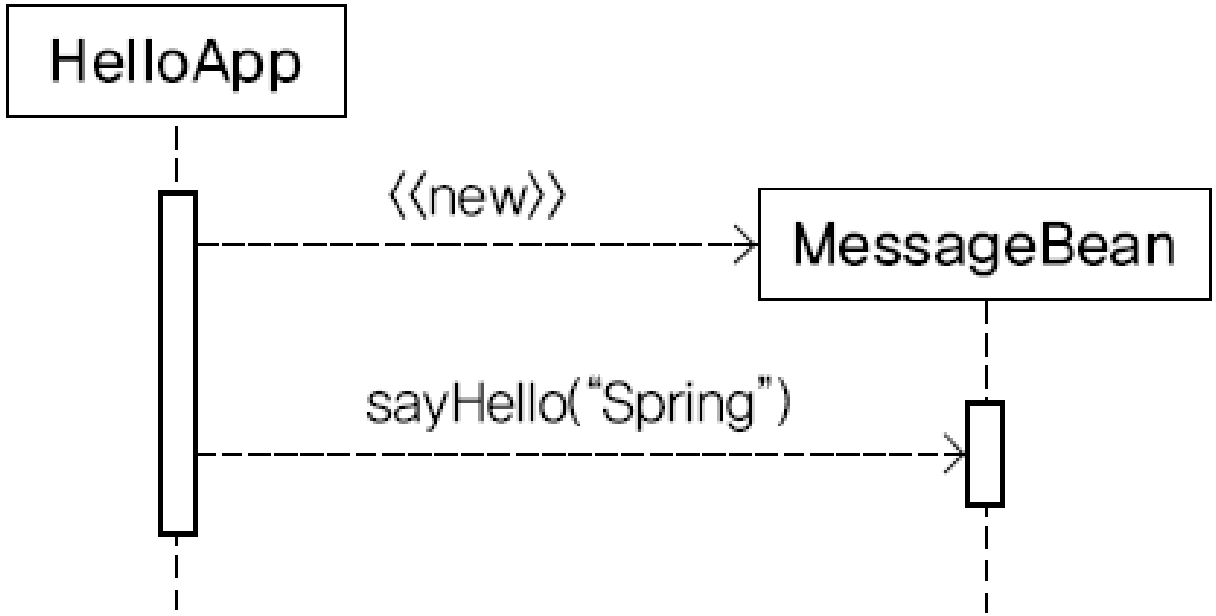
클래스

◈ MessageBean.java

```
package sample1;
public class MessageBean {
 public void sayHello(String name) {
    System.out.println("Hello," +name+ "!");
 }
}
```

◈ HelloApp.java

```
package sample1;
public class HelloApp {
 public static void main(String[ ] args) {
    MessageBean bean = new MessageBean( );
    bean.sayHello("Spring");
 }
}
```



▲ Sample 샘플 어플리케이션 시퀀스 그림

◈ MessageBean.java

```
package sample2;
public interface MessageBean {
 void sayHello(String name);
}
```

◈ MessageBeanEn.java

```
package sample2;
public class MessageBeanEn implements MessageBean {
 @Override
 public void sayHello(String name) {
    System.out.println("Hello," +name+ "!");
 }
}
```

```
MessageBean bean = new MessageBeanEn( );
Bean.sayHello("Spring");
```

수정하지 않아도 되는 코드

```
MessageBean bean = new MessageBeanKo( );
Bean.sayHello("Spring");
```

아직은 수정해야 하는 코드가 존재

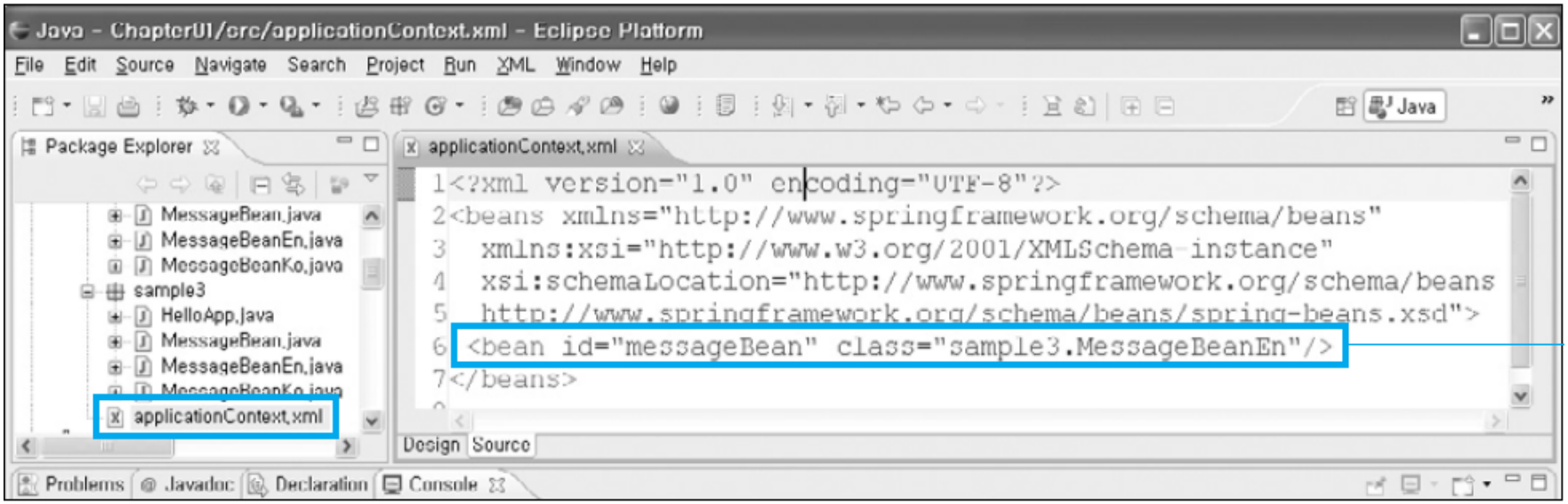
수정하지 않아도 되는 코드

HelloApp.java

```
package sample3;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class HelloApp {
  public static void main(String[ ] args) {
    Resource resource = new FileSystemResource("applicationContext.xml");····· ❶
    BeanFactory factory = new XmlBeanFactory(resource);
    MessageBean bean = (MessageBean)factory.getBean("messageBean");  ····· ❷
    bean.sayHello("Spring");
  }
}
```

〈bean id = “messageBean” class = “sample3.MessageBeanEn” /〉



▲ applicationContext.xml 파일에 〈bean〉 요소 추가

AOP

- 스프링은 DI 컨테이너로써 뿐만 아니라 AOP 프레임워크로서의 기능도 제공하고 있다. AOP(Aspect Oriented Programming : Aspect 지향 프로그래밍)는 최근 각광받는 새로운 프로그래밍 기법에 대한 개념이다

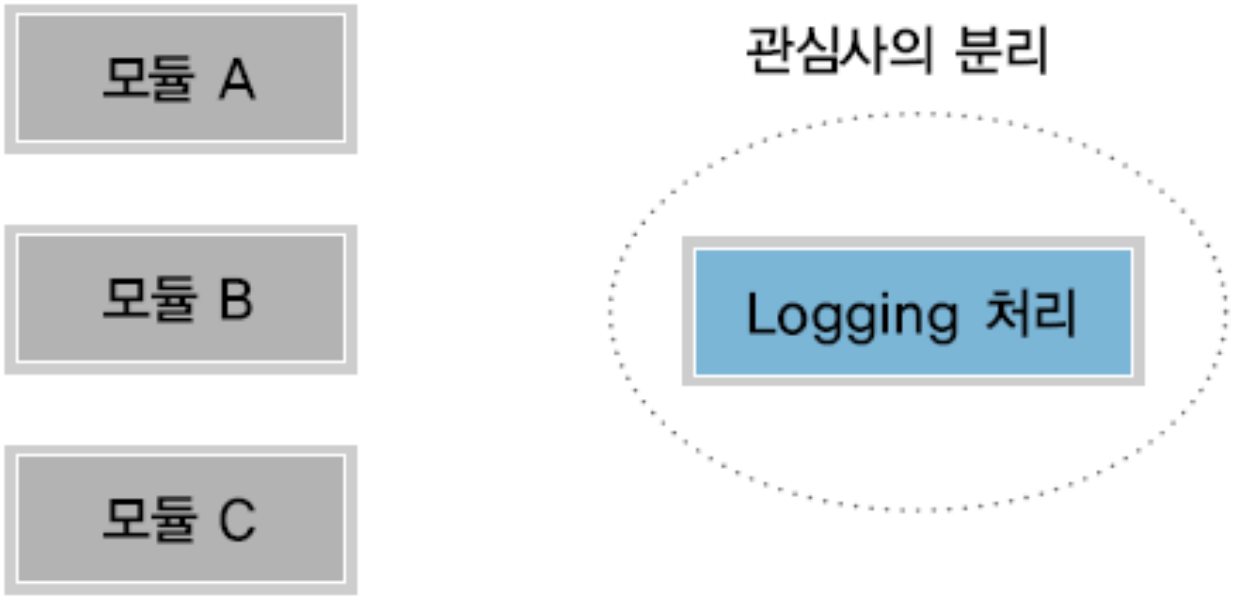
- 관점 지향 프로그래밍(Aspect Oriented Programming, 이하 AOP)은 결국 객체 지향 프로그래밍(Object Oriented Programming의 뒤를 잇는 또 하나의 프로그래밍 언어 구조라고 생각될 수 있다.

- OOP를 더욱 OOP답게 만들어 준다.

- Aspect 지향에서 중요한 개념은「횡단 관점의 분리(Separation of Cross-Cutting Concern)」이다. 이에 대한 이해를 쉽게 하기 위해서 은행 업무를 처리하는 시스템을 예를 들어 보겠다.

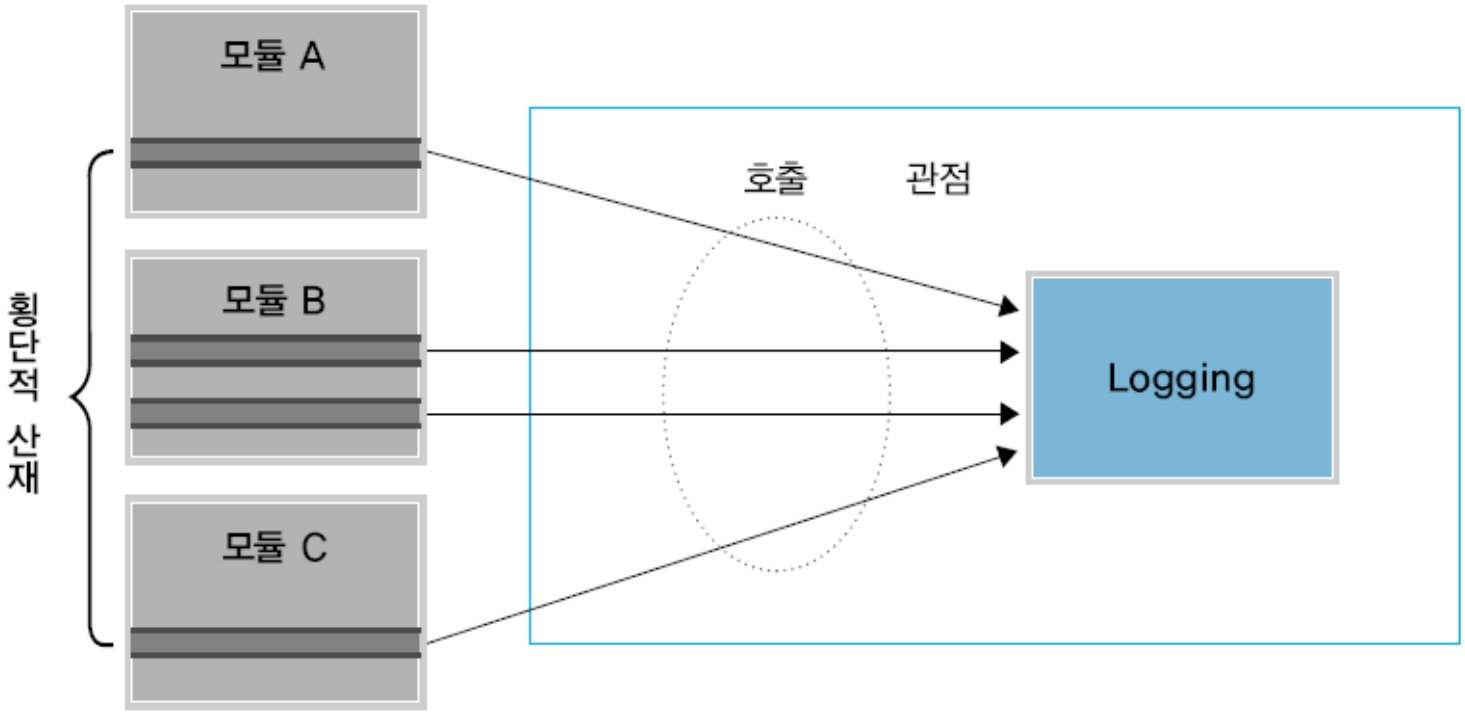
- 은행 업무 중에서 계좌이체, 이자계산, 대출처리 등은 주된 업무(핵심 관점, 핵심 비즈니스 기능)로 볼 수 있다. 이러한 업무(핵심 관점)들을 처리하는데 있어서「로깅」,「보안」,「트랜잭션」등의 처리는 어플리케이션 전반에 걸쳐 필요한 기능으로 핵심 비즈니스 기능과는 구분하기 위해서 공통 관심사항(Cross-Cutting Concern)이라고 표현한다.

- 오브젝트 지향에서는 이들 업무들을 하나의 클래스라는 단위로 모으고 그것들을 모듈로부터 분리함으로써 재이용성과 보수성을 높이고 있다.



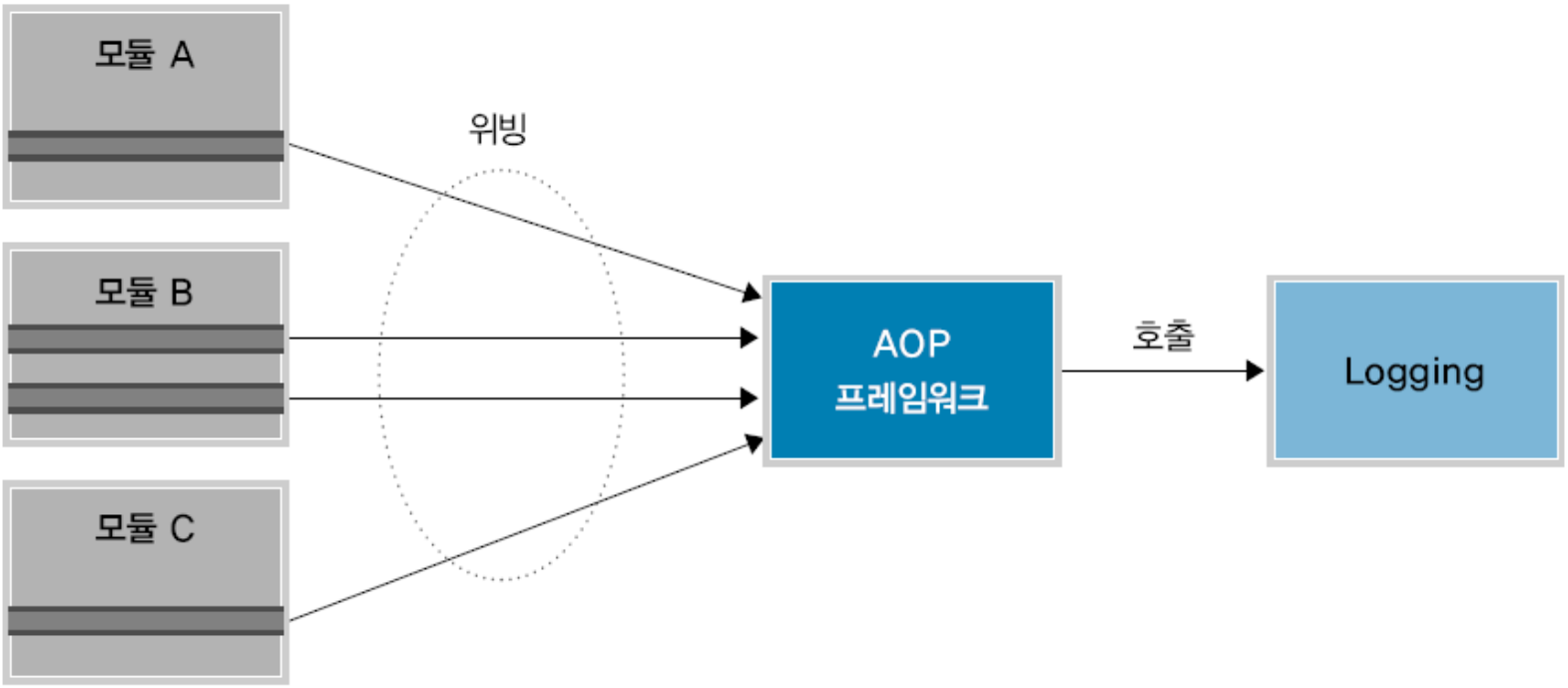
▲ 기존의 객체 지향에서 관점의 분리

- 오브젝트 지향에서는 로깅이라는 기능 및 관련하는 데이터 자체는 각 모듈로부터 분리하는 것으로 성공했지만 그 기능을 사용하기 위해서 코드까지는 각 모듈로부터 분리할 수 없다. 그렇기 때문에 분리한 기능을 이용하기 위해서 코드가 각 모듈에 횡단으로 산재하게 된다.



▲ 횡단적으로 산재하는 '기능의 호출'

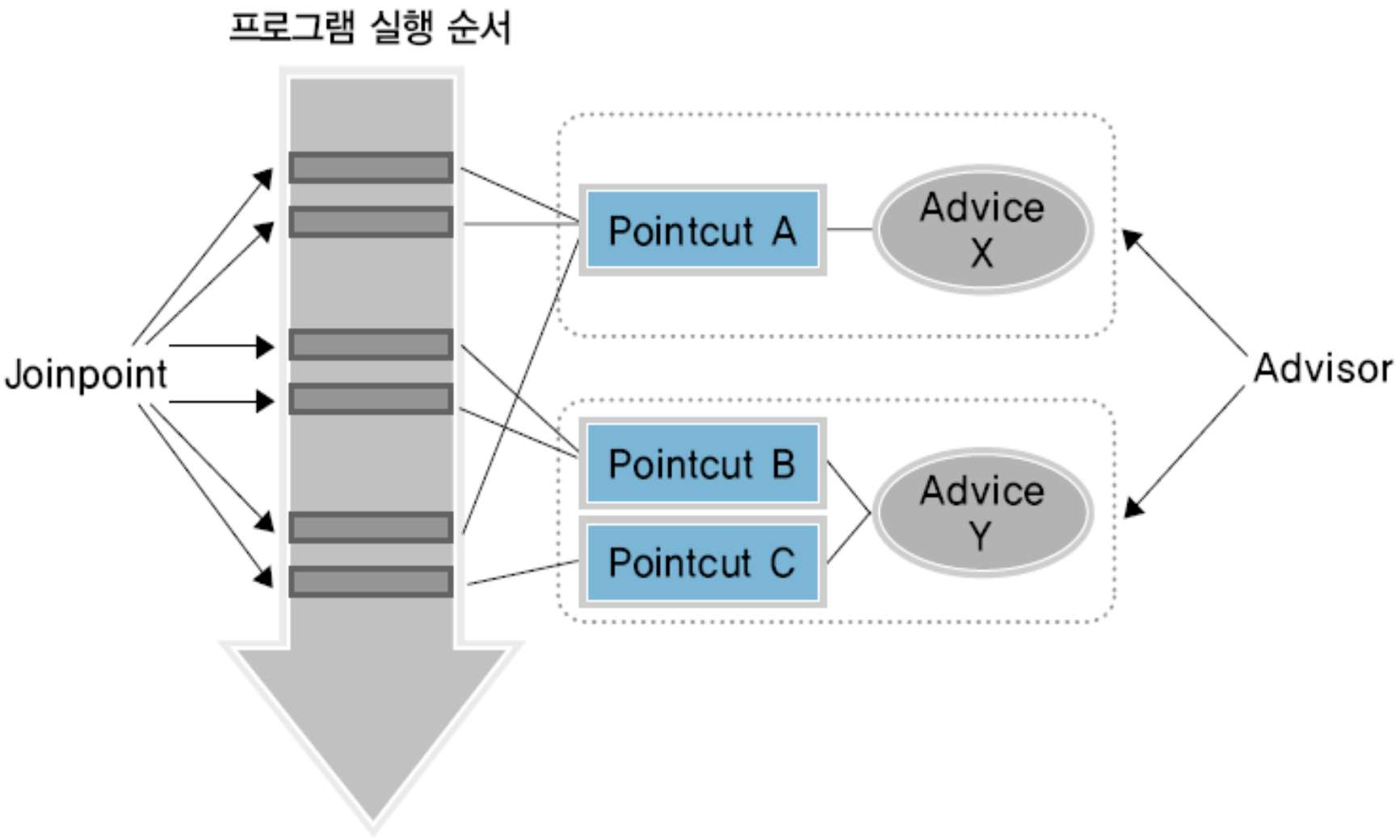
- AOP에서는 분리한 기능의 호출도 포함하여 「관점」으로 다룬다. 그리고 이러한 각 모듈로 산재한 관점을 「횡단 관점」라 부르고 있다.
- AOP에서는 이러한「횡단 관점」까지 분리함으로써 각 모듈로부터 관점에 관한 코드를 완전히 제거하는 것을 목표로 한다.



▲ AOP의 횡단 관점의 분리와 위빙

스프링 AOP 용어

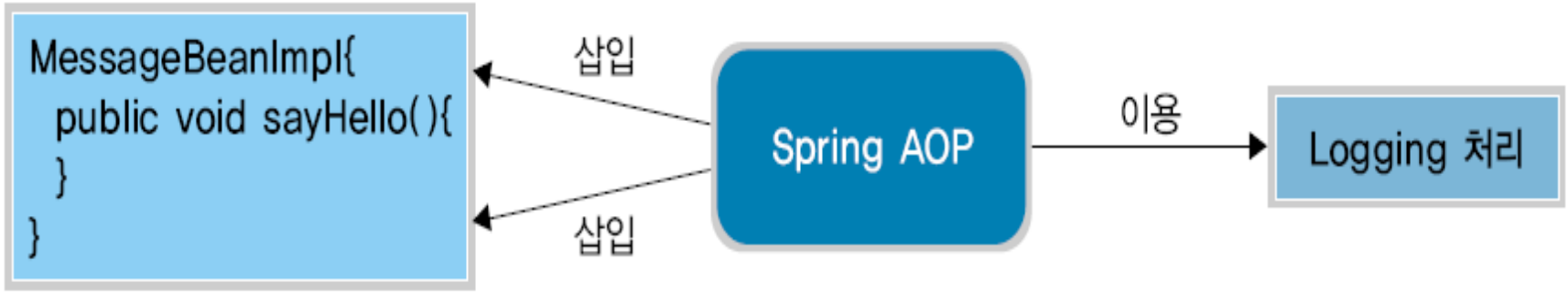
- Joinpoint -「클래스의 인스턴스 생성 시점」,「메소드 호출 시점」및「예외 발생 시점」과 같이 애플리케이션을 실행할 때 특정 작업이 시작되는 시점을 Joinpoint라 한다.
- Advice - Joinpoint에 삽입되어져 동작할 수 있는 코드를 Advice라 한다.
- Pointcut - 여러 개의 Joinpoint를 하나로 결합한(묶은) 것을 Pointcut이라고 부른다.
- Advisor - Advice와 Pointcut를 하나로 묶어 취급한 것을 Advisor라 부른다.
- Weaving - Advice를 핵심 로직 코드에 삽입하는 것을 Weaving이라 부른다.
- Target - 핵심 로직을 구현하는 클래스를 말한다.
- Aspect - 여러 객체에 공통으로 적용되는 공통 관점 사항을 Aspect라 부른다.



▲ 스프링 AOP에서의 용어와 개념

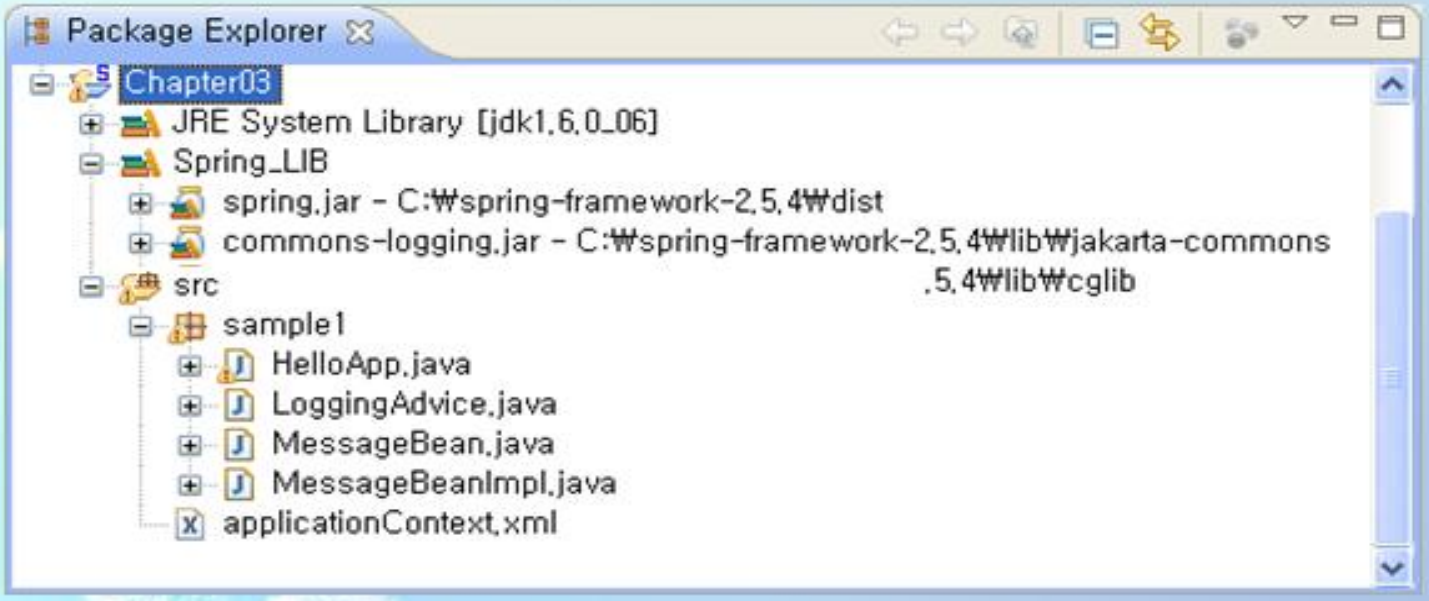
AOP를 이용한 logging 구현 예제

- 이 예제에서는 AOP 구조를 활용하여 메소드 트레이스 정보의 Logging 처리를 MessageBeanImpl의 sayHello() 메소드 호출 전후에 삽입한다.
- 로깅 처리 자체 및 그 호출은 MessageBeanImpl에는 기술하지 않는다.
- 스프링이 제공하는 기능인「스프링 AOP」가 그 역할을 담당한다.



▲ 예제 개요 그림

▼ 파일 구성



sayHello() 메소드가 핵심 로직이고 이 메소드를 멤버로 갖는 MessageBeanImpl는 타켓 클래스가 된다.
LoggingAdvice 클래스가 로깅처리를 담당하고 있게 된다.

스프링에서 AOP를 구현하는 과정

1. Advice 클래스를 작성한다.
2. 설정 파일에 Pointcut을 설정한다.
3. 설정 파일에 Advice와 Pointcut을 묶어 놓는 Adviseor를 설정한다.
4. 설정 파일에 ProxyFactoryBean 클래스를 이용하여 대상 객체에 Adviseor를 적용한다.
5. getBean( ) 메소드로 빈 객체를 가져와 사용한다.