2. 의존성 주입

1. DI와 IOC 컨테이너
2. 빈생성
3. 의존성 주입방법

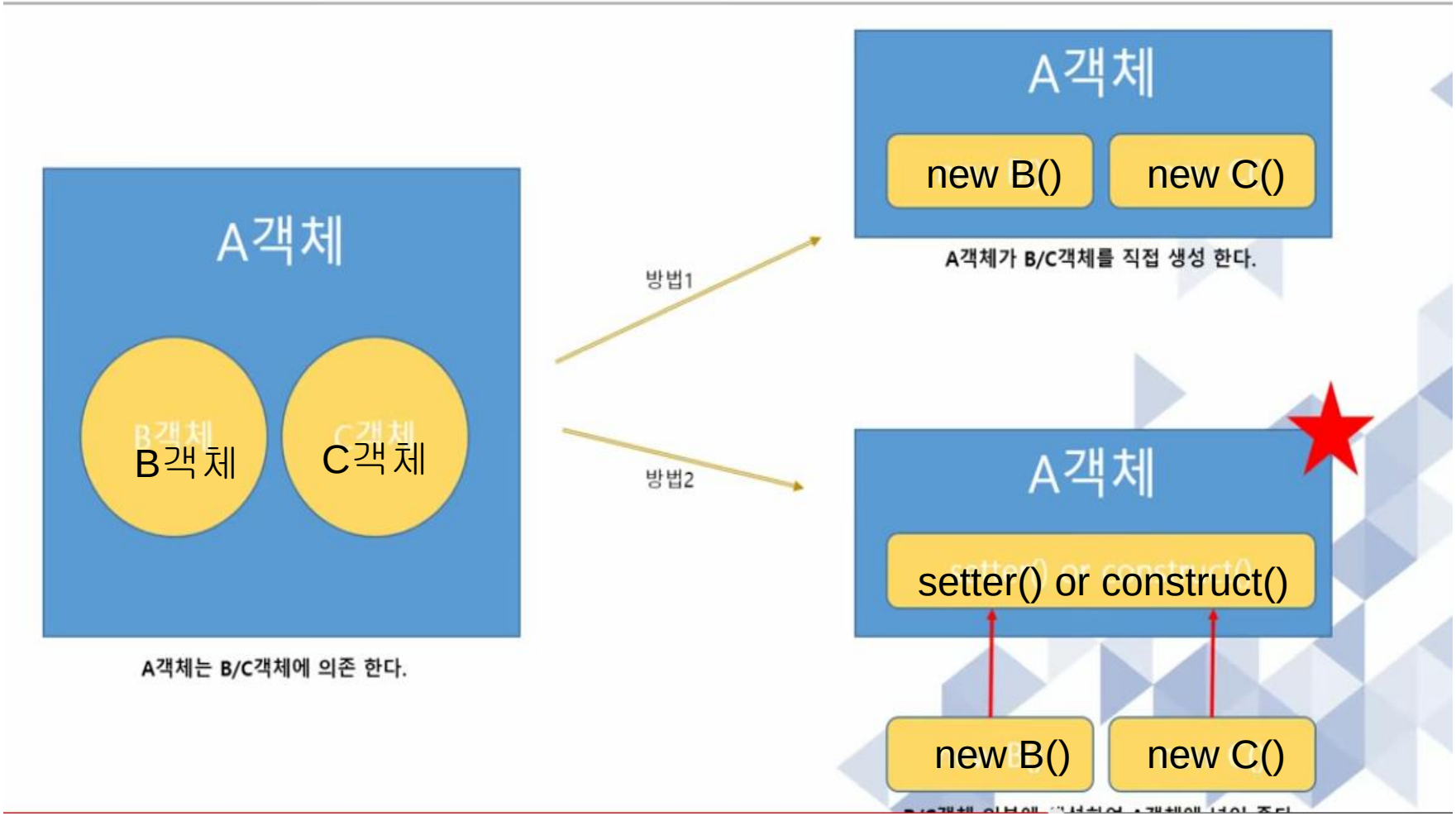
1. DI(Dependence Injection)와 IOC 컨테이너

- 객체간의 의존성
  - 객체의 사용에 의해 발생
  - Interface에 의한 의존성 제거
  - factory에 의한 의존성 제거
  - Spring Framework 사용

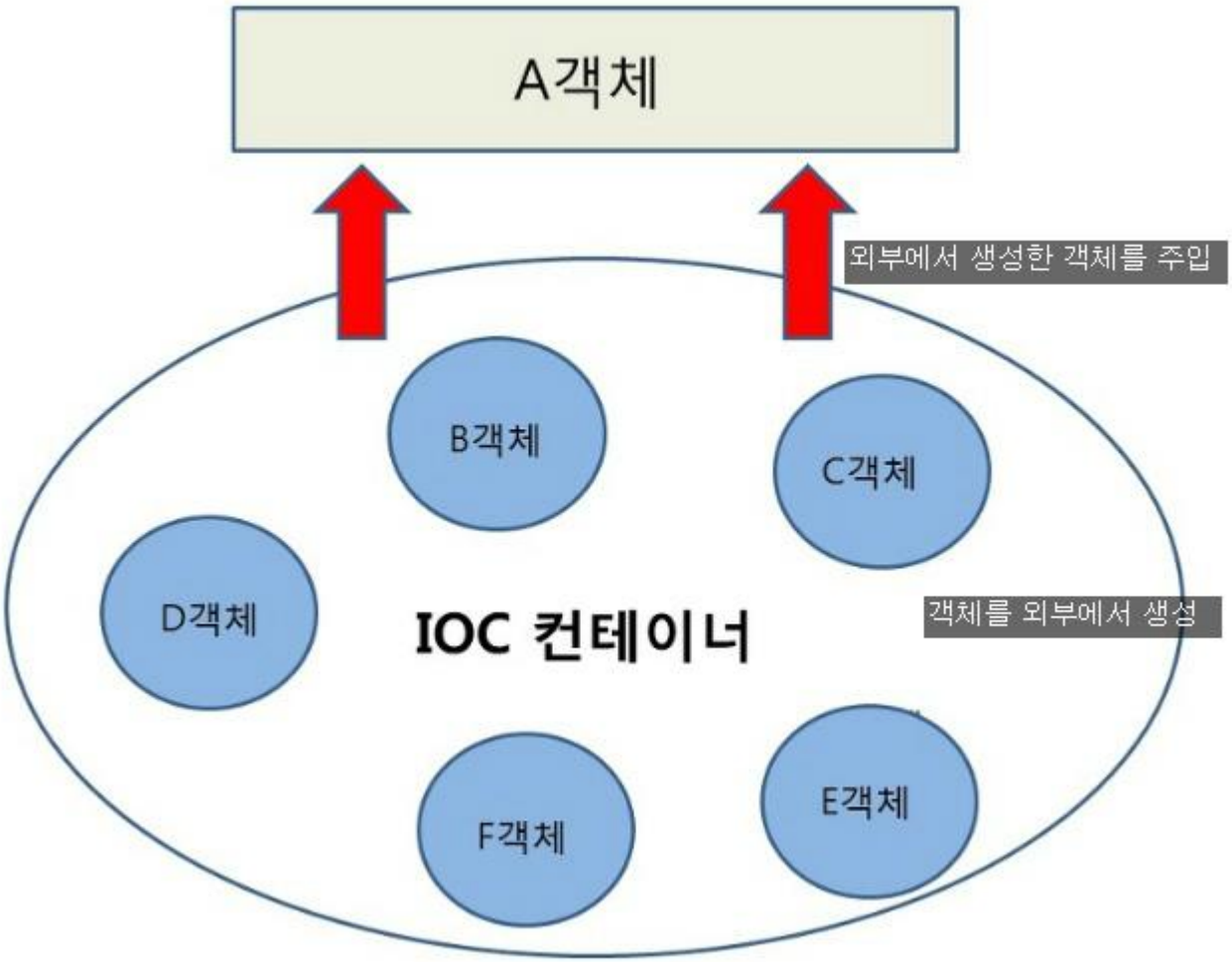
1. DI(Dependence Injection)와 IOC 컨테이너

- DI(Dependence Injection)
  - 의존성을 주입. 즉, Spring 프레임워크에서 지원하는 IoC의 형태
  - 객체를 직접 생성하는 게 아니라 외부에서 생성한 후 주입을 시켜주는 방식
- IOC(Inversion of Control)의 개념
  - 초기에 객체 생성, 객체간의 의존관계 연결, 연결 생명주기 관리 등의 제어권을 개발자가 가짐
  - 서블릿, EJB가 등장, 개발자들의 독점하던 제어권이 서블릿과 EJB를 관리하는 컨테이너에게 넘어감.
  - 객체의 제어권이 바뀐 것을 제어권의 역전, 즉 Ioc라고 한다.

1. DI(Dependence Injection)와 IOC 컨테이너



1. DI와 IOC 컨테이너



1. DI와 IOC 컨테이너

스프링을 사용하지 않는 프로젝트

```
public static void main(String[] args){
    Student s1=new Student();
    s1.setName("홍길동");
    s1.setSno(1);
    s1.setMajor("컴공");
    System.out.println(s1);
}
```

스프링을 사용하는 프로젝트

```
public static void main(String[] args){
    ApplicationContext ctx=new ClassPathXmlApplicationContext("bean1.xml");
    Student s1=ctx.getBean("stu1", Student.class);
    System.out.println(s1)
}
```

실습

- 새(MVC)프로젝트 작성 : 프로젝트명: diExam
- 새 패키지 작성(DI와 관련된 내용 패키지로 분류)

  패키지명 : com.pgm.di01

  MessageBean 클래스 di01에 작성

```
public class MessageBean{
    public void sayHello(String name){
        System.out.printl("How are You" + name + "님");
    }
}
```

- App 클래스 수정

```
public class App{
    public static void main(String[] args){
        MessageBean bean=new MessageBean();
        bean.sayHello("홍길동");
    }
}
```

Interface에 의한 의존성 해결

- com.pgm.di02 패키지 작성
- App클래스와 MessageBean클래스 복사
- IMessageBean 작성

```
public interface IMessageBean{
    public void sayHello(String name);
}
```

- MessageBean 수정

```
public class MessageBean implements IMessageBean{
    public void sayHello(String name){
        System.out.printl("How are You" + name + "님");
    }
}
```

Interface에 의한 의존성 해결

- App 수정

```
public class App{
    public static void main(String[] args){
        IMessageBean bean=new MessageBean();
        bean.sayHello("홍길동");
    }
}
```

- MessageBeanKr 클래스 추가

```
public class MessageBeanKr implements IMessageBean{
    public void sayHello(String name){
        System.out.printl("안녕하세요 " + name + "님");
    }
}
```

Interface에 의한 의존성 해결

- App에서 MessageBeanKr 사용시 다음과 같이 수정

```
public class App{
    public static void main(String[] args){
        IMessageBean bean=new MessageBeanKr();
        bean.sayHello("홍길동");
    }
}
```

- Interface의 의존성 제거의 한계

  interface를 구현한 클래스 객체만 사용가능

Factory Pattern에 의한 의존성 제거

- Factory Pattern 와 Singleton Pattern
- com.pgm.di03 패키지 작성
- factory : 인스턴스를 대신해 줌
- main은 factory의 주소만 알고 있으면 됨

```
public class MessageFactory{
    private MessageFactory(){}
    private staric MessageFactory factory=new MessageFactory();
    public static MessageFactory getInstance(){
        return factory;
    }
    public IMessage createMessage(String str){
        if(str.equal("kr"){
            return new MessageBeanKr();
        }else{
            return new MessageBean();
        }
    }
}
```

Factory Pattern에 의한 의존성 제거

- App수정

```
public class App{
    public static void main(String[] args){
        MessageFactory factory=MessageFactory.getInstance();
        IMessageBean bean=factory,createMessage(“kr”)
        bean.sayHello(“홍길동”);
        bean=factory.createMessage(“en”);
        bean.sayHello(“Hong Gil Dong”);
    }
}
```

환경설정.xml 파일 작성

- 자동으로 factory 가지고 있음
- factory클래스를 가져올 설정파일 작성
- 프로젝트의 src/main/resource에 설정파일 작성
- new->Spring Bean Configuration File 선택 or
- other->spring-> Spring Bean Configuration File 선택
- next ->위치 프로젝트 루트->file명 : bean_config.xml(임의로 작성)
- namespace 사용 상자에서 beans 선택
- 설정파일에 다음을 추가한다.

```
<bean id="msgBean" class="com.pgm.exam.Messagebean" />
```

환경설정.xml 파일 작성

- ApplicationContext 사용
  - XML설정 컨테이너

1) 생성
  - ApplicationContext ctx = new ClassPathXmlApplicationContext(“bean_config.xml”);

2) 설정
  - 생략가능, Context 생성시 config 파일 지정하지 않았을 때만 수행
  - ctx.load(...);
  - ctx.refresh();

3) 사용
  - SomeBean bean = ctx.getBean("someBean", SomeBean.class);

4) 종료
  - ctx.close()

환경 설정 파일 살펴보기

```
<bean id="msgBean"  name="msg"
     class="com.pgm.test01.di01.MessageBean>
```

- id 속성 변수 역할
- name : 별명(alias), 여러 개 가능
- class : 사용한 클래스(패기지명.클래스)

```
<bean id="msgBean"  name="msg, msg1, msg2, msg3"
     class="com.pgm.exam.MessageBean>
```

환경설정.xml 파일 작성

- 여러 개의 name 속성 사용 예

```
public class App{
    public static void main(String[] args){
     ApplicationContext ctx =new
     ClassPathXmlApplicationContext(“bean_config.xml”);
        IMessageBean bean=ctx.getBean(“msg”, IMessageBean.class);
        IMessageBean bean1=ctx.getBean(“msg1”, IMessageBean.class);
        IMessageBean bean2=ctx.getBean(“msg2”, IMessageBean.class);
        IMessageBean bean3=ctx.getBean(“msg3”, IMessageBean.class);

        bean.sayHello(“홍길동”);
        bean1.sayHello(“kim”);
        bean2.sayHello(“lee”);
        bean3.sayHello(“park”);

    }
}
```

클래스 이용 환경설정

- 환경설정을 위한 클래스 작성
  - Anotation: 기능을 가진 주석

```
@Configuration
public class ApplicationContextConfiguration{
        @Bean
        public IMessageBean  messageBean(){
                return new MessageBean);
        }
}
```

- App 파일

```
public class App{
    public static void main(String[] args){
        ApplicationContext  ctx=new
          AnnotationConfigApplicationContext(ApplicationContextConfiguration.class)
        IMessageBean bean=ctx.getBean("messageBean", IMessageBean.class);
        bean.sayHello("홍길동");
    }
}
```

클래스 이용 환경설정

- @Bean의 name 속성
  - 사용 예

    @Bean(name="msg")

    @Beaa(name={"msg","msg1","msg2"}
  - App 클래스에서 사용 예

    IMessageBean bean=ctx.getBean("msg", IMessageBean.class);
- @Bean의 init 및 destroy 속성
  - init : 가장 먼저 부르는 함수
  - destory : 마지막에 부르는 함수
  - @Bean(init="initFnc")
  - MessageBean 클래스에 InitFnc() 메소드 작성

Bean의 의존성 주입

- IoC 컨테이너
  - 컨테이너에서 Spring 빈 인스턴스를 생성하고,생성된 빈인스턴스의 의존성을 관리
  - ApplicationContext 인터페이스는 IoC 컨테이너를 표현하고 스프링 빈 인스턴스를 생성, 설정하고 조합하는 기능 수행
  - IoC 컨터이너는 ApplicationContext 라고도 함

- 의존성 주입 방법
  - 생성자를 이용한 방식
  - 필드를 사용하는 방식

생성자의 의한 의존성 주입

- 새 패키지 작성(com.pgm.di03)
- App 클래스, IMessageBean, MessageBean 작성
- IMessageBean, MessageBean 클래스 변경

```
public interface IMessageBean{
        public void sayHello();
}
```

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;
    public void sayHello(){
        System.out.printl(greeting +"!! " +  name + "님 나이가"
                + age +"입니다" );
    }
}
```

생성자의 의한 의존성 주입

- 설정파일 작성
  - src/main/resource에 bean_config.xml 작성

```
<bean id="msg"  class="com.pgm.di03.MessageBean>
```

- App 수정

```
public class App{
    public static void main(String[] args){
        ApplicationContext  ctx=new
            ClassPathXmlApplicationContext("bean_config.xml");
        IMessageBean bean=ctx.getBean("msg", IMessageBean.class);
        bean.sayHello();
    }
}
```

생성자의 의한 의존성 주입

- MessageBean 클래스에 생성자 추가

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;

    public MessageBean(String name, int age, String greeting){
        this.name=name;
        this.age=age;
        this.greeting=greeting;
    }
    public void sayHello(){
        System.out.printl(greeting +"!! " +  name + "님 나이가"
                + age +"입니다" );
    }
}
```

생성자의 의한 의존성 주입

- 설정파일에서 생성자 값 주입

```
<bean id="msg"  class="com.pgm.di03.Messagebean>
        <constructor arg>
                <value>홍길동</value>
        </constructor>
        <constructor arg>
                <value>25</value>
        </constructor>
        <constructor arg>
                <value>안녕하세요</value>
        </constructor>

</bean>
```

생성자의 의한 의존성 주입

- index, type 속성 사용하기

```
<bean id="msg"  class="com.pgm.di03.Messagebean>
        <constructor arg index="0">
                <value>홍길동</value>
        </constructor>
        <constructor arg type="int">
                <value>25</value>
        </constructor>
        <constructor arg>
                <value>안녕하세요</value>
        </constructor>

</bean>
```

Setter에 의한 의존성 주입

- setXXX()메소드에 값 전달

```
<bean id="msg"  class="com.pgm.di03.Messagebean>
        <property name="name" value="임꺽정"/>
        <property name="age" value="30"/>
        <property name="greeting" value="Hi"/>

        또는
        <property name="name">
                <value>임꺽정</value>
        </property>
        이하생략

</bean>
```

Setter에 의한 의존성 주입

- MessageBean에 setter 메소드 추카

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;

    public void setName(Strong name){
        this,name=name;
    }
    ….. 이하생략---
```

빈 참조하기

- xml 설정 파일에 같은 클래스의 객체 bean 2개 작성

```
<bean id="myInfo" class="com.example.pgm.MyInfo">
        <constructor-arg value="홍길동" />
        <constructor-arg value="180" />
        <constructor-arg value="70" />
        <property name="hobby">
                <list>
                        <value>수영</value>
                        <value>요리</value>
                        <value>독서</value>
                </list>
        </property>
        <property name="bmiCalculator">
            <ref bean="bmiCalculator"/>
        </property>
</bean>

<bean id="myInfo2" class="com.example.pgm.MyInfo">
        <constructor-arg value="홍길순" />
        … 이하 생략
```

실습

com.pgm.di03에 다음을 작성

```
public class Score{
      private String subject
      private double point;;
       public Score(String subject, double point){
              this.subject=subject;
              this.point=point;
       }
       getter, setter, toString 작성
}
```

```
public class Student{
      private String name;
      private int sno;
      private String major;
      private ArrayList<Score> scoreList;
      public Student(String name, int sno, String major){
            this.name =name;
            this.sno=sno;
            this.major=major;
      }
      getter, setter, toString 작성
```

src/main/resource에 bena_config3.xml 작성

```
<bean id="score1" class="com.pgm.exam.di01.Score">
	<constructor-arg value="java"/>
	<constructor-arg value="85"/>
</bean>
<bean id="score2" class="com.pgm.exam.di01.Score">
	<constructor-arg value="DB"/>
	<constructor-arg value="90"/>
</bean>

<bean id="stud1" class="com.pgm.exam.di01.Student">
	<constructor-arg value="홍길동"/>
	<constructor-arg value="1"/>
	<constructor-arg value="컴공"/>
	<property name="scoreList">
		<list>
			<ref bean="score1"/>
			<ref bean="score2"/>
		</list>
	</property>
</bean>
```

- App3

```
public class App3{

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ApplicationContext ctx=new
            ClassPathXmlApplicationContext("bean_config.xml");
        System.out.println(ctx.getBean("stud1",Student.class));
    }
}
```

Java를 이용한 DI 설정 실습

@Configuration
'이 클래스는 스프링 설정에 사용되는 클래스 입니다.' 라고 명시해 주는 어노테이션

```
@Configuration
public class ApplicationConfig {
```

```
@Bean
public Student student10{
    ArrayList<String> hobbys = new ArrayList<String>();
    hobbys.add("수영");
    hobbys.add("요리");

    Student student = new Student("홍길동", 20, hobbys);
    student.setHeight(180);
    student.setWeight(80);

    return student;
}
```

@Bean - 객체 생성

생성자에 설정

프로퍼티에 설정

의존성 자동 주입

- XML 설정 최소화
  - Spring 빈 클래스가 많아지면 설정이 많아지고 개발자 어플리케이션 개발이 복잡해짐
  - 이러한 문제를 극복하기 위해 애노테이션 사용

  - 자동 와이어링(auto wiring)

    XML 설정파일을 사용하면서 빈 설정을 최소한으로 사용함
  - 어노텡이션 와이어링(annotation wiring)

    XML 설정 파일을 사용하지 않거나 억제하여 Bean 설정

- auto wiring
  - xml 설정파일에 의존성을 주입할 빈을 지정하는 대신 autowire 속성에 autowire 방식을 지정
  - Spring 프레임 워크가 해당 방식에 맞는 빈을 자동으로 연결
  - Spring 프레임워크 4가지 자동 와이어링 방식

방식	autowire 속성	설명
이름	byName	필드와 같은 이름(또는 ID)를 가지는 빈과 자동 와이어링
타입	byType	필드와 같은 타입의 빈과 자동 와이어링
생성자	construnctor	생성자 매개변수의 타입과 일치하는 빈과 자동 와이어링
자동탐색	autodetect	먼저 생성자 자동 와이어링이 수행되고 실패하면 타입 와이어링이 수행됨

예제

```
public class TestA{
        public static void main(String[] args){
        }
}
```

```
public class TestB{
        public void display(){
                Sysem.out.println("TestB 입니다");
        }
}
```

```
public class TestC{
        public void display(){
                Sysem.out.println("TestC 입니다");
        }
}
```

- 최기 환경설정 파일 작성

```
<bean id="testb" class="com.pgm.exam.di02.TestB"/>
<bean id="testc" class="com.pgm.exam.di02.TestC"/>
<bean id="testa" class="com.pgm.exam.di02.TestA">
<property name="testB" ref="testb" />
<property name="testC" ref="testc" />
</bean>
```

- TestA 클래스 변경

```
public class TestA{
    private TestB b;
    private TestC c;
    public void setB(TestB b){
        this.b=b;
    }
    public void setC(TestC c){
        this.c=c;
    }
    public static void main(String[] args){
        ApplicationContext ctx=new
                ClassPathXmlApplicationContext("bean.xml");
        TestA bean=(TestA)ctx.getBean("testa", TestA.class);
        bean.b.display();
        bean.b.display();
    }
}
```

- bean 수정

bean_config.xml의 네임스페이스에서 context 네임스페이스 추가한 후 다음 코드 추가

```
<context:annotation-config/>

<bean id="testb" class="com.pgm.exam.di02.TestB"/>
<bean id="testc" class="com.pgm.exam.di02.TestC"/>
<bean id="testa" class=com.miya.section3.annotation.TestA"
        autowire="byName"> byType 도 가능
        <!– default : 사용하지 않음 -->
</bean>  또는
<bean id="testa" class=com.miya.section3.annotation.TestA" />
```

어노테이션	제공자
@Autowired	Spring
@Inject	JSR-330

- 어노테이션 와이어링

```
public class TestA{
    private TestB b;
    private TestC c;
    @Autowired
    public void setB(TestB b){
        this.b=b;
    }
    @Autowired
    public void setC(TestC c){
        this.c=c;
    }
    public static void main(String[] args){
        ApplicationContext ctx=new
                ClassPathXmlApplicationContext(“bean.xml”);
        TestA bean=(TestA)ctx.getBean(“testa”, TestA.class);
        bean.b.display();
        bean.b.display();
    }
}
```

- 와이어링할 빈이 없는 경우
  - @Autowired(required=false) : 선택적 와이어링, null 값 허용
  - @Qualifirer(‘beanid’) : 와이어링 할 빈지정

- 필드에 @Autowired 지정, setXX() 생략 가능

```
public class TestA{
    @Autowired
    private TestB b;
    @Autowired
    private TestC c;

    public static void main(String[] args){
        ApplicationContext ctx=new
                ClassPathXmlApplicationContext(“bean.xml”);
        TestA bean=(TestA)ctx.getBean(“testa);
        bean.b.display();
        bean.b.display();
    }
}
```

자동 빈 발견

- @Component(value="id")
- <context:component-scan base-package="패키지명"/>
- @Component(value="testa")
- <context:component-scan base-package="com.miya.section3.annotation"/>

- XML파일은 간단해짐, POJO 클래스는 복잡해짐