



제5장 스프링 프레임워크 고급 빈 와이링

- 알고 갑시다.
- 부모와 자식 빈(bean) 선언
- 메소드 주입 적용



알고 갑시다..



- ▶ 빈(bean)이란?
- ▶ DI(=IoC)란?
- ▶ Xml 설정 파일의 구성은?
- ▶ Bean 요소의 속성은?





빈(bean)이란?



- ▶ 애플리케이션에 존재하는 객체
- ▶ 객체=빈=bean



DI(=IoC) 이란?(1/3)

- ▶ IoC : Inversion of Control (제어의 역전)
- ▶ 스프링IoC컨테이너 or 스프링컨테이너
 - ▶ 스프링 애플리케이션에서 애플리케이션에 존재하는 객체(빈, bean)를 생성하고 의존관계를 주입하는 일을 담당
 - ▶ 애플리케이션의 설정 메타 데이터를 읽어서 애플리케이션 객체들과 그들의 의존관계를 인스턴스화함
 - ▶ 스프링 컨테이너에 설정 메타 데이터를 제공하는 방법: XML, 자바 애너테이션, 자바코드사용
- ▶ DI : Dependency Injection (의존 관계 주입)
 - ▶ 객체간의 관계를 생성자 인수나 setter 메소드 인수로 명시, 객체를 생성할 때 생성자나 setter를 통해 의존 관계를 주입하는 디자인 패턴
 - ▶ Constructor Injection, Method Injection(Setter Injection)

DI(=IoC) 이란? (2/3)

▶ Constructor Injection : 생성자 주입

```
public class Abc{  
    private Bbc bar;  
}
```

- ▶ Abc 클래스가 Bbc 클래스에 의존하고 있고,
- ▶ Bbc에 대한 인스턴스 참조는 의존 관계(Bbc에 대한 인스턴스 참조를 가지고 있음)라고 할 수 있음

▶ 생성자를 통한 주입 : 생성자를 사용해 의존 관계를 주입한다는 뜻

```
public class Abc{  
    private Bbc bar;  
    public Abc(Bbc bar){  
        this.bar = bar;  
    }  
}
```

// 생성자를 사용해 Bbc 인스턴스 참조를 지정

DI(=IoC) 이란?(3/3)

▶ Method Injection(Setter Injection) : 메소드 주입

- ▶ 설정 메소드를 사용해 의존 관계를 주입하는 것

```
public class Abc{  
    private Bbc bar;  
    public void setBar(Bbc bar){  
        this.bar = bar;  
    }  
}
```

// **Bbc** 타입 인수를 가지는
// 설정 메소드 **setBar()**를 통해
// **Bbc**에 대한 인스턴스 참조를 지정

Xml 설정 파일의 구성은?

```
<? xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
  <bean id="식별자" class="패키지명.클래스명" singleton="false" >
    <property name="속성명" value="기본형 데이터" ref="참조형 데이터" />
```

```
  </bean>
```

```
  <bean ... />
```

```
  <bean .....>
```

```
  </bean>
```

```
</beans>
```

Bean요소의 속성은?

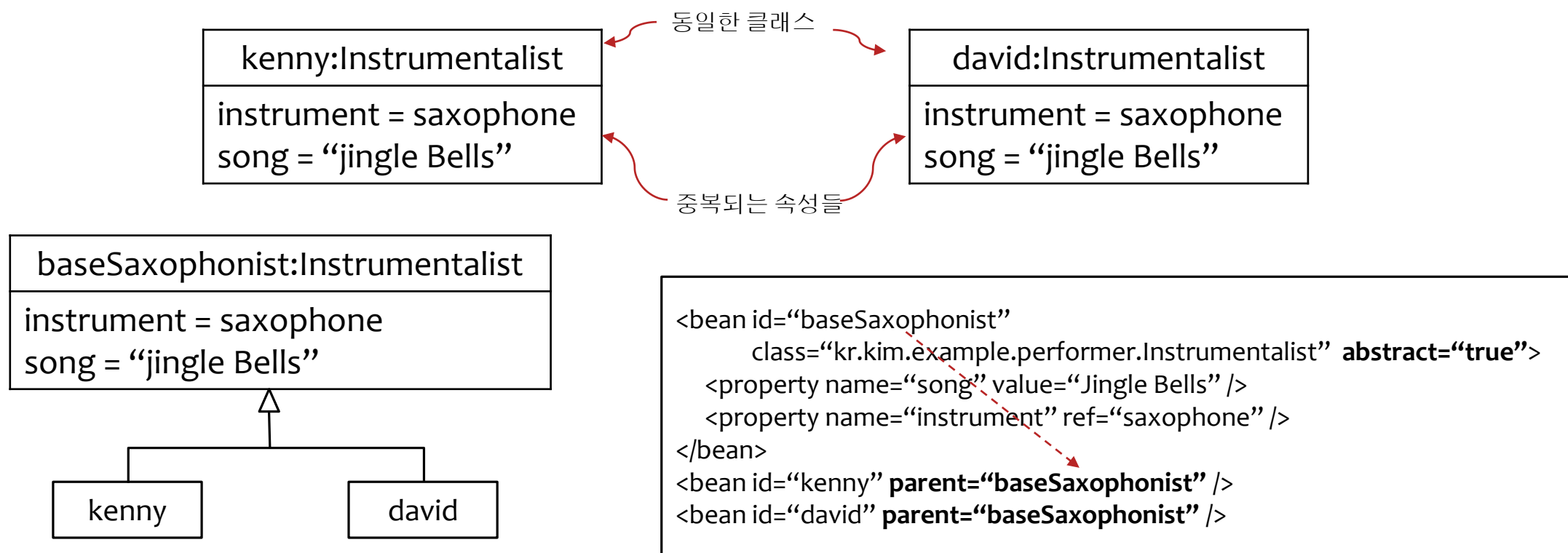
속성	의미	기본값
id	Bean에 붙이는 식별자	
name	Id에 대한 별칭, 복수 정의 가능	
class	Bean 클래스 이름	
parent	Bean 정의를 상속하는 경우 지정하는 새로운 Bean의 id	
abstract	Bean클래스가 추상 클래스 인지 여부	false
singleton	Bean이 싱글턴으로 관리되는지 여부	true
lazy-init	Bean의 로딩을 지연시킬지 여부	default
autowire	오토와이어 설정	default
dependency-check	의존 관계 확인 방법	default
depends-on	Bean이 의존할 Bean 이름, 먼저 초기화 되는 것을 보장	
init-method	Bean 초기화시 실행시킬 메소드	
destroy-method	Bean컨테이너 종료시 실행시킬 메소드	

○○○ 부모와 자식 빈(bean) 선언 (1/4) ○○○

- ▶ 공통적은 부분을 공유
 - ▶ <bean>의 두가지 속성
 - ▶ parent – java의 extends이며, 공통 부분을 공유하고자 하는 <bean> 의 id 속성을 지정함
 - ▶ abstract – true로 세팅 되면 <bean> 선언은 추상화(abstract)됨. Spring에 의해서 절대 초기화되지 않음

부모와 자식 빈(bean) 선언 (2/4)

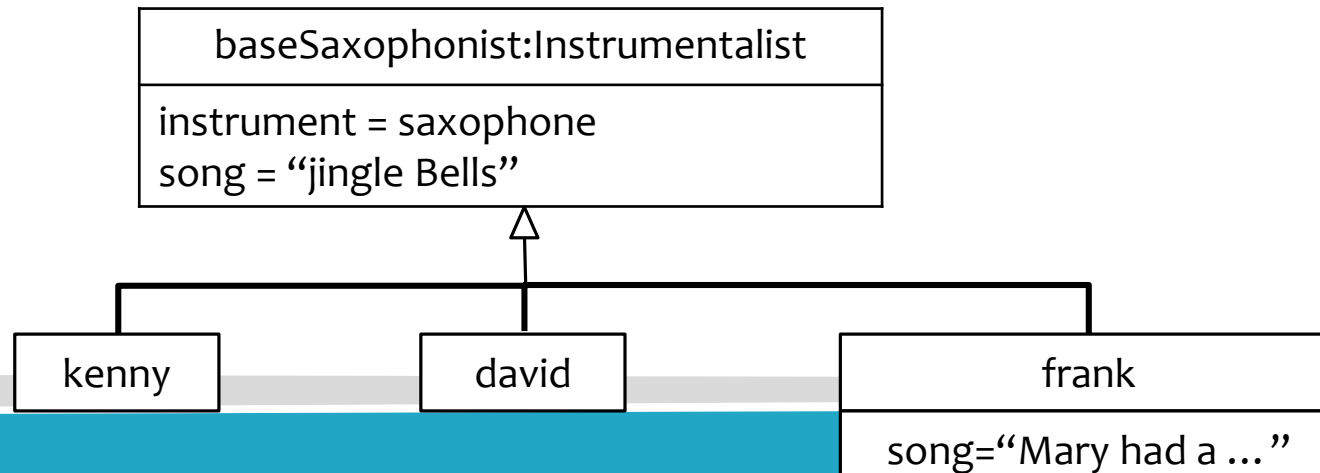
- ▶ 기본 빈(**base bean**) 타입의 추상화
 - ▶ 동일한 역할을 수행하는 빈(bean)



부모와 자식 빈(bean) 선언 (3/4)

- ▶ 기본 빈(**base bean**) 타입의 추상화
 - ▶ 추상 빈(bean)
 - ▶ Spring은 컨테이너로부터 명시적으로 요청을 해도 초기화하지 않음
 - ▶ 부모 빈(bean)이 추상화 될 필요는 없음
 - ▶ 추상 빈(bean)이 아닌 구체화된 빈(concrete bean)을 확장하여 하위 빈(bean) 생성이 가능
 - ▶ 상속되는 속성의 오버라이딩
 - ▶ 자식 빈(bean)은 부모 빈(bean)에 정의된 속성을 재정의하여 사용 가능

```
<bean id="frank" parent="baseSaxophonist" />  
    <property name="song" value="Mary had a little lamb" />  
</bean>
```

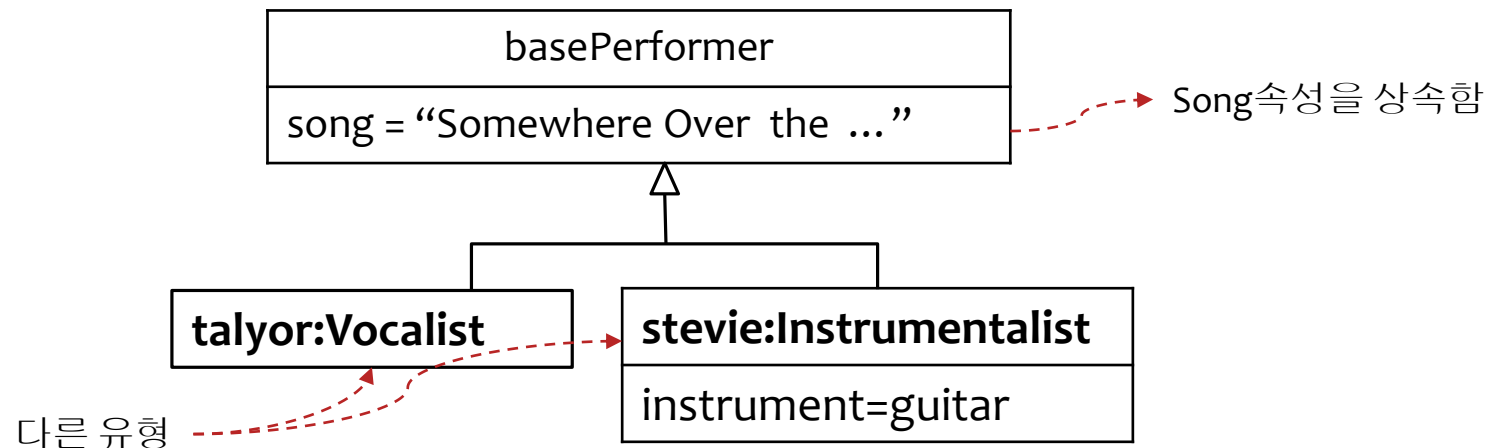


frank bean은 song 속성을
오버라이딩하고 있음

부모와 자식 빈(bean) 선언 (4/4)

- ▶ 공통 속성에 대한 추상화
 - ▶ 서로 다른 유형(클래스 타입)을 가지고 공통된 속성을 공유하는 것이 가능
 - ▶ 서브 빈(sub-bean)

```
<bean id="basePerformer" abstract="true">  
    <property name="song" value="Somewhere Over the Rainbow"/>  
</bean>  
<bean id="taylor" class="kr.kim.example.performer.Vocalist" parent="basePerformer" />  
<bean id="stevie" class="kr.kim.example.performer.Instrumentalist" parent="basePerformer">  
    <property name="instrument" ref="guitar"/>  
</bean>
```



메소드 주입 적용 (1/9)

- ▶ 메소드 주입 (**method injection**)
- ▶ 전체 메소드 정의를 빈(bean)으로 주입이 가능

```
public Object getContents() {  
    return "Hi Hi Hi";  
}
```

주입

MagicBoxImpl

getContents(): Object

- ▶ 두가지 형태
 - ▶ 메소드 대체(method replacement)
 - ▶ Getter 주입(getter injection)

메소드 주입 적용 (2/9)

▶ 기본적인 메소드 대체 (method replacement)

```
public class Musician implements Performer {
    public Musician() {}

    @Override
    public void perform() throws PerformanceException {
        System.out.println(magicWords);
        System.out.println("The magic box contains...");
        System.out.println(magicBox.getContents());
    }
    private MagicBox magicBox;
    public void setMagicBox(MagicBox magicBox) {
        this.magicBox = magicBox;
    }
    private String magicWords;
    public void setMagicWords(String magicWords) {
        this.magicWords = magicWords;
    }
}
```

```
public interface MagicBox {
    String getContents();
}
```

```
public class MagicBoxImpl implements MagicBox
{
    public MagicBoxImpl() {}

    @Override
    public String getContents() {
        return "A beautiful assistant";
    }
}
```

메소드 주입 적용 (3/9)

▶ 기본적인 메소드 대체 (method replacement)

```
<bean id="harry" class="kr.kim.example.performer.Musician">
    <property name="magicBox" ref="magicBox"/>
    <property name="magicWords" value="boo boo boo"/>
</bean>
<bean id="magicBox" class="kr.kim.example.performer.MagicBoxImpl"/>
```

▶ 메소드 대체

```
import org.springframework.beans.factory.support.MethodReplacer;
public class TigerReplacer implements MethodReplacer {
    @Override
    public Object reimplement(Object target, Method method, Object[] args) throws Throwable {
        return "Hi Hi Hi";
    }
}
```

```
<bean id="magicBox" class="kr.kim.example.performer.MagicBoxImpl">
    <replaced-method name="getContents" replacer="tigerReplacer"/> //대체할 메소드 지정
</bean>
<bean id="tigerReplacer" class="kr.kim.example.performer.TigerReplacer"/>
```

메소드 주입 적용 (4/9)

- ▶ 기본적인 메소드 대체 (**method replacement**)
 - ▶ MethodReplacer 인터페이스의 reimplement() 메소드
 - ▶ target: 대체할 메소드가 위치한 객체
 - ▶ method: 대체할 메소드
 - ▶ args: 메소드에 넘겨지는 인자들
 - ▶ 메소드의 실질적인 구현을 배포시까지 알지 못하는 경우에 유용

메소드 주입 적용 (5/9)

▶ Getter 주입 사용

- ▶ 특정 유형의 빈(bean)을 반환하는 메소드에 대한 주입
 - ▶ 메소드는 보통 abstract
 - ▶ 실제 빈(bean)은 Spring 컨텍스트에서 설정되어서 반환

```
public abstract class AbstractInstrumentalist implements Performer {  
    public AbstractInstrumentalist() {}  
  
    @Override  
    public void perform() throws PerformanceException {  
        System.out.println("Playing " + song + " : ");  
        getInstrument().play();  
    }  
    private String song;  
    public void setSong(String song) {  
        this.song = song;  
    }  
    public abstract Instrument getInstrument();           //주입할 메소드  
}
```

메소드 주입 적용 (6/9)

▶ Getter 주입 사용

▶ <lookup-method> 사용

- ▶ 실행시 새로운 구현체로 메소드를 대체
- ▶ getter 에 대해서만 사용
- ▶ MethodReplacer를 구현할 필요가 없음

```
<bean id="stevie" class="kr.kim.example.performer.AbstractInstrumentalist">  
    <lookup-method name="getInstrument" bean="guitar" />  
    <property name="song" value="Greensleeves"/>  
</bean>  
<bean id="guitar" class="kr.kim.example.performer.Guitar"/>
```

▶ 참조하는 guitar 빈(bean)의 범위가 prototype인 경우

- ▶ <bean id="guitar" class="... Guitar" scope="prototype"/>
- ▶ Setter 주입인 경우 속성에 주입되는 시점만 인스턴스가 한번 생성
- ▶ Getter 주입은 getInstrument() 를 호출할 때마다 새로운 인스턴스가 생성

메소드 주입 적용 (7/9)

- ▶ 사용자 정의 속성(**property**) 에디터 등록
 - ▶ `java.beans.PropertyEditor` 인터페이스
 - ▶ `String` 값을 `String`이 아닌 타입으로 매핑하는 방법을 제공
 - ▶ `java.beans.PropertyEditorSupport` 클래스
 - ▶ `getAsText()` – 속성값에 대한 `String` 표현
 - ▶ `setAsText(String value)` – 넘어온 `String` 값을 빈(bean) 속성 값으로 세팅
- ▶ Spring 제공 속성 에디터들

속성 데이터	내용
ClassEditor	속성값이 fully qualified 클래스 명인 <code>String</code> 타입을 <code>java.lang.Class</code> 로 세팅
CustomDateEditor	사용자 정의 <code>java.text.DateFormat</code> 객체를 사용하는 <code>String</code> 타입을 <code>java.util.Date</code> 로 세팅
FileEditor	파일의 경로를 포함하는 <code>String</code> 타입을 <code>java.io.File</code> 로 세팅
LocalEditor	텍스트로 표현되는 지역(en_US)을 포함하는 <code>String</code> 을 <code>java.util.Locale</code> 로 세팅
StringArrayProperty Editor	콤마로 구분된 <code>String</code> 을 <code>String</code> 배열 속성으로 변경
StringTrimmerEditor	빈 <code>String</code> 을 <code>null</code> 로 변환하는 옵션을 포함하여 자동으로 <code>String</code> 속성을 trim
URLEditor	URL형태를 포함하는 <code>String</code> 타입을 <code>java.net.URL</code> 로 세팅

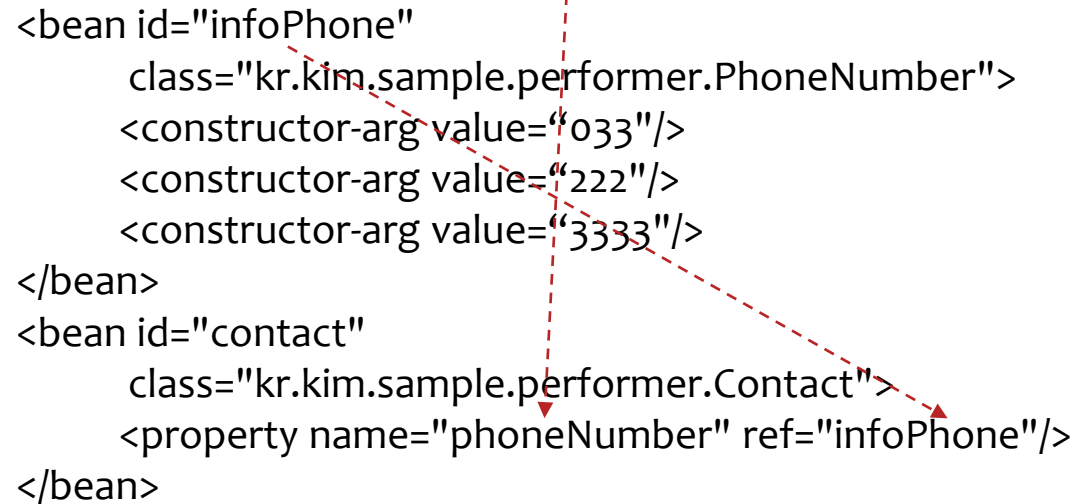
메소드 주입 적용 (8/9)

▶ 사용자 정의 속성(property) 에디터 등록

```
public class PhoneNumber {  
    private String areaCode;  
    private String prefix;  
    private String number;  
  
    public PhoneNumber() {}  
    public PhoneNumber(String areaCode,  
                        String prefix, String number) {  
        this.areaCode = areaCode;  
        this.prefix = prefix;  
        this.number = number;  
    }  
}
```

```
public class Contact {  
    private PhoneNumber phoneNumber;  
    public void setPhoneNumber(PhoneNumber phoneNumber){  
        this.phoneNumber = phoneNumber;  
    }  
}
```

```
<bean id="infoPhone"  
      class="kr.kim.sample.performer.PhoneNumber">  
    <constructor-arg value="033"/>  
    <constructor-arg value="222"/>  
    <constructor-arg value="3333"/>  
</bean>  
<bean id="contact"  
      class="kr.kim.sample.performer.Contact">  
    <property name="phoneNumber" ref="infoPhone"/>  
</bean>
```

A dashed red line with arrowheads at both ends connects the 'infoPhone' bean in the first XML block to the 'phoneNumber' property of the 'contact' bean in the second XML block, illustrating the dependency injection.

메소드 주입 적용 (9/9)

▶ 사용자 정의 속성(property) 에디터 등록

```
public class PhoneEditor extends PropertyEditorSupport {
    @Override
    public void setAsText(String textValue) throws IllegalArgumentException
    {
        String stripped = stripNonNumeric(textValue);
        String areaCode = stripped.substring(0, 3);
        String prefix = stripped.substring(3, 6);
        String number = stripped.substring(6);
        PhoneNumber phone = new PhoneNumber(areaCode, prefix, number);
        setValue(phone);
    }

    private String stripNonNumeric(String original) {
        StringBuilder allNumeric = new StringBuilder();
        for (int i = 0; i < original.length(); i++) {
            char c = original.charAt(i);
            if (Character.isDigit(c))
                allNumeric.append(c);
        }
        return allNumeric.toString();
    }
}
```

```
<bean
    class="org.springframework.beans.factory.config.CustomEditorConfigurer">
    <property name="customEditors">
        <map>
            <entry key="kr.kim.example.performer. PhoneNumber">
                <bean id="phoneEditor" class="kr.kim.example.performer.PhoneEditor"/>
            </entry>
        </map>
    </property>
</bean>
<bean id="contact" class="kr.kim.performer.Contact">
    <property name="phoneNumber" value="033-555-7777"/>
</bean>
```