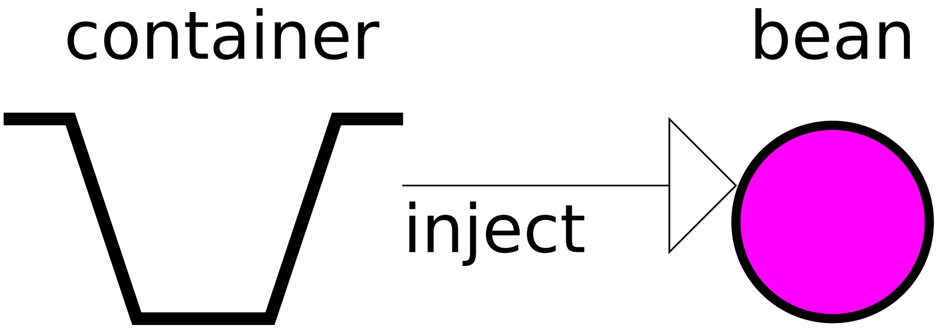
Dependency Injection

- Setter Injection
- Constructor Injection
- Annotations

Introduction to Dependency Injection

- **Dependency injection** is a technique in which the container provides resource references for beans
- To inject a resource, the container either calls a bean's "set" method or constructor
- DI lessens coupling between beans and decouples beans from services such as JNDI



Dependency injection is a form of Inversion of Control</i> where the container supplies a bean with its dependencies.

The term "Dependency injection" was coined by Martin Fowler.

Spring developers typically write an XML file to configure their beans and how the Spring container should inject dependencies.

Inversion of Control

- IoC is an architectural pattern in which the flow of control of a system is inverted in comparison to procedural programming
- Sometimes referred to as "The Hollywood Principle"
- Dependency injection is a form of IoC in which a container provides resources instead of the traditional practice of performing lookups

For a good read on Inversion of Control and Dependency Injection, go to:

`http://martinfowler.com/articles/injection.html`

Sample Application for this Chapter

Student.java

Advisor.java

TestDI.java

spring.xml

```java
package university;

public class Student
{
    private int id;
    private String name;
    private double gpa;
    private Advisor theAdvisor;

    public Student(String name, Advisor advisor)
    {
        this.name = name;
        this.theAdvisor = advisor;
    }

    public Student(int id, double gpa)
    {
        this.id = id;
        this.gpa = gpa;
    }

    public Student() {}

    public Advisor getTheAdvisor()
    {
        return theAdvisor;
    }

    public void setTheAdvisor(Advisor theAdvisor)
    {
        this.theAdvisor = theAdvisor;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public double getGpa()
    {
        return gpa;
    }

    public void setGpa(double gpa)
    {
        this.gpa = gpa;
    }

    @Override
    public String toString()
    {
        return "ID: " + id + " name: " + name + " GPA: " + gpa;
    }
}
```

```java
package university;

public class Advisor
{
    private String name;
    private double salary;

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public double getSalary()
    {
        return salary;
    }

    public void setSalary(double salary)
    {
        this.salary = salary;
    }

    @Override
    public String toString()
    {
        return "Name: " + name + " salary: " + salary;
    }
}
```

```java
package main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import university.Student;

public class TestDI
{

    public static void main(String[] args)
    {
        ApplicationContext ctx =
            new FileSystemXmlApplicationContext("spring.xml");

        Student s0 = (Student)ctx.getBean("student0");
        System.out.println(s0);

        Student s1 = (Student)ctx.getBean("student1");
        System.out.println(s1);

        Student s2 = (Student)ctx.getBean("student2");
        System.out.println(s2);

        Student s3 = (Student)ctx.getBean("student3");
        System.out.println(s3);

        Student s4 = (Student)ctx.getBean("student4");
        System.out.println(s4);

        Student s5 = (Student)ctx.getBean("student5");
        System.out.println(s5);
    }
}
```

```xml
<?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-2.0.xsd">

   <bean id="advisor1" class="university.Advisor"/>

   <bean id="student0" class="university.Student">
     <property name="name" value="Harry"/>
     <property name="id" value="77"/>
     <property name="gpa" value="3.14"/>
     <property name="theAdvisor" ref="advisor1"/>
   </bean>

   <bean id="student1" class="university.Student">
     <constructor-arg value="Joe"/>
     <constructor-arg ref="advisor1"/>
   </bean>

   <bean id="student2" class="university.Student">
     <constructor-arg ref="advisor1"/>
     <constructor-arg value="Sue"/>
   </bean>

   <bean id="student3" class="university.Student">
     <constructor-arg value="12"/>
     <constructor-arg value="3.33"/>
   </bean>

   <bean id="student4" class="university.Student">
     <constructor-arg value="3.33" index="1"/>
     <constructor-arg value="12" index="0"/>
   </bean>

   <bean id="student5" class="university.Student">
     <constructor-arg value="3.33" index="1"/>
     <constructor-arg value="12" index="0"/>
     <property name="name" value="George"/>
     <property name="theAdvisor" ref="advisor1">
     </property>
   </bean>
</beans>
```

spring.xml

Setter Injection

- The JavaBean specification defines a **property** as data exposed via get/set methods
- Spring can use the "set" method on JavaBean-style properties to inject simple values and bean references

Student bean

```
public class Student
{
  public void setId(int id) {...}
  public void setName(String name) {...}
  public void setTheAdvisor(Advisor advisor) {...}
  public void setGpa(double gpa) {...}
 . . .
}
```

spring.xml

```
<bean id="advisor1" class="university.Advisor"/>

<bean id="student0" class="university.Student">
  <property name="name" value="Harry"/>
  <property name="id" value="77"/>
  <property name="gpa" value="3.14"/>
  <property name="theAdvisor" ref="advisor1"/>
</bean>
```

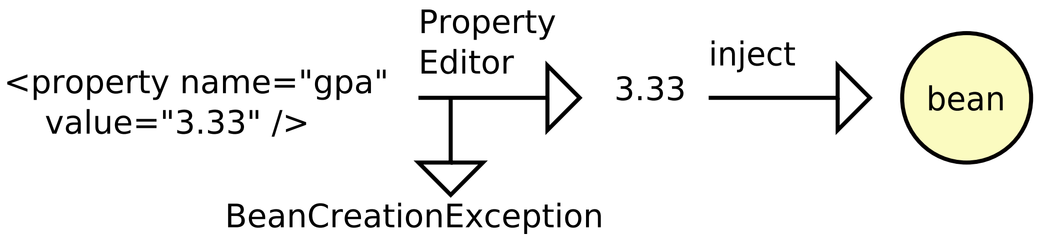
This is the most common form of injection.

Note that for this to work, the bean must have a "set" method for the property that matches the JavaBean rules. The order of the "property" elements in the XML is not important.

Spring only uses the "set" method, so a "get" method is only required if your own logic needs it. Note that other frameworks, e.g. JBoss Seam, also support "out-jection", where the framework does need the "get" method.

Property Conversions

- In the XML configuration, values are defined as character strings
- Spring converts the character strings as required to match the property type using standard JavaBean **property editors**



PropertyEditors are part of JavaBeans spec, but the spec just specifies the interface. However, Spring provides several PropertyEditors, many of which are from the sun.beans.editors package:

```
BooleanEditor.java
ByteEditor.java
ColorEditor.java
DoubleEditor.java
FloatEditor.java
FontEditor.java
IntegerEditor.java
LongEditor.java
NumberEditor.java
ShortEditor.java
StringEditor.java
```

If you search for "DoubleEditor.java" with Google, you can find the source code for this class and see how simple the conversion process is.

Constructor Injection

- Instead of, or in addition to using **setter** injection, Spring supports **constructor** injection that provides properties to a bean during instantiation
- The bean must define an appropriate constructor

Student bean

```
public Student(int id, double gpa){...}
```

spring.xml

```
<bean id="student3" class="university.Student">
  <constructor-arg value="12"/>
  <constructor-arg value="3.33"/>
</bean>
```

If you do NOT use constructor injection, then the bean MUST have a public zero-argument constructor.

Constructor Injection Resolution

- When you specify constructor arguments, Spring examines the bean for a matching constructor
- Spring attempts to use the types used in the XML file, but note that **values** don't have a type (they are character strings in XML)

Student bean

```
public Student(String name, Advisor advisor) {...}
```

spring.xml

```
<bean id="student1" class="university.Student">
  <constructor-arg value="Joe"/>
  <constructor-arg ref="advisor1"/>
</bean>

<bean id="student2" class="university.Student">
  <constructor-arg ref="advisor1"/>
  <constructor-arg value="Sue"/>
</bean>
```

In this case, since one of the constructor-args is a reference to an Advisor bean, Spring doesn't care about the order in which you write the constructor-arg elements. In other words, since Spring can determine the type of the "ref" constructor-arg, and there are two constructor-args, Spring can match up with the actual Java constructor.

Constructor Injection Resolution, cont'd

- To aid resolution, you can use the optional **type** and/or **index** attributes in the XML
- Note that if you specify **constructor-args** in the same order as they appear in the bean's constructor, this is often not necessary

Student bean

```
public Student(int id, double gpa){...}
```

spring.xml

```
<bean id="student3" class="university.Student">
  <constructor-arg value="12"/>
  <constructor-arg value="3.33"/>
</bean>

<bean id="student4" class="university.Student">
  <constructor-arg value="3.33" index="1"/>
  <constructor-arg value="12" index="0"/>
</bean>
```

For "student4", both of the constructor-args are values, not references, so Spring cannot determine the type. And since we put a string with a decimal point ("3.3") as the first argument, Spring would try to convert it to an integer and fail. Thus, the "index" attribute is necessary.

Mixing Injection Types

- It's OK to use both setter and constructor injection on a bean
- Spring will first create the bean using the constructor arguments, then invoke the set method(s)

Student bean

```
public Student(int id, double gpa) {...}
public void setName(String name) {...}
public void setTheAdvisor(Advisor advisor) {...}
```

spring.xml

```
<bean id="student5" class="university.Student">
  <constructor-arg value="3.33" index="1"/>
  <constructor-arg value="12" index="0"/>
  <property name="name" value="George"/>
  <property name="theAdvisor" ref="advisor1">
  </property>
</bean>
```

Setter vs Constructor Injection

- The Spring documentation favors setter injection over constructor injection since it's simpler and more intuitive
- Constructor injection does allow you to guarantee intialization (especially if you ONLY allow constructor injection)
- In Spring 2.5 and later, you can use the **@Required** annotation on a "set" method to guarantee initialization

Note that if you want to use the @Required annotation, you will also need to define this bean so that Spring can process the annotation:

```
<bean class=
  "org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>
```

Chapter Summary

In this chapter, you learned:

- How to inject values and references via **setter** injection
- How to inject values and references via **constructor** injection