The Spring Container

- Spring is a container based framework
- The container must be configured to tell it what beans it should contain, instantiate, and assemble

- Three Styles of providing the Container Metadata
  - **XML** based (traditional format)
  - **Annotation** based (introduced in Spring 2.5)
  - **Java** based (introduced in Spring 3.0)

Wiring Your Applications

"Wiring" – creating the associations between application objects

- Spring uses the metadata to specify the actual implementations used
- Specific implementations can then be injected into objects that require their services
- It resembles connecting your components with "wires"
- The wiring can easily be changed at any time (even without any recompilation) by modifying the .xml file

Namespaces (Schemas) in the Config File

- The root element of the XML file is <beans> from Spring's beans schema
  - An XML schema describes the structure of an XML document (expected tags, attributes, etc)
  - The namespaces must be specified to be able to use tags such as <beans> and <bean>
  - At times, you'll need to declare additional schemas which give you additional tag sets

```
<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-3.1.xsd">

  <!-- add your bean definitions here! -->

</beans>
```

Specifying Beans

- The <bean> tag configures one bean
  - Use the **id** or **name** attribute to give the bean a Name
  - Provide a package-qualified **class** name
  - Additional attributes set the beans properties and set the bean's lifecycle

```
<!-- Spring XML configuration file -->
<beans>
   <bean id="acctServiceIntlRules"
           class="nvz.services.AccountingServiceIntlRules" />

   <bean name="billingService"
           class="nvz.services.UsaBillingService" />

</beans>
```

- It is not mandatory to give a bean an id or name. If no name or id is given, Spring generates a unique name for the bean.
- The XML **id** attribute is preferred, but does limit the characters that can be used

Dependency Injection

A mechanism for supplying components with their dependencies and managing the dependency objects throughout their lifecycle

- Objects obtain their dependencies without having to request them
- In the case of Spring the dependency objects are created and maintained by its **IoC Container**

Dependency Injection

- Dependencies that may be injected
  - Collaborator objects
  - Simple values (Strings, integers, etc.)

- Types of Injection
  - Setter
  - Constructor
  - Method
    - Specify implementation for an abstract method or replace a method (implementation method is in another class)
    - Rarely used

Setter Injection (Beans)

- Specify the property as a reference (ref)

```
public class InvoiceGeneratorImpl implements InvoiceGenerator {
    private String companyName;
    private int companyId;
    private double salesTax;
    private ShippingChargeCalculator shipChargeCalc;

    public void setShipChargeCalc(ShippingChargeCalculator shipChargeCalc) {
        this.shipChargeCalc = shipChargeCalc;
    }
    ...
}
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
            "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <!-- add your bean definitions here! -->
    <bean id="shippingCalculator"
            class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

    <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
        <property name="shipChargeCalc" ref="shippingCalculator" />
    </bean
</beans>
```

Setter Injection (Simple Values)

- Specify the property as a **value**
  - Values are automatically converted to any primitive type or wrapper class

```
public class InvoiceGeneratorImpl implements InvoiceGenerator {
   private String companyName;
   private int companyId;
   private double salesTax;
   private ShippingChargeCalculator shipChargeCalc;

   public void setSalesTax(double salesTax) {
      this.salesTax = salesTax;
   }
   ...
}
```

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
            class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
      <property name="companyName" value="ZBooks" />
      <property name="companyId" value="129" />
      <property name="salesTax" value=".0875" />
      <property name="shipChargeCalc" ref="shippingCalculator" />
   </bean>
</beans>
```

p: notation (p namespace)

- Spring 2.5 introduced a shortcut to reduce XML
  - You'll need to configure the p namespace schema in the beans tag of your xml file
  - Eliminates the property tag
  - Simple Value Form: **p:**name="value"
  - Reference Form: **p:**bean-**ref**="value"

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
         class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
        p:companyName="ZBooks"
        p:companyId="129"
        p:salesTax=".0875"
        p:shipChargeCalc-ref="shippingCalculator">
   </bean>

</beans>
```

Constructor Injection

- Instead of calling the default constructor, the IoC container can call a different constructor and provide the collaborators
- Use the **constructor-arg** tag in the XML to specify constructor arguments
- A complication – there can be more than one parameter
  - How to match up the values to the parameters?

➢ 3 Ways to do Constructor parameter matching
  - Index (Recommended Practice)
  - Type
  - Name

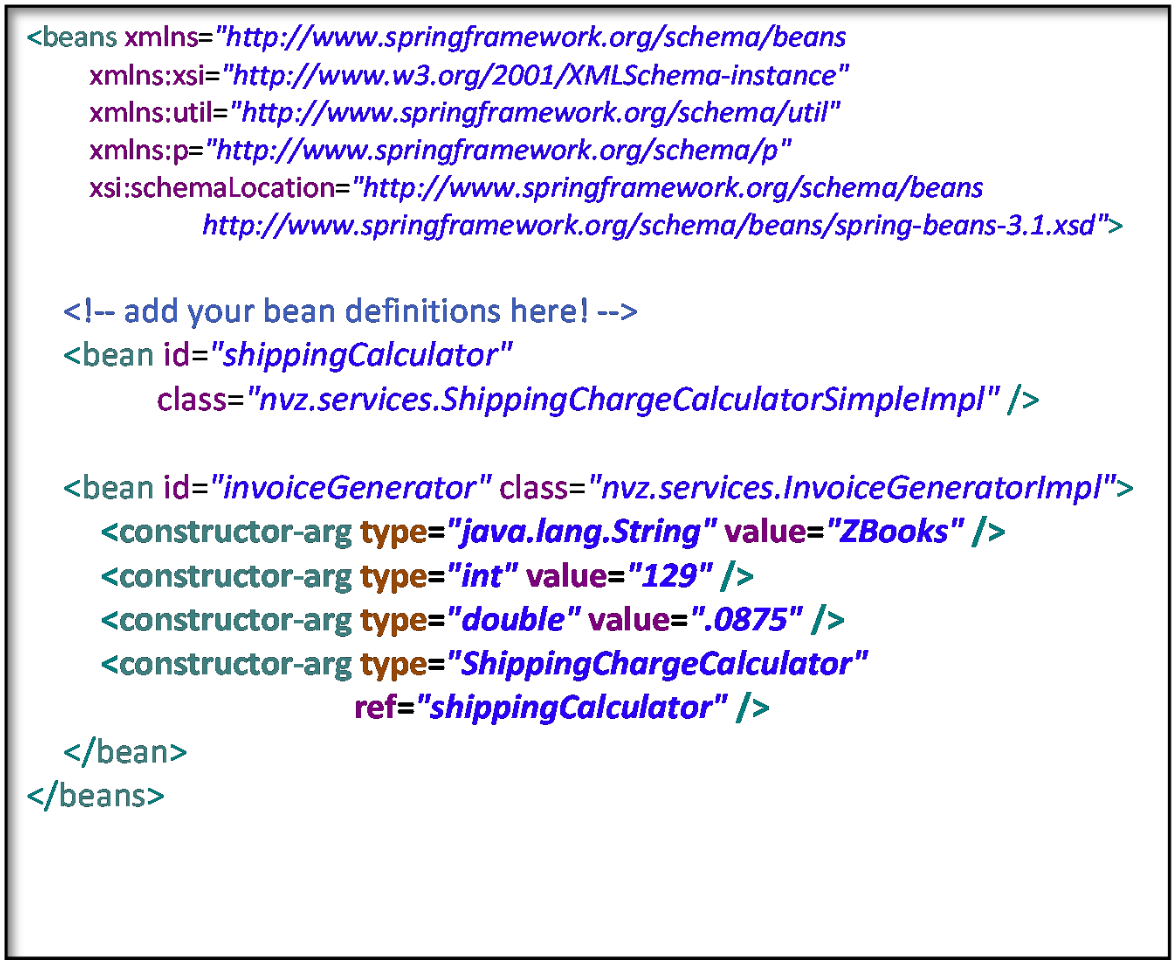
Constructor Injection (by Index)

```
public class InvoiceGeneratorImpl implements InvoiceGenerator {
   private String companyName;
   private int companyId;
   private double salesTax;
   private ShippingChargeCalculator shipChargeCalc;

   public InvoiceGeneratorImpl(String companyName, int companyId,
         double salesTax, ShippingChargeCalculator shipChargeCalc)
   {
      this.companyName = companyName;
      this.companyId = companyId;
      this.salesTax = salesTax;
      this.shipChargeCalc = shipChargeCalc;
   }
}
```

```
<beans xmlns="http://www.springframework.org/schema/beans
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
         class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
      <constructor-arg index="0" value="ZBooks" />
      <constructor-arg index="1" value="129" />
      <constructor-arg index="2" value=".0875" />
      <constructor-arg index="3" ref="shippingCalculator" />
   </bean>
</beans>
```

Constructor Injection (by Type)

- Specify type
- Spring will try to match that with a constructor parameter
- Spring cannot always tell which constructor to use



```
<beans xmlns="http://www.springframework.org/schema/beans
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                 http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
          class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
      <constructor-arg type="java.lang.String" value="ZBooks" />
      <constructor-arg type="int" value="129" />
      <constructor-arg type="double" value=".0875" />
      <constructor-arg type="ShippingChargeCalculator"
                           ref="shippingCalculator" />
   </bean>
</beans>
```

Constructor Injection (by Name)

- Specify parameter name
- Add @ConstructorProperties to constructor in case names lost in bytecode (if the –g debug flag is not used)

```
<beans xmlns="http://www.springframework.org/schema/beans
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
         class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
      <constructor-arg name="companyName" value="ZBooks" />
      <constructor-arg name="companyId" value="129" />
      <constructor-arg name="salesTax" value=".0875" />
      <constructor-arg name="shipChargeCalc" ref="shippingCalculator" />
   </bean>
</beans>
```

Constructor Injection (by Name)

- @ConstructorProperties
  - Available as of JDK 6
  - Can be used to match up property names with parameter names if they didn't match
  - Can be used if debug flags are off

```
import java.beans.ConstructorProperties;

public class InvoiceGeneratorImpl implements InvoiceGenerator {
   private String companyName;
   private int companyId;
   private double salesTax;
   private ShippingChargeCalculator shipChargeCalc;

   @ConstructorProperties({"companyName", "companyId", "salesTax",
                   "shipChargeCalc"})
   public InvoiceGeneratorImpl(String companyName, int companyId,
         double salesTax, ShippingChargeCalculator shipChargeCalc)
   {
      this.companyName = companyName;
      this.companyId = companyId;
      this.salesTax = salesTax;
      this.shipChargeCalc = shipChargeCalc;
   }
```

c: notation (c namespace)

- Spring 3.1 introduced a shortcut to reduce XML
  - You'll need to configure the c namespace schema in the beans tag of your xml file
  - Eliminates the <constructor-arg> tag
  - Simple Value Form: **c**:style="value"
  - Reference Form: **c**:style-**ref**="value"

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:c="http://www.springframework.org/schema/c"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
          class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
      c:_0="ZBooks"
      c:_1="129"
      c:_2=".0875"
      c:_3-ref="shippingCalculator" />

</beans>
```

c: notation (c namespace)

- Using name instead of index

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:c="http://www.springframework.org/schema/c"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

   <!-- add your bean definitions here! -->
   <bean id="shippingCalculator"
          class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
      c:companyName="ZBooks"
      c:companyId="129"
      c:salesTax=".0875"
      c:shipChargeCalc-ref="shippingCalculator" />
</beans>
```

Injecting Collections

- Spring provides tags for properties that use a Java Collections Class:
  - **<list>** for java.util.List
  - **<set>** for java.util.Set
  - **<map>** for java.util.Map
  - **<props>** for java.util.Properties

```
public class NewsletterSender {
   private String emailServers[];

   public void setEmailServers(String emailServers[]) {
      this.emailServers = emailServers;
   }
}
```

```
<bean id="newsletterSender" class="nvz.services.NewsletterSender">
   <property name="emailServers">
      <list>
         <value>"smtp.gmail.com"</value>
         <value>"smtp.mail.yahoo.com"</value>
         <value>"smtp.live.com"</value>
      </list>
   </property>
</bean>
```

Injecting Collections

- Use <ref> for bean references instead of simple values

```
public class NewsletterSender {
    private List<String> emailServers;

    public void setEmailServers(List<String> emailServers) {
        this.emailServers = emailServers;
    }
```

```
<bean id="newsletterSender" class="nvz.services.NewsletterSender">
    <property name="emailServers">
        <list>
            <ref bean="gmailServer” />
            <ref bean="yahooServer” />
            <ref bean="windowsLiveServer” />
        </list>
    </property>
</bean>
```

Spring Expression Language (SpEL)

Spring doesn’t limit you to constant values, you may also use expressions

- Format

  #{SpEL expression goes here}

- SpEL expressions may include
  - References to bean ids

    ```
    <property name=”invGenerator” value=”#{invoiceGenerator}” />
    ```

  - Calling methods of beans

    ```
    <property name="salesTax" value="#{invoiceGenerator.salesTax}" />
    <property name="salesTax" value="#{invoiceGenerator.getSalesTax()}" />
    ```

  - Mathematical, Relational, and Logical operators

    ```
    <property name=”totalTax” value=”#{invoiceGenerator.salesTax + 0.25}” />
    <property name=”presidentMsg” value=”#{‘The President is:’ + president.name}” />
    ```

  - Regular expression matching

    ```
    <property name=”validPhoneNum” value=”#{officeInfo.phoneNumber matches ‘\d{3}-\d{4}-\d{3}’}” />
    ```

  - Manipulation of Collection objects

    ```
    <property name=”VNameClients” value=”#{Clients.?[Name.startsWith(‘V’)]}” />
    ```

    (retrieves all clients whose name starts with ‘V’)

Resolving Dependencies

- Spring will automatically determine the dependencies and instantiate objects in the required order
  - The order of beans in your .xml does not matter

```
<bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
        <property name="shipChargeCalc" ref="shippingCalculator" />
</bean>

<bean id="shippingCalculator" class="nvz. ShippingChargeCalculator" />
```

- If for some reason Spring is unable to determine a dependency you could use the depends-on attribute

```
<bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
        depends-on=”shippingCalculator”>
        <property name="shipChargeCalc" ref="shippingCalculator" />
</bean>

<bean id="shippingCalculator" class="nvz. ShippingChargeCalculator" />
```

Another Shortcut: Autowiring

- You don't always have to explicitly specify how components are wired together
- Autowiring is where Spring attempts to wire your components automatically
- Turned off by default
- Not recommended for production code
  - Becomes difficult to determine where the problem is when something goes wrong
  - Hides dependency information
  - More commonly used for prototyping to test quickly

Autowiring by Name

- Spring attempts to match all properties of a bean with beans of the same name
- Properties that have no match will remain unwired
  - The order of beans in your .xml does not matter

```
public class InvoiceGeneratorImpl implements InvoiceGenerator {
   private ShippingChargeCalculator shippingCalculator;
   private String companyName;
```

```
<bean id="shippingCalculator" class="nvz. ShippingChargeCalculator" />

<bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
        <property name="shippingCalculator" ref="shippingCalculator" />
        <property name="companyName" value="ZBooks" />
</bean>
```

⬇

```
<bean id="shippingCalculator" class="nvz. ShippingChargeCalculator" />

<bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
      autowire="byName">
        <property name="companyName" value="ZBooks" />
</bean>
```

Autowiring by Name

- You can mix autowiring with explicitly specified references
- Any property explicitly provided for will not be autowired

```
public class InvoiceGeneratorImpl implements InvoiceGenerator {
   private ShippingChargeCalculator shippingCalculator;
   private SalesTaxCalculator salesTaxCalculator;
```

```
<bean id="shippingCalculator" class="nvz. ShippingChargeCalculator" />

<!--      shippingCalculator is autowired, salesTaxCalculator is explicitly wired    -->
<bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl"
       autowire="byName">
          <property name="salesTaxCalculator" ref="SalesTaxCalcImpl" />
          <property name="companyName" value="ZBooks" />
</bean>
```

Types of Autowiring

- byName
  - match property name to bean name
- byType
  - match property type to bean type
  - fails if multiple types match
- constructor
  - attempt to match up constructor arguments with beans whose **type** matches
- autodetect
  - Attempt **constructor** autowiring; if that fails try **byType** for properties

Default Autowiring

- You can apply autowiring to all beans by adding a **default-autowire** attribute to the root **<beans>** element
- No need to add **autowire** tag to every (or most) beans in your .xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd"
       default-autowire="byName">

   <!-- add your bean definitions here! -->

   <bean id="shippingCalculator"
          class="nvz.services.ShippingChargeCalculatorSimpleImpl" />

   <bean id="invoiceGenerator" class="nvz.services.InvoiceGeneratorImpl">
         <property name="companyName" value="ZBooks" />
   </bean>

</beans>
```

Property Files

- A collection of key-value pairs that can be parsed by the java.util.Properties class
  - A way in Java to put constant properties that may change in a file instead of Java code
  - Changes don't require compilation
  - Don't have to dig through Java code to find them
- Each line represents one property. The format may be:
  - key=value or key = value
  - key: value
  - key value

invoice.properties

```
invoice.companyName = zBooks
invoice.companyId:129
invoice.salesTax .0875
```

Property Files

- Spring makes it easy to use property files
- Pull in properties by one of the following methods:
  - Adding a PropertyPlaceHolderConfigurer bean in your .xml file
  - Use the context:property-placeholder shortcut

```
<bean id="propertyPlaceholderConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="location"
            value="classpath:invoice.properties" />
</bean>
```

Or use the shortcut:

```
<context:property-placeholder
    location="classpath:invoice.properties"/>
```

Property Files

- Can provide a list of property files

```
<bean id="propertyPlaceholderConfigurer"
   class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="locations">
    <list>
      <value>classpath:invoice.properties</value>
      <value>classpath:database.properties</value>
    </list>
  </property>
</bean>
```

Or multiple context:property-placeholder tags:

```
<context:property-placeholder
    location="classpath:invoice.properties"/>

<context:property-placeholder
    location="classpath:database.properties"/>
```

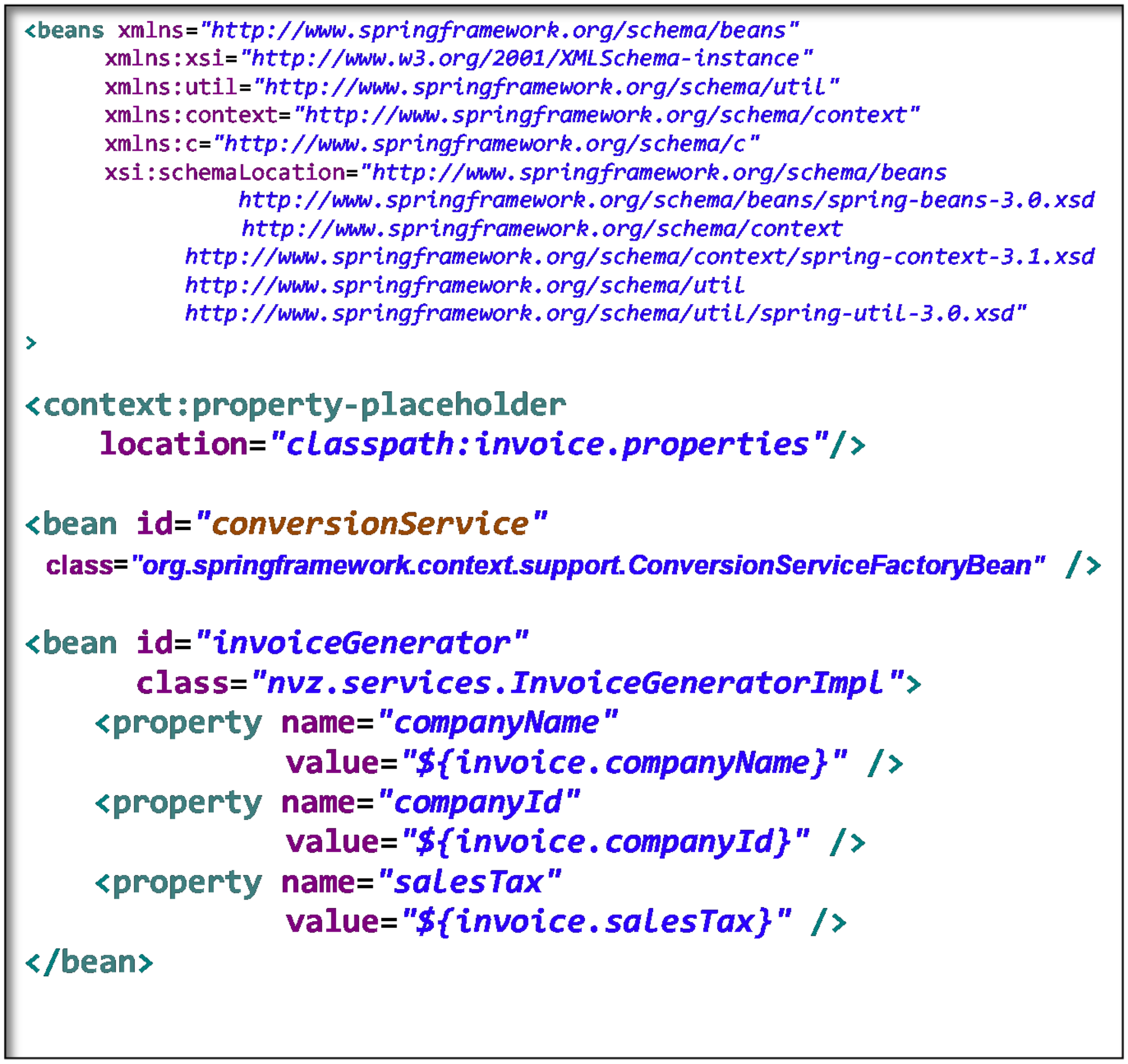
Property Files

- Property files hold String values
- To access String properties

```
<bean id="invoiceGenerator"
      class="nvz.services.InvoiceGeneratorImpl">
     <property name="companyName"
               value="${invoice.companyName}" />
</bean>
```

Accessing Non String Properties

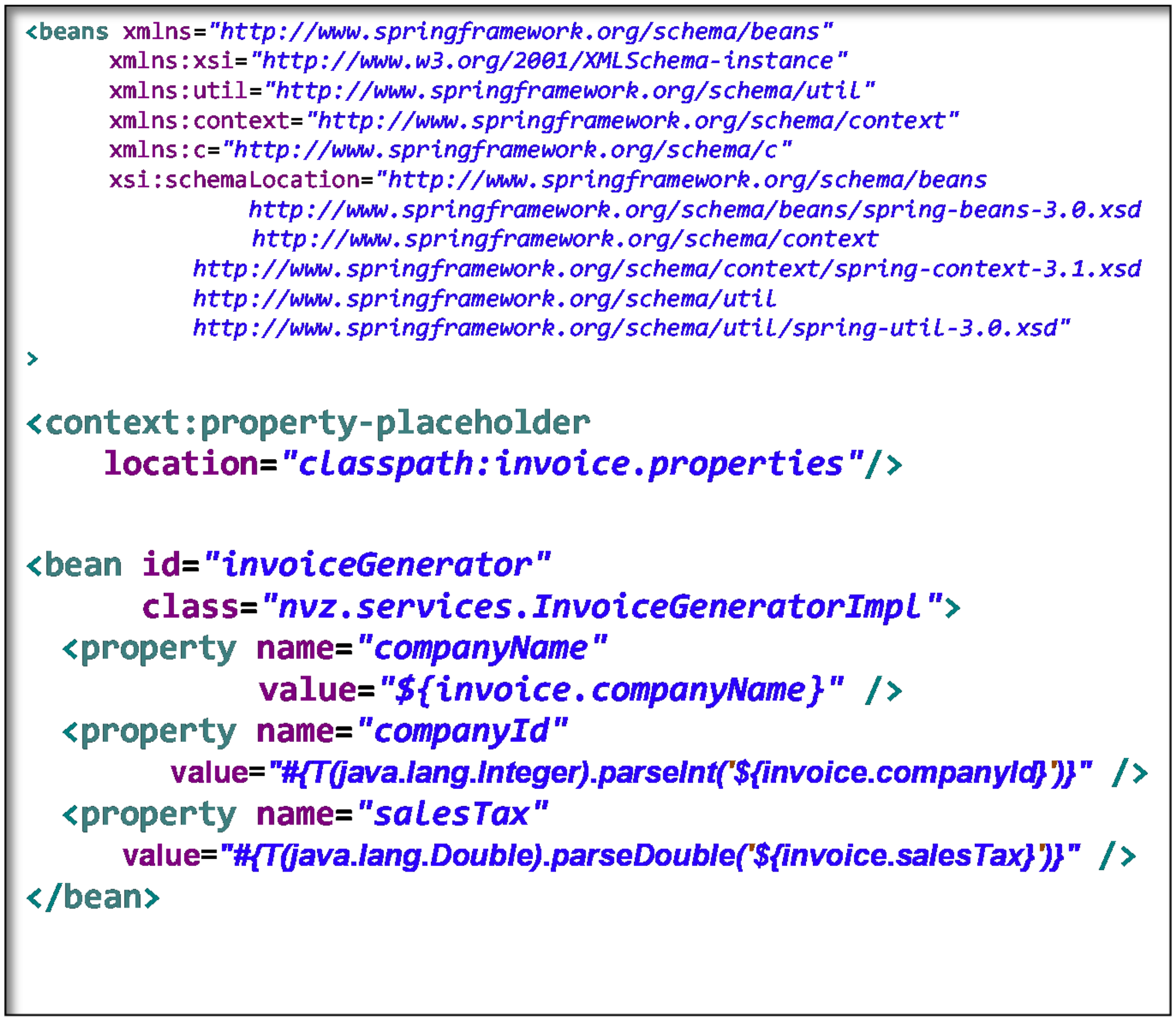
- To get the default type converters installed, use a ConversionService bean
  - It will be picked up by Spring and used for conversion from String to other types



```xml
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:c="http://www.springframework.org/schema/c"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                 http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.1.xsd
            http://www.springframework.org/schema/util
            http://www.springframework.org/schema/util/spring-util-3.0.xsd"
>

<context:property-placeholder
     location="classpath:invoice.properties"/>

<bean id="conversionService"
  class="org.springframework.context.support.ConversionServiceFactoryBean" />

<bean id="invoiceGenerator"
      class="nvz.services.InvoiceGeneratorImpl">
    <property name="companyName"
              value="${invoice.companyName}" />
    <property name="companyId"
              value="${invoice.companyId}" />
    <property name="salesTax"
              value="${invoice.salesTax}" />
</bean>
```

Accessing Non String Properties

- An alternative: Use SpEL to convert



```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:c="http://www.springframework.org/schema/c"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                  http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.1.xsd
            http://www.springframework.org/schema/util
            http://www.springframework.org/schema/util/spring-util-3.0.xsd"
>

<context:property-placeholder
    location="classpath:invoice.properties"/>


<bean id="invoiceGenerator"
      class="nvz.services.InvoiceGeneratorImpl">
  <property name="companyName"
            value="${invoice.companyName}" />
  <property name="companyId"
        value="#{T(java.lang.Integer).parseInt('${invoice.companyId}')}" />
  <property name="salesTax"
      value="#{T(java.lang.Double).parseDouble('${invoice.salesTax}')}" />
</bean>
```

When are Beans Created?

- By default the Spring container instantiates and configures all beans as part of its initialization process
  - This means errors in the configuration metadata or environment are discovered immediately
  - The default behavior can be changed or the behavior can be changed on a bean by bean basis

- **lazy-init** attribute
  - Create a bean instance when it is requested, not at startup

```
<bean id="shippingCalculator"
        class="nvz.services.ShippingChargeCalculatorSimpleImpl"
        lazy-init="true" />
```

Changing the default behavior of the container:

```
<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-3.1.xsd"
    default-lazy-init=”true”>
```

Bean Scopes

- singleton
  - One instance of the bean per Spring container
  - This is the default bean scope
- prototype
  - Allows multiple instances of a bean in a Spring container

```
<bean id="shippingCalculator"
      class="nvz.services.ShippingChargeCalculatorSimpleImpl"
      scope="prototype" />
```

  - By default prototype beans are lazy-init
- request
  - One bean is created for every HTTP request
- session
  - One bean is created for every HTTP session