Mastering the Spring Framework

Chapter 1:
Introducing the Spring Framework

Chapter Objectives

In this chapter, we will:

- Provide a basic introduction to XML
- Introduce the Spring Tool Suite
- Introduce the Spring Framework and its core areas of functionality

What This Course Covers

- We will begin by discussing several technologies and tools that will be used in our software development:
  - XML which will be used for several configuration files
  - Maven which will control the project dependencies and build process
  - Spring Tool Suite which provides Eclipse plugins that are Spring aware
- We will then introduce the Spring object factory and its role in development
  - Spring will create objects (beans) for us and inject their dependencies on other objects

Chapter Concepts

What Is XML?

The Spring Tool Suite

The Spring Object Factory

Annotation-Based Factory Configuration

Chapter Summary

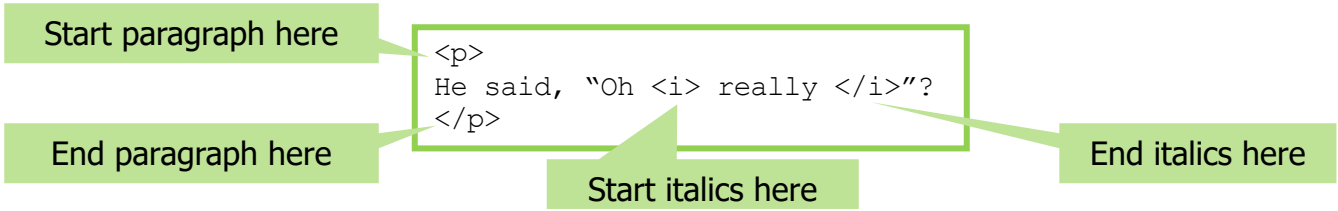
What Is XML?

- XML is an acronym for Extensible Markup Language
  - A standard from the World Wide Web Consortium (w3c)
- XML documents are in plain text
  - White space usually doesn't matter
  - Indentation is a way of improving readability

```
<!-- This is a comment -->
<project basedir="." default="build" name="ex1">
   <target name="clean">
        <delete dir="build"/>
   </target>
   <target name="build">
        <mkdir dir="build"/>
        <javac destdir="build">
            <src path="src"/>
            <classpath refid="project.classpath"/>
        </javac>
   </target>
</project>
```

What Is a Markup Language?

- Markup means that different parts of the document itself are annotated
  - For example, consider the Hyper-Text Markup Language (HTML):
    - Makes it easy for a computer program to decode content and display it
    - No need to consult separate file that says "characters 32-41" are italics



- Many markup languages are textual
  - Makes it easy for humans to view/edit documents
  - Documents are portable across platforms

What Is Extensibility?

- HTML is not an extensible language
  - The tags allowed (`p`, `b`, `it`, `br`, etc.) are part of a standard
- What happens if you insert a made-up tag into your HTML document?

```
<p>
He said, "Oh <irony> really </irony>"?
</p>
```

  - The HTML standard specifies that a browser should ignore such tags
    - So that older browsers can continue to operate on documents that are written to newer versions of the HTML standard
- To add a tag to HTML, the W3C has to approve it
  - Cannot simply create new tags to mark up proprietary content
- Why is being able to define your own tags important?

XML Is Extensible

- By allowing producers to define their own tags:
  - It is possible for the producers to describe the data they are producing
  - Possible to transmit "self-describing" data
- XML is a standard, but it is extensible
  - Only the markup syntax is fixed
  - The tags allowed (the "grammar") are not fixed
- Because tags are not fixed:
  - Producers and consumers of XML documents have to agree on what each tag means
  - Any data can be represented using XML
    - Not always concisely
    - JPEG, GIF, PNG, etc. may be better suited for large amounts of numerical or image data

XML Is Hierarchical

- In XML, "element" and "tag" are often used interchangeably
  - In this XML document, the project element contains a target element
  - XML is hierarchical
    - Every XML document has only one "root" element

```
<project basedir="." default="build" name="ex1">
   <target name="clean">
        <delete dir="build"/>
   </target>
</project>
```

- Because XML is hierarchical, easy to map XML documents to Java classes
  - Classes: `Project`, `Target`, `Delete`
  - Aggregation (has-a) relationships:
    - `Project` has a field of type `Target`
    - `Target` has a field of type `Delete`
  - If `Project` can have many `Target` fields, use `List<Target>`

Elements, Attributes, Character Data

- The data in an XML document is held mostly in elements
  - One top-level element called the root element
  - The root element of this document is the project element
- An element:
  - Has a tag name; e.g., the element with a tag name "quote"
  - May have attributes within its start tag; e.g., the `asof` attribute of quote
  - May have child elements; e.g., the symbol element is a child of target
  - May have textual content; e.g., "IBM" is the text content of symbol
  - Maybe empty; e.g., the exchange element is empty

```
<quote asof="20070321081423UTC">
   <symbol>IBM</symbol>
   <price>35.23</price>
   <exchange name="NYSE"/>
</quote>
```

XML vs. HTML

- Easy for automated algorithm to find necessary piece of data in XML
  - If document consumer knows the structure of the XML document

```
<quote>
   <symbol>IBM</symbol>
   <price>35.23</price>
</quote>
```

Find the quote element whose symbol element's content is "IBM" and get the content of its price element

- HTML is only a presentation format
  - The consumer is a web browser which knows the HTML standard
  - XHTML is HTML written in strict XML syntax
  - Hard to find necessary piece of data, even if it is displayed on screen

```
<table>
   <tr><td>IBM</td><td><i>35.23</i></td></tr>
   <tr><td>Sun Micro</td><td><b>3.12</b></td></tr>
</table>
```

  - Logic breaks every time page format changes

Find the "tr" whose first "td" has inside it a string named "IBM" or "International Business Machines" and then find the content of next "td", disregarding any "i" or "b" elements but use their text

Chapter Concepts

What Is XML?

The Spring Tool Suite

The Spring Object Factory

Annotation-Based Factory Configuration

Chapter Summary

Spring Tool Suite

- The Spring Tool Suite (STS) is an Eclipse-based development environment that is customized to provide assistance to developers building a Spring-based application
- The STS understands the meaning of the various files in a Spring application
  - It can parse a Spring configuration file and understand the bean definitions
  - It will display detailed information about the Spring managed beans and their dependencies
  - It will validate your Spring project, which will allow it to catch configuration errors at design time instead of at runtime
- The STS provides some Spring-specific refactorings, such as renaming of a Spring managed bean
- Plus many more features
  - See the STS website for more info: https://spring.io/tools/sts

Chapter Concepts

What Is XML?

The Spring Tool Suite

The Spring Object Factory

Annotation-Based Factory Configuration

Chapter Summary

What Is Spring?

The Spring Framework is an open-source application framework and inversion of control container for the Java platform*

- The major focus of Spring is on simplifying application development
- At the core is an object factory—known as the container
- Supplemented by extensive support for application development
  - Data access
  - Web application development—MVC framework
  - Aspect-oriented programming
  - Transaction control
  - Security
  - Batch processing
  - Much more

*Source: Wikipedia

The Spring Object Factory

- Spring provides an object factory
  - Creates and manages lifecycle of application objects
  - Principle is known as *Inversion of Control* (IoC)
    - You hand-over control of object creation to Spring
- Object factory can also perform *Dependency Injection* (DI)
  - Establish links between objects it creates when dependencies exist
- Object factory needs to be configured
  - With which objects to create
  - Which dependencies to establish
- Configuration can be performed with:
  - Annotations or
  - XML

Using Spring's Object Factory

- Consider the following simple plain Java code
  - The `PopupGreeter` class has a dependency on the Visitor interface
    - An interface is used to maintain loose coupling between greeter and visitor

```
public class PopupGreeter implements Greeter {
  private Visitor visitor;

  public Visitor getVisitor() {  return visitor;  }
  public void setVisitor(Visitor visitor) { this.visitor = visitor;  }
  public void greet() {
    JOptionPane.showMessageDialog(null, visitor.getGreeting() + "," +
                                  visitor.getFullName());
  }
}
```

Has dependency on Visitor

```
public interface Greeter {
 void greet();
 void setVisitor(Visitor v);
}
```

```
public interface Visitor {
  String getFullName();
  public String getGreeting();
}
```

Using Spring's Object Factory (continued)

```
public class AmarilloVisitor implements Visitor {
  private String fullName;
  private String greeting;

  public AmarilloVisitor(){ this.greeting = "Howdy"; }

  public String getGreeting() {return greeting;  }

  public String getFullName() { return fullName; }

  public void setFullName(String name) { this.fullName = name;
}
}
```

- To use the `PopupGreeter` class with the `AmarilloVisitor` above, we need to:
  - Create an instance of `PopupGreeter`
  - Create an instance of the `AmarilloVisitor`
    - Call `setVisitor` on `PopupGreeter` passing the instance of `AmarilloVisitor`

Using Spring's Object Factory for the `PopupGreeter`

- Spring's object factory will create the two objects for us – IoC
  - And pass the `AmarilloVisitor` to `setVisitor()` of the `PopupGreeter` – DI
  - Pass a name configured to `setFullName()` of the `AmarilloVisitor` – DI
- To use Spring, we need to:
  - Configure the factory
  - Instantiate the factory and ask it for the `PopupGreeter`
- Springs factory creates objects referred to as Beans
  - JavaBeans are:
    - Plain Java classes
    - Normally have a default constructor (although not necessary for Spring)
    - Provide access to data values via properties
      - Properties are get/set methods

JavaBean Example

- `AmarilloVisitor` is a JavaBean providing the following properties
  - A read and write property called *fullName*
  - A read-only property named greeting
- Property names are defined by the name of the get/set methods
  - Not the data member they access

```
public class AmarilloVisitor implements Visitor {
  private String fullName;
  private String greeting;

  public AmarilloVisitor(){ this.greeting = "Howdy"; }
  public String getGreeting() {return greeting;  }
  public String getFullName() { return fullName; }
  public void setFullName(String name) { this.fullName = name;
}
}
```

Spring Factories: `ApplicationContext`

- Spring provides several factories that may be used by an application
  - All of them implement the `ApplicationContext` interface
  - `org.springframework.context.ApplicationContext`
  - Configuration is provided in XML file(s) or by annotations or both
- Any `ApplicationContext` implementation is capable of:
  - Instantiating a bean (IoC)
  - Injecting its dependencies (DI)
  - Providing access to bean to any code that requires it
- Each factory is its own "container"
  - Manages the lifecycle of the beans that it creates
  - The factory constructs beans as needed
    - Can also dispose of beans that are no longer needed
- Objects created by factory are known as *Spring-managed beans*

ClassPathXmlApplicationContext

- This factory expects the configuration in XML file(s)
  - File(s) should be present somewhere in the classpath:

Should be on the classpath

```
AbstractApplicationContext factory
      = new ClassPathXmlApplicationContext("greeter-beans.xml");

// Use factory

factory.close();
```

- The `close()` method is not available in the `ApplicationContext` interface
  - So we declare the factory as `AbstractApplicationContext`

Using Beans from Factory

- Beans are provided by the `getBean` method on `factory`
  - An id/name is provided for each bean when configured
    - Id is used when asking for the bean from the factory

```
AbstractApplicationContext factory =
     new ClassPathXmlApplicationContext("greeter-beans.xml");

Greeter greeter = factory.getBean("greeter", Greeter.class);

greeter.greet();

factory.close();
```

Use bean

Id of bean requested

Interface bean should implement

XML Configuration File

- XML configuration file has `<beans>` root element
- Every bean to be created is configured using `<bean>` element
  - Id is provided to enable access by clients requesting bean from factory
  - Fully qualified class name so Spring knows which type of object to create

```
<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.xsd">

   <bean id="vis" class="com.masteringspring.greeter.AmarilloVisitor”/>

   <bean id="greeter” class="com.masteringspring.greeter.PopupGreeter”/>

</beans>
```

Id of bean when requested from factory

Configuring Dependency Injection in XML

- Spring will automatically call setter methods on properties if configured
- Values passed to properties can be:
  - Other Spring created objects(beans)
  - Absolute values configured in XML

```
<beans …>
  <bean id="vis" class="com.masteringspring.greeter.AmarilloVisitor">
    <property name="fullName" value="Joe Bob Springstein"/>
  </bean>
  <bean id="greeter" class="com.masteringspring.greeter.PopupGreeter">
    <property name="visitor" ref="vis"/>
  </bean>
</beans>
```

Call setFullName on AmarilloVisitor with value as parameter

Call setVisitor on PopupGreeter with bean whose id is vis as parameter

How Many Beans Will Be Created?

- The `getBean` method may:
  - Keep returning the same object each time
    - "singleton"
    - One bean per Spring container (`ApplicationContext` instance)
  - Return a freshly instantiated object each time
    - Uses the XML configuration as a "prototype"
    - Return a fresh object for every new request from the factory
- The number of beans created is controlled by the *scope* configuration of the bean
  - Five scopes are available
    - Singleton and prototype
    - Request and session for web applications
    - Global session for Portlets

Bean Scope Example

- The scope attribute of the <bean> element is used to define the scope
  - If omitted, singleton is the default

```
<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.xsd">

   <bean id="vis" class="com.masteringspring.greeter.AmarilloVisitor"
                 scope="singleton"/>

   <bean id="greeter" class="com.masteringspring.greeter.PopupGreeter"
                 scope="prototype"/>

</beans>
```

New bean per request from factory

Chapter Concepts

What Is XML?

The Spring Tool Suite

The Spring Object Factory

Annotation-Based Factory Configuration

Chapter Summary

Configuration Using Annotations

- Possible to use annotations for configuration rather than XML
- To use annotation-based configuration requires:
  1. Annotating any class(es) that are to be Spring-managed beans
  2. Enable annotation-based configuration in XML configuration file
- `@Component` is used to mark a class as a Spring-managed bean
  - `@Autowired` or `@Inject` is used to indicate dependency injection
  - Both work the same way
- `@Autowired` is Spring-specific
  - `@Inject` is a Java JEE standard annotation
    - Needs JEE jar file in classpath to use this

Annotation Example

- Can use annotations, instead of XML configuration for every class:

Bean Id is value passed to annotation

```
@Component ("greeter")
public class PopupGreeter implements Greeter {
  private Visitor visitor;

  public Visitor getVisitor() { return visitor; }

  @Autowired
  public void setVisitor(Visitor visitor) {  this.visitor = visitor; }

  public void greet() {
    JOptionPane.showMessageDialog(null, visitor.getGreeting() + "," +
                          visitor.getFullName());
  }
}
```

Annotation Configuration in XML

- In the XML configuration file, need to tell Spring to scan the package(s) that contains the bean

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

    <context:annotation-config/>

    <context:component-scan base-package="com.masteringspring.greeterannotations" />

</beans>
```

Scan XML defined classes for annotations

Package to scan for annotated classes

Annotation Stereotypes

- When configuring a class using annotations the following annotations can be used:
  - `@Component`
  - `@Controller`
  - `@Service`
  - `@Repository`
- A Component is a catch-all
  - Controllers are associated with the presentation tier
  - Services are associated with the business tier
  - Repositories are associated with the integration tier

Complete Configuration Using Annotations

- XML can be completely removed from a Spring application
- Spring allows for a class to be annotated with `@Configuration`
  - Creation of objects is completed in code

```
@Configuration
public class AppConfig {
          @Bean
          public Visitor createVisitor(){
                    return new AmarilloVisitor();
          }
          @Bean(name="greeter")
          public Greeter createGreeter(){
                    return new PopupGreeter();
          }
}
```

Annotation-Based Factory Creation

- The Configuration class can be used to create an `ApplicationContext`
  - Use `AnnotationConfigApplicationContext`

```
AbstractApplicationContext factory =
        new AnnotationConfigApplicationContext(AppConfig.class);

Greeter g = factory.getBean("greeter", Greeter.class);

g.greet();

factory.close();
```

Chapter Concepts

What Is XML?

The Spring Tool Suite

The Spring Object Factory

Annotation-Based Factory Configuration

Chapter Summary

Chapter Summary

- Spring provides a general-purpose object factory
  - Factory performs dependency injection when configured to do so
- To use the Spring factory:
  1. Configure the factory using XML or annotations(or both)
  2. Create a `ApplicationContext`
  3. Get beans from the `ApplicationContext`
     - Using the id of the bean and its interface type