Spring Framework Notes

Vaman Deshmukh

Link to this document -
https://docs.google.com/document/d/1gjd0EmZKpZsIGuU7zTre9IZaLqHeqwjyQtUixNDbIVU/edit?usp=sharing

Introduction Spring | Home

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

Why Spring?

Spring makes programming Java quicker, easier, and safer for everybody. Spring's focus on speed, simplicity, and productivity has made it the world's most popular Java framework. Spring framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform.

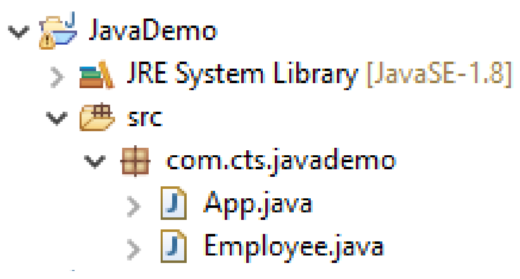
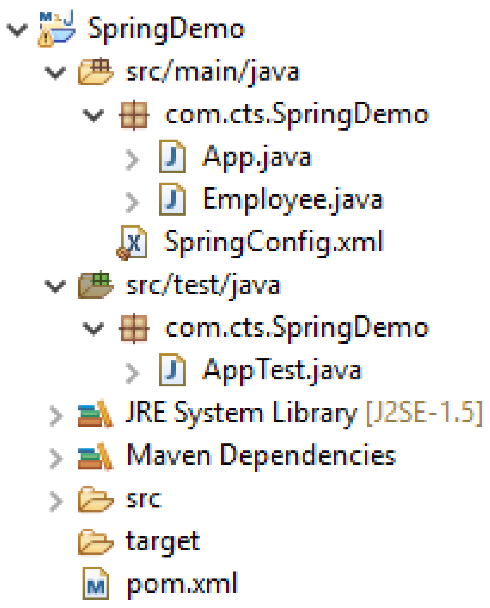
Spring Versions Spring Framework

Version	Year
0.9	2002
1.0	2003
2.0	2006
3.0	2009
4.0	2013
5.0	2017

Advantages of Spring

- Lightweight container
- Modular architecture
- Lazy creation of beans
- Easy Initialization of properties
- Easier unit testing of codebase
- Dependency Injection approach
- Configuration management service
- No special deployment needed
- POJO Programming - continuous integration and testability
- Open source

Java project vs Spring Maven Project

Java Project Structure	Java Spring Maven Project Structure
 JavaDemo JRE System Library [JavaSE-1.8] src com.cts.javademo App.java Employee.java	 SpringDemo src/main/java com.cts.SpringDemo App.java Employee.java SpringConfig.xml src/test/java com.cts.SpringDemo AppTest.java JRE System Library [J2SE-1.5] Maven Dependencies src target pom.xml

Spring Project Configuration Core Technologies

A spring project is basically a Java project which needs to be configured to take its full effect.

Spring project can be configured in many ways -

- XML based configuration (Separate XML file(s))
- Java based configuration (No XML needed)
- Annotation based configuration (No XML needed)

Are annotations better than XML for configuring Spring?
https://docs.spring.io/spring/docs/5.2.7.RELEASE/spring-framework-reference/core.html#beans-annotation-config

The following concepts have been discussed using xml based configuration. Same discussion is applicable to annotation based configuration as well as Java based configuration.

- Dependency Injection (IoC - Inversion of Control)
- Setter Injection (property injection)
- Constructor Injection
- Autowiring
- Injecting Collections
- Inner Beans
- Inheritance, Interfaces, Scope

Example code for xml based spring project:

Create a maven project in eclipse as follows:

pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>

      <groupId>com.vaman.spring.demo</groupId>
      <artifactId>spring-framework-project</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <packaging>jar</packaging>
      <name>spring-framework-project</name>
      <url>http://maven.apache.org</url>

      <properties>
            <java.version>1.8</java.version>
            <maven.compiler.source>1.8</maven.compiler.source>
            <maven.compiler.target>1.8</maven.compiler.target>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      </properties>

      <dependencies>
            <dependency>
                  <groupId>org.springframework</groupId>
                  <artifactId>spring-context</artifactId>
                  <version>5.2.12.RELEASE</version>
            </dependency>
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>3.8.1</version>
                  <scope>test</scope>
            </dependency>
      </dependencies>
</project>
```

src/main/java/SpringConfig.xml

```
<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-4.0.xsd">

<bean id="employee" class="com.vaman.spring.demo.Employee">
</bean>

</beans>
```

Employee.java

```
package com.vaman.spring.demo;

public class Employee {

	public void work() {
		System.out.println("Employee works.");
	}
}
```

App.java

```java
package com.vaman.spring.demo;

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

public class App {

	public static void main(String[] args) {

//		Employee obj = new Employee();
//		obj.work();

		ApplicationContext context = new
		ClassPathXmlApplicationContext("SpringConfig.xml");
		Employee emp = context.getBean("employee", Employee.class);
		emp.work();
	}
}
```

Output

```
Jun 01, 2020 8:22:56 PM org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@4566e5bd: startup date [Mon Jun 01 20:22:56 IST 2020];
root of context hierarchy
Jun 01, 2020 8:22:57 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [SpringConfig.xml]
Employee works.
Jun 01, 2020 8:22:57 PM org.springframework.context.support.ClassPathXmlApplicationContext doClose
INFO: Closing org.springframework.context.support.ClassPathXmlApplicationContext@4566e5bd: startup date [Mon Jun 01 20:22:56 IST 2020];
root of context hierarchy
```

<Code analysis>

<Beans>

<Bean and object>

<Syntax>

<xml>

<Java code>

Dependency injection

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container "injects" objects into other objects or "dependencies". Simply put, **this allows for loose coupling of components** and moves the responsibility of managing components onto the container.

Dependencies can be injected for

- Primitive data values
- String data values
- Object values (Inner beans)
- Collection values
- Other types of values

Dependency Injection can be achieved in **two ways** in Spring framework:

1. By Setter method (Setter injection) **(property injection)***
2. By Constructor (**Constructor injection**)

Setter injection (property injection)

Dependency can be injected by using **<property>** subelement of **<bean>** element.

Syntax

```
<bean id="objectName" class="fully.qualified.ClassName">
      <property name="field1">
           <value>Value1</value>
      </property>
      <property name="fieldN">
           <value>ValueN</value>
      </property>
</bean>
```

Example

Following is an example of injecting values using **property injection** for dependency injection. Modify the code as follows:

Employee.java

```java
package com.vaman.spring.demo;

public class Employee {

	private int id;
	private String name;
	private double salary;

	// constructors, getters, setters, toString etc
 }
```

App.java

```java
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

	public static void main(String[] args) {

	  ApplicationContext context = new
	  ClassPathXmlApplicationContext("SpringConfig.xml");

	  Employee emp = context.getBean("employee", Employee.class);
	  System.out.println(emp.toString());

	  ((AbstractApplicationContext) context).close();
	}
}
```

SpringConfig.xml

```xml
<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-4.0.xsd">

     <!-- property injection / setter injection -->
     <bean id="employee" class="com.vaman.spring.demo.Employee">
          <property name="id">
               <value>101</value>
          </property>
          <property name="name">
               <value>Sonu</value>
          </property>
          <property name="salary">
               <value>10.5</value>
          </property>
     </bean>
</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5]
```

Constructor injection

Dependency can also be injected by a constructor using **<constructor-arg>** subelement of **<bean>**.

Syntax

```
 <bean id="objectName" class="fully.qualified.ClassName">
<constructor-arg value="value1" type="datatype1">
</constructor-arg>
</bean>
```

Example

Following is an example of injecting values using **constructor injection** for dependency injection. Modify the code as follows:

App.java

```
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

      public static void main(String[] args) {

        ApplicationContext context = new
        ClassPathXmlApplicationContext("SpringConfig.xml");

        Employee emp = context.getBean("employee", Employee.class);
        System.out.println(emp.toString());

        Employee emp2 = context.getBean("employee2", Employee.class);
        System.out.println(emp2.toString());

        ((AbstractApplicationContext) context).close();
      }
}
```

SpringConfig.xml

```
<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-4.0.xsd">

      <!-- property injection / setter injection -->
      <bean id="employee" class="com.vaman.spring.demo.Employee">
            <property name="id">
                  <value>101</value>
            </property>
            <property name="name">
                  <value>Sonu</value>
            </property>
            <property name="salary">
                  <value>10.5</value>
            </property>
      </bean>

      <!-- constructor injection -->
      <bean id="employee2" class="com.vaman.spring.demo.Employee">
            <constructor-arg value="102" type="int"></constructor-arg>
            <constructor-arg value="Monu"
type="java.lang.String"></constructor-arg>
            <constructor-arg value="20.6"
type="double"></constructor-arg>
      </bean>
</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5]
Employee [id=101, name=Sonu, salary=10.5]
```

Injecting Collections

Java collection values can be injected using spring framework **both** by setter injection and constructor injection. Each of the collections can contain either String or on-String values.

Following three of Java collection values can be inserted:

1. <list>
2. <set>
3. <map>
4. Properties <props>

Add/ replace the code as follows:

Department.java

```
package com.vaman.spring.demo;

import java.util.List;

public class Department {

	private int id;
	private String name;
	private List<String> functions;

	// constructors, getters, setters, toString etc
}
```

App.java

```
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

	public static void main(String[] args) {

	  ApplicationContext context = new
	  ClassPathXmlApplicationContext("SpringConfig.xml");
	  Employee emp = context.getBean("employee", Employee.class);
	  System.out.println(emp.toString());
	  Employee emp2 = context.getBean("employee2", Employee.class);
	  System.out.println(emp2.toString());

   Department dept = context.getBean("department", Department.class);
   System.out.println(dept.toString());
	  ((AbstractApplicationContext) context).close();
	}
}
```

SpringConfig.xml

```xml
<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-4.0.xsd">
       <!-- property injection / setter injection -->
       <bean id="employee" class="com.vaman.spring.demo.Employee">
              <property name="id"><value>101</value>/property>
              <property name="name"><value>Sonu</value></property>
              <property name="salary"><value>10.5</value></property>
       </bean>
       <!-- constructor injection -->
       <bean id="employee2" class="com.vaman.spring.demo.Employee">
              <constructor-arg value="102" type="int"></constructor-arg>
              <constructor-arg value="Monu"
type="java.lang.String"></constructor-arg>
              <constructor-arg value="20.6" type="double"></constructor-arg>
       </bean>
<!-- collection injection -->
       <bean id="department" class="com.vaman.spring.demo.Department">
              <property name="id" value="10"></property>
              <property name="name" value="HR"></property>
              <property name="functions">
                     <list>
                            <value>HRM</value>
                            <value>Payroll</value>
                            <value>CSR</value>
                            <value>EE</value>
                     </list>
              </property>
       </bean>
</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5, dept=null]
Employee [id=102, name=Monu, salary=20.6, dept=null]
Department [id=10, name=HR, functions=[HRM, Payroll, CSR, EE]]
```

Inner Beans
Inner beans are the beans that are defined within the scope of another bean.

Syntactically, a **<bean/>** element inside the **<property/>** or **<constructor-arg/>** elements is called an inner bean.

Example:
Add / replace the following code.

App.java

```
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

     public static void main(String[] args) {

       ApplicationContext context = new
       ClassPathXmlApplicationContext("SpringConfig.xml");

       Employee emp = context.getBean("employee", Employee.class);
       System.out.println(emp.toString());
       Employee emp2 = context.getBean("employee2", Employee.class);
       System.out.println(emp2.toString());
   Department dept = context.getBean("department", Department.class);
   System.out.println(dept.toString());
       Employee emp3 = context.getBean("employee3", Employee.class);
       System.out.println(emp3.toString());

       ((AbstractApplicationContext) context).close();
     }
}
```

SpringConfig.xml

```
<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-4.0.xsd">
       <!-- property injection / setter injection -->
       <!-- constructor injection -->
       <!-- collection injection -->
       <!-- inner beans -->
       <bean id="employee3" class="com.vaman.spring.demo.Employee">
              <property name="id"><value>103</value></property>
              <property name="name"><value>Tonu</value></property>
              <property name="salary"><value>15.8</value></property>
              <property name="dept">
              <bean id="department"
class="com.vaman.spring.demo.Department">
                     <property name="id" value="20"></property>
                     <property name="name" value="Marketing"></property>
                            <property name="functions">
                                   <list>
                                          <value>Sales</value>
                                          <value>Promotions</value>
                                          <value>Customer Care</value>
                                   </list>
                            </property>
                     </bean>
              </property>
       </bean>
</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5, dept=null]
Employee [id=102, name=Monu, salary=20.6, dept=null]
Department [id=10, name=HR, functions=[HRM, Payroll, CSR, EE]]
Employee [id=103, name=Tonu, salary=15.8, dept=Department [id=20,
name=Marketing, functions=[Sales, Promotions, Customer Care]]]
```

Autowiring

Autowiring is used to inject object dependency **implicitly** in Java Spring applications.

- It internally uses constructor injection or setter injection (property injection).
- Autowiring is used to inject object reference.
- It can not be used to inject primitive and string type values.
- It needs less code to inject dependency.

Modes in Autowiring

Mode	**Usage**
no	default autowiring mode; no autowiring
byName	Injects the object dependency using the name of the bean property name same as bean name Internally calls setter method
byType	Injects the object dependency using the type property name same / different as bean name Only one bean of the type is allowed Internally calls setter method
constructor	Injects the dependency by calling the constructor

Example

Add / replace the code as follows:

App.java

```
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

     public static void main(String[] args) {

       ApplicationContext context = new
       ClassPathXmlApplicationContext("SpringConfig.xml");

       Employee emp = context.getBean("employee", Employee.class);
       System.out.println(emp.toString());
       Employee emp2 = context.getBean("employee2", Employee.class);
       System.out.println(emp2.toString());
   Department dept = context.getBean("department", Department.class);
   System.out.println(dept.toString());
       Employee emp3 = context.getBean("employee3", Employee.class);
       System.out.println(emp3.toString());

       Employee emp4 = context.getBean("employee4", Employee.class);
       System.out.println(emp4.toString());

       ((AbstractApplicationContext) context).close();
     }
}
```

SpringConfig.xml

```xml
<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-4.0.xsd">
	<!-- property injection / setter injection -->

	<!-- constructor injection -->

	<!-- collection injection -->

	<!-- inner beans -->

		<!-- autowiring -->
	<bean id="employee4"
	class="com.vaman.spring.demo.Employee" autowire="byName">
		<property name="id">
			<value>104</value>
		</property>
		<property name="name">
			<value>Ponu</value>
		</property>
		<property name="salary">
			<value>22.0</value>
		</property>
	</bean>
</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5, dept=null]
Employee [id=102, name=Monu, salary=20.6, dept=null]
Department [id=10, name=HR, functions=[HRM, Payroll, CSR, EE]]
Employee [id=103, name=Tonu, salary=15.8, dept=Department [id=20,
name=Marketing, functions=[Sales, Promotions, Customer Care]]]
Employee [id=104, name=Ponu, salary=22.0, dept=null]
```

Other modes

byType

```
	<bean id="department" class="com.vaman.spring.demo.Department">
	</bean>
	<bean id="employee5" class="com.vaman.spring.demo.Employee"
	autowire="byType"></bean>
```

constructor

```
	<bean id="department" class="com.vaman.spring.demo.Department">
	</bean>
	<bean id="employee6" class="com.vaman.spring.demo.Employee"
	autowire="constructor"></bean>
```

no

```
	<bean id="department" class="com.vaman.spring.demo.Department">
	</bean>
	<bean id="employee7" class="com.vaman.spring.demo.Employee"
	autowire="no"></bean>
```

<default>

```
	<bean id="department" class="com.vaman.spring.demo.Department">
	</bean>
	<bean id="employee8" class="com.vaman.spring.demo.Employee"
	autowire="default"></bean>
```

Scope of Bean

Bean definition can optionally contain scope of the bean. Scope of the bean is the context in which the bean is accessed.

Spring supports following bean scopes.

Bean scope	**Description**
singleton	Provides single instance of the bean per container <default>
prototype	Provides any number of instances of the bean per container
request	Provides the bean to an HTTP request
session	Provides the bean to an HTTP session
global-session	Provides the bean to a global HTTP session

Note:

prototype scope is used for ***stateful beans*** and singleton scope is used for ***stateless beans***.

Syntax

```
      <bean id="beanName" class="fully.qualified.ClassName"
      scope="scopeValue"></bean>
```

Add / replace the following code.

```java
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

      public static void main(String[] args) {

        ApplicationContext context = new
        ClassPathXmlApplicationContext("SpringConfig.xml");

        Employee emp = context.getBean("employee", Employee.class);
        Employee emp2 = context.getBean("employee", Employee.class);

        System.out.println(emp.toString());
        System.out.println(emp2.toString());

        emp.setName("Monu");
        emp.setId(102);

        System.out.println(emp.toString());
        System.out.println(emp2.toString());

        ((AbstractApplicationContext) context).close();
      }
}
```

SpringConfig.xml

```
<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-4.0.xsd">

      <!-- scope of a bean -->

      <bean id="employee"
      class="com.vaman.spring.demo.Employee" scope="singleton">
            <property name="id">
                  <value>101</value>
            </property>
            <property name="name">
                  <value>Sonu</value>
            </property>
            <property name="salary">
                  <value>10.5</value>
            </property>
      </bean>

</beans>
```

Output

```
Employee [id=101, name=Sonu, salary=10.5, dept=null]
Employee [id=101, name=Sonu, salary=10.5, dept=null]
Employee [id=102, name=Monu, salary=10.5, dept=null]
Employee [id=102, name=Monu, salary=10.5, dept=null]
```

Inheritance in beans

Beans can inherit other beans in Spring.

- Bean contains property values, constructor arguments and other data.
- This data can be inherited by other beans.
- The child bean can refer to the parent bean.
- Child bean can override values or can add its own values to the properties.
- This is similar to **class-to-class** inheritance in Java.

Syntax

```
<bean id="parentBean" class="fully.qualified.ClassName">
      <property name="field1" value="value1"></property>
</bean>
<bean id="childBean" parent="parentBean">
      <property name="field2" value="value2"/>
      <property name="fieldN" value="valueN"></property>
</bean>
```

Example

Add / replace the code as follows:

Person.java

```
package com.vaman.spring.demo;

public interface Person {

	public abstract void eat();

}
```

Employee.java

```
package com.vaman.spring.demo;

public class Employee implements Person {

	private int id;
	private String name;
	private double salary;

	 @Override
	public void eat() {
		System.out.println("Employee eats...");
	}

	// constructors, getters, setters, toString etc
 }
```

App.java

```
package com.vaman.spring.demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

	public static void main(String[] args) {

		ApplicationContext context = new
		ClassPathXmlApplicationContext("SpringConfig2.xml");

	  Person person = context.getBean("employee", Employee.class);
	  person.eat();

		((AbstractApplicationContext) context).close();
	}
}
```

Output

```
Employee eats...
```

Java based configuration
Spring applications can also be configured based on a java class.

Add / modify the code as follows.

App.java

```
package com.vaman.spring.jv;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

/**
 *
 * @author Vaman Deshmukh
 *
 */

public class App {

	public static void main(String[] args) {
		System.out.println("Start");
		ApplicationContext ctx = new
		AnnotationConfigApplicationContext(SpringConfig.class);

		Employee emp = ctx.getBean(Employee.class);
		emp.work();

		((AbstractApplicationContext) ctx).close();
	}
}
```

SpringConfig.java

```
package com.vaman.spring.jv;

import org.springframework.context.annotation.Bean;

public class SpringConfig {

	@Bean
	public Employee employee() {
		System.out.println("Employee bean");
		return new Employee();
	}
}
```

Output

```
Start
Employee bean
Employee works...
```

Annotation based configuration

Spring applications can also be configured based on annotations.

Add / edit the code as follows:

Employee.java

```java
package com.vaman.spring.ano;

import org.springframework.stereotype.Component;

@Component
public class Employee {
	public void work() {
		System.out.println("Employee works...");
	}
}
```

App.java

```java
package com.vaman.spring.ano;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan
public class App {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext ctx = new
		AnnotationConfigApplicationContext(App.class);
		Employee emp = ctx.getBean(Employee.class);
		emp.work();
	}
}
```

Output

```
	Employee works...
```

<Code analysis>

@Autowired

Beans can be injected using @Autowired annotation.

Add / edit the code as follows:

Department.java

```
package com.vaman.spring.ano;

import org.springframework.stereotype.Component;

@Component
public class Department {

	public void work() {
		System.out.println("Employee works in department...");
	}
}
```

Employee.java

```java
package com.vaman.spring.ano;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Employee {

//	@Autowired
	private Department department;

//	public Employee() {
//		super();
//		System.out.println("default constructor");
//	}

//	@Autowired
	public Employee(Department department) {
		super();
		System.out.println("parameterized constructor");
		this.department = department;
	}

	public Department getDepartment() {
		return department;
	}

//	@Autowired
	public void setDepartment(Department department) {
		System.out.println("setter method");
		this.department = department;
	}
}
```

App.java

```
package com.vaman.spring.ano;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan
public class App {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext ctx = new
		AnnotationConfigApplicationContext(App.class);
		Employee emp = ctx.getBean(Employee.class);
		emp.getDepartment.work();
	}
}
```

Output

```
	Employee works in the department...
```

Basic Concepts
https://docs.spring.io/spring/docs/5.2.7.RELEASE/spring-framework-reference/core.html#beans-java-basic-concepts

Annotation / Java based configuration in Spring applications used annotations on elements like classes, methods, fields, constructors etc in the Java code.

@Bean
It is used for a **method** which creates a new object to be managed by the Spring IoC container.

Following two pieces of code serve the same purpose.

Config.java

```java
@Configuration
public class SpringConfig {

	@Bean
	public Employee employee() {
		return new Employee();
	}
}
```

SpringConfig.xml

```xml
 <bean id="employee" class="com.vamandeshmukh.demo.spring.Employee">
```

The **@Bean** annotation plays the same role as the **<bean/>** element. The @Bean-annotated methods can be used with any Spring @Component, but they are primarily used with @Configuration beans.

@Bean method can also be declared with an interface (or base class) return type.

@Configuration
It is used for a **class** which, primarily, is a source of bean definitions. This class allows **inter-bean dependencies** to be defined by calling other @Bean methods in the same class.

@ComponentScan
This annotation enables component scanning. It is used for the class which uses the beans.

@Component
This annotation is used for the class for which beans are created by spring.

@Autowired
It is used to indicate that autowiring is required. It is applied to fields, constructors and methods.

@Scope
This annotation defines the scope of a bean. Its **value** can be e.g. singleton or prototype.

Annotations for Bean Lifecycle Methods

@PostConstruct

@PreDestroy

AnnotationConfigApplicationContext AnnotationConfigApplicationContext

- This class creates a new AnnotationConfigApplicationContext bean (object).
- It gets bean definitions from the component classes.
- It accepts @Configuration and @Component classes.
- It works the same as SpringConfig.xml.

Note: When @Configuration classes are provided as input, the @Configuration class itself is registered as a bean definition and all declared @Bean methods within the class are also registered as bean definitions.

AnnotationConfigApplicationContext bean can be created using different constructors.

register() method
The AnnotationConfigApplicationContext bean can **also** be created by using the **default constructor.** This bean is configured by using the register() method.

App.java

```
@Configuration
@ComponentScan
public class App {
	public static void main(String[] args) {
		AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext();
		ctx.register(Config.class);
		ctx.refresh();
		Employee emp = ctx.getBean(Employee.class);
		emp.work();
	}
}
```

AnnotationConfigWebApplicationContext AnnotationConfigWebApplicationContext
This class is used for configuring the Spring ContextLoaderListener servlet listener, Spring MVC DispatcherServlet, and so forth.

Code repository https://github.com/vamandeshmukh/spring-framework-project

- - - end - - -