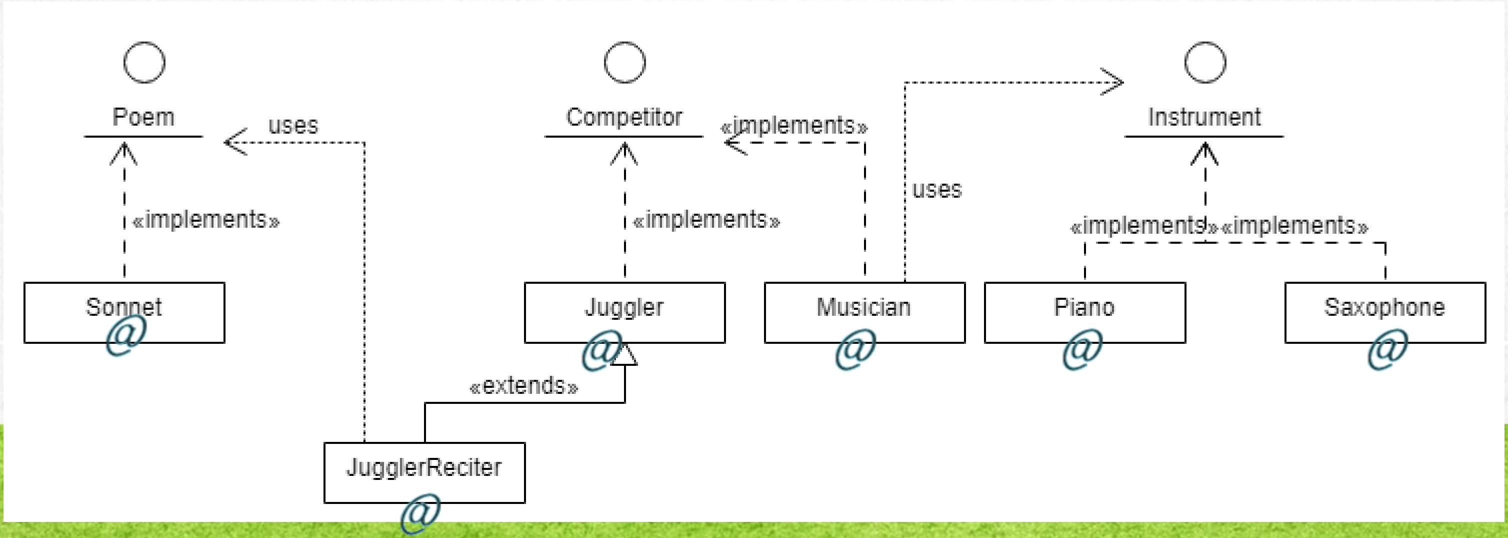
EXERCISE

TALENT CONTEST V4 WITH SPRING FRAMEWORK

EXERCISE OBJECTIVE

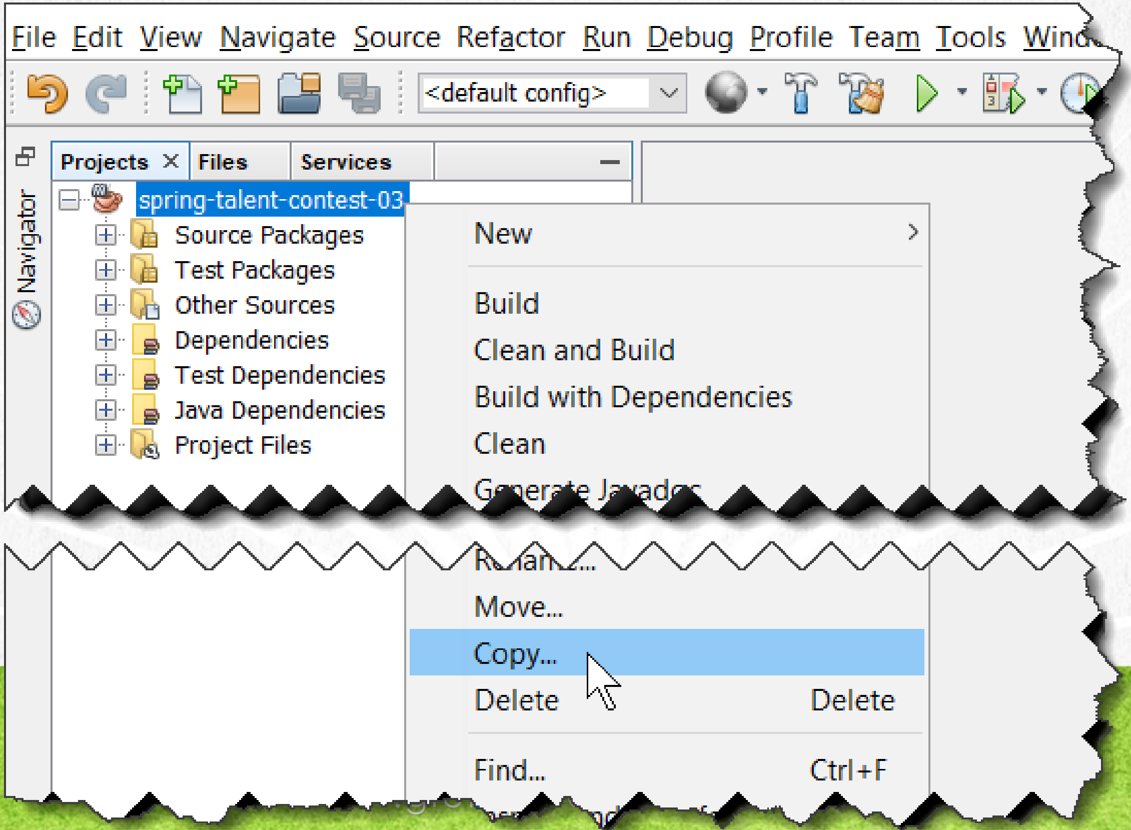
•The objective of the exercise is to modify the Talent Contest project to put into practice the concept of the Use of Annotations with Spring.

•At the end we must have the Talent Contest Project with the following classes, but each class using annotations.



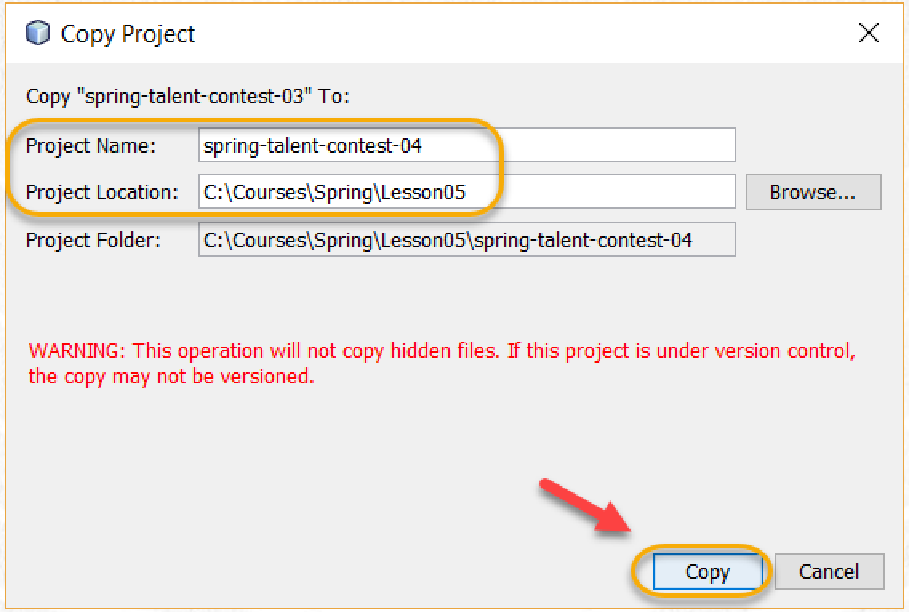
1. COPY THE PROJECT

Copy the Project spring-talent-contest-03:



1. COPY THE PROJECT

Change the name to spring-talent-contest-04:



Copy "spring-talent-contest-03" To:

Project Name: spring-talent-contest-04

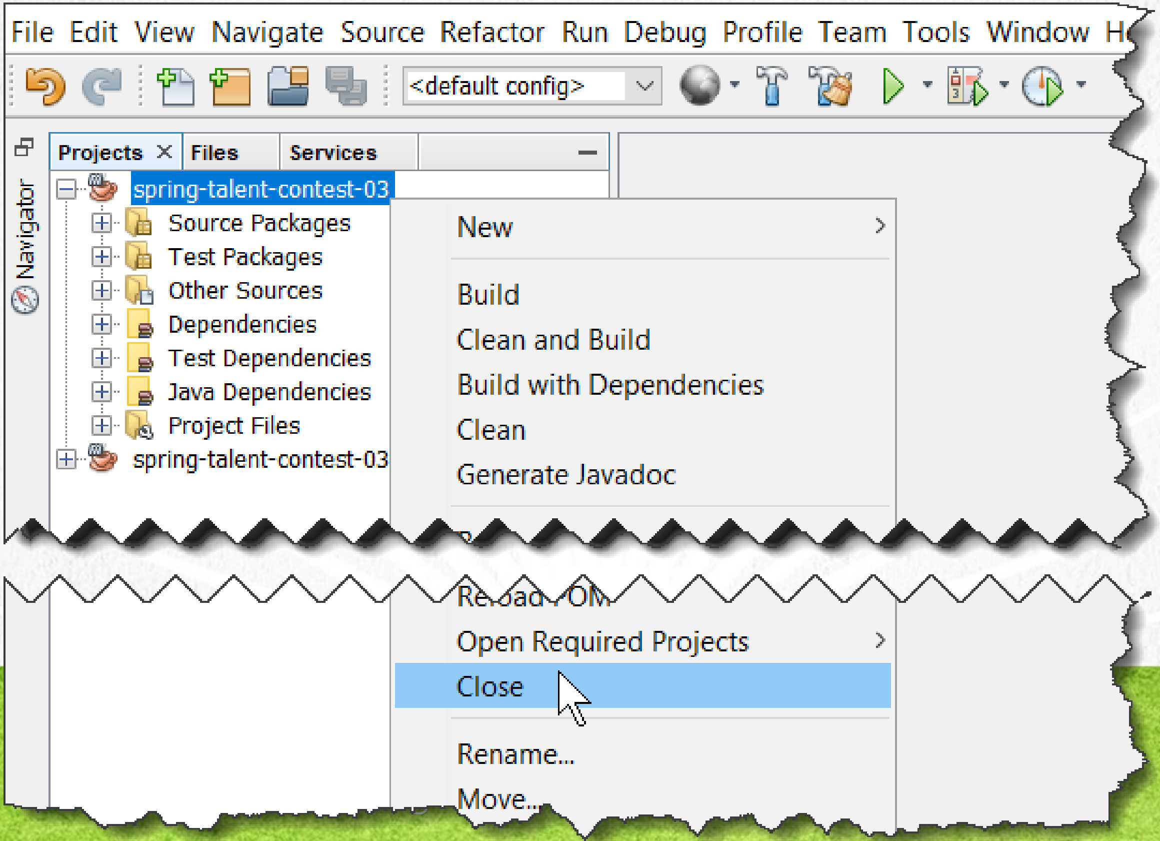
Project Location: C:\Courses\Spring\Lesson05

Project Folder: C:\Courses\Spring\Lesson05\spring-talent-contest-04

WARNING: This operation will not copy hidden files. If this project is under version control, the copy may not be versioned.

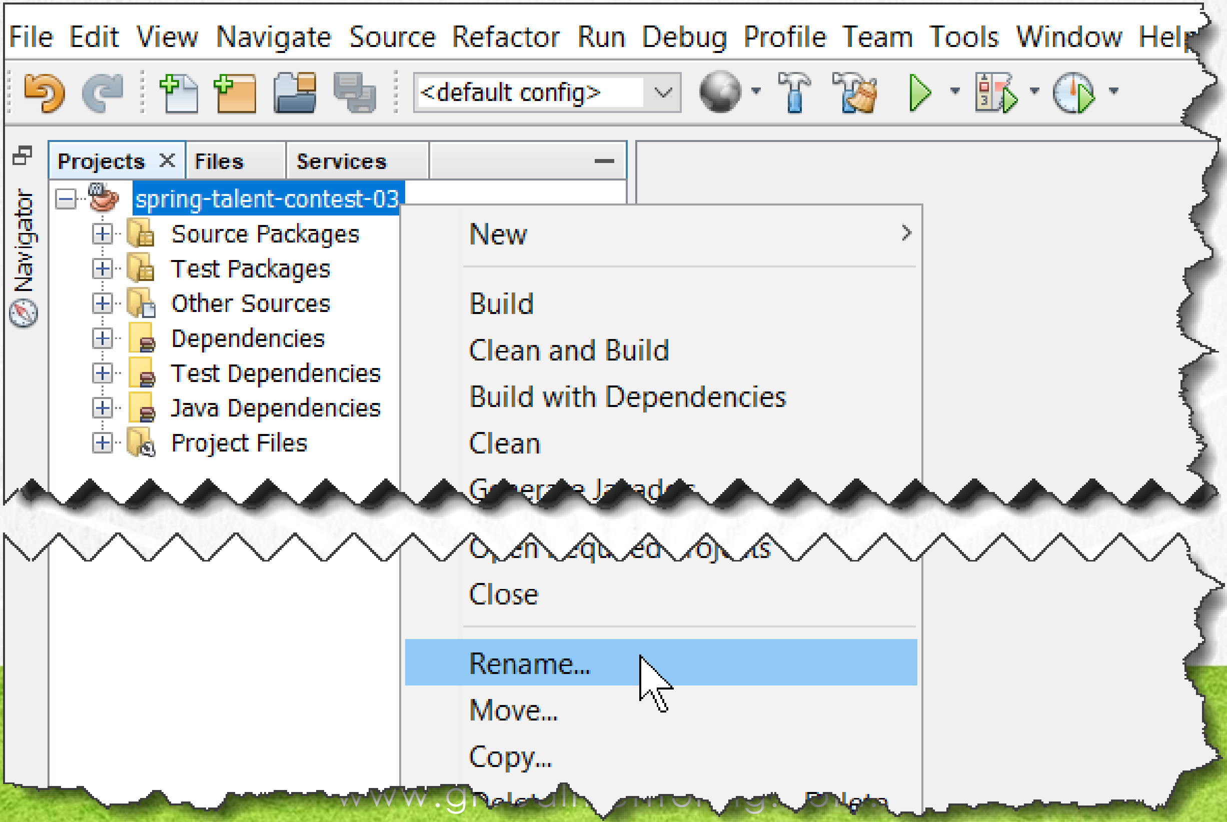
2. CLOSE THE PROJECT

Close the previous Project and keep open the new one:



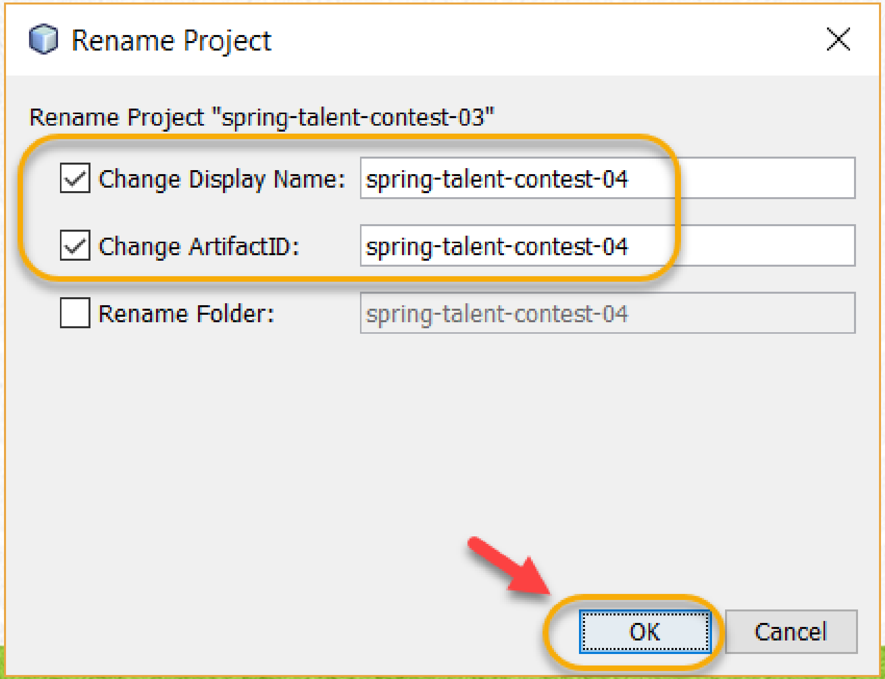
3. RENAME THE PROJECT

Rename the Project to spring-talent-contest-04:



3. RENAME THE PROJECT

Rename the Project to spring-talent-contest-04:



Rename Project "spring-talent-contest-03"

[✓] Change Display Name: spring-talent-contest-04

[✓] Change ArtifactID: spring-talent-contest-04

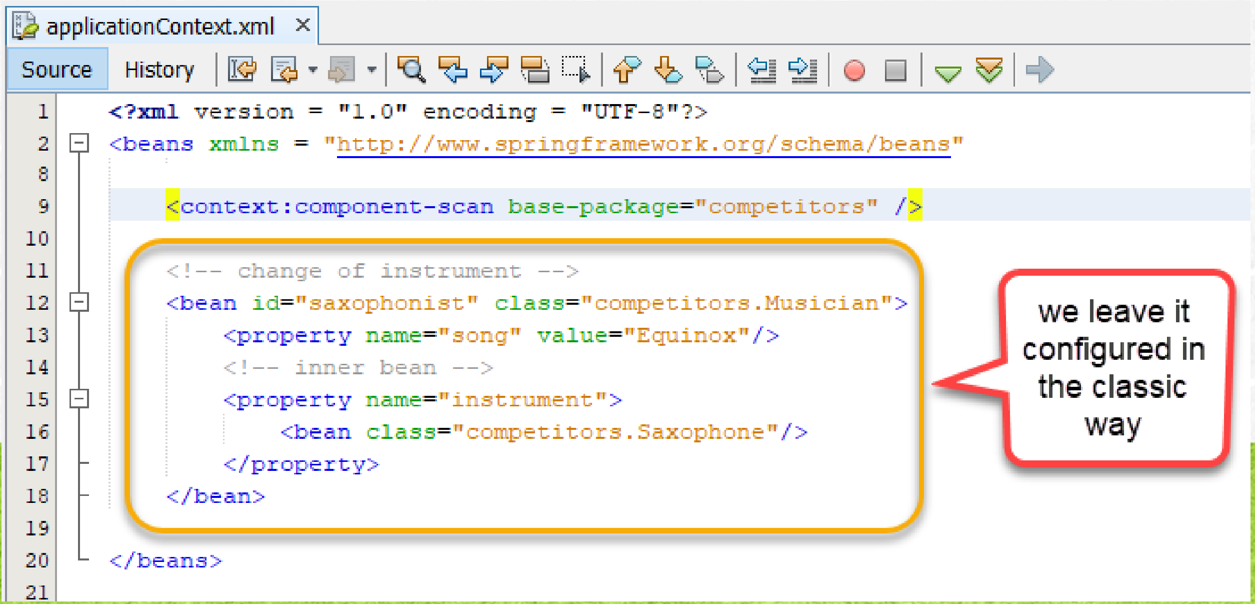
[ ] Rename Folder: spring-talent-contest-04

4. MODIFY THE APPLICATIONCONTEXT.XML FILE

We must put between comments each of the beans (except the bean of "saxophonist") defined in the applicationContext file, and add the definition of autodiscovering:

```
<context:component-scan base-package="competitors" />
```

We also add the context name space in the beginning of the file. The result is similar to the following :



```
<?xml version = "1.0" encoding = "UTF-8"?>
<beans xmlns = "http://www.springframework.org/schema/beans"

    <context:component-scan base-package="competitors" />

    <!-- change of instrument -->
    <bean id="saxophonist" class="competitors.Musician">
        <property name="song" value="Equinox"/>
        <!-- inner bean -->
        <property name="instrument">
            <bean class="competitors.Saxophone"/>
        </property>
    </bean>

</beans>
```

5. MODIFY THE FILE

applicationContext.xml:

Click to download

```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"
       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:component-scan base-package="competitors" />

    <!-- change of instrument -->
    <bean id="saxophonist" class="competitors.Musician">
        <property name="song" value="Equinox"/>
        <!-- inner bean -->
        <property name="instrument">
            <bean class="competitors.Saxophone"/>
        </property>
    </bean>

</beans>
```

6. Modify the Juggler.java file

Let's start the modifications for the autodiscovering of each of the components using Spring. In this first Juggler class, we will add the @Component annotation, which will help us to avoid the configuration we had in the applicationContext.xml file.

The name of the bean will be the same as that of the class starting with lowercase letter, in this case: juggler. If we want the name of the bean to be different, we have to write the string of the name. Ex. @Component ("myBean")

6. MODIFY THE FILE

Juggler.java:

Click to download

```
package competitors;

import org.springframework.stereotype.Component;

@Component
public class Juggler implements Competitor {

    private int balls = 10;

    public Juggler() {
    }

    public Juggler(int balls) {
        this.balls = balls;
    }

    @Override
    public void execute() throws ExecutionException {
        System.out.println("juggling " + this.balls + " balls");
    }

    public int getBalls() {
        return balls;
    }

    public void setBalls(int balls) {
        this.balls = balls;
    }
}
```

7. MODIFY THE SONNET.JAVA FILE

Let's now modify the Sonnet class. This class if we are going to modify the name to be called reciter, so we add the component annotation but with the name of "reciter".

```java
@Component("reciter")
public class Sonnet implements Poem{
```

Let's see how our class is:

7. MODIFY THE FILE

Sonnet.java:

Click to download

```java
package competitors;

import org.springframework.stereotype.Component;

@Component("reciter")
public class Sonnet implements Poem{

    @Override
    public void recite(){
        String sonnet = "A thing of beauty is a joy forever.\n" +
                        "Its loveliness increases; it will never\n" +
                        "pass into nothingness ...\n";

        System.out.println("\nSonnet:" + sonnet);
    }
}
```

8. MODIFY THE JUGGLERRECITER.JAVA FILE

In this JugglerReciter class, in addition to adding the component annotation, we will begin to inject the references that we have created.

The dependency injection that we are going to use will be via the class constructor. So we are going to inject the value of 15 into the argument of balls, and then automatically bean the bean whose type is Poem, that is the Sonnet bean, this is done automatically because it is the only bean of this guy in the whole Spring factory.

If there were more beans of type Poem, we would have to indicate which is the bean that we want to inject. We will see later an example of how to indicate the bean that interests us in case there is more than one.

8.MODIFY THE FILE

JugglerReciter.java:

Click to download

```java
package competitors;

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

@Component
public class JugglerReciter extends Juggler{

    private Poem poem;

    public JugglerReciter(Poem poem) {
        super();
        this.poem = poem;
    }

    //Automatic injection by constructor, we inject primitive values and references
    @Autowired
    public JugglerReciter(@Value("15") int balls, Poem poem) {
        super(balls);
        this.poem = poem;
    }
```

8.MODIFY THE FILE

JugglerReciter.java:

Click to download

```
    @Override
    public void execute() throws ExecutionException {
        super.execute();
        System.out.println("while reciting...");
        poem.recite();
        System.out.println("Ends recitation...");
    }
}
```

9. MODIFY THE FILE

Modify the Piano.java class as follows. Just add the @Component annotation at the begginning of the class:

9. MODIFY THE CODE

Piano.java:

Dar click para ir al código

```java
package competitors;

import org.springframework.stereotype.Component;

@Component
public class Piano implements Instrument {

    @Override
    public void play() {
        System.out.println("Clin clin clin clin...");
    }
}
```

10. MODIFY THE FILE

Modify the Saxophone.java class as follows. Just add the @Component annotation at the begginning of the class:

10. MODIFY THE FILE

Saxofon.java:

Click to download

```java
package competitors;

import org.springframework.stereotype.Component;

@Component
public class Saxophone implements Instrument{

    @Override
    public void play() {
        System.out.println("Tuu tuu tuu tuu...");
    }
}
```

11. MODIFY THE FILE

Modify the Musician.java class as follows.

Add the @Component annotation at the begginning of the class.

In addition to the definition of the component, in this case we will perform the injection of dependencies using the set methods of the properties.

On the one hand we use the annotation of value to inject the value of a String, and on the other hand we inject the instrument value, but in this case because there is more than one object of type Instrument (piano and saxophone) it is necessary to specify which bean we want to use, for this we use the Qualifier annotation, and we indicate the name of the bean we want to use.

11. MODIFY THE CODE

Musician.java:

Click to download

```java
package competitors;

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

@Component("pianist")
public class Musician implements Competitor {

    //Inyeccion de valor primitivo (podemos eliminar el metodo set)
    @Value("Silent Night")
    private String song;

    //Inyeccion automatica por propiedad (podemos eliminar el metodo set)
    @Autowired
    @Qualifier("piano")
    private Instrument instrument;

    public Musician() {
    }

    @Override
    public void execute() throws ExecutionException {
        System.out.println("Playing " + song + ": ");
        instrument.play();
    }
```

11. MODIFY THE CODE

Musician.java:

Click to download

```
    public String getSong() {
        return song;
    }

    public void setSong(String song) {
        this.song = song;
    }

    public Instrument getInstrument() {
        return instrument;
    }

    public void setInstrument(Instrument instrument) {
        this.instrument = instrument;
    }
}
```

12. MODIFY THE FILE

Finally we are going to modify the pom.xml file. In this case, in order to perform the tests using Spring, it is necessary to add the spring-test dependency.

12. MODIFY THE FILE

pom.xml:

Click to download

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>beans</groupId>
    <artifactId>spring-talent-contest-04</artifactId>
    <version>1</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <spring.version>5.1.0.RELEASE</spring.version>
        <log4j.version>2.11.1</log4j.version>
        <junit.version>5.3.1</junit.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
```

12. MODIFY THE FILE

pom.xml:

Click to download

```
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>${log4j.version}</version>
         </dependency>
         <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>${log4j.version}</version>
         </dependency>
         <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
         </dependency>
         <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
         </dependency>
```

12. MODIFY THE FILE

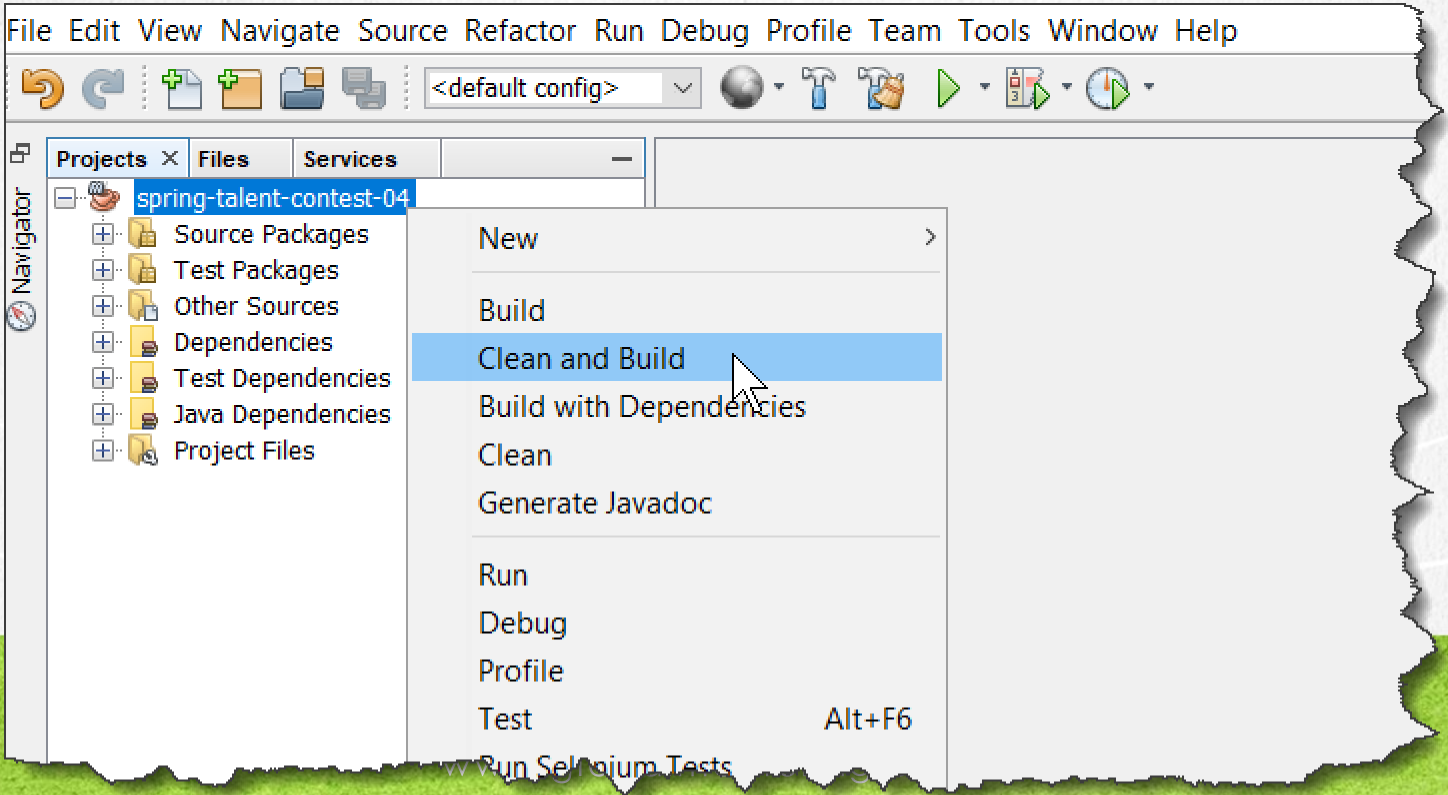
pom.xml:

Click to download

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
            <scope>test</scope>
            <type>jar</type>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.0</version>
            </plugin>
        </plugins>
    </build>
    <name>spring-talent-contest-04</name>
</project>
```

13. EXECUTE CLEAN & BUILD

We do a clean & build to download the missing libraries:



14. MODIFY THE CODE

TestJUnitTalentContest.java:

Click to download

```java
package test;

import competitors.*;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.apache.logging.log4j.*;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@ContextConfiguration(
  { "/applicationContext.xml" })
public class TestJUnitTalentContest {

    Logger log = LogManager.getRootLogger();
    private Competitor competitor1;
    private Competitor competitor2;
    private Competitor musician1;
    private Competitor musician2;
```

14. MODIFY THE CODE

TestJUnitTalentContest.java:

Click to download

```
    @BeforeEach
    public void before() {
        log.info("Starting Spring Framework");
        ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
        log.info("getting the first Competitor");
        competitor1 = (Competitor) ctx.getBean("juggler");
        competitor2 = (Competitor) ctx.getBean("jugglerReciter");
        musician1 = (Competitor) ctx.getBean("pianist");
        musician2 = (Competitor) ctx.getBean("saxophonist");
    }

    @Test
    public void testJuggler() {
        log.info("Start executing Juggler");

        int ballsTest = 10;
        competitor1.execute();
        assertEquals(ballsTest, ((Juggler) competitor1).getBalls());

        log.info("Finish executing Juggler");

        log.info("Start executing JugglerReciter");

        ballsTest = 15;
        competitor2.execute();
        assertEquals(ballsTest, ((Juggler) competitor2).getBalls());

        log.info("Finish executing JugglerReciter");
```

14. MODIFY THE CODE

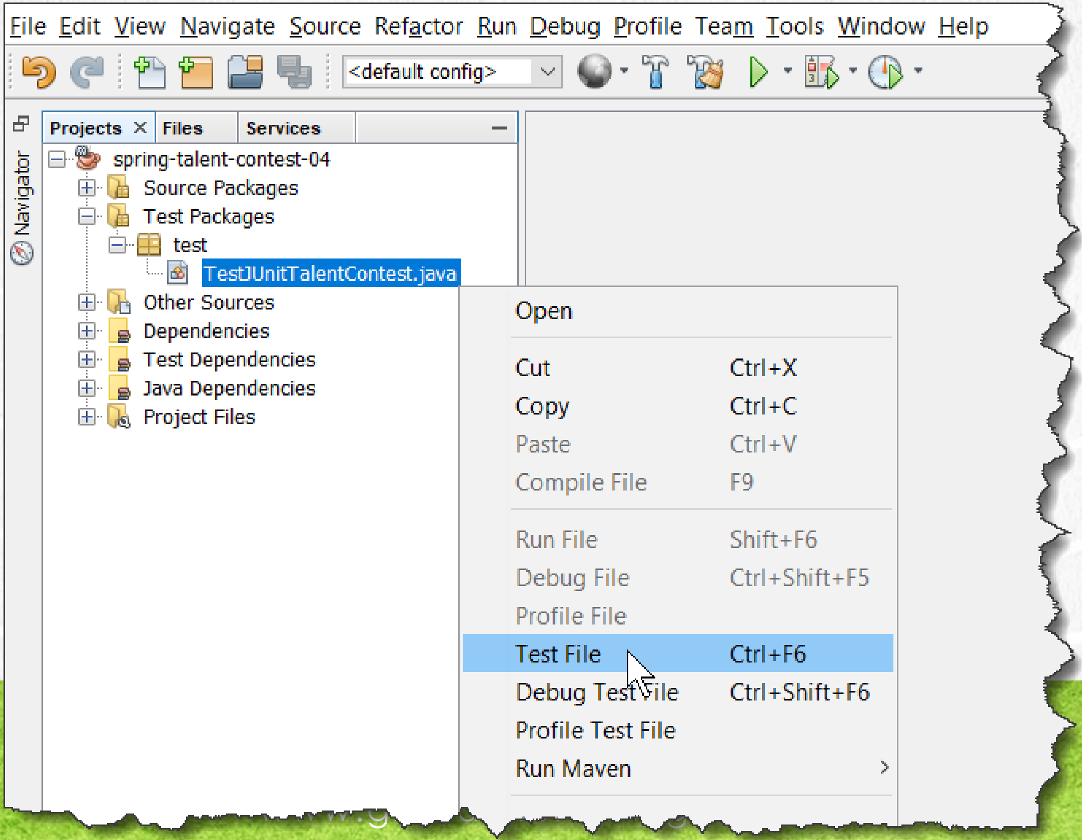
TestJUnitTalentContest.java:

Click to download

```
        log.info("Start Executing Pianist");

        String song = "Silent Night";
        musician1.execute();
        assertEquals(song, ((Musician) musician1).getSong());

        log.info("Finish Executing Pianist");

        log.info("Start Executing Saxophonist");

        song = "Equinox";
        musician2.execute();
        assertEquals(song, ((Musician) musician2).getSong());

        log.info("End Executing Saxophonist");
    }
}
```

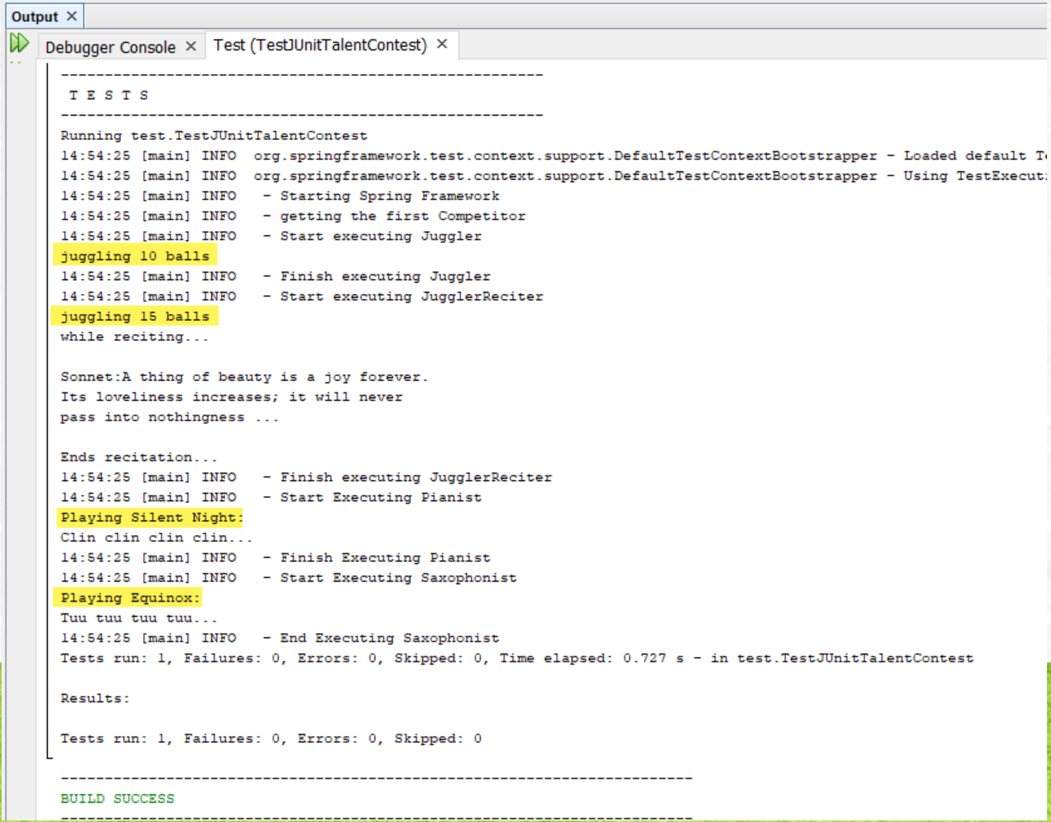
15. EXECUTE THE TEST

Execute the test:



15. EXECUTE THE TEST

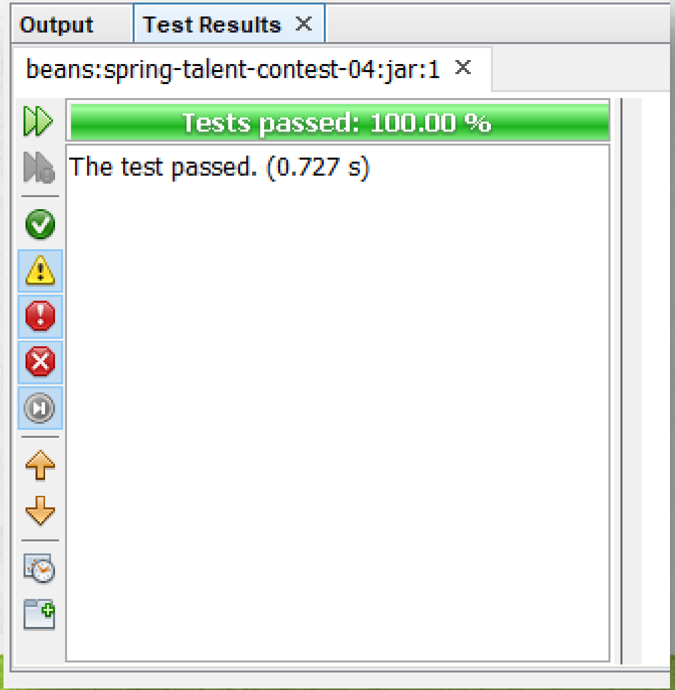
We execute the test. The result is as follows:



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running test.TestJUnitTalentContest
14:54:25 [main] INFO  org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default T
14:54:25 [main] INFO  org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecut
14:54:25 [main] INFO   - Starting Spring Framework
14:54:25 [main] INFO   - getting the first Competitor
14:54:25 [main] INFO   - Start executing Juggler
juggling 10 balls
14:54:25 [main] INFO   - Finish executing Juggler
14:54:25 [main] INFO   - Start executing JugglerReciter
juggling 15 balls
while reciting...

Sonnet:A thing of beauty is a joy forever.
Its loveliness increases; it will never
pass into nothingness ...

Ends recitation...
14:54:25 [main] INFO   - Finish executing JugglerReciter
14:54:25 [main] INFO   - Start Executing Pianist
Playing Silent Night:
Clin clin clin clin...
14:54:25 [main] INFO   - Finish Executing Pianist
14:54:25 [main] INFO   - Start Executing Saxophonist
Playing Equinox:
Tuu tuu tuu tuu...
14:54:25 [main] INFO   - End Executing Saxophonist
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.727 s - in test.TestJUnitTalentContest

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

15. EXECUTE THE TEST

We execute the test. The result is as follows:



```
Tests passed: 100.00 %
The test passed. (0.727 s)
```

EXERCISE CONCLUSION

With this exercise we put into practice the use of annotations with Spring, and how to replace the XML configuration of the applicationContext.xml file.

With this we have already the bases of the use of applications Spring with annotations, injection of dependencies and the relationship between the different beans, including the concept of autowire, and how the applicationContext.xml file should be configured.

Finally, we created a unit test, which served to prove that the Spring factory worked correctly with the annotation and autowire changes we made in the configuration of the Spring beans.

ONLINE COURSE

SPRING FRAMEWORK

By: Eng. Ubaldo Acosta

Global Mentoring

SPRING FRAMEWORK COURSE

www.globalmentoring.com.mx