VaanNila

Spring IoC

In Spring, the `Inversion of Control (IoC)` principle is implemented using the `Dependency Injection (DI)` design pattern. Let's understand dependency injection with the help of an example. First we will see a java version of the example and later we will add spring functionalities to it. As far as the example go, its pretty simple. The `QuizMater` interface exposes the popQuestion() method. To keep things simple, our `QuizMaster` will generate only one question.

```
QuizMaster.java
----------------
package com.vaannila;

public interface QuizMaster {

    public String popQuestion();
}
```

The `StrutsQuizMaster` and the `SpringQuizMaster` class implements `QuizMaster` interface and they generate questions related to struts and spring respectively.

```
StrutsQuizMaster.java
----------------------
package com.vaannila;

public class StrutsQuizMaster implements QuizMaster {

    @Override
    public String popQuestion() {
        return "Are you new to Struts?";
    }

}
```

```
SpringQuizMaster.java
----------------------
package com.vaannila;

public class SpringQuizMaster implements QuizMaster {

    @Override
    public String popQuestion() {
        return "Are you new to Spring?";
    }

}
```

We have a `QuizMasterService` class that displays the question to the user. The `QuizMasterService` class holds reference to the `QuizMaster`.

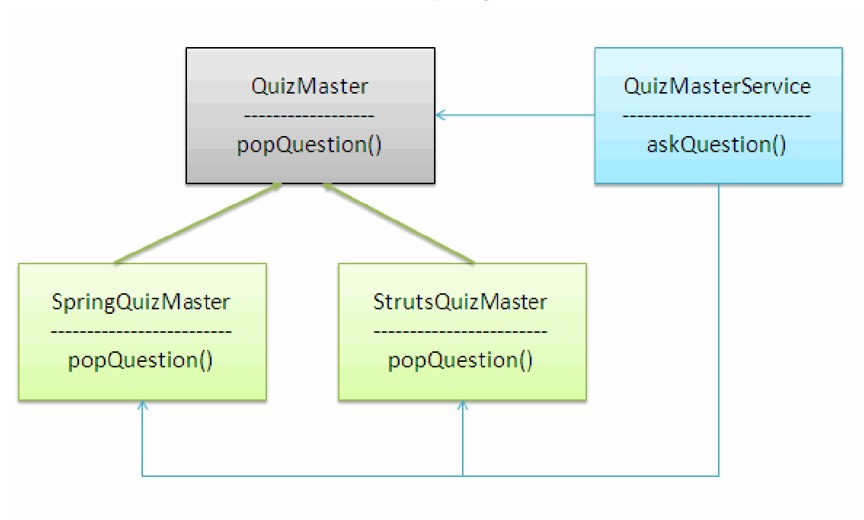
```
QuizMasterService.java
-----------------------
package com.vaannila;

public class QuizMasterService {

    private QuizMaster quizMaster = new SpringQuizMaster();

    public void askQuestion()
    {
        System.out.println(quizMaster.popQuestion());
    }
}
```

Finally we create the `QuizProgram` class to conduct quiz.

```
QuizProgram.java
----------------
package com.vaannila;

public class QuizProgram {

    public static void main(String[] args) {
        QuizMasterService quizMasterService = new QuizMasterService();
        quizMasterService.askQuestion();
    }

}
```

As you can see it is pretty simple, here we create an instance of the `QuizMasterService` class and call the `askQuestion()` method. When you run the program as expected "`Are you new to Spring?`" gets printed in the console.

Let's have a look at the class diagram of this example. The green arrows indicate generalization and the blue arrows indicates association.



As you can see this architecture is tightly coupled. We create an instance of the `QuizMaster` in the `QuizMasterService` class in the following way.

```java
private QuizMaster quizMaster = new SpringQuizMaster();
```

To make our quiz master Struts genius we need to make modifications to the `QuizMasterService` class like this.

```java
private QuizMaster quizMaster = new StrutsQuizMaster();
```

So it is tightly coupled. Now lets see how we can avoid this by using the `Dependency Injection` design pattern. The Spring framework provides powerful container to manage the components. The container is based on the Inversion of Control (IoC) principle and can be implemented by using the Dependency Injection (DI) design pattern. Here the component only needs to choose a way to accept the resources and the container will deliver the resource to the components.

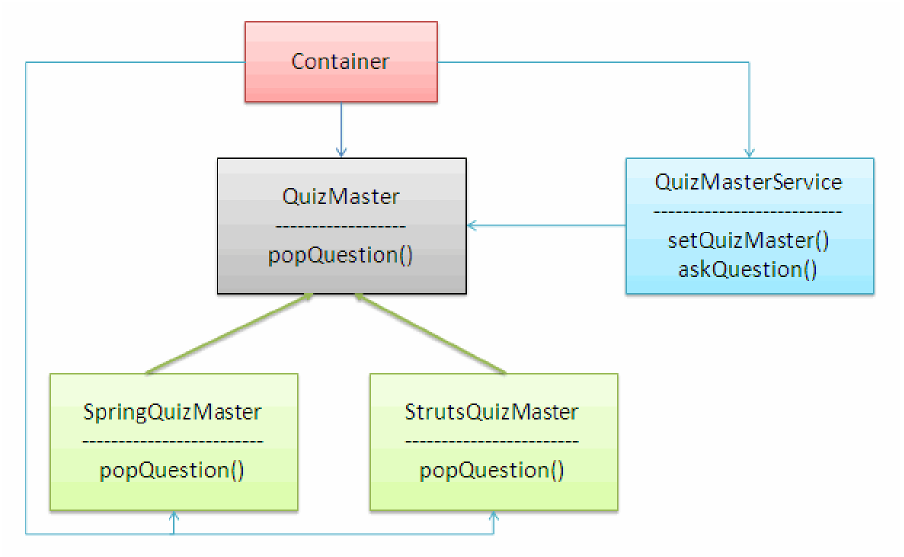
In this example instead of we, directly creating an object of the `QuizMaster` bean in the `QuizMasterService` class, we make use of the container to do this job for us. Instead of hard coding any values we will allow the container to inject the required dependancies.

We can inject the dependancies using the setter or constructor injection. Here we will see how we can do this using the setter injection.

```java
QuizMasterService.java
-----------------------
package com.vaannila;

public class QuizMasterService {

    QuizMaster quizMaster;

    public void setQuizMaster(QuizMaster quizMaster) {
        this.quizMaster = quizMaster;
    }

    public void askQuestion()
    {
        System.out.println(quizMaster.popQuestion());
    }
}
```

The value for the `QuizMaster` will be set using the `setQuizMaster()` method. The QuizMaster object is never instantiated in the `QuizMasterService` class, but still we access it. Usually this will throw a `NullPointerException`, but here the container will instantiate the object for us, so it works fine.

After making all the changes, the class diagram of the example look like this.



The container comes into picture and it helps in injecting the dependancies.

The bean configuration is done in the `beans.xml` file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=" http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="springQuizMaster"
        class="com.vaannila.SpringQuizMaster"></bean>
    <bean id="strutsQuizMaster"
        class="com.vaannila.StrutsQuizMaster"></bean>
    <bean id="quizMasterService" class="com.vaannila.QuizMasterService">
        <property name="quizMaster">
        <ref local="springQuizMaster"/>
        </property>
    </bean>

</beans>
```

We define each bean using the `bean` tag. The `id` attribute of the bean tag gives a logical name to the bean and the `class` attribute represents the actual bean class. The `property` tag is used to refer the property of the bean. To inject a bean using the setter injection you need to use the `ref` tag.

Here a reference of `SpringQuizMaster` is injected to the `QuizMaster` bean. When we execute this example, "`Are you new to Spring?`" gets printed in the console.

To make our `QuizMaster` ask questions related to Struts, the only change we need to do is, to change the bean reference in the `ref` tag.

```xml
<bean id="quizMasterService" class="com.vaannila.QuizMasterService">
    <property name="quizMaster">
        <ref local="strutsQuizMaster"/>
    </property>
</bean>
```

In this way the Dependency Injection helps in reducing the coupling between the components.

To execute this example add the following jar files to the classpath.

```
antlr-runtime-3.0
commons-logging-1.0.4
org.springframework.asm-3.0.0.M3
org.springframework.beans-3.0.0.M3
org.springframework.context-3.0.0.M3
org.springframework.context.support-3.0.0.M3
org.springframework.core-3.0.0.M3
org.springframework.expression-3.0.0.M3
```

You can download and try the DI example by clicking the download link below.

Source :**Download**

Source + Lib :**Download**

Contact Us | Copyright © 2009 vaannila.com