**Spring IoC Tutorial**

The **Dependency Injection (DI)** can be used to decouple the dependencies between different components.

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 IQuizMaster will generate only one question.

1.IQuizMaster.java

2.----------------

3.package com.tutorials4u.quiz;

4.

5.public interface IQuizMaster {

6.

7.    public String popQuestion();

8.}

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

01.StrutsQuizMaster.java

02.----------------------

03.package com.tutorials4u.quiz;

04.

05.public class StrutsQuizMaster implements IQuizMaster {

06.

07.    @Override

08.    public String popQuestion() {

09.        return "Are you new to Struts?";

10.    }

11.

12.}

01.SpringQuizMaster.java

02.----------------------

03.package com.tutorials4u.quiz;

04.

05.public class SpringQuizMaster implements IQuizMaster {

06.

07.    @Override

08.    public String popQuestion() {

09.        return "Are you new to Spring?";

10.    }

11.

12.}

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

01.QuizMasterService.java

02.-----------------------

03.package com.tutorials4u.quiz;

04.

05.public class QuizMasterService {

06.

07.    private IQuizMaster quizMaster = new SpringQuizMaster();

08.

09.    public void askQuestion()

10.    {

11.        System.out.println(quizMaster.popQuestion());

12.    }

13.}

Finally we create the QuizProgram class to conduct quiz.

01.QuizProgram.java

02.----------------

03.package com.tutorials4u.quiz;

04.

05.public class QuizProgram {

06.

07.    public static void main(String[] args) {

08.        QuizMasterService quizMasterService = new QuizMasterService();

09.        quizMasterService.askQuestion();

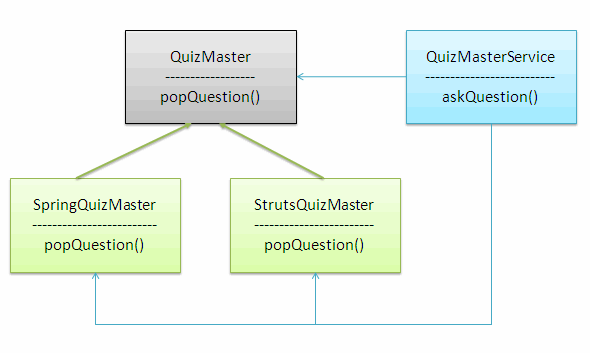
10.    }

11.

12.}

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 IQuizMaster in the QuizMasterService class in the following way.

1.private IQuizMaster quizMaster = new SpringQuizMaster();

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

1.private IQuizMaster 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 prowerful 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 IQuizMaster 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.

01.QuizMasterService.java

02.-----------------------

03.package com.tutorials4u.quiz;

04.

05.public class QuizMasterService {

06.

07.    IQuizMaster quizMaster;

08.

09.    public void setQuizMaster(IQuizMaster quizMaster) {

10.        this.quizMaster = quizMaster;

11.    }

12.

13.    public void askQuestion()

14.    {

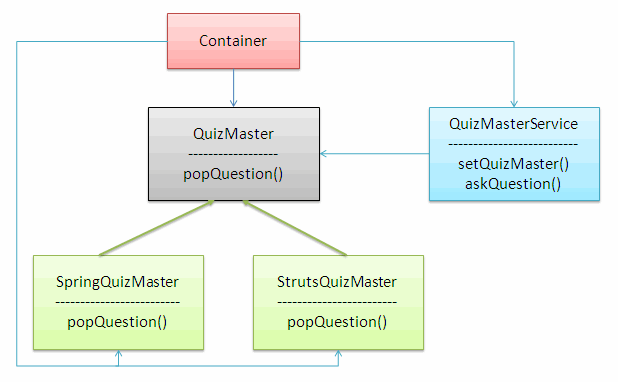
15.        System.out.println(quizMaster.popQuestion());

16.    }

17.}

The value for the IQuizMaster will be set using the setQuizMaster() method. The IQuizMaster 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.

01.<?xml version="1.0" encoding="UTF-8"?>

02.<beans xmlns="http://www.springframework.org/schema/beans"

03.xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

04.xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

05.

06.    <bean id="springQuizMaster" class="com.tutorials4u.quiz.SpringQuizMaster"></bean>

07.    <bean id="strutsQuizMaster" class="com.tutorials4u.quiz.StrutsQuizMaster"></bean>

08.

 <bean id="quizMasterService" class="com.tutorials4u.quiz.QuizMasterService">

09.        <property name="quizMaster">

10.        <ref local="springQuizMaster"/>

11.

12.        </property>

13.    </bean>

14.

15.</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 IQuizMaster bean. When we execute this example, "Are you new to Spring?" gets printed in the console.

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

1.<bean id="quizMasterService" class="com.tutorials4u.quiz.QuizMasterService">

2.    <property name="quizMaster">

3.        <ref local="strutsQuizMaster"/>

4.    </property>

5.</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.

1.antlr-runtime-3.0

2.commons-logging-1.0.4

3.org.springframework.asm-3.0.0.M3

4.org.springframework.beans-3.0.0.M3

5.org.springframework.context-3.0.0.M3

6.org.springframework.context.support-3.0.0.M3

7.org.springframework.core-3.0.0.M3

8.org.springframework.expression-3.0.0.M3