|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld* and *MainApp* under the *com.tutorialspoint*package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

**File Beans.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-3.0.xsd">

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message" value = "Hello World!"/>

</bean>

</beans>

**Scope : singleton, protype …**

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

**Init, destroy method**

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" init-method = "init"

destroy-method = "destroy">

<property name = "message" value = "Hello World!"/>

</bean>

public class MainApp {

public static void main(String[] args) {

AbstractApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

**context.registerShutdownHook();**

}

}

**Default init, destroy method**

<beans ...

**default-init-method = "init"**

**default-destroy-method = "destroy**">

</beans>

**Kế thừa bean**

A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed.

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld">

<property name = "message1" value = "Hello World!"/>

<property name = "message2" value = "Hello Second World!"/>

</bean>

<bean id =" helloIndia" class = "com.tutorialspoint.HelloIndia" **parent** = "helloWorld">

<property name = "message1" value = "Hello India!"/>

<property name = "message3" value = "Namaste India!"/>

</bean>

Dependency Injection

public class TextEditor {

private SpellChecker spellChecker;

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

**Anotations**

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

**Tương đương**

<beans>

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" />

</beans>

**ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");**

**Chuyển thành**

**ApplicationContext ctx = new AnnotationConfigApplicationContext(HelloWorldConfig.class);**

**Load nhiều configuration khác nhau:**

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

## **Lifecycle Callbacks**

Init and destroy

public class Foo {

public void init() {

// initialization logic

}

public void cleanup() {

// destruction logic

}

}

@Configuration

public class AppConfig {

@Bean(initMethod = "init", destroyMethod = "cleanup" )

public Foo foo() {

return new Foo();

}

}

**Bean scope**

@Configuration

public class AppConfig {

@Bean

@Scope("prototype")

public Foo foo() {

return new Foo();

}

}