Spring Containers

➢ **Introduction**

1. The central component of a Spring application is the containers.
2. Spring containers manage the life cycle of the bean in the applications,and are responsible for wiring of these beans.
3. Spring provides two interfaces that act as containers,namely **BeanFactory** and **ApplicationContext.**

➢ **BeanFactory**

1. BeanFactory is an interface and available in org.springframework.beans.factory package.
2. BeanFactory container is the root container that loads all the beans and provides dependency injection.
3. BeanFactory is the basic container.
4. BeanFactory is a lazy container,which means that it instantiate the bean and configures it only when the getBean() method is called.

❖ **Loading the configuration file using BeanFactory**

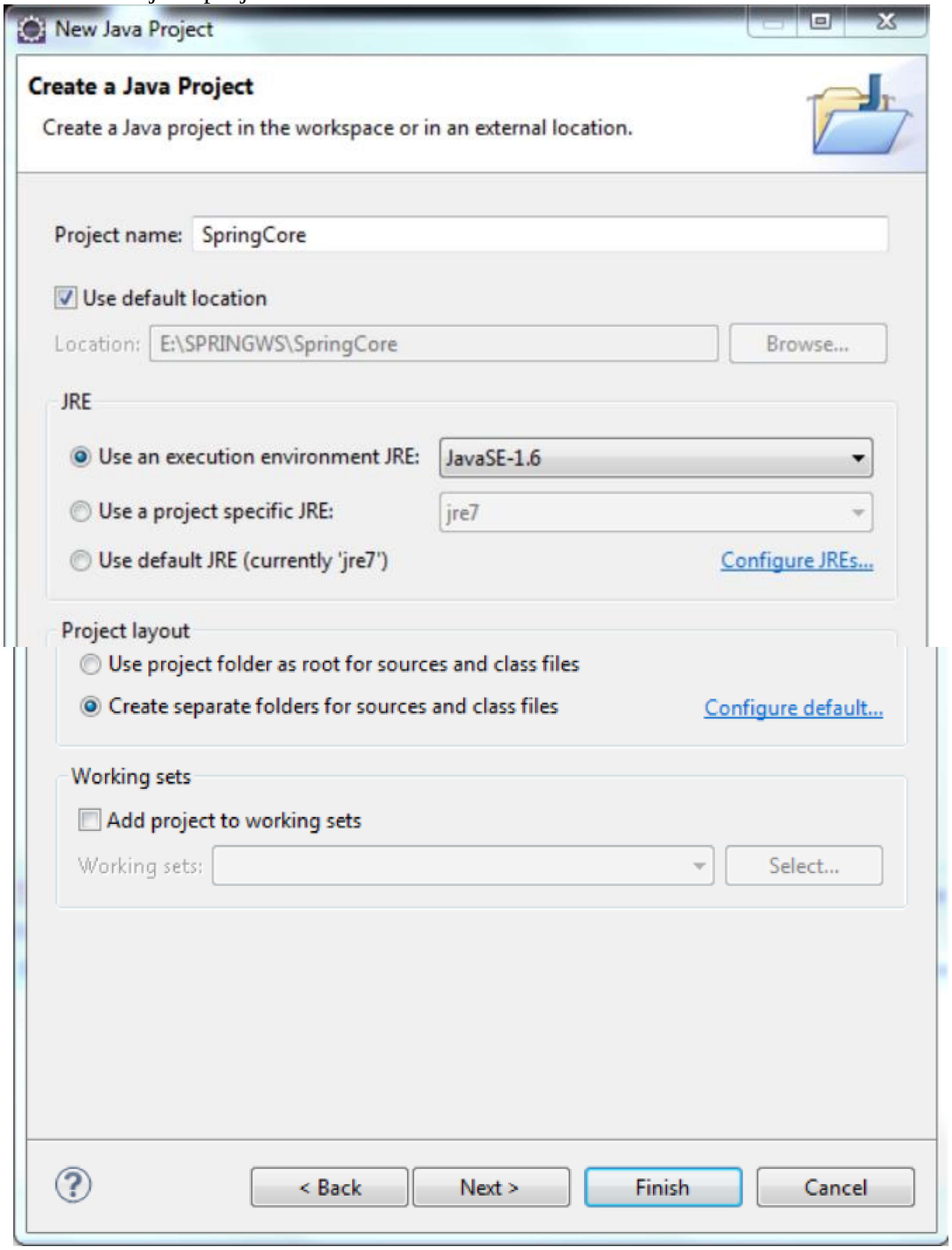
```
BeanFactory factory=new XmlBeanFactory(new ClassPathResource("beans.xml"));
```

❖ **Example:**

1. Create the java project.
2. Go to configure build path option.
3. Add the spring jar files to the project using build.
4. Create a pojo class named Message.java
5. Create the Spring bean configuration file and add Message class as bean entry.
6. Create client program and run it.
7. Final project structure.

❖ **Implementation**

1. Create the java project.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: SpringCore

[✓] Use default location

Location: E:\SPRINGWS\SpringCore

JRE

[●] Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre7

( ) Use default JRE (currently 'jre7')

Project layout

( ) Use project folder as root for sources and class files

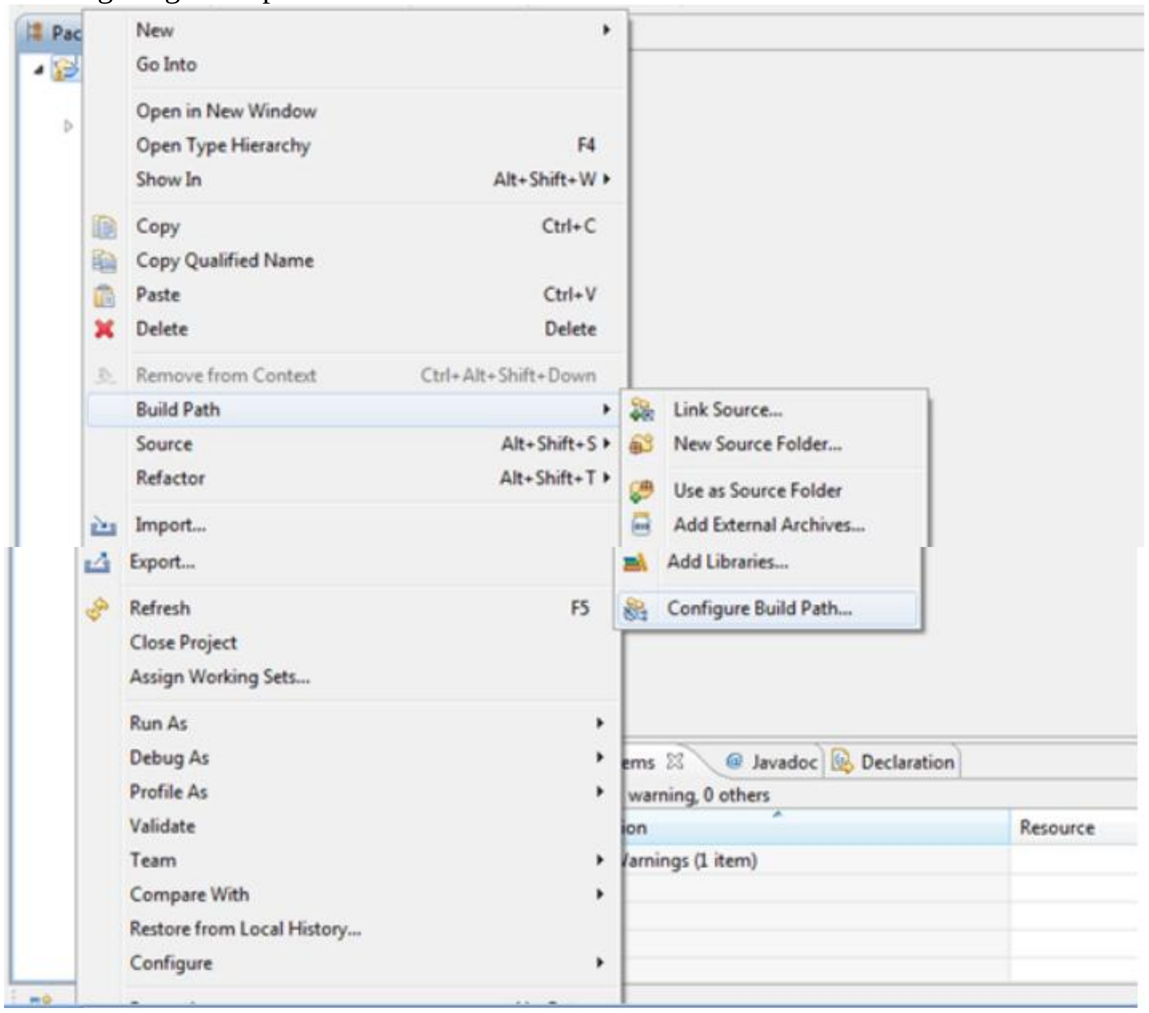
[●] Create separate folders for sources and class files

Working sets

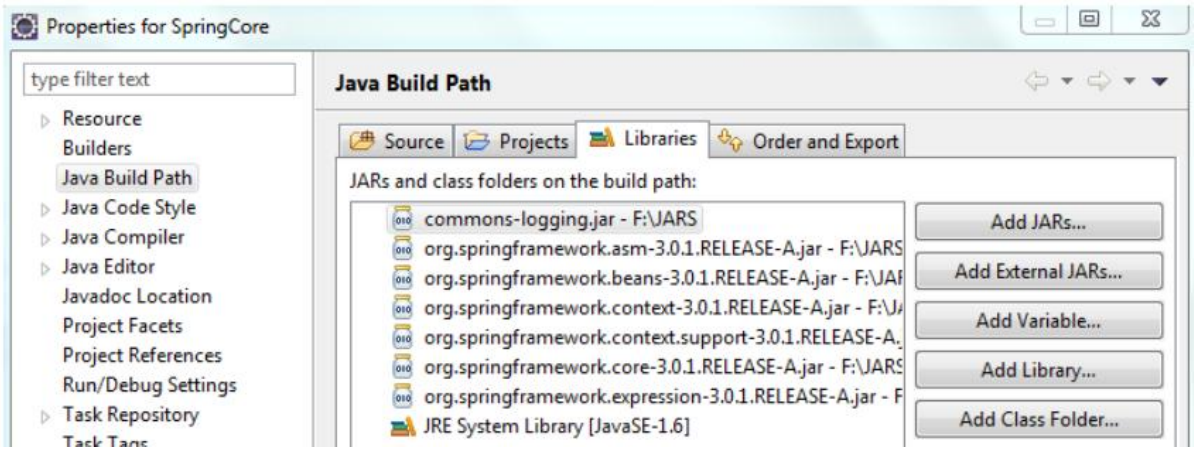
[ ] Add project to working sets

Working sets:

2.Configuring build path

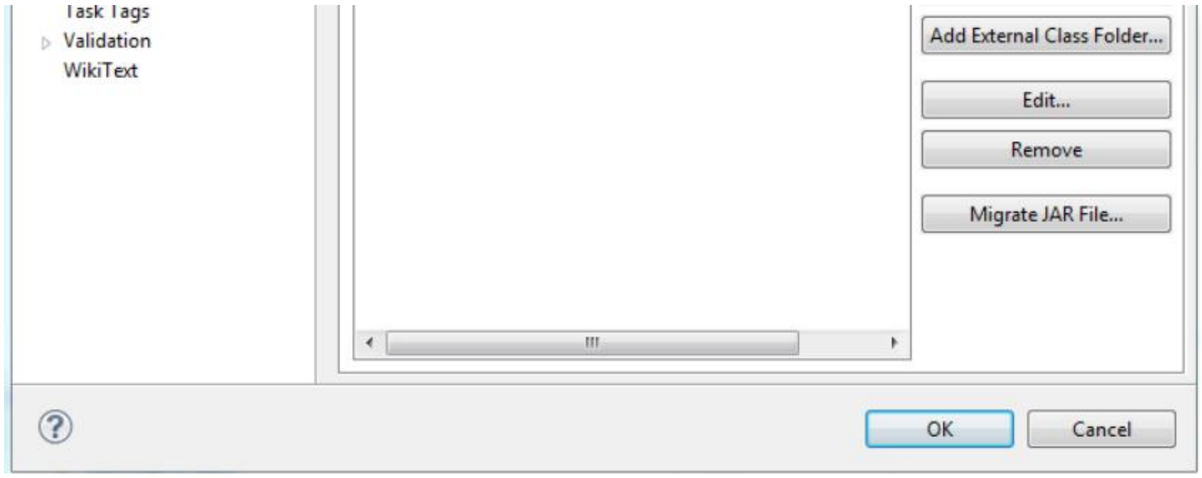


3. Add the spring jar files to the project using build.



JARs and class folders on the build path:

- commons-logging.jar - F:\JARS
- org.springframework.asm-3.0.1.RELEASE-A.jar - F:\JARS
- org.springframework.beans-3.0.1.RELEASE-A.jar - F:\JAI
- org.springframework.context-3.0.1.RELEASE-A.jar - F:\J
- org.springframework.context.support-3.0.1.RELEASE-A.
- org.springframework.core-3.0.1.RELEASE-A.jar - F:\JARS
- org.springframework.expression-3.0.1.RELEASE-A.jar - F
- JRE System Library [JavaSE-1.6]



4. Create a pojo class named Message.java

```java
package com.java2learn.core;

public class Message {

    private String message;

    public void setMessage(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }
}
```

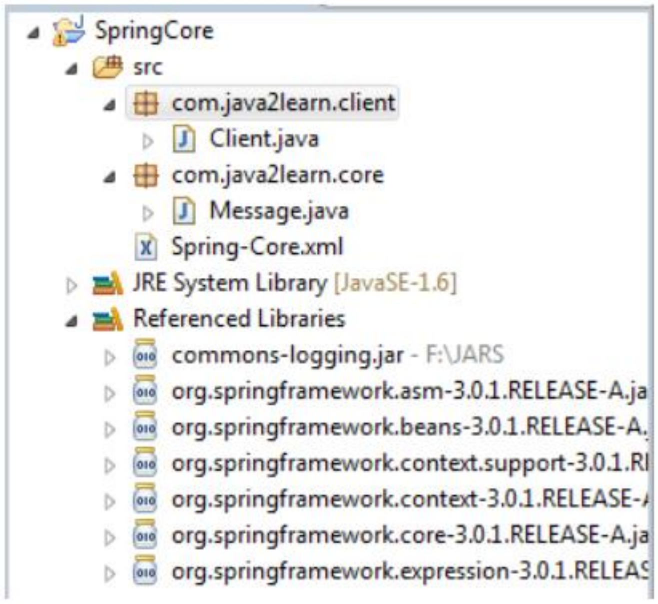
5. Create the Spring bean configuration file and add Message class as bean entry.

```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-3.0.xsd">

    <bean id="message" class="com.java2learn.core.Message">
    <property name="message" value="Hello World"></property>
    </bean>

</beans>
```

6. Create client program and run it.

```
package com.java2learn.client;

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

import com.java2learn.core.Message;

public class Client {

    public static void main(String[] args) {

 BeanFactory factory=new XmlBeanFactory(new ClassPathResource("Spring-
Core.xml"));
        Message message=(Message)factory.getBean("message");
        System.out.println("Message Name::"+message.getMessage());

    }

}
```

7. Final Project structure.



```
SpringCore
  src
    com.java2learn.client
      Client.java
    com.java2learn.core
      Message.java
    Spring-Core.xml
  JRE System Library [JavaSE-1.6]
  Referenced Libraries
    commons-logging.jar - F:\JARS
    org.springframework.asm-3.0.1.RELEASE-A.ja
    org.springframework.beans-3.0.1.RELEASE-A.
    org.springframework.context.support-3.0.1.RI
    org.springframework.context-3.0.1.RELEASE-/
    org.springframework.core-3.0.1.RELEASE-A.ja
    org.springframework.expression-3.0.1.RELEAS
```

❖ **Output:**

```
Message Name::Hello World
```

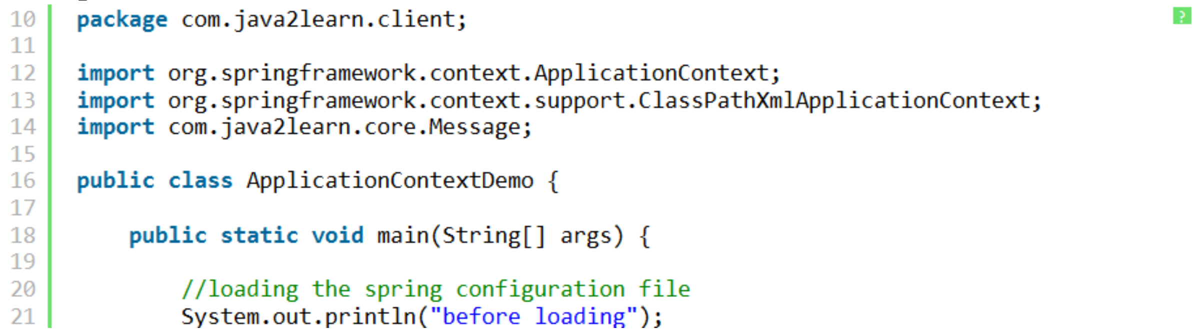
➢ **ApplicationContext**

1. ApplicationContext is an interface and available in org.springframework.contexty package.
2. ApplicationContext is the sub interface of BeanFactory.
3. ApplicationContext used as a container in the enterprise applications with a number of features.

4. ApplicationContext container instantiate all the beans while loading the configuration file.

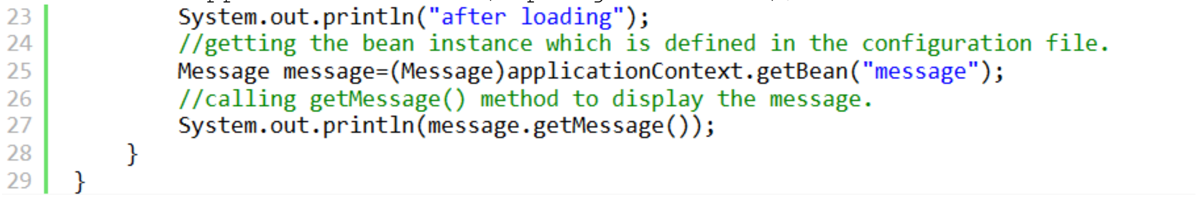
❖ **Loading the configuration file using ApplicationContext container.**
Use the above example for the ApplicationContext container demo except client program.

❖ **Example:**



```java
package com.java2learn.client;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.java2learn.core.Message;

public class ApplicationContextDemo {

    public static void main(String[] args) {

        //loading the spring configuration file
        System.out.println("before loading");
```

```
ApplicationContext applicationContext=new
ClassPathXmlApplicationContext("Spring-Core.xml");
```



```java
        System.out.println("after loading");
        //getting the bean instance which is defined in the configuration file.
        Message message=(Message)applicationContext.getBean("message");
        //calling getMessage() method to display the message.
        System.out.println(message.getMessage());
    }
}
```

❖ **Output:**

```
before loading

Message object is creating..

after loading

Hello World
```