[Spring Core - Bean Definition](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Explain the need and benefits of Spring Core
  + Enterprise Application Development, Pitfalls of Application Servers and EJB. Features of Spring Framework: Lightweight, loose coupling with dependency injection, Inversion of Control (IoC), Aspect Oriented Programming, Container, MVC, Transaction Management; Benefits - Solve Enterprise Application development difficulties, development through POJOs, eliminates creation of singleton and factory classes.
    - Ref - https://www.onlinetutorialspoint.com/spring/advantages-of-spring-framework.html

* Demonstrate creation of Spring Boot Application
  + Spring initializr, https://start.spring.io, @SpringBootApplication, SpringApplication.run()
    - Ref - https://start.spring.io

* Explain the need and benefits of Spring Boot
  + Makes Java development easy, avoids tedious development steps, reduces development time, avoids writing boilerplate code, provides embedded tomcat server, avoid XML configuration
    - Ref - https://www.journaldev.com/7969/spring-boot-tutorial

* Demonstrate loading bean from spring configuration file
  + Spring configuration xml, spring xml schema spring-beans.xsd, <bean>, id, class, <constructor-arg>, <property>, name, value, ClassPathXmlApplicationContext, ApplicationContext, context.getBean(), singleton scope, prototype scope
    - Ref - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html
    - IoC Container - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans
    - Scopes - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-factory-scopes
    - Constructor Injection - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-constructor-injection
    - Setter method injection - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-setter-injection

* Demonstrate inclusion of logging in Spring Boot Application
  + application.properties, logging.level, logging.pattern, server.port, LoggerFactory, Logger, log levels (trace, debug, info, warn, error)
    - Ref - https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html

**Create a Spring Web Project using Maven**  
  
Follow steps below to create a project:

1. Go to <https://start.spring.io/>
2. Change Group as “com.cognizant”
3. Change Artifact Id as “spring-learn”
4. Select Spring Boot DevTools and Spring Web
5. Create and download the project as zip
6. Extract the zip in root folder to Eclipse Workspace
7. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
8. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
9. Include logs to verify if main() method of SpringLearnApplication.
10. Run the SpringLearnApplication class.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. SpringLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

**Spring Core – Load SimpleDateFormat from Spring Configuration XML**  
  
SimpleDateFormat with the pattern ‘dd/MM/yyyy’ is created in multiple places of an application. To avoid creation of SimpleDateFormat in multiple places, define a bean in Spring XML Configuration file and retrieve the date.  
  
Follow steps below to implement:

* Create spring configuration file date-format.xml in src/main/resources folder of 'spring-learn' project
* Open https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-factory-metadata
* Copy the XML defined in the section of previous step URL and paste it into date-format.xml
* Define bean tag in the XML with for date format. Refer code below.

<?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

        https://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="dateFormat" class="java.text.SimpleDateFormat">

<constructor-arg value="dd/MM/yyyy" />

</bean>

</beans>

* Create new method displayDate() in SpringLearnApplication.java
* In displayDate() method create the ApplicationContext. Refer code below:

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");

* Get the dateFormat using getBean() method. Refer code below.

SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);

* Using the format variable try to parse string '31/12/2018' to Date class and display the result using System.out.println.
* Run the application as 'Java Application' and check the result in console log output.

**Troubleshooting Tips**  
  
If the tomcat port has a conflict and the server is not starting include the below property in application.properties file in src/main/resources folder.

server.port=8083

**Spring Core - Incorporate Logging**  
  
Incorporate logging in the Spring Boot project created in previous hands on. Refer steps below:

* Create application.properties if not yet created in src/main/resources folder
* Add below lines in application.properties

logging.level.org.springframework=info

logging.level.com.cognizant.springlearn=debug

logging.pattern.console=%d{yyMMdd}|%d{HH:mm:ss.SSS}|%-20.20thread|%5p|%-25.25logger{25}|%25M|%m%n

* In SpringLearnApplication.java include the following imports:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

* Include the below static variable in SpringLearnApplication.java:

private static final Logger LOGGER = LoggerFactory.getLogger(SpringLearnApplication.class);

* Include info log on start and end of method. Debug log for displaying the date (refer code below)

public void displayDate() {

    LOGGER.info(“START”);

    //..

    LOGGER.debug(date);

    //..

    LOGGER.info(“END”);

}

**IMPORTANT NOTE:** Going forward all methods should incorporate logging as specified above. **Never** use System.out.println().

**Spring Core – Load Country from Spring Configuration XML**  
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country.

|  |  |
| --- | --- |
| **Code** | **Name** |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.  
  
Steps to implement

* Pick any one of your choice country to configure in Spring XML configuration named country.xml.
* Create a bean tag in spring configuration for country and set the property and values

    <bean id="country" class="com.cognizant.springlearn.Country">

        <property name="code" value="IN" />

        <property name="name" value="India" />

    </bean>

* Create Country class with following aspects:
  + Instance variables for code and name
  + Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”
  + Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.
  + Generate toString() method
* Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = (Country) context.getBean("country", Country.class);

LOGGER.debug("Country : {}", country.toString());

* Invoke displayCountry() method in main() method of SpringLearnApplication.java.
* Execute main() method and check the logs to find out which constructors and methods were invoked.

SME to provide more detailing about the following aspects:

* bean tag, id attribute, class attribute, property tag, name attribute, value attribute
* ApplicationContext, ClassPathXmlApplicationContext
* What exactly happens when context.getBean() is invoked

**Spring Core – Demonstration of Singleton Scope and Prototype Scope**  
  
The Country bean done in the previous hands on will be used to demonstrate the scopes in Spring. Implement the steps below.  
  
**Follow steps below to demonstrate Singleton Scope**

* Include a line in displayCountry() to get country bean reference one more time from the same application context. Only the third line of the below code snippet should be copied and pasted.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = context.getBean("country", Country.class);

Country anotherCountry = context.getBean("country", Country.class);

* The constructor will be called only once, which means that only one instance of Country bean is created

**Follow steps below to demonstrate Prototype Scope**

* Include scope="prototype" attribute in bean definition xml.

<bean id="country" class="com.cognizant.springlearn.Country" scope="prototype">

* Run the application
* Constructor will be called twice, which means that two instances of country is created.

**Spring Core – Load list of countries from Spring Configuration XML**  
  
Our main objective was to retrieve the list of four countries for the airlines website. Refer steps below to get this incorporated.

* Create a separate bean for each of the four country in country.xml.
* Create an ArrayList of Country in country.xml. Refer code below.

    <bean id="countryList" class="java.util.ArrayList">

        <constructor-arg>

            <list>

                <ref bean="in"></ref>

                <ref bean="us"></ref>

                <ref bean="de"></ref>

                <ref bean="jp"></ref>

            </list>

        </constructor-arg>

    </bean>

* Include new method displayCountries() in SpringLearnApplication.java
* In displayCountries() read the country list created above
* Display the list of countries as debug log.

SME to provide detailing on below aspects:

* <list>
* <ref>
* bean attribute

**IMPORTANT NOTE**: Do not forget to include the start and end logs in this new method.

**Spring Core – Implement data model for Employee using Spring XML Configuration**  
  
Implement data model for employee based on instructions provided below:

* Create package com.cognizant.spring-learn.bean
* Create class com.cognizant.spring-learn.bean.Employee with following attributes

|  |  |
| --- | --- |
| **Type** | **Attribute Name** |
| int | id |
| String | name |
| double | salary |
| boolean | permanent |
| Date | dateOfBirth |

* Generate getters, setters and toString() methods.
* Include empty parameter constructor with debug log.
* Define values for the attributes in a new spring xml configuration file named employee.xml
* Display the employee details in a new method displayEmployee() in main method of SpringLearnApplication.java

**Spring Core - Implement data model for Department in Employee**  
  
In continuation to the previous problem, let us add department details to Employee. Refer instructions below.

* Create com.cognizant.spring-learn.bean.Department
* Add attributes id and name
* Include empty parameter constructor with log
* Generate getters, setters and toString()
* Include department as an attribute in Employee class
* Include getters/setters for department
* Regenerate toString() method in Employee for inclusion of department
* In employee.xml, using ref attribute include department to employee. Refer sample spring xml code below.

   <bean id="departmentBean" class="com.cognizant.bean.Department">

        <property name="id" value="1" />

        <property name="name" value="Payroll" />

    </bean>

    <bean id="employee" class="com.cognizant.bean.Employee">

  ... other attributes of employee goes here

        <property name="department" ref="departmentBean" />

    </bean>

**Spring Core - Include Skill details for Employee**  
  
In continuation to the previous problem, incorporate list of skills for each Employee instance:

* Create new class com.cognizant.spring-learn.bean.Skill
* Create instance variables id and name with data types int and String respectively.
* Include, empty parameter constructor with log, getters, setters and toString()
* Add skills as instance variable of type Skill array in Employee.
* In Employee class, include getters, setters and update the toString() method for inclusion of Skill[]
* Create multiple skill instances in employee.xml and add them to array of Skills.
* Set the array of Skills to skills of Employee bean in configuration xml file
* Load the employee class from employee.xml and display the data.
* Refer sample code below inclusion of skills property for employee in employee.xml

       <property name="skills">

            <array>

                <ref bean="skillHtml" />

                <ref bean="skillCss" />

                <ref bean="skillJs" />

            </array>

        </property>

**Spring Core - Inversion of Control / Dependency Injection**  
  
In order to build a web service for managing employee details, controller, service and dao classes needs to be defined.  
  
As part of this hands on, creation of these classes needs to be done using Spring Core framework's Inversion of Control (IoC) and Dependency Injection.  
  
At the same time we need to implement this using annotations without spring configuraton xml file. Refer steps below to gradually implement this.

* Create all necessary classes based on the below specified class diagram. Do not include any constructor, toString() and getter methods.
* Include logs in constructors and setter methods on all these classes.
* Implement dependency injection in a staged manner as specified below one by one:
  + **Spring XML Configuration:**
    - In employee.xml define the beans for EmployeeController, EmployeeService and EmployeeDao using bean reference.
    - Load the EmployeeController bean in a new method getEmployeeController() in the SpringLearnApplication.java
    - Check the logs to see if EmployeeController, EmployeeService and EmployeeDao are loaded
  + **Spring XML Configuration with Autowiring:**
    - Include new bean configuration in employee.xml for the three classes using autowire property.
    - Modify getEmployeeController() method to load the new autowired bean from xml configuration.
    - Check the logs to see if all three classes are loaded.
  + **Loading beans using Annotations:**
    - Include @Component annotation at class level for EmployeeController, EmployeeService and EmployeeDao
    - Include @Autowired annotation in the respective setter method of EmployeeController and EmployeeService
    - Include a new method in SpringLearnApplication.java with below signature:

public static void displayEmployeeControllerAnnotation(ApplicationContext applicationContext)

* Use the below code in this method to load the controller bean:

        EmployeeController employeeController = (EmployeeController) applicationContext

                .getBean("employeeController");

* Invoke displayEmployeeControllerAnnotation() method in main method passing the applicaitonContext.

|  |
| --- |
| https://cognizant.e-box.co.in/uploads/Image/01fseangular/ioc1.png |

SME to explain how Inversion of Control works.