**Objectives**

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

Spring Boot simplifies Java application development by offering embedded servers, default configurations, and no requirement for XML setup. Applications can be quickly started using Spring Initializr.

To create a Spring Boot application:

1. Open Spring Initializr and set Group to com.cognizant and Artifact to spring-learn.
2. Add dependencies: Spring Boot DevTools and Spring Web.
3. Download and extract the project.
4. Import into Eclipse as a Maven project.
5. In the SpringLearnApplication.java class, add the following:

Example:

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

System.out.println("Application started");

}

}

This uses @SpringBootApplication to enable auto-configuration, component scanning, and Java-based configuration.

* 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

Spring Boot provides a framework to avoid boilerplate configuration and setup. It simplifies development by automatically handling dependency resolution, configuration, and embedded servers.

**Benefits of Spring Boot:**

* Eliminates boilerplate code.
* Comes with embedded servers like Tomcat.
* Reduces development time.
* Provides starter dependencies for easy setup.
* Avoids the need for external XML configuration files.
* Supports auto-configuration based on classpath.
* Integrates easily with modern development tools.
* 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

To avoid repetitive code like creating a new SimpleDateFormat object multiple times, it can be defined in a Spring configuration XML.

**Steps:**

1. Create a file named date-format.xml in the src/main/resources folder with the following content:

<?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>

1. **Add this code to SpringLearnApplication.java:**

public static void displayDate() throws ParseException {

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

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

Date date = format.parse("31/12/2018");

System.out.println("Parsed Date: " + date);

}

1. Call displayDate(); inside the main() method.

This demonstrates the use of Spring's IoC container with XML configuration and constructor-based 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

Spring Boot uses SLF4J with Logback for logging. Logging can be configured in the application.properties file and used programmatically with Logger.

Steps:

1. In src/main/resources/application.properties, include:

logging.level.root=info

logging.level.com.cognizant=debug

server.port=8082

1. Create a class with logging:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggerDemo {

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

public void process() {

LOGGER.info("Info log");

LOGGER.debug("Debug log");

LOGGER.error("Error log");

}

}

Log Levels:

* TRACE: Detailed internal logs
* DEBUG: Debugging information
* INFO: General information
* WARN: Potential problems
* ERROR: Errors that need attention