Unit 12

Q) Spring Framework

Sping is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable and reusable code.

Spring framework is an open source Java platform.

Spring is lightweight when it comes to size and transparency. The basic version of ~~Sp~~ Spring framework is around 2MB.

The core features of the Spring framework can be used in developing any

Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a Plain Old Java Object (POJO) -based programming model.

Benefits of Using the Spring Framework

Following is the list of few of the great benefits of using Spring Framework -

- Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.

- Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.

- Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK

timers and other view technologies.

- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.

- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other engineered or less popular web frameworks.

- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

- Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.

- Spring provides a consistent transaction

management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example.)

Dependency Injection (DI)

The technology that Spring is most identified with is the Dependency Injection (DI) flavor of Inversion of Control. The Inversion of Control (IoC) is a general concept and it can be expressed in many different ways. Dependency Injection is merely one concrete example of Inversion of Control.

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection helps in gluing these classes together and at the same time keeping them independent.

What is dependency injection exactly? Let's look at these two words separately. Here the dependency part translates into an association between two classes. For example, class A is dependent of class B. Now, let's

look at the second part, injection. All this means is, class B will get injected into class A by the IoC. Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods. Dependency Injection is the heart of Spring framework.

Aspect Oriented Programming (AOP)

One of the key components of Spring is the Aspect Oriented Programming (AOP) framework. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects including logging, declarative transactions, security, caching etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. DI helps you decouple your application objects from each other, while AOP helps you decouple cross-cutting concerns from the objects that they affect. The AOP module of Spring framework provides an aspect-oriented programming implementation allowing you

to define method-interceptors and pointcuts to clearly decouple code that implements functionality that should be separated.

Q) JoinPoint and PointCut

JoinPoint

A JoinPoint represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework. Consider the following examples -

- All methods classes contained in a package(s).
- A particular methods of a class.

PointCut

PointCut is a set of one or more JoinPoint where an advice should be executed. You can specify PointCuts using expressions or patterns. In Spring, PointCut helps to use specific JoinPoints to apply the advice.

Consider the following examples -

- @PointCut("execution(* com.tutorialspoint.*.*(..))").
- @PointCut("execution(* com.tutorialspoint.Student.getName(..))")

PAGE No.
DATE / /

Syntax

```
@Aspect
public class Logging{
   @PointCut("execution(* com.tutorialpoint.*.*(..))")
   private void selectAll(){}
}
```

where,

- @Aspect - Mark a class as a class containing advice methods.
- @PointCut - Mark a function as a PointCut
- execution(expression) - Expression covering methods on which advice is to be applied.

To understand the above-mentioned concepts related to JoinPoint and PointCut, let us write an example which will implement few of the PointCuts.

Following is the content of Logging.java file. This is actually a sample of aspect module, which defines the methods to be called at various points.

```
package com.tutorialspoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.PointCut;
import org.aspectj.lang.annotation.Before;
@Aspect
public class Logging{
@PointCut("execution(* com.tutorialspoint.*.*(...))")
private void selectAll(){}
```

```
@Before("selectAll()")
public void beforeAdvice(){
 System.out.println("Going to setup student
 profile.");
    }
  }
```

Following is the content of the Student.java file.

```
package com.tutorialspoint;
public class Student{
 private Integer age;
 private String name;
public void setAge(Integer age){
  this.age=age;
 }
public Integer getAge(){
 System.out.println("Age:" + age);
 return age;
}
public void setName(String name){
  this.name=name;
 }
 public String getName(){
  System.out.println("Name: " +name);
  return name;
 }
 public void printThrowException(){
  System.out.println("Exception raised");
  throw new IllegalArgumentException();
 }
 }
```

Following is the content of the MainApp.java file.

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MainApp {
   public static void main(String[] args) {
   ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
   Student student = (Student) context.getBean("student");
   student.getName();
   student.getAge();
   }
}
```

Following is the configuration file Beans.xml.

```
<bean id = "student" class = "com.tutorialspoint.Student">
<property name = "name" value = "Zara" />
<property name = "age" value = "11" />
</bean>
<bean id = "logging" class = "com.tutorialspoint.Logging" />
</beans>
```

Run Project

Once you are done creating the source and configuration files, run your application. Rightclick on MainApp.java in

your application and use run as Java Application command. If everything is fine with your application, it will print the following message.

```
Going to setup student profile.
Name: Zara
Going to setup student profile.
Age: 11
```

The above-defined @pointcut uses an expression to select all the methods defined under the package com.tutorialspoint. @Before advice uses the above-defined pointcut as a parameter. Effectively beforeAdvice() method will be called before every method covered by above pointcut.