# MODULE- 11
# SPRING

# Course Topics

- Module 1
  - Introduction to Java
- Module 2
  - Data Handling and Functions
- Module 3
  - Object Oriented Programming in Java
- Module 4
  - Packages and Multi-threading
- Module 5
  - Collections
- Module 6
  - XML
- Module 7
  - JDBC
- Module 8
  - Servlets
- Module 9
  - JSP
- Module 10
  - Hibernate
- Module 11
  - **Spring**
- Module 12
  - Spring, Ajax and Design Patterns
- Module 13
  - SOA
- Module 14
  - Web Services and Project

# Objectives

At the end of this module, you will be able to

→ Understand the usage of Spring Framework

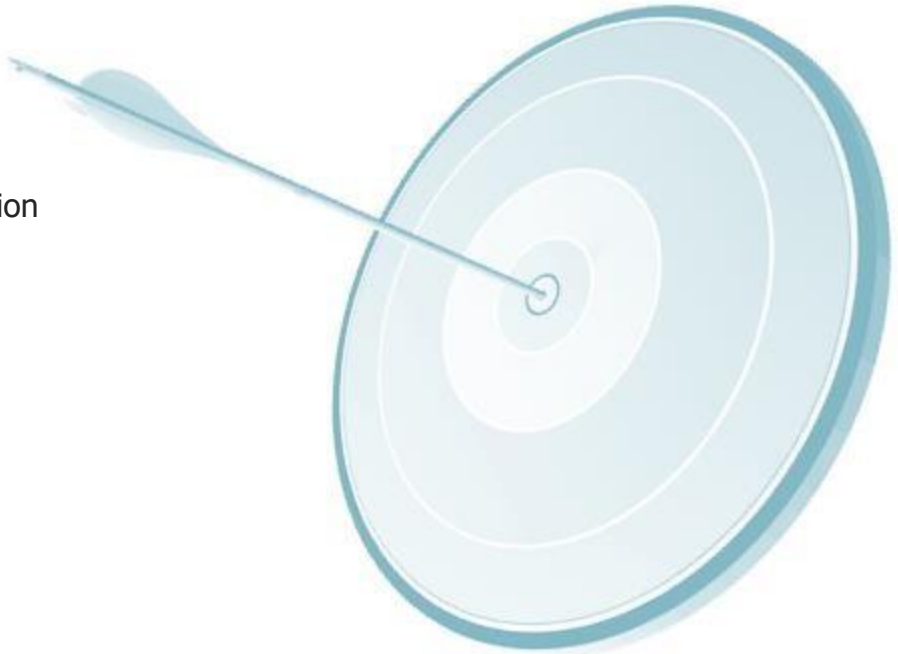
→ Understand Spring Architecture

→ Learn Bean Scope and Bean Lifecycle Methods

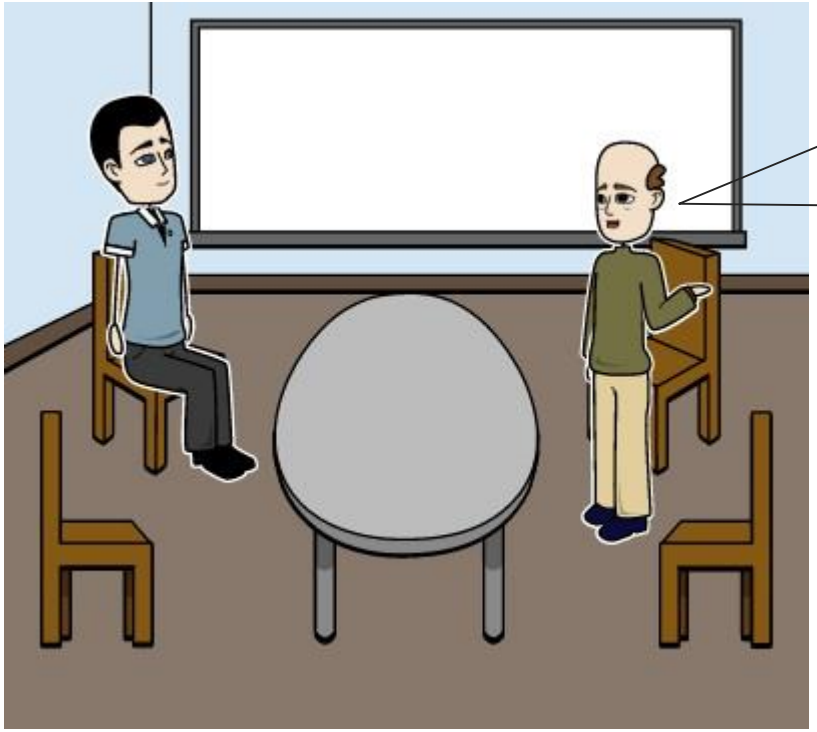
→ Analyze BeanPostProcessor

→ Understand Bean inheritance and Dependency Injection

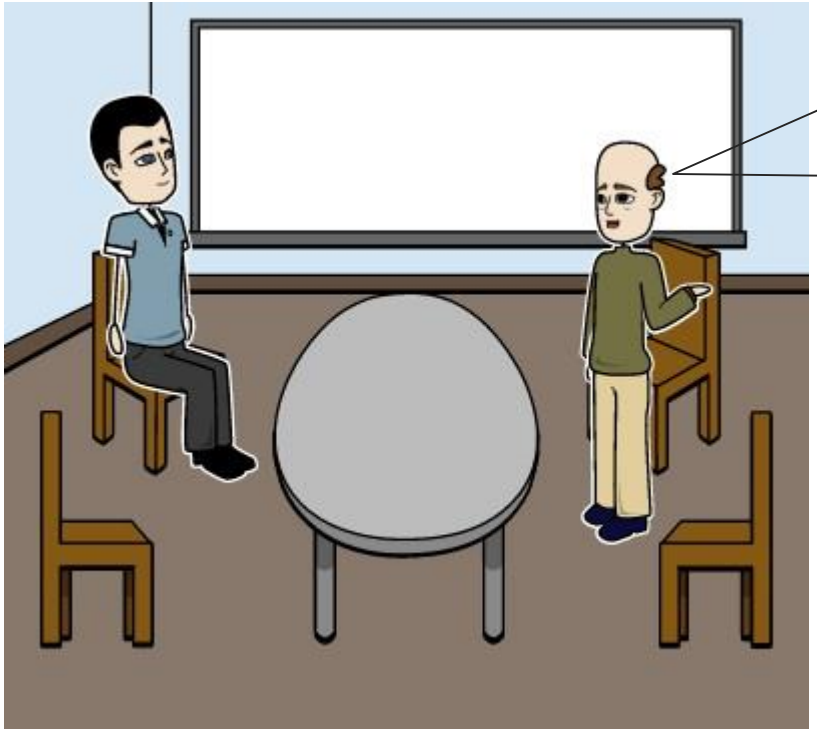
→ Learn Autowiring



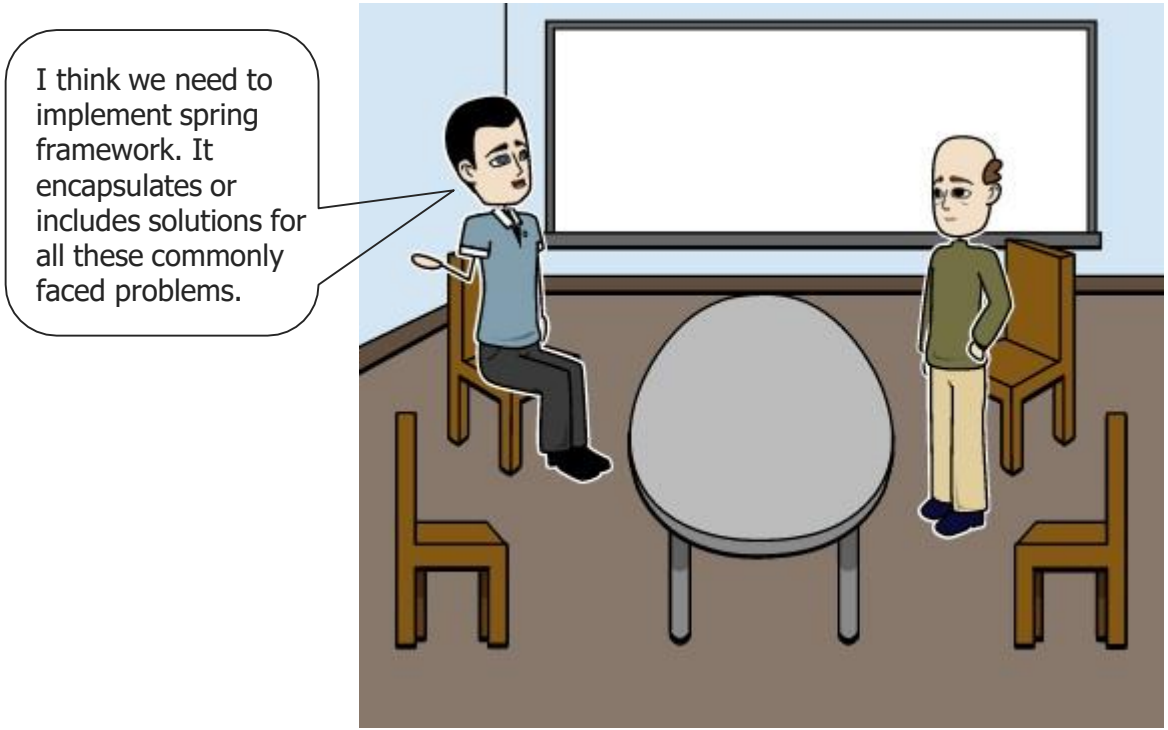
# A new Task for John!



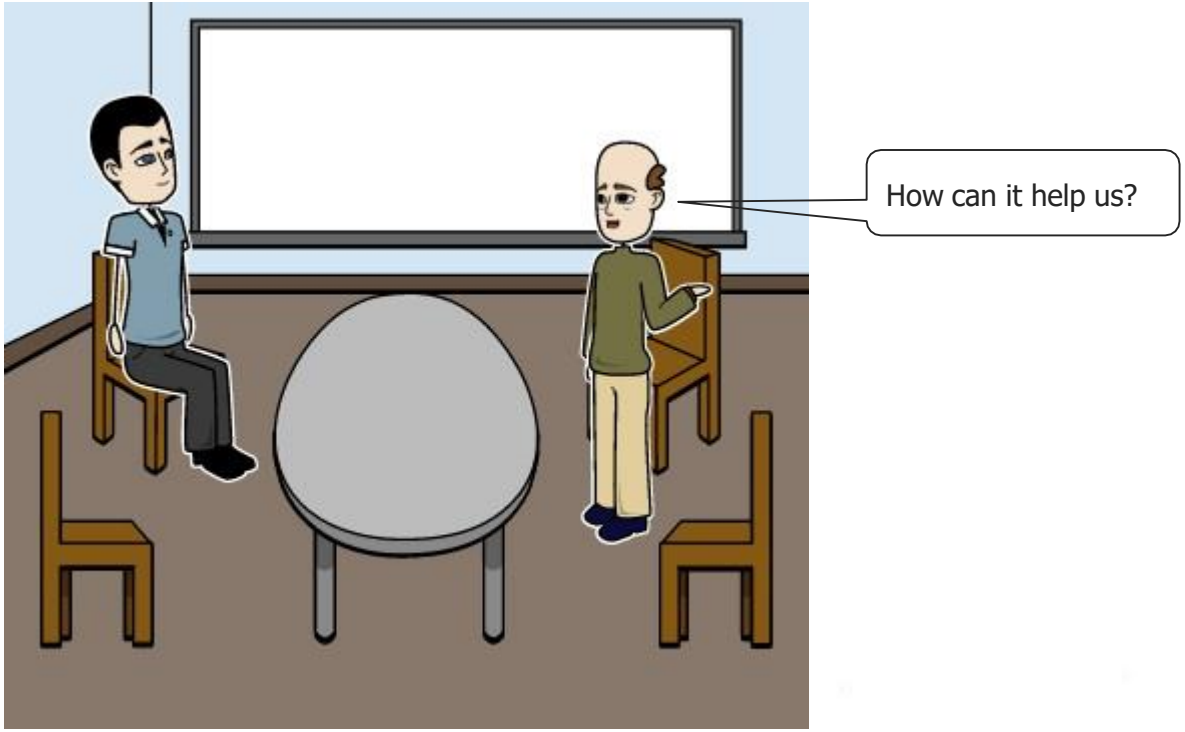
# A new Task for John!



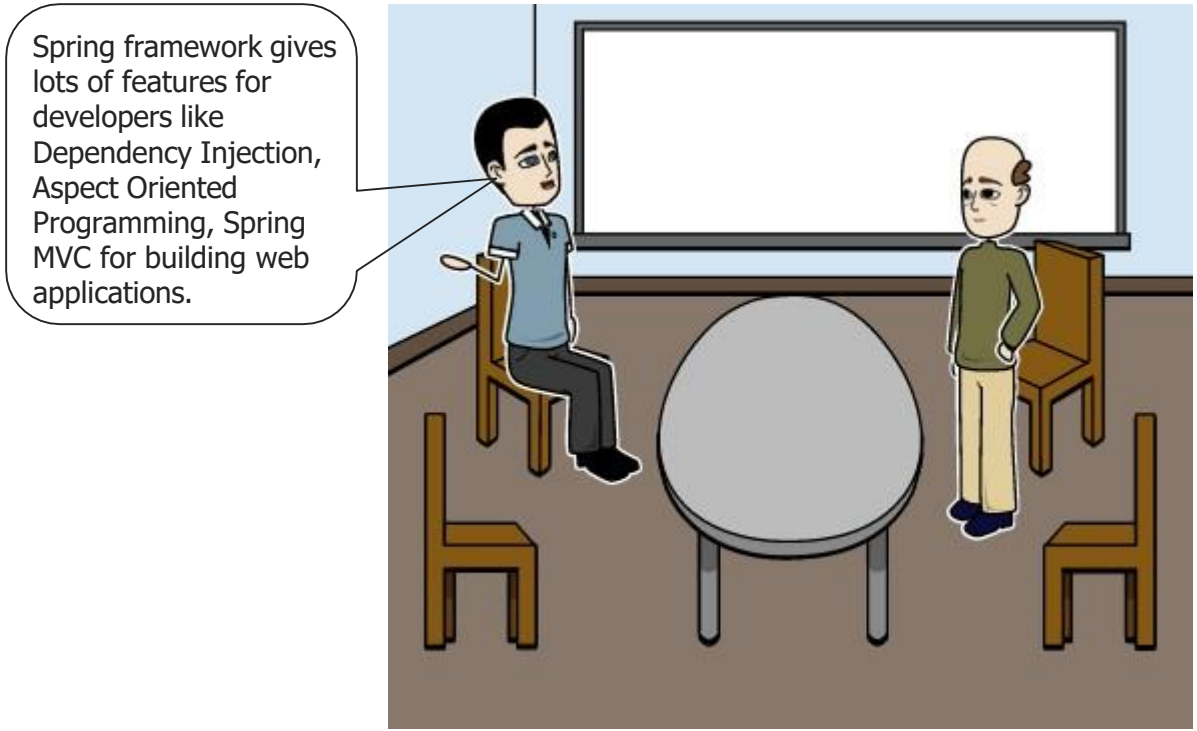
# John gives the Solution…



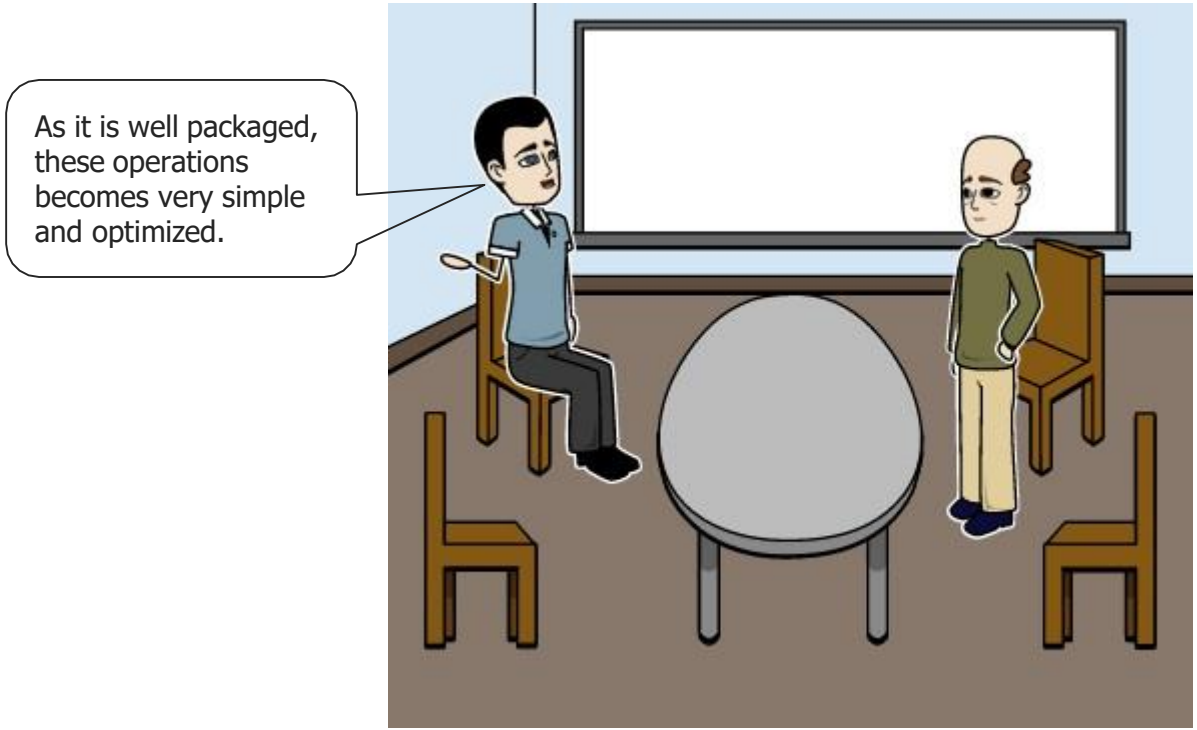
# John gives the Solution…



# John gives the Solution…



# John gives the Solution…




Let's Learn More….

# Introduction

→ Spring is a framework like Struts and Hibernate.

→ Spring is famous in the industry.

→ Learner should focus on learning more of Spring compared to other technologies now. Spring and Hibernate is extremely popular and is required in the industry.

→ Spring framework is lightweight.

→ Rod Johnson started SpringSource company and developed Spring framework for Apache 2.0 in June 2003.



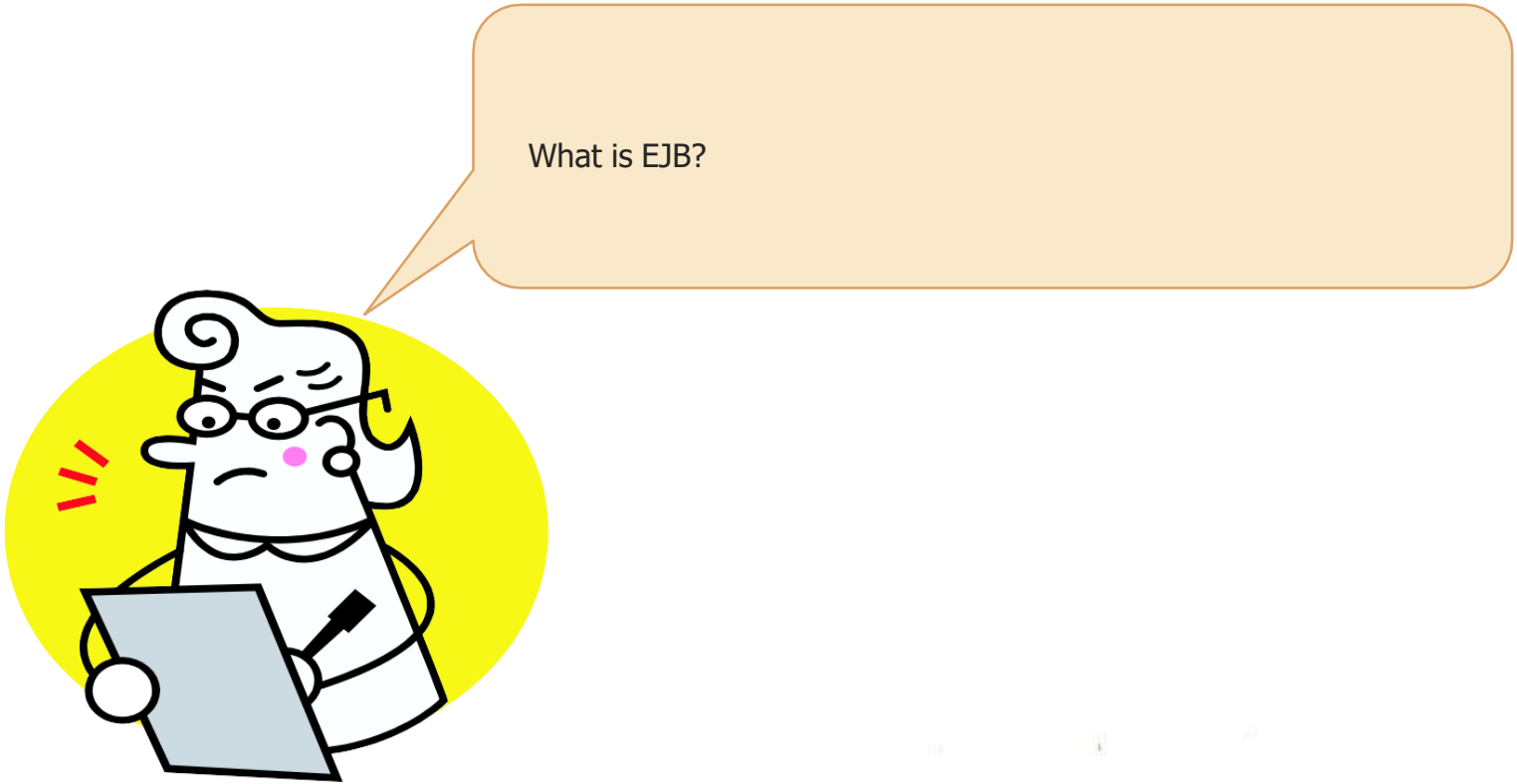
http://www.eweek.com/c/a/Application-Development/Top-10-Application-Development-Products-for-2008/

# Why Spring?

→ It is treated equivalent to EJB or the replacement of EJB or addition to EJB.

→ Spring can work on Tomcat itself instead of application server like Jboss, Glass Fish, Web Logic or Web Sphere. This is one more reason to consider Spring as Light Weight.

# Annie’s Question



# Annie’s Answer

EJB stands for Enterprise Java Beans. It is also one of the Java technology. It works on Application Server. Using EJB, we can develop enterprise applications.

# What is Spring?

→ Spring uses the existing technologies like JDBC, Struts, Transactions and Hibernate.

→ Spring has testing framework. So the testing becomes very easy.

# Annie’s Question

What are the advantages of a light weight application?

# Annie’s Answer

Memory and CPU consumption will be less, hence the performance of the system goes up.

# Spring Features/Modules

Inversion of Control → Done through Dependency Injection.

Aspect Oriented Programming → Enabling implementing cross-cutting concerns.

Data Access → Works with JDBC and Hibernate.

Model View Controller → Provides MVC Support through servlets and struts. This is web framework.

Remote Access Framework → Supports remote access through RMI, web service through SOAP and REST.

Convention over configuration → Spring Roo is another framework which eases the development. This is Rapid Application Development tool.

Authentication and Authorization → Spring security module provides this.

# Annie’s Question

What is the difference between Authentication and Authorization?

# Annie’s Answer

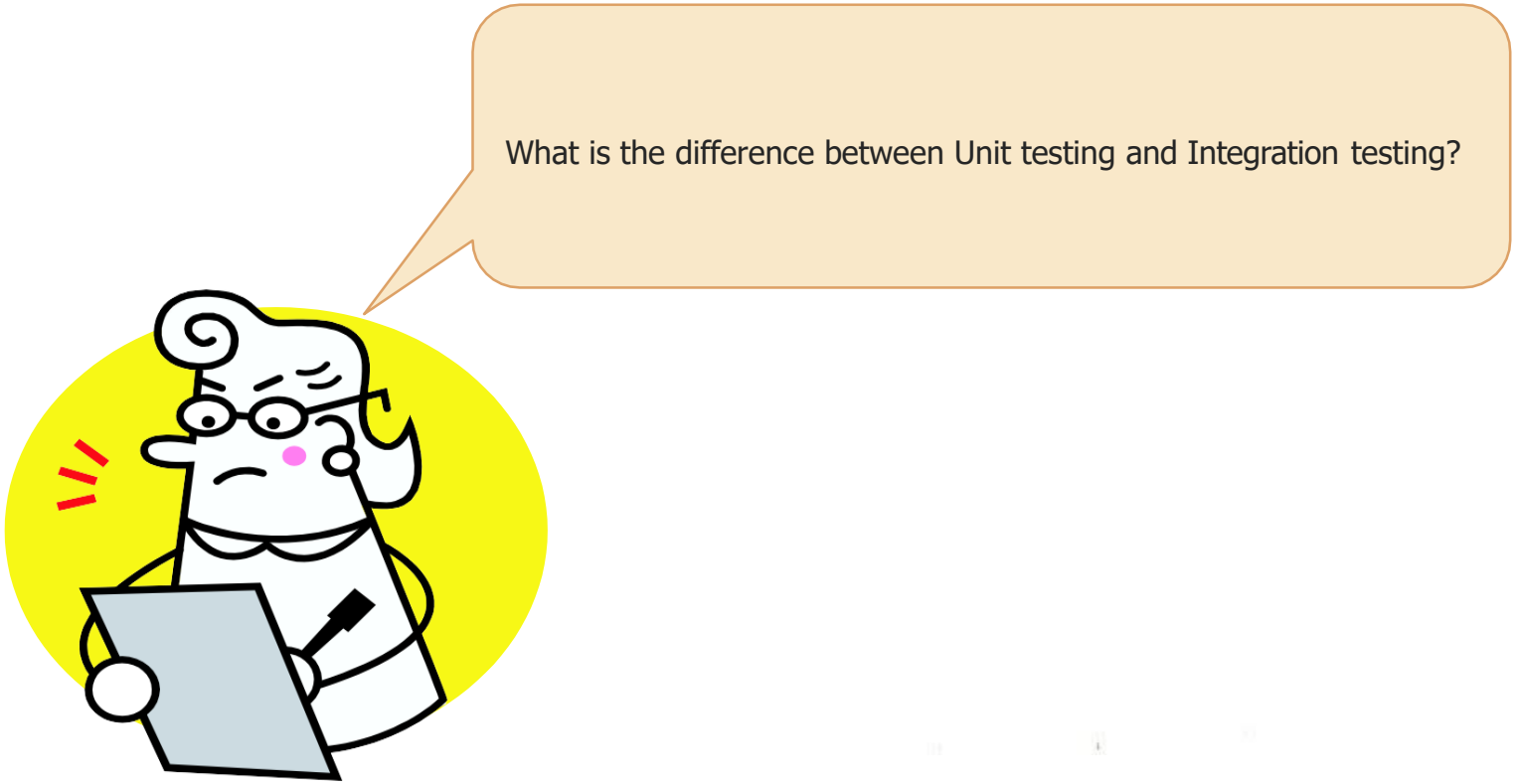
# Spring Features/Modules

Remote Management → JMX is the technology that is used to manage the system objects and devices like printers.

Messaging → Uses JMS to communicate through Message Queues.

Testing → Supports classes for writing unit test cases and integration test cases.

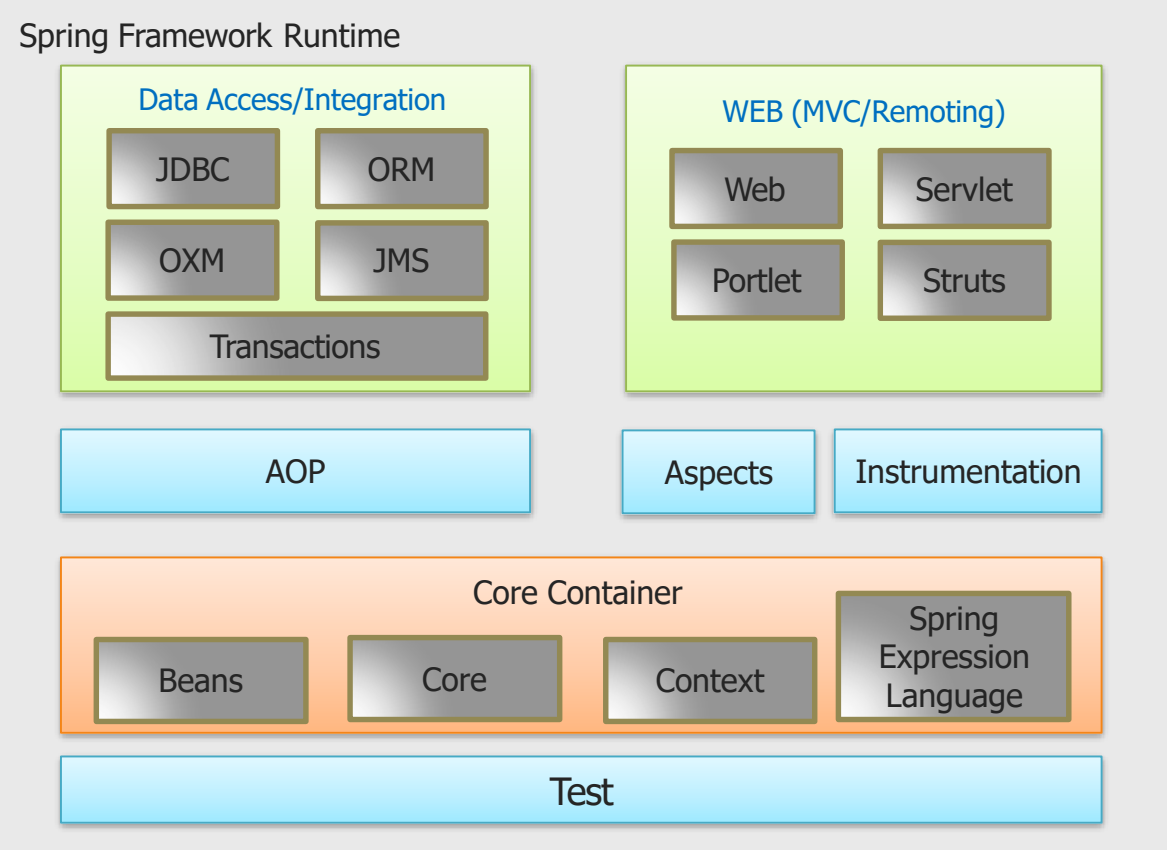
# Annie’s Question



# Annie’s Answer

Testing every module individually is Unit testing. When all the modules are combined and the project is integrated then testing the entire project is termed as integration testing.

# Spring Architecture



# Spring Architecture (Contd.)

Architecture has 5 Layers:

(1) Data Access/Integration
(2) Web
(3) Aspect
(4) Core
(5) Test

Data Access/Integration has JDBC, ORM, OXM, JMS and Transaction modules.

JDBC → JDBC code can be done using JDBC Template.

ORM → Can develop code using ORM tools like Hibernate, iBatis etc.

OXM → OXM Stands for Object to XML Mapping. It supports technologies like JAXB.

JMS → Can send JMS operations in this module.

Transaction → Supports both Programmatic and Declarative Transaction Management. Programmatic is handling in the program and declarative is handling either as part of Annotation or using XML.

# Spring Architecture – Web Layer

Web layer has following modules

(1) Web
(2) Servlet
(3) Portlet
(4) Struts modules.

Web → This module will initialize the IOC Container using servlet listener.

Servlet → Used for servlets.

Struts → Can write struts applications.

Portlet → Can write portlets. Portlet is a standalone application that can be used in a Web Portal.

# Spring Framework – Aspect and Test layer

AOP – Used for Aspect Oriented Programming. This will allow to define interceptors and point-cuts to separate them as required.

Aspects – This modules helps to integrate with AspectJ. AspectJ is another AOP framework.

Instrumentation – Instrumentation is the ability to monitor the level of products performance, to diagnose the errors and write the trace information. This module supports these.

Testing – Supports to test the spring modules using Junit framework.

# Spring Framework – Core Layer

This layer has

(1) Core
(2) Beans
(3) Context
(4) Spring Expression Language(SpEL)

Core → It has Dependency Injection(DI) and Inversion Of Control(IoC) features.

Beans → It implements BeanFactory through factory pattern.

Context → Used for ApplicationContext. It uses Core and Beans module.

Spring Expression Language → Used for querying and manipulating objects at runtime.

# IOC Container

→ IOC container/Spring container is the core component. The entire spring framework depends on this.

→ It creates the objects, configures them and makes it available for use.

→ Configuration information about objects is in XML file. Container reads this information and configures the objects.

→ Container manages the object from creation to destruction. It is like servlet container.

# Spring Container

Spring provides two containers:

(a) BeanFactory
(b) ApplicationContext

BeanFactory is used to create beans.

ApplicationContext → This is built on top of BeanFactory interface.

This interface provides internationalization support, publish events to the even listeners, integration with AOP.

It is better to use ApplicationContext instead of BeanFactory.

# Spring Containers – BeanFactory

BeanFactory can be created like this:

```
Resource resource=new ClassPathResource("applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

These two lines will load the applicationContext.xml which has configuration information and creates the specified beans given in the xml file.

# Spring Containers – ApplicationContext

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
```

This will load the applicationContext.xml and creates the context using ApplicationContext.

# Spring Containers – ApplicationContext (contd.)

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
```

This will load the applicationContext.xml and creates the context using ApplicationContext.

applicationContext.xml can define beans like this:

```
<bean id="hello" class="co.edureka.Hello">
    <property name="msg" value="Hello Spring Learners!"/>
</bean>
```

# XML file

→ XML file defines the id of the bean as hello.

→ Class to execute for this bean is co.edureka.Hello.

→ For the msg property "Hello Spring Learners!" value has been set.

```
package co.edureka;
public class Hello {

        String msg;

        public void setMsg(String message){
            msg=message;
        }

        public String getMsg(){
            return msg;
        }

      }
```

# Client to Test

Now we are all set to write the Test class where we will get the Hello type bean:

```
public class Test {

        public static void main(String[] args) {
           ApplicationContext c= new ClassPathXmlApplicationContext("applicationContext.xml");
           Hello hw_obj = (Hello) c.getBean("hello");
           hw_obj.getMsg();
        }


  }
```

Above code loads applicationContext.xml file. We have defined Hello type bean with id hello in the xml file.

We are calling the getBean(String s) method of applicationContext to get Hello type object.

getBean(String s) method returns object, so we have to cast it into Hello and then we are calling getMsg() method on the retrieved object.

# Bean Scope

Bean can be defined in the following scopes:

singleton → Only one instance of the of the bean can be created in the entire application.

prototype → Can create any number of instances of the bean.

request → One bean can be created per request of HTTP.

session → One bean can be created per session of HTTP.

global-session → Useful in portlet application. It is same as session in servlet based application.

# Bean – Life Cycle

init-method in the xml file specifies a method which needs to be invoked immediately after bean instantiation.

destroy-method in the xml file says what to do in a method just before bean is dying.

Syntax in the xml file is as follow:

```
<bean id="hello" class="co.edureka.Hello"  init-method="init"  destroy-method="destroy">
     <property name="msg" value="Welcome to bean life cycle..." />
</bean>
```

Define the init() method and destroy() method in the bean class.

# Bean – Postprocessor

→ BeanPostProcessor provides methods to be called before and after initializing the bean.

→ If something is to be done before and after initializing the bean then this facility can used.

# Bean – Postprocessor Example

```
package co.edureka;

public class SampleBean {



 public void init() {

     System.out.println("Bean is going through init.");
 }

 public void destroy() {

     System.out.println("Bean is getting destroyed.");
 }


 public SampleBean(){
     System.out.println("I am the constructor of SampleBean class");
 }

}
```

# Bean – Postprocessor Example (contd.)

```
package co.edureka;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;


public class DemoBeanPostProcessor implements BeanPostProcessor {

 public Object postProcessAfterInitialization(Object bean, String beanName)
         throws BeansException {

     System.out.println("After Initialization just after init life cycle event ");
     return bean;
 }

 public Object postProcessBeforeInitialization(Object bean, String beanName)
         throws BeansException {

     System.out.println("Before initialization just before init life cycle event ");
     return bean;
 }

}
```

# Bean – Postprocessor Example (contd.)

```
<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="bean" class="co.edureka.SampleBean" init-method="init" destroy-method="destroy"/>

<bean class="co.edureka.DemoBeanPostProcessor"/>

</beans>
```

# Bean – Postprocessor Example (contd.)

```
package co.edureka;

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

public class TestClass {

@SuppressWarnings({ "resource", "unused" })
public static void main(String args[]){
    ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
    SampleBean obj=(SampleBean)context.getBean("bean");
 }

}
```

When you run the TestClass you will get following output:

I am the constructor of SampleBean class
Before initialization just before init life cycle event
Bean is going through init.
After Initialization just after init life cycle event.

Methods of DemoBeanPostProcessor runs before and after the init method.

# LAB

# Bean – Inheritance

Bean can have configuration information like constructor arguments, initialization methods etc.

A child bean inherits the configuration information from parent bean.

```
<bean id="helloDerived" class="bean_inheritance.HelloDerived" parent="helloBase">
 <property name="message2" value="Derived class string 1"/>
 <property name="message3" value="Derived class string 2"/>
</bean>
```

Here bean helloBase is used as the parent bean. All the configuration properties of the base bean can be used.

# Bean – Inheritance

applicationContext.xml

```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="child"  parent="father">
   <property name="first_name" value="Shane"/>
  </bean>

  <bean id="father" class="co.edureka.Father">
  <property name="first_name" value="Krick"/>
  <property name="last_name" value="Watson"/>
  </bean>

 </beans>
```

Father class

```java
package co.edureka;

public class Father {
   String first_name;
   String last_name;

   public String getFirst_name(){
       return first_name;
   }

   public String getLast_name(){
       return last_name;
   }

   public void setFirst_name(String first_name){
       this.first_name=first_name;
   }
   public void setLast_name(String last_name){
       this.last_name=last_name;
   }

 }
```

# Bean – Inheritance

```
package co.edureka;

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

public class TestClass {

  public static void main(String args[])
  {
       ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
       Father parent=(Father)context.getBean("father");

       Father child=(Father)context.getBean("child");
       System.out.println(child.getFirst_name()+" "+child.getLast_name());
  }

 }
```

Note for the child bean father bean have been set as Parent Bean in xml file. So child bean can access its last_name property from parent bean.

When you run the TestClass you will get output as Shane Watson.

# Dependency Injection

→ Dependency Injection is a pattern for creating the objects that depends on other objects not knowing at compile time.

→ DI reduces the programming code.

→ DI helps the code to be loosely coupled.

There are two types of Dependency Injection:

(1) Constructor based
(2) Setter Method based

# Dependency Injection (contd.)

Observe the following code:

```
package co.edureka;

public class A {
 private B b;
 public A(){
     b=new B();
 }

}
```

Observe the following code and tell which is beneficial compared to both.

```
package co.edureka;

public class A {
 private B b;
 public A(B b){
     this.b=b;
 }

}
```

# Dependency Injection (contd.)

In the second example:

B object is passed to A constructor. A need not worry about B functionality at all. B functionality is implemented separately.

Object of B will be passed from XML file.

Add **<constructor-arg ref="b"/>** in the A Bean definition as shown:

```
<beans>
 <bean id="a" class="co.edureka.A">
   <constructor-arg ref="b"></constructor-arg>
 </bean>
 <bean id="b" class="co.edureka.B"></bean>
</beans>
```

This means B object is passed for A constructor.

# LAB

# Dependency Injection – Setter based

Instead of having the object passed to the constructor, set the property of the A as part of setter method and pass the property value in the XML file for A Bean.

This is Dependency Injection through setter method.

```
package co.edureka;

public class A {
 private B b;
 public void setB(B b){
     b=this.b;
 }

}
```

And specify the below in XML file:

```
<beans>
 <bean id="a" class="co.edureka.A">
   <property  name="b" ref="bBean"/>
 </bean>
 <bean id="bBean" class="co.edureka.B"></bean>
</beans>
```

# Spring DI - Collections

Apart from setting a single value for a property in XML, it is possible to set the list of values to a property in XML through collections.

List, Map and Set can be used for setting the collection values in XML.

Code snippet is like this:

```
<bean>
<!- This will call setNoSqls(java.util.List) setter method in class -->
<property name="noSqls">
     <list>
      <value>HBase</value>
      <value>MongoDB</value>
      <value>Neo4j</value>
     </list>
</property>
</bean>
```

# Spring DI – Collections Example

applicationContext.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="db" class="co.edureka.Database">

   <!-- This will result in a call to setNoSqls(java.util.List) method -->
   <property name="noSqls">
     <list>
        <value>HBase</value>
        <value>MongoDB</value>
        <value>Neo4j</value>
     </list>
   </property>
  </bean>

 </beans>
```

Database class

```
package co.edureka;

import java.util.List;

public class Database {
List<String> noSqls;

public void setNoSqls(List<String> l){
noSqls=l;
  }
public List<String> getNoSqls(){
return noSqls;
  }

}
```

# Spring DI – Collections Example(contd.)

TestClass

```
package co.edureka;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestClass {

 public static void main(String[] args) {
     @SuppressWarnings("resource")
     ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
     Database b=(Database)context.getBean("db");
    System.out.println(b.getNoSqls());

 }

}
```

When you run the TestClass it will print [HBase, MongoDB, Neo4j]

# Spring – Auto Wiring

→ Spring container can set relationships between beans without declaring them in XML file using Autowiring feature.

→ Autowiring can be done using @Autowired annotation.

→ Use @Autowired above property to have Autowiring.

# Spring – Auto Wiring Example

Class A

```
package co.edureka;

public class A {

public void aMethod(){
    System.out.println("This is aMethod of class A");
 }

}
```

Class B

```
package co.edureka;
import org.springframework.beans.factory.annotation.Autowired;

public class B {
private A a;

 @Autowired
   public void setA( A a ){
      this.a = a;
   }
   public A getA(){ return a;}

   public void bMethod(){
    System.out.println("This is class B method");
    a.aMethod();
  }

}
```

# Spring – Auto Wiring Example(contd.)

TestClass

```
package co.edureka;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestClass {

 public static void main(String[] args) {
     ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");
     B b=(B)context.getBean("b");
     b.bMethod();
 }

}
```

applicationContext.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"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-3.0.xsd">

 <context:annotation-config/>

<bean id="a" class="co.edureka.A"/>
<bean id="b" class="co.edureka.B"/>

</beans>
```

# Spring – Auto Wiring Example(contd.)

## Lets go through the code:

We have two classes under package co.edureka, class A and B. Class A have a method aMethod.

→ In Class B we have declared a Class A type reference and setter and getter method along with a method bMethod.

→ So Class B have dependency on class A.
Note the @Autowired annotation on setter method in class B.

→ In TestClass we are getting the bean with id b that we have defined in applicationContext.xml file.

→ Then we are calling the bMethod() on that bean.

Following is the o/p produced on the execution of TestClass.

This is class B method

This is aMethod of class A

# QUESTIONS

# Agenda of the Next Class

In the next module you will be able to:

→ Understand what is AOP and how to integrate AOP with Spring.
→ Understand how to use JDBC with Spring
→ Learn how to integrate hibernate with Spring
→ Understand Ajax and its usage
→ Understand core and J2EE design patterns.

# Pre-work

- Preparation for the next module: Go through the concepts and programs of spring.


Thank you!