Sitesbay.com

A Perfect Place for All Tutorials Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring |

Java Projects | C | C++ | DS | Interview Questions | Html | CSS | Html5 | JavaScript | Ajax | Angularjs | Basic Programs | C Project

www.sitesbay.com

[SPRING]

[Thursday, June 05, 2014] **Faculty: Mr. Sekhar Sir**

Inventor of SPRING: Rod Johnson

1995	---	Java Applets	---	Gaming
1996	---	Java Beans	---	Business Logic
1998	---	EJB	---	Business Logic + Services
2003	---	Spring	---	Business Logic + Services(Extra Advantages)

Q. Why Spring?

Ans:-

> Java is introduced in 1995 and at the initial days of JAVA the famous programming is JAVA applets.
> With Java applets industry developers assumed that java is only for gaming applications using applets.
> To come-out of industry path on java, SUN Microsystems introduced JAVA Beans.
> With Java Beans, industry have identified adding services to the business is a problem.
> To solve problem with beans, SUN Microsystems introduced EJB (Enterprise Java Beans) technology.
> With EJB business logic and services can be added together but again there are some problems are identified with EJB.
  (1) Many files are needed to develop an application.
  (2) EJB is a server-side technology some if any problem is occurred then we need to reload the application after modification, and then we need to restart the server. It is a time taken process and a burden on developers.
> To solve the problems of working with EJB, ***Rod Johnson*** has created Spring Framework.
> Spring Framework makes application development easy by injecting services to business logic created in ordinary java classes (POJO).
> Finally Spring Framework is introduced as an alternate for EJB.

[Friday, June 06, 2014]

Q. What is Framework?

Ans:-

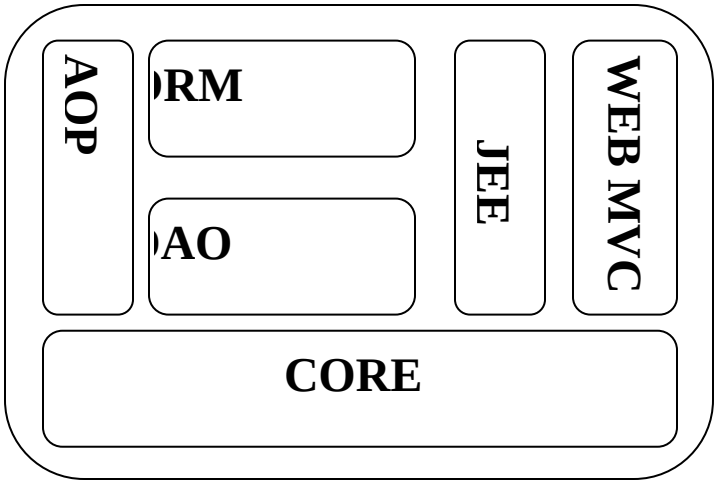
> While developing java projects, most of the projects contain some common functionality.
> If the common functionalities are coded by the developers in every project then it will increase the burden on developers.
> In order to reduce the burden on developers, third party vendors started providing frameworks.
> *A framework is defined as an abstraction layer on top of the existing technologies and concept of java.*
> With the help of frameworks, the burden on developer reduces because around 50% of code of a project will be given by a framework only.
> With the help of frameworks, a project can be developed fastly, delivered to the clients as early as possible and it can be easily maintained.
> A framework will reduce the burden on developers. So today almost all java real-time projects are developing through frameworks.
> A framework is not a technology. It is a layer on top of existing technologies.
> Frameworks are 2 types
  (a) Invasive (b) Non-invasive

> Invasive framework will force a developer to extend a framework or implement a framework interface while creating projects.
For example, Struts framework is invasive.
> Non-invasive framework does not force developers to extend a framework class or implement a framework interface.
For example, Hibernate and spring are non-invasive framework.

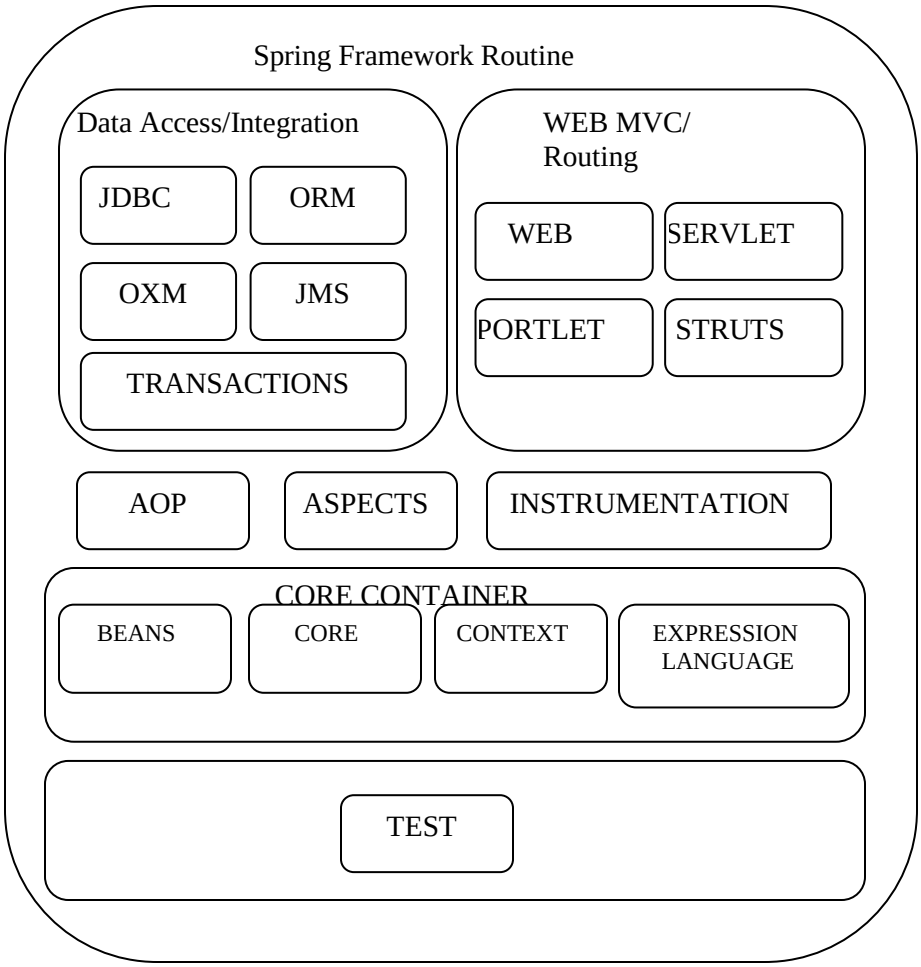
[Monday, June 09, 2014]

Spring Framework Modules

> Spring framework is a complete and modular framework. It means, we can develop either all modules of the application using spring framework or we can use spring framework for some modules of the application.
> Spring is a light-weight, non-invasive and a loosely-coupled framework for building java projects.
> Spring is very popular framework, due to the following reasons.
(1) It is a light-weight framework.
(2) It is a loosely-coupled framework
(3) It reduces boiler-plate code.
(4) It is easily testable, because of its own container.
> Boiler-plate code is nothing but it is a repetitive code. The framework has reduced the repetitive code with help of template classes.
> If we consider the technology by SERVLET, JSP or EJB then for testing them a third-party provided container is needed. But in case of spring framework, the framework itself provided a container and it will be automatically started and automatically stopped with our application.
> Spring framework is a modular framework which contains different modules for developing application.
> In spring 2.x, the modules are divided like the following.



> In Spring 3.x, the modules are shuffled that is DAO and ORM modules are combined as data access and integration and JEE module is removed and it is added as a some part into data access and integration and some part into web.
> In Spring 3.x, a new module is added called **TEST.**



Spring Core Module:-

- This core module is the fundamental module of the framework.
- This core module tells about how to configure spring beans, their dependencies in XML file.
- This core module also provides Spring Inversion of Control (IOC) container (Spring Container).

[Tuesday, June 10, 2014]

AOP Module (Aspect Oriented Programming):-

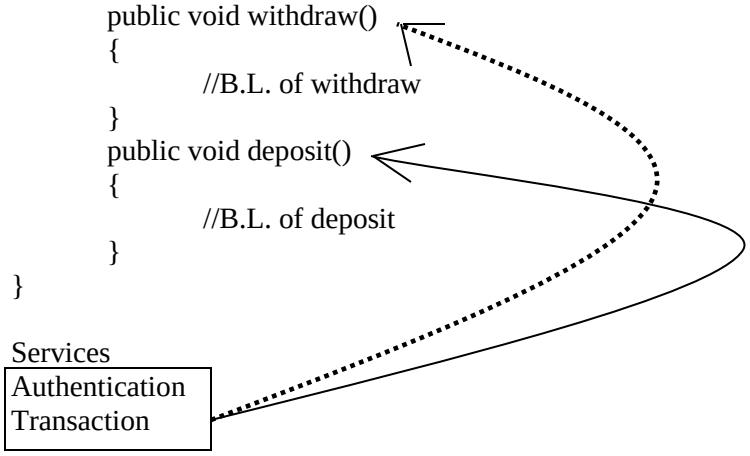
- While implementing business logics in a project, to make business logics as efficient, we attach some services to the business logics.
- If the same services are added to multiple business methods then we will get redundancy code in the project.
- To make ***cross-cutting-concerns (services) as re-usable*** for multiple business methods, we separate the services from business.
- ***AOP is a mechanism, which adds services to business at runtime.***

  Before applying the AOP:-

```
AOP Module
public class Account
{
        public void withdraw()
        {
                //Authenticaton
                //Transaction
```

```
                //B.L. of withdraw

}
        public void deposit()
        {
                //Authentication
                //Transaction
                //B.L. of deposit
        }
}
//After applying AOP:-
public class Account
{
        public void withdraw()
        {
                //B.L. of withdraw
        }
        public void deposit()
        {
                //B.L. of deposit
        }
}
```



Data Access & Integration module:-

- > This module is an abstraction layer on top of JDBC technology and ORM tools.
- > This module makes database access as simple by avoiding boiler-plate code
- > Boiler-plate code is a repetitive code.
- > This module of Spring framework provides some template classes, for reducing boiler-plate code problem and also memory leakage problem.

Web and remoting Module:-

- > This module is for distributed application development using spring framework.
- > Spring web allows spring developers to integrate Struts with Spring framework or it allows to the developer to write MVC application also with Spring framework.
- > Spring remoting tells about how to invoke the services of other beans which are running at some other JVM across network.

Test Module:-

- > This module is newly introduced in Spring 3.x
- > This test module prepares mock objects needed for developing test cases.
- > In real-time this spring test module is not used. Instead some third party tools like Easymock, mockito, TestNG etc are used for creating mock objects.

[Wednesday, June 11, 2014]

Spring Core Module:-

POJO Class: - Plain Old Java Object

- > A POJO class is nothing but, it is an ordinary java class but not a special class.
- > An ordinary java class means it is a java class which is not going beyond of java API.
- > A POJO class will not be bind to any technology or a framework.

> A POJO class does not have any other special rules.

Example1:-

```
class MyClass implements Serilizable
{}
```

----> Serilizable is a marker interface from java.lang package. So my class is a POJO.

Example2:-

```
class MyClass extends GenericServlet
{}
```

--> GenericServlet is an abstract class of Servlet API. So MyClass is not a POJO Class

Example3:-

```
class public class MyClass extends Exception
{}
```

--> Exception is a class of java API. So MyClass is aPOJO

Example4:-

```
class MyClass
{
        void process(ServletRequest request
        {}
}
```

----> Her my class is a POJO class because class is not binded to the servlet technology.

Example5:-

```
public class MyClass extends org.apache.struts.action.ActionForm
{}
```

-->Here MyClass is not a POJO class, because it is binded with struts framework.

Q. Can we call a JAVA class with annotations as a POJO class or not?

Ans:- After avoiding annotations, if a java class is acting as a pojo then that java class with annotation is also a pojo class.

Example1:-

```
@Entity
public class Student
{
@Id
private int studentID;
...
}
```

--> This is a POJO class

Example2:-

```
@WebServlet
public class MyClass extend HttpServlet
{}
```

--> My class is not a POJO class.

JAVA Bean:-

> Re-usable classes in java are termed as beans.

> A class is called as a java bean, if it follows the below rules.

(1) Class must be public.
(2) Class must contain default constructer.
(3) Each private property should contain a setter or a getter or both methods.
(4) Class should implements Serializable interface not other interface.

> *** *Every java bean is a POJO class, but every POJO class is not a java bean*.

```
public class MyClass implements Serializable
{
        public MyClass(){}
        ……..
}
```
---->MyClass is java bean and also POJO class.
```
class MyClass implements Serilizable
{
        public MyClass(){}
}
```
---->MyClass is not a java bean because it is not public, but a POJO class

[Thursday, June 12, 2014]

Spring Bean:-

- Every java class which is a part of a spring application is called as a **Spring Bean**.
- A spring bean does not have any special rules.
- In spring framework, we call either a user-defined class or a pre-defined class as spring beans.
- If a spring bean class is a public and has a default constructer then it will become as a java bean.
- **** The difference between a spring bean and a java bean is, a java bean must be public but a spring bean may not be public and a java bean must contain a default constructer but a spring bean may not contain default constructer.*

Example1:-
```
class Travel
{
        void startJourney(){}
}
```
---> Here Travel class is a spring bean and a pojo, but not a java bean.

Example2:-
```
class Reporter implements ReportWriter
{
}
```
ReportWriter is an pre-defined interface belongs to java API

---> Here Reporter is a spring bean and pojo class, but it is not a java bean.

Example3:-
```
public class Device implements org.springframework.beans.factory.BeanFactory
{

}
```
---> Here Device is a spring bean not a pojo and also not a java bean

Tight Coupling and Loose Coupling between Objects:-

- While developing applications, it contains many classes and one class calls another class, to provide a service to the clients.
- In spring framework, if one class calls another class then we call it as ***bean collaboration***.
- This bean collaboration is also called as bean dependency.
- For example, if class A calls services of class B and class C then we call class B and class C are dependencies of A.

[Friday, June 13, 2014]

> In java, a class can get its *dependencies*by using one of the following 4 ways.
(1) A class can directly create its dependency object with the new keyword.
(2) A class can get its dependencies by calling a factory method.
(3) A class can collect its dependency object from external memory like a registry.
(4) Some external person (entity) can inject class to object into class1.

> When one object depends on another and if modifications done on dependency object is demanding the modifications in a caller object also then it is said to be ***tight-coupling*** between objects.

> For example, consider the following 2 classes

```
class Travel
{
        private Car car = new Car();
        void startJourney()
        {
                car.move();
        }
}
class Car
{
        void move();
}
```

- In the above example, if a method in Car class is changed from move() to go() then, in Travel class also we need to change in startJourney() method. It means if dependencies object i.e. Car is changed then caller i.e. Travel also need changes. It is a tight-coupling between objects.
- In the above example, suppose if Travel class want another class, says Bike then we need to modify the travel class by replacing Car with Bike. This is also ***tight-coupling***.

> In order to avoid tight-coupling between objects, we should follow the below principles.
(1) Use ***POJI-POJO*** model for creating dependencies.
(2) Apply ***dependency-injection*** mechanism.

- For example, create an interface Vehicle and define implementation classes like Car, Bike, and Flight.
- The importance of this model is, methods names in multiple classes will be same and a class cannot change method name.
- In Travel Class, take a reference of type Vehicle interface, so that it can store any implantation class object of that interface

```
interface Vehicle
{
    void move();
}
class Car implements Vehicle
{
    public void move()
    {
```

```
    }
}
class Bike implements Vehicle
{
    public void move()
    {
    }
}
class Flight implements Vehicle
{
    public void move()
    {
    }
}
class Travel
{
    private Vehicle vehicle;
    //setter method
    public void setVehilce(Vehicle vehicle)
    {
            this.vehicle=vehicle;
    }
    void startJourney()
    {
    vehicle.move();
    }

}
```

- A container injects one vehicle object needed from Travel class. So there is no need to make any changes in Travel class.
- If the container injects the dependencies required for a class then it is called dependencies injection.
- For example, every servlet object needs two dependencies request and response. A servlet container injects the dependencies to the servlet object. So it is also called dependency injection.

[Monday, June 16, 2014]

Types of Dependencies Injection:-

- *Setter injection*
- *Constructer injection*
- *Interface injection*

- If the dependencies are injected through setter methods defined in a caller class by container then it is called *setter injection*.
- If the dependencies are injected through constructer (s) of caller class by container then it is called *constructer injection*.
- If the dependencies are injected by calling the interface methods by container then it is called *interface injection*.

- In spring framework, interface injection is only possible at few cases during the life cycle of a bean. So we say that spring framework doesn't support interface injection.
- A spring container identifies whether a setter injection is needed or constructer injection is needed through spring configuration file (XML file).
- Spring configuration means defining the spring beans and their dependencies into XML file.
- A spring configuration file will act as a mediator between a spring programmer and spring container.
- In a spring configuration, a spring configuration file can be names as <anyname>.xml

[Tuesday, June 17, 2014]

```
public class Travel
{
        private Car car;
}
public class Car
{
}
<beans>
<bean id="id1" class="com.sathya.spring.Travel">
<property name="car" ref="id2"/>
</bean>
<bean id="id2" class="com.sathya.spring.Car">
</bean>
</beans>
-------------------------------------------------
public class Travel
{
        public Travel(Car car)
        {
                this.car=car;
        }
}
public class Car
{
}
<beans>
<bean id="id1" class="com.sathya.spring.Travel">
<constructer-arg name="car" ref="id2"/>
</bean>
<bean id="id2" class="com.sathya.spring.Travel">
</bean>
</beans>
```