Spring Framework

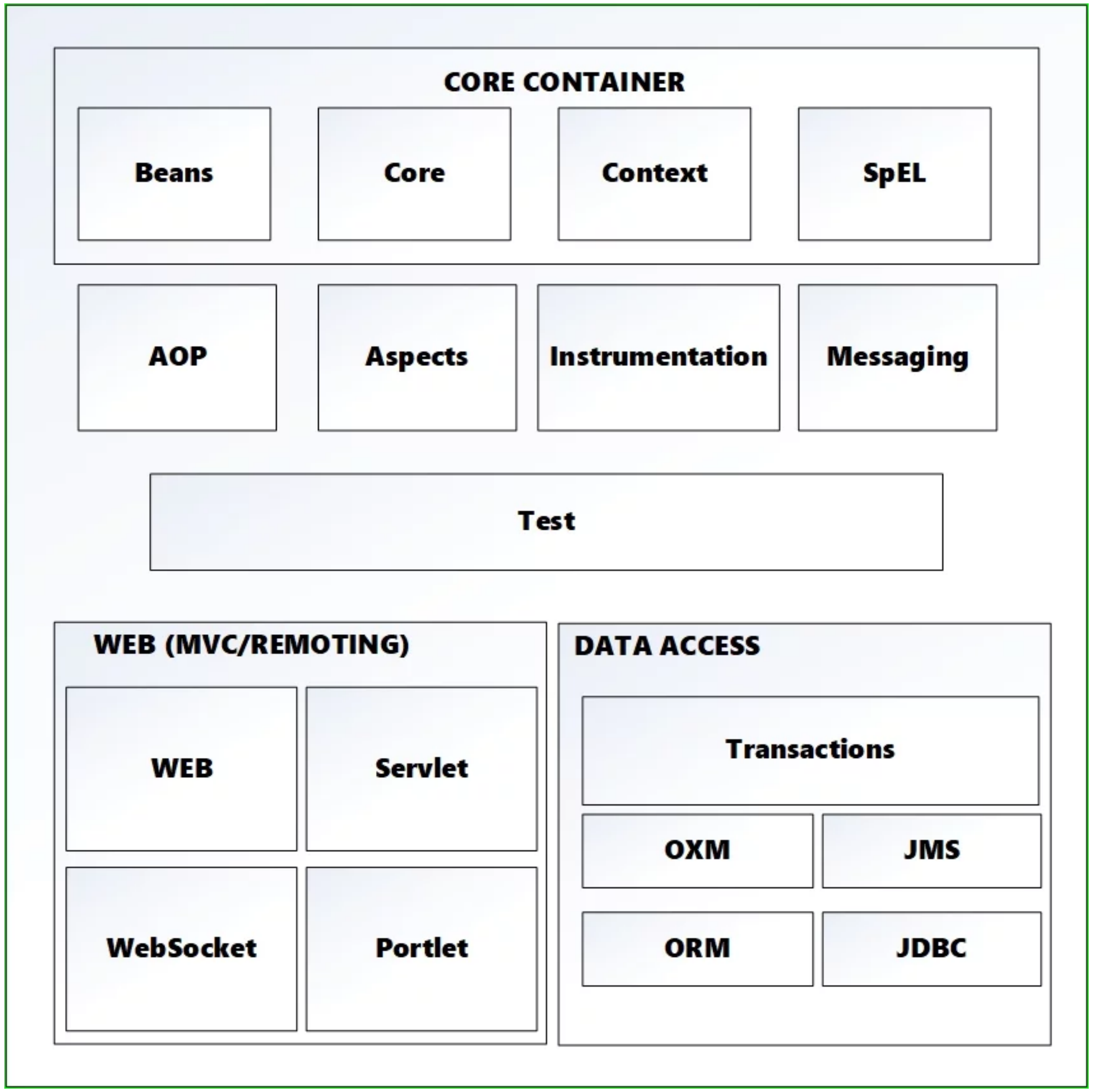
Introduction to Spring Framework

- The **Spring Framework** is an open source application development **framework**
- Spring Provides comprehensive infrastructural support for developing enterprise Java applications very easily
- Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, EJB, JSF etc.
- The spring framework was designed by **Rod Johnson**.

Benefits of Using Spring Framework

- Develop enterprise-class applications using POJOs
- Spring makes use of some of the existing technologies like ORM frameworks, Logging frameworks, etc..
- Good web MVC framework
- Best Transaction Management
- Light weight comparing EJBs
- Provides good Testing support
- Good security and logging service etc

Spring Framework Architecture



Core Components

- The **Bean** module is responsible for creating and managing Spring Beans – is application context structure unit

- The **Core** module provides key parts of the framework including IoC and DI properties.

- **Context** is built on the basis of Beans and Core and allows you to access any object that is defined in the settings. The key element of the Context module is the ApplicationContext interface.

- The **SpEL** module provides a powerful expression language for manipulating objects during execution.

Web Components

- The **Web** module provides functions such as multipart file-upload functionality, downloading files, creating web application, rest web service etc.

- **Web-MVC** contains a Spring MVC implementation for web applications.

- **Web-Socket** provides support for communication between the client and the server, using Web-Sockets in web applications.

- **Web-Portlet** provides MVC implementation with portlet environment

Data Access

- **JDBC** provides an abstract layer of JDBC and eliminates the need for the developer to manually register the monotonous code associated with connecting to the database.

- Spring **ORM** provides integration with popular ORMs such as Hibernate, JDO, which are implementations of JPA.

- The **OXM** module is responsible for linking the Object / XML – XMLBeans, JAXB, etc.

- The **JMS** (Java Messaging Service) module is responsible for creating, sending and receiving messages.

- **Transactions** supports transaction management for classes that implement certain methods and POJOs.

AOP, Aspects and Instrumentation

- These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code.
- The aspects module provides support to integration with AspectJ.

Test

- This layer provides support of testing with JUnit and TestNG.

Spring Core Concepts

- **Inversion of Control** – this is the principle of object-oriented programming, in which objects of the program do not depend on concrete implementations of other objects, but may have knowledge about their abstractions (interfaces) for later interaction.

- **Dependency Injection** – 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.

- **Aspect oriented programming** – a programming paradigm that allows you to distinguish cross-through (functional) functionality in application. These functions, which span multiple application nodes, are called cross-cutting concerns and these cross-cutting notes are separated from the immediate business logic of the application. In OOP, the key unit is the class, while in AOP, the key element is the aspect. DI helps to separate application classes into separate modules, and AOP helps to separate cross-cutting concerns from the objects they affect

IoC Container

The core of the Spring Framework is its **Inversion of Control** (Ioc) container.

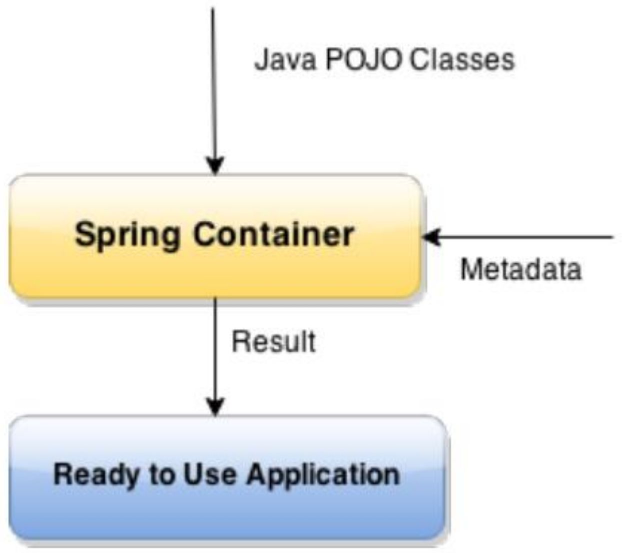
The container will create the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction

It is responsible

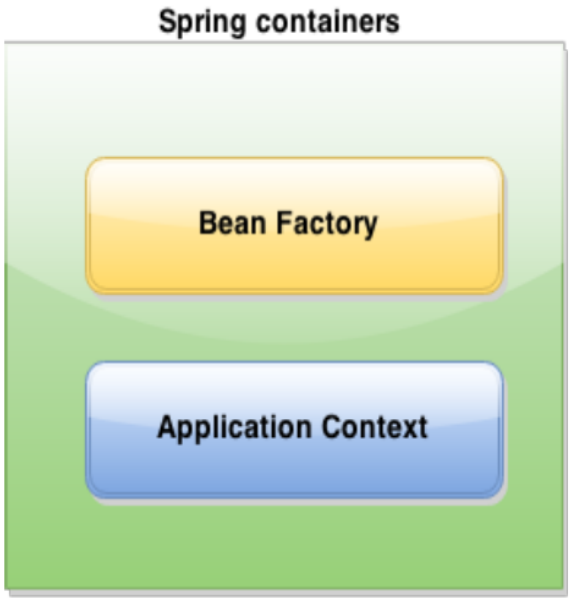
- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

Types of IoC containers are:

- **BeanFactory**
- **ApplicationContext**



Contd..



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

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

Dependency Injection

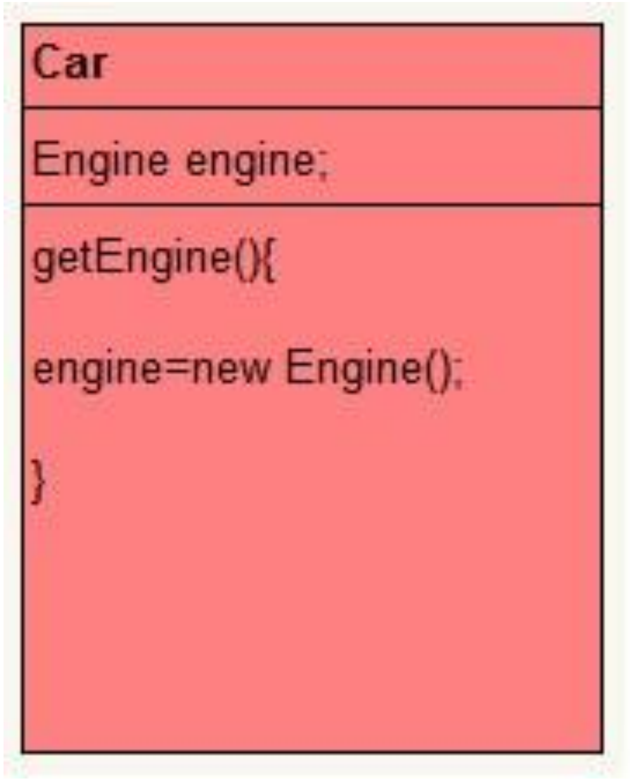
- It is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.

- Dependency Injection makes our programming code loosely coupled.

- The basic concept of the dependency injection (also known as Inversion of Control pattern) is that you do not create your objects but describe how they should be created.

Normal Way:

- There are many ways to instantiate a object.
- So here **Car** class contain object of Engine and we have it instantiated using new operator.

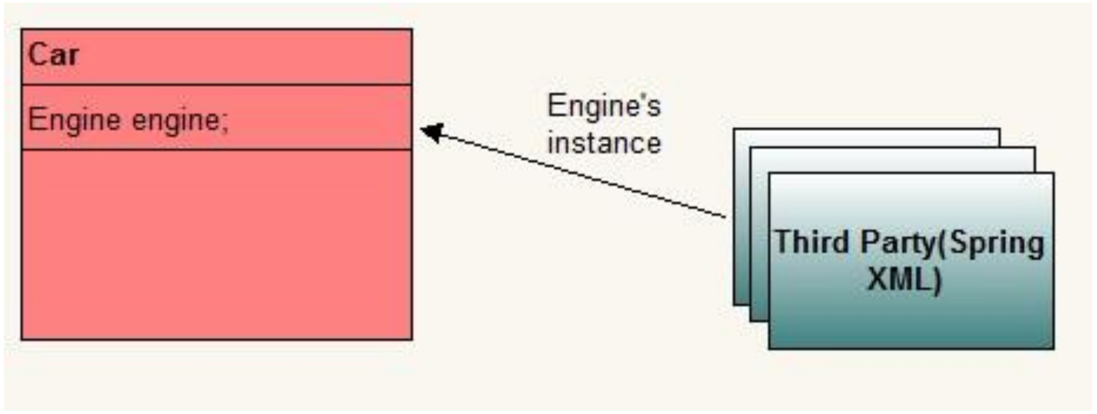


```
Car
Engine engine;
getEngine(){

engine=new Engine();

}
```

Without DI

With help of Dependency Injection:

- **Car** needs object of **Engine** to operate but it outsources that job to some third party.
- The designated third party, decides the moment of instantiation and the type to use to create the instance.
- The dependency between class **Car** and class **Engine** is injected by a third party.
- Whole of this agreement involves some configuration information too. This whole process is called dependency injection.



With DI

Dependency Injection

Spring framework provides two ways to inject dependency

- By **Constructor:** Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

- By **Setter method:** Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

Constructor vs Setter injection

- **Partial dependency**: can be injected using setter injection. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.

- **Overriding**: Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.

- **Changes**: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

Spring Bean scopes

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

In Spring, bean scope is used to decide which type of bean instance should be return from Spring container back to the caller.

1. **singleton** – Scopes a single bean definition to a single object instance per Spring IoC container.
2. **prototype** – Return a new bean instance each time when requested
3. **request** – Return a single bean instance per HTTP request.
4. **session** – Return a single bean instance per HTTP session.
5. **globalSession** – Return a single bean instance per global HTTP session.

In many cases, spring's core scopes i.e. singleton and prototype are used. By default scope of beans is singleton.

Spring Configuration

There are two ways via which you can inject dependency in spring

- By configuring XML.
- By using annotation.

- If you have done both i.e. used annotations and XML both. In that case, XML configuration will override annotations because XML configuration will be injected after annotations.
- Annotations based configuration is turned off by default so you have to turn it on by entering into spring XML file.

```
<context:annotation-config/>
<!-- beans declaration goes here -->
</beans>
```

Annotation based Configuration

@Required:

- The @Required annotation applies to bean property setter methods.

@Autowired:

- The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.

@Qualifier:

- The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.

Autowiring

- Autowiring feature of spring framework enables you to inject the object dependency implicitly.

- To enable it, just define the “**autowire**” attribute in.

- The Spring container can **autowire** relationships between collaborating beans without using and elements which helps cut down on the amount of XML configuration

- It internally uses setter or constructor injection.

- Autowiring can't be used to inject primitive and string values. It works with reference only.

Autowiring modes

- **no:** Default, no auto wiring, set it manually via “ref” attribute as we have done in dependency injection via settor method post.

- **byName:** Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file and it tries to match it with name of bean in xml configuration file.

- **byType:** Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType* in the XML configuration file. It then tries to match and wire a property if its **type** matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown.

- **contructor:** byType mode in constructor argument.

- **autodetect:** Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*.

Spring JDBC Template

- It is a mechanism to connect to the database and execute SQL queries

JDBC Template class:

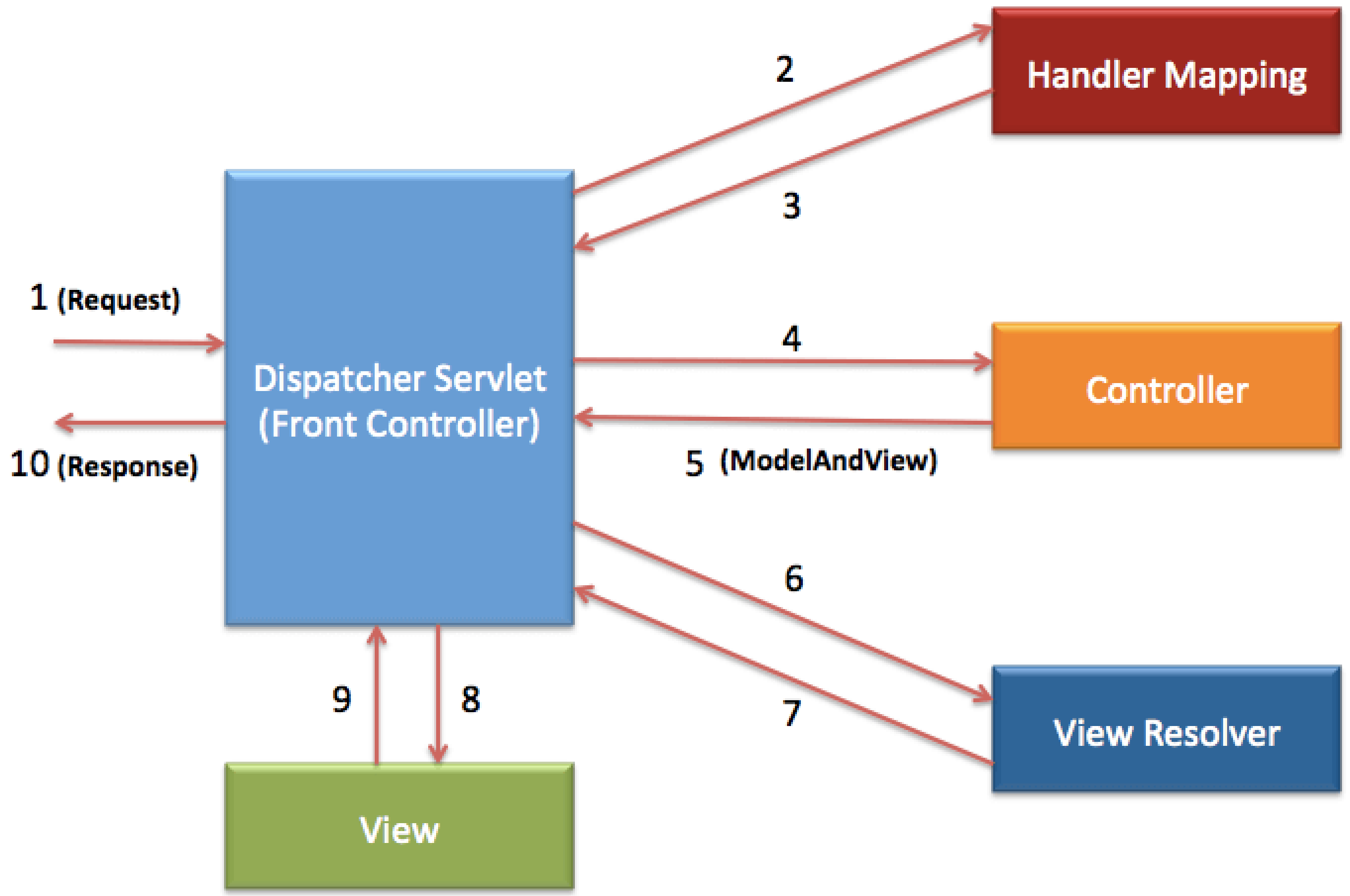
  - It is the central framework class that manages all the database communication and exception handling.
  - It takes care of creation and release of resources such as creating and closing of connection object etc.
  - So it will not lead to any problem if you forget to close the connection.

Spring MVC

Spring MVC framework is a robust Model view controller framework which helps us to develop a loosely coupled web application. It separates different aspects of web applications with the help of MVC architecture.

- **Model:** Model carries application data. It generally includes POJO in the form of business objects
- **View:** View is used to render User interface (UI). It will render application data on UI. For example JSP
- **Controller:** Controller takes care of processing user request and calling back end services.

Spring MVC workflow



Spring MVC workflow

1. The request will be received by Front Controller i.e. **DispatcherServlet**.
2. DispatcherServlet will pass this request to HandlerMapping. **HandlerMapping** will find suitable Controller for the request
3. **HandlerMapping** will send the details of the controller to DispatcherServlet.
4. DispatcherServlet will call the **Controller** identified by HandlerMapping. The **Controller** will process the request by calling appropriate method and prepare the data. It may call some business logic or directly retrieve data from the database.
5. The **Controller** will send **ModelAndView**(Model data and view name) to **DispatcherServlet**.
6. Once DispatcherServlet receives ModelAndView object, it will pass it to **ViewResolver** to find appropriate View.
7. **ViewResolver** will identify the view and send it back to **DispatcherServlet**.
8. **DispatcherServlet** will call appropriate **View** identified by ViewResolver.
9. The **View** will create Response in form of **HTML** and send it to **DispatcherServlet**.
10. **DispatcherServlet** will send the response to the **browser**. The browser will render the html code and display it to **end user**.