**Spring Notes**

**What is Spring?**

* It is a lightweight, loosely coupled and integrated framework for developing enterprise applications in java.
* Spring Framework is built on top of two design concepts – Dependency Injection and Aspect Oriented Programming.

**Use of Spring**

* It’s been noticed a lot of complexity of code is present in development which can be avoided, such as code to handle transaction, JDBC, interaction with ORM, Exception Handling, Testing and many others.
* Springs provides support for existing J2EE technologies and provide easy and efficient way to do the development.
* Dependency Injection or Inversion of Control to write components that are independent of each other, spring container takes care of wiring them together to achieve our work.

**Spring Modules**

1. **Spring Core Model**: - The core container provides the essential functionality of the Spring framework. A primary component of the core container is the Bean Factory, an implementation of the Factory pattern.
2. **Spring DAO (JDBC) Module**: This Module will provide an abstraction layer on top of JDBC technology. The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors.
3. **Spring ORM Module**: -This ORM Module is an abstraction layer on top of the ORM tools and which makes a spring programmer to transfer a data between application and database in the form of objects.
4. **Spring ORM Module**: -This ORM Module is an abstraction layer on top of the ORM tools and which makes a spring programmer to transfer a data between application and database in the form of objects.
5. **Spring ORM Module**: -This ORM Module is an abstraction layer on top of the ORM tools and which makes a spring programmer to transfer a data between application and database in the form of objects.
6. **Spring Web Module**: -This module is either for integrating with web applications created in other frame works (or) for creating complete MVC to based web application in spring frame work.

**Spring IOC Containers**

|  |  |
| --- | --- |
| **Sr.No.** | **Container & Description** |
| 1 | [Spring BeanFactory Container](https://www.tutorialspoint.com/spring/spring_beanfactory_container.htm)  This is the simplest container providing the basic support for DI and is defined by the *org.springframework.beans.factory.BeanFactory* interface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purpose of backward compatibility with a large number of third-party frameworks that integrate with Spring. |
| 2 | [Spring ApplicationContext Container](https://www.tutorialspoint.com/spring/spring_applicationcontext_container.htm)  This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the *org.springframework.context.ApplicationContext* interface. |

**Spring Bean Life Cycle**

To define setup and teardown for a bean, we simply declare the <bean> with initmethod and/or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroymethod specifies a method that is called just before a bean is removed from the container.

Initialization callbacks-

The org.springframework.beans.factory.InitializingBean interface specifies a single method-

void afterPropertiesSet() throws Exception;

Destruction callbacks

The org.springframework.beans.factory.DisposableBean interface specifies a single method −

void destroy () throws Exception;

Steps to create a Spring Application-

|  |  |
| --- | --- |
| **Steps** | **Description** |
| 1 | Create a project with a name *SpringExample* and create a package of any name under the **src** folder in the created project. |
| 2 | Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter. |
| 3 | Create Java classes *HelloWorld* and *MainApp* under the package. |
| 4 | Create Beans configuration file *Beans.xml* under the **src** folder. |
| 5 | The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below. |

**Spring-Dependency Injection**

|  |  |
| --- | --- |
| **Sr.No.** | **Dependency Injection Type & Description** |
| 1 | [Constructor-based dependency injection](https://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm)  Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class. |
| 2 | [Setter-based dependency injection](https://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm)  Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean. |

**AOP:**

One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework. Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns.

**AOP Terminologies**

|  |  |
| --- | --- |
| **Sr.No** | **Terms & Description** |
| 1 | **Aspect**  This is a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. |
| 2 | **Join point**  This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework. |
| 3 | **Advice**  This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework. |
| 4 | **Pointcut**  This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples. |
| 5 | **Introduction**  An introduction allows you to add new methods or attributes to the existing classes. |
| 6 | **Target object**  The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object. |
| 7 | **Weaving**  Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime. |

**Spring JDBC Framework**

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

**JdbcTemplate class**

It is the central class in the Spring JDBC support classes. 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.

It handles the exception and provides the informative exception messages by the help of excepion classes defined in the org.springframework.dao package.

**Data Access Object (DAO)**

DAO stands for Data Access Object, which is commonly used for database interaction. DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them.

The DAO support in Spring makes it easy to work with data access technologies like JDBC, Hibernate, JPA, or JDO in a consistent way.

**Spring MVC Framework**

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

* The **Model** encapsulates the application data and in general they will consist of POJO.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering

**DispatcherServlet**

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses.

**Defining a Controller**

The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it. The @Controller annotation indicates that a particular class serves the role of a controller. The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

**Creating JSP Views**

Spring MVC supports many types of views for different presentation technologies. These include - JSPs, HTML, PDF, Excel worksheets, XML, Velocity templates, XSLT, JSON, Atom and RSS feeds, JasperReports, etc. But most commonly we use JSP templates written with JSTL.