**Spring Framework**

It is an open-source java platform that provides comprehensive infrastructure support for developing robust java applications in very easy way with rapid development.

It was first released under Apache 2.0 License in 2003.

Spring Framework is lightweight. The core feature of the Spring Framework is dependency injection.

* **POJO Based:** Spring enables you to develop enterprise solutions using POJOs. The benefit is that we no longer require EJB containers.
* **Modular:** Spring is organized in modules.
* **Integration with Existing Framework:**

ORM Framework, Logging Framework [ Log4J ],Unit Testing [ Junit ] & others

* **Web MVC**
* **Lightweight**
* **Transaction Management**

**Dependency Injection**

* DI is a part of Inversion of Control. Inversion of control is a general concept. This concept explains
  + The classes in java programs should be as independent as possible from other java classes to increase the possibility of reusing other classes and testing them independently.
  + Dependency Injection can be helpful in binding classes together and keeping them independent at the same time.

class Employee{

@Autowired // xml declaration in beans.xml

private Address add; // field member

}

class Employee{

@Autowired

public void setAddress(Address add){

}

}

A picture containing calendar

Description automatically generated

**Core Container**

Core Container consists of Core, Beans, Context, and Expression Language.

* **Core** Module provides a fundamental part of the framework that includes IoC and Dependency Injection.
* **Bean** Module provides BeanFactory [ based on Factory Pattern ]
* **Context**  module refers to runtime environment that Spring Core is providing and Bean is being managed.
* **SpEL** provides powerful expression language to query and manipulate an object graph at runtime.

**Data Access / Integration**

1. Spring JDBC module
2. Spring ORM module [ Spring JPA, Spring JDO, Hibernate & IBatis ]
3. Spring OXM module [ Object / XML mapping ]
4. JMS
5. Spring Transaction

**Web**

1. Spring Web
2. Web – MVC
3. Web Socket [ duplex ]
4. Web – Portlet

**Others**

1. Spring AOP
2. Spring Instrumentation
3. Spring Messaging
4. Spring Test
5. Spring Batch
6. Spring Rest
7. Spring Security with JWT

**Spring Bean Scope**

1. **singleton**

By Default Spring Bean is Singleton. This means it provides injection for the same object.

1. **prototype :** A new bean instance will be injected for every dependency
2. **request**
3. **Session Available with Web Programming**
4. **global-session**

**Spring IoC Container**

It is the core of the spring framework. The container takes care of life-cycle management of **Spring Bean** Component. Spring Container is using DI to manage the components.

Diagram

Description automatically generated

**Types of Containers**

1. **Spring Bean Factory Container**

It uses the BeanFactory interface. It is lightweight

1. **Spring ApplicationContext Container**

This container uses all functionalities from BeanFactory Container and the ability to resolve properties using text messages. This uses ApplicationContext interface.

Initialization CallBack Method

Interface InitializingBean{

Void afterPropertiesSet() throws Exception

}

**Types of Dependency Injection**

1. Constructor-Based
2. Setter-Based / Property-Based

Inner Bean – Setter Based Dependency Injection

<bean class=*"com.example.Employee"* id=*"emp"* abstract=*"true"*>

<property name=*"id"* value=*"1"*></property>

<property name=*"name"* value=*"Guest"*></property>

<property name=*"salary"* value=*"1000.00"*></property>

<property name=*"add"*>

<bean id=*"add"* class=*"com.example.Address"*>

<property name=*"street"* value=*"7th Cross"*/>

<property name=*"city"* value=*"Jersey"*/>

</bean>

</property>

</bean>

**Constructor Based – Injection**

<bean class=*"com.example.Manager"* id=*"mgr"*>

<constructor-arg index=*"0"* value=*"2"*/>

<constructor-arg index=*"1"* value=*"Manager"*/>

<constructor-arg index=*"2"* value=*"1200.00"*/>

<constructor-arg index=*"3"* value=*"Human Resource"*/>

<constructor-arg index=*"4"* ref=*"add"*/>

</bean>

<bean class=*"com.example.Admin"* id=*"adm"* parent=*"emp"*>

<constructor-arg index=*"0"* value=*"1"* type=*"int"*/>

<constructor-arg index=*"1"* value=*"Guest"*/>

<constructor-arg index=*"2"* value=*"1000.00"*/>

<constructor-arg index=*"3"* ref=*"add"*/>

</bean>

<bean id=*"add"* class=*"com.example.Address"*>

<property name=*"street"* value=*"7th Cross"*/>

<property name=*"city"* value=*"Jersey"*/>

</bean>

**Injecting Collections**

<list>

<set>

<map>

<prop>

**public** **abstract** **class** Employee **implements** InitializingBean {

**private** **int** id;

**private** String name;

**private** **double** salary;

**private** List<Address> add;

//constructor setter getter toString

}

**public** **class** Address {

**private** String street;

**private** String city;

**private** List phone;

//constructor setter getter toString

}

<bean id=*"add"* class=*"com.example.Address"*>

<property name=*"street"* value=*"7th Cross"*/>

<property name=*"city"* value=*"Jersey"*/>

<property name=*"phone"*>

<list>

<value>910-0011-432</value>

<value>610-0011-432</value>

</list>

</property>

</bean>

<bean class=*"com.example.Manager"* id=*"mgr"*>

<constructor-arg index=*"0"* value=*"2"*/>

<constructor-arg index=*"1"* value=*"Manager"*/>

<constructor-arg index=*"2"* value=*"1200.00"*/>

<constructor-arg index=*"3"* value=*"Human Resource"*/>

<constructor-arg index=*"4"*>

<list>

<ref bean=*"add"*/>

</list>

</constructor-arg>

</bean>

Declarative Approach [ XML based configuration ]

**Autowiring enables** you to perform dependency injection and allows you to cut down on the amount of XML configurations.

**Autowiring Mode**

No : by default no

byName

byType

constructor

<bean class=*"com.example.Employee"* id=*"emp"* autowire=*"constructor"*>

<constructor-arg index=*"0"* value=*"2"*/>

<constructor-arg index=*"1"* value=*"Manager"*/>

<constructor-arg index=*"2"* value=*"1200.00"*/>

</bean>

<bean class=*"com.example.Employee"* id=*"emp"* autowire=*"byName"*>

<property name=*"id"* value=*"2"*/>

<property name=*"name"* value=*"Manager"*/>

<property name=*"salary"* value=*"1200.00"*/>

</bean>

**Annotation Based Configuration**

**@Autowired**

**@Required**

**@Qualifier**

<context:annotation-config/>

<bean id=*"emp"* class=*"com.example.Employee"*></bean>

<bean id=*"add1"* class=*"com.example.Address"*>

<property name=*"city"* value=*"Jersey"*></property>

<property name=*"street"* value=*"7th Cross"*/>

</bean>

<bean id=*"add2"* class=*"com.example.Address"*>

<property name=*"city"* value=*"Dallas"*></property>

<property name=*"street"* value=*"Lu Feng Cross"*/>

</bean>

**public** **class** Employee **implements** InitializingBean {

**private** **int** id;

**private** String name;

**private** **double** salary;

**private** Address add;

//all constructors/getter/setter/toString

**public** Address getAdd() {

**return** add;

}

@Autowired @Qualifier("add1")

**public** **void** setAdd(Address add) {

**this**.add = add;

}

**}**

Assignment

Timeline

Description automatically generated

**Obtain order object from Spring Container and print order, item & customer details.**

**Java Based Configuration**

Helps you to write most of your spring configurations without XML.

**@Configuration:** A class annotated with @Configuration will be used by IoC container as a source of bean definition.

**@Bean :** This is used to registers POJOs as Spring Bean by container.

@Configuration

public class MyConfig{

@Bean

public Employee emp(){

return new Employee();

}

}

ApplicationContext context=**new** AnnotationConfigApplicationContext(MyConfig.class);

**Equivalent to below XML configurations**

<beans>

<bean id=”emp” class=”com.example.Employee”/>

<beans>

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

**Spring JDBC**

Spring JDBC framework is all about taking care of low-level details starting from opening connections, preparing & execute the SQL statement processing queries, handling exceptions & transactions, and finally closing the resources.

JdbcTemplate class is one-stop solution for everything that we need.

You will be configuring DataSource in the spring configuration file using JdbcTemplate

Instance of JdbcTemplate is ThreadSafe

**Spring Web MVC**

Spring Web MVC framework provides built-in APIs to develop web application using MVC architecture.

**DispatcherServlet**

Spring Web MVC is designed around a dispatcher servlet that handles all the HTTP request & response.

* To handle all the HTTP request & response.
* After receiving the request the dispatcher servlet consults the HandlerMapping to call the appropriate controller [ Spring Bean ]
* Controller takes the request and calls the service [ spring bean ] method
* DispatcherServlet takes help from the View-Resolver to pick up the defined view for the- response.

Diagram, schematic

Description automatically generated

Text

Description automatically generated

Logging is a powerful mechanism to capture **runtime behaviour** of your application that can help understanding and debugging programs.

If you want to enable logging service in your application – you need logging API [ Library ]

JDK has a package that provides logging API.

**java.util.logging**

and there are several logging framework also available

1. **Log4J**
2. SL4J
3. Logback

**Introduction on Java Logging**

Handler :  Handler object takes log messages from a Logger and exports them to File, Console or to Sockets

FileHandler

ConsoleHandler == System.out

Most of the enterprise application have some common cross-cutting concerns that are application on various operation( methods ] , objects and modules.

**Example of Cross-Cutting Concerns**

1. Logging
2. Transaction Management
3. Data Validation
4. Security

* **Aspect** is a class that implements Enterprise Application Concern that cut across multiple classes and operation.
* **JoinPoint** a specific point in the application e.g. method , exception, objects
* **Advice** are action taken for a particular join point.

1. Before
2. After
3. After Returning
4. After Throwing
5. Around

* **Pointcut** is expression that are matched with join point to determine whether advice needs to be executed or not.

* **Target Object :** Object on which advices are applied.
* **AOP Proxy:**
* **Weaving** It is the process of linking aspects with other objects to create the advised proxy objects.