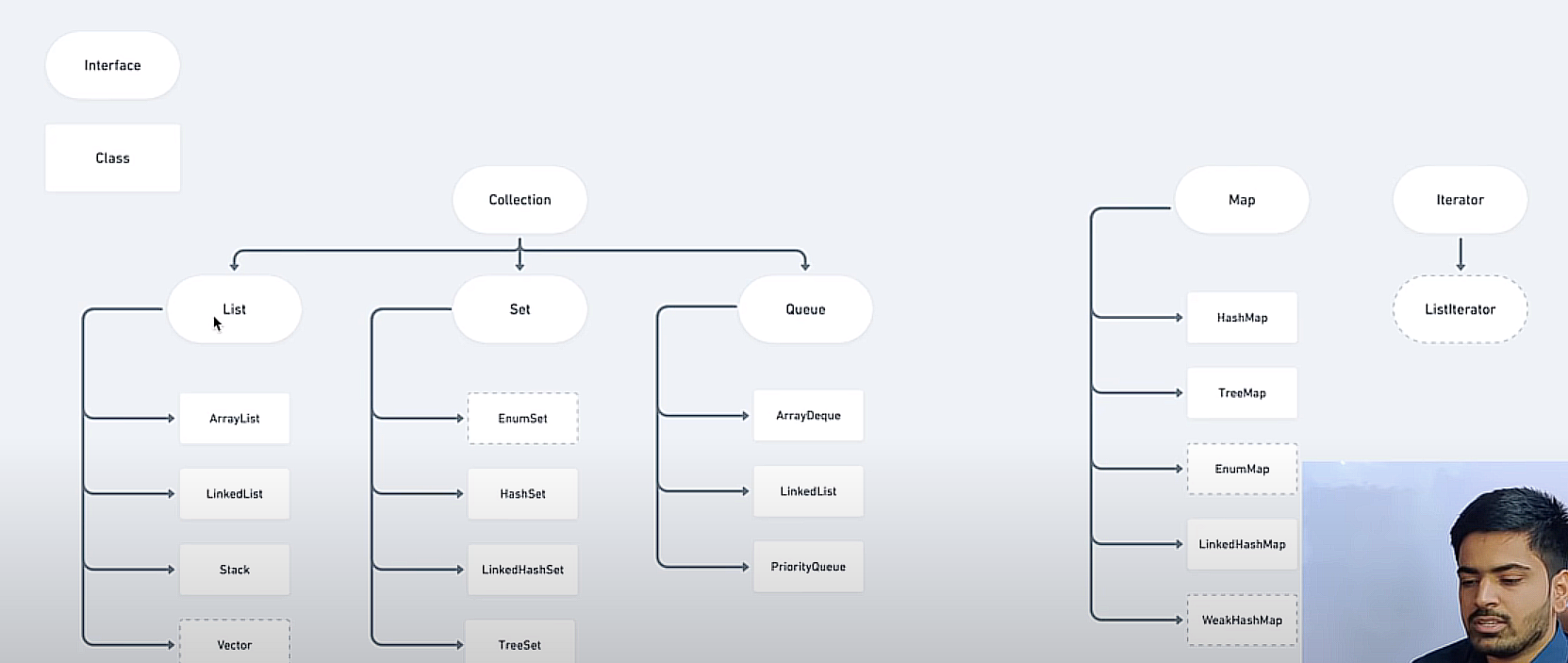
JAVA Collections FRAMEWORK



Dotted once are least used

ArrayList<String> str = new ArrayList<>();

List.add(element)

List.add(index:10, element: 40) -> list.add(10,40)

List.addAll() -> append another list behind, get() -> to fetch, similarly remove(index) and

List.remove(Integer.value0f(element));

List.clear(), contains(element) -> O(n)

--------------------------------------------------------------------------------------------------------------------------------------

Iterator<Integer> it = list.iterator();

While(list.hasNext()){

Sys(“element is : ” + it.next());

}

STACK -> LIFO

Stack<String> animals = new Stack<>();

animals.push(“lion”); animals.peek(), animals.pop()

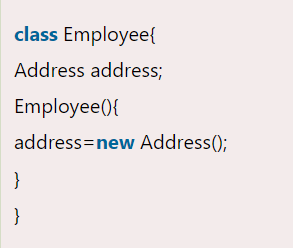
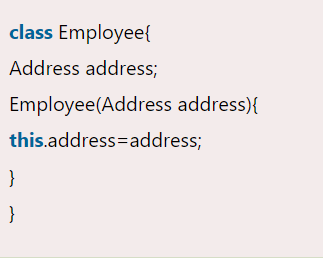
JAVA SPRING BOOT

Spring boot is made up of Spring and Auto configuration, embedded servers, external configurations,

Spring initializr -> start.spring.io

This spring tutorial provides in-depth concepts of Spring Framework with simplified examples. It was developed by Rod Johnson in 2003. 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, Tapestry, EJB, JSF, etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems. The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

Inversion Of Control (IOC) and Dependency Injection -> These are the design patterns that are used to remove dependency from the programming code.

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this: B)

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

**Advantage of Dependency Injection**

* **makes the code loosely coupled so easy to maintain**
* **makes the code easy to test**

Advantages of Spring Framework

**Predefined Templates ->** Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

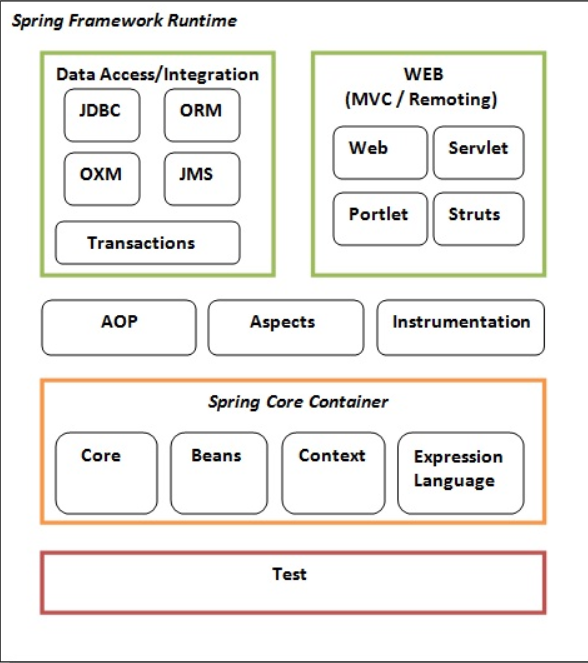
**The Spring applications are loosely coupled because of dependency injection**.

**The Dependency Injection makes easier to test the application**. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA. It provides declarative support for caching, validation, transactions and formatting.

**The Spring framework comprises of many modules** such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



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

Spring Core Container -> The Spring Core container contains core, beans, context and expression language (EL) modules.

Core and Beans -> These modules provide IOC and Dependency Injection features.

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.

The instrumentation module provides support to class instrumentation and classloader implementations.

Data Access / Integration -> This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

Web ->This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

Dependency Injection

**The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly**. The main tasks performed by IoC container are:

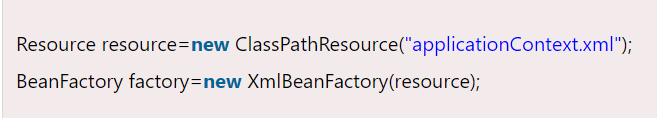
* to instantiate the application class
* to configure the object
* to assemble the dependencies between the objects

There are two types of IoC containers. They are:

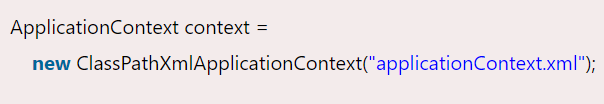
* **BeanFactory**
* **ApplicationContext**

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

The XmlBeanFactory is the implementation class for the BeanFactory interface. To use the BeanFactory, we need to create the instance of XmlBeanFactory class as given below:



The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.



The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

**Dependency Injection (DI) 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. To understand the DI better, Let's understand the Dependency Lookup (DL) first:**

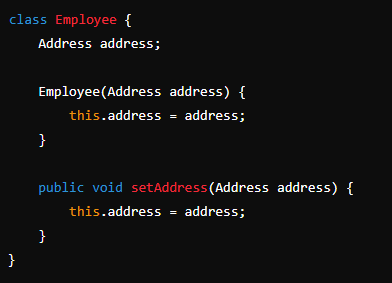
The Dependency Lookup is an approach where we get the resource after demand. There can be various ways to get the resource for example: **A obj = new AImpl();**

In such way, we get the resource(instance of A class) directly by new keyword. Another way is factory method: **A obj = A.getA(); (by calling the static factory method getA().)**

There are mainly two problems of dependency lookup.

* **tight coupling The dependency lookup approach makes the code tightly coupled. If resource is changed, we need to perform a lot of modification in the code.**
* **Not easy for testing This approach creates a lot of problems while testing the application especially in black box testing.**

The Dependency Injection is a design pattern that removes the dependency of the programs.



When an Employee object is created, an Address object is passed to it. This means the Employee object is not responsible for creating the Address object, adhering to the principle of dependency injection. This is Constructor Injection.

In such case, instance of Address class is provided by external souce such as XML file either by constructor or setter method. Dependency injection is a design pattern that allows an object to receive its dependencies from an external source rather than creating them itself

**Two ways to perform Dependency Injection in Spring framework -> By Constructor & By Setter method**

**Dependency example – constructor injection -** [**https://www.javatpoint.com/spring-tutorial-dependency-injection-by-constructor**](https://www.javatpoint.com/spring-tutorial-dependency-injection-by-constructor)