**1)Autowiring and its types-------------**

In Spring framework, you can wire beans automatically with auto-wiring feature. To enable it, just define the “**autowire**” attribute in xml file. The Spring container can **autowire** relationships between collaborating beans without using and elements which helps cut down on the amount of XML configuration

**Autowiring modes** – no,byName,byType,Constructor,autodetect.

No : Default, no auto wiring, set it manually via “ref” attribute as we have done in dependency injection via setter 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.

Constructor : 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*.

**2)Autowiring byName -------------**

Spring autowiring byName means autowiring on the basis of name. If any bean property matches with exact one other bean’s name declaration then spring will automatically takes care of dependency.

Syntax----

<bean id="country" **class**="org.Country" autowire="byName">

<property name="countryName" value="India"/>

</bean>

<bean id="capital" **class**="org.Capital">

<property name="capitalName" value="Delhi" />

</bean>

Here in ,we have used autowire attribute and set it to “byName”.So now spring container will match “capitalObj” reference in Country class with id or name of other beans in XML configuration file. So here you can see we have bean with id as “capitalObj”

**3)Autowiring byType-----------------**

Spring autowiring byType means autowiring on the basis of datatype. if any bean property datatype matches with exact one other bean’s datatype declaration then spring will automatically takes care of dependency. If more than one bean property matches, then it throws fatal exception.

Syntax-

<bean id="country" **class**="org.Country" autowire="byType">

<property name="countryName" value="India"/>

</bean>

<bean id="capital" **class**="org.Capital">

<property name="capitalName" value="Delhi" />

</bean>

we have used autowire attribute and set it to “byType”.So now spring container will match Capital datatype in Country class with type of other beans in XML configuration file. So here you can see we have bean with class as Capital.

**4)Constructor Injection-----------------**

As the name implies, using constructor spring container will inject the dependencies. Required components are passed into a class at the time of instantiation. We can use constructor-arg tag here.

Syntax:

<bean id="empid" class="com.constructormode.Employee" autowire="constructor">

<constructor-arg name="eid" value="202"></constructor-arg>

<constructor-arg name="ename" value ="rachel"></constructor-arg>

<constructor-arg> <ref bean="dept1"/> </constructor-arg>

</bean>

Advantages--------

### All Required Dependencies Are Available at Initialization Time.

* The IoC container makes sure that all the arguments provided in the constructor are available before passing them into the constructor. This helps in preventing the infamous NullPointerException.
* we do not have to write separate business logic everywhere to check if all the required dependencies are loaded, thus simplifying code complexity.

**5)Collection Objects---------------**

Java Collection means a single unit of objects. Collection framework provides many interfaces like Set, List, Map

and classes like ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet, HashMap, LinkedHashMap.

List:

List is an Inteface in java and child interface of Collection which is present in java.util package.If we want to represent a group of individual objects as a single entity where "duplicates are allow and insertion order must be preserved" then we should go for List interface.

Vector and Stack classes are re-engineered in 1.2 versions to implement List interface.

Set:

It is the child interface of Collection which is present in java.util package.If we want to represent a group of individual objects as single entity "where duplicates are not allow and insertion order is not preserved" then we should go for Set interface.

Set has child intefaces and child classes which are implemented (SortedSet and NavigableSet)and inherited by Set interface (HashSet, LinkedHashSet and TreeSet)

Map:

If we want to represent a group of objects as key-value pairs then we should go for Map.Map is not child interface of Collection.

If we want to represent a group of objects as key-value pairs then we should go for Map interface.Duplicate keys are not allowed but values can be duplicated.

Map Interface also have many child classes (HashMap, LinkedHashMap, TreeMap, IdentityHashMap and WeakHashMap, Dictionary (It is an Abstract Class), hashTable and Properties) and child interfaces (SortedMap and NavigableMap).

Properties:

We also have a Properties class which is also in java.util package. It is a subclass of Hashtable.It is used to maintain a list of values in which the key is a string and the value is also a string i.e; it can be used to store and retrieve string type data from the properties file.Properties class can specify other properties list as it’s the default. If a particular key property is not present in the original Properties list, the default properties will be searched.Properties object does not require external synchronization and Multiple threads can share a single Properties object.Also, it can be used to retrieve the properties of the system.