**Java Notes:**

1. When an object is created it is associated a unique id. This id is used in an encrypted manner, called ***hashCode.***

We can use object.hashCode() to get the info about the hash code of the object.

1. **List vs Set**

|  |  |
| --- | --- |
| **List** | **Set** |
| It’s an ordered collection of values of a certain type | It’s an unordered collection of ***distinct*** values of a certain type |

1. **Map**

It’s a collection having the data stored as key-value pairs. Each key will be unique but the values may or may not be unique.

***Note: If there’s a key-value pair provided for an existing key, its value will be updated to the new value.***

**Spring Notes:**

1. **What’s Spring?**

Spring is a Java based framework to develop web applications.

1. **What are Spring projects?**

Spring Projects gives us a convenient way to implement the functionality in our application.

1. **Why do we need Spring/ advantages of Spring?**
2. To develop the web applications in an easier way.
3. It’s lightweight as it has the capability to use only the required modules.
4. It works with POJOs(**P**lain **O**ld **J**ava **O**bjects).
5. It has many projects which can be implemented to make the application development easier.
6. It works on the principle of IoC, using DI.
7. **Define Dependency Injection.**

It is the process of injecting the required objects.

1. Since IoC works inversely to normal control, i.e., by providing the full-fledged ready to use objects when required. Hence, it’s called IoC.
2. **What are Bean?**

Beans are Java objects which are created, managed and processed by **Spring Containers**.

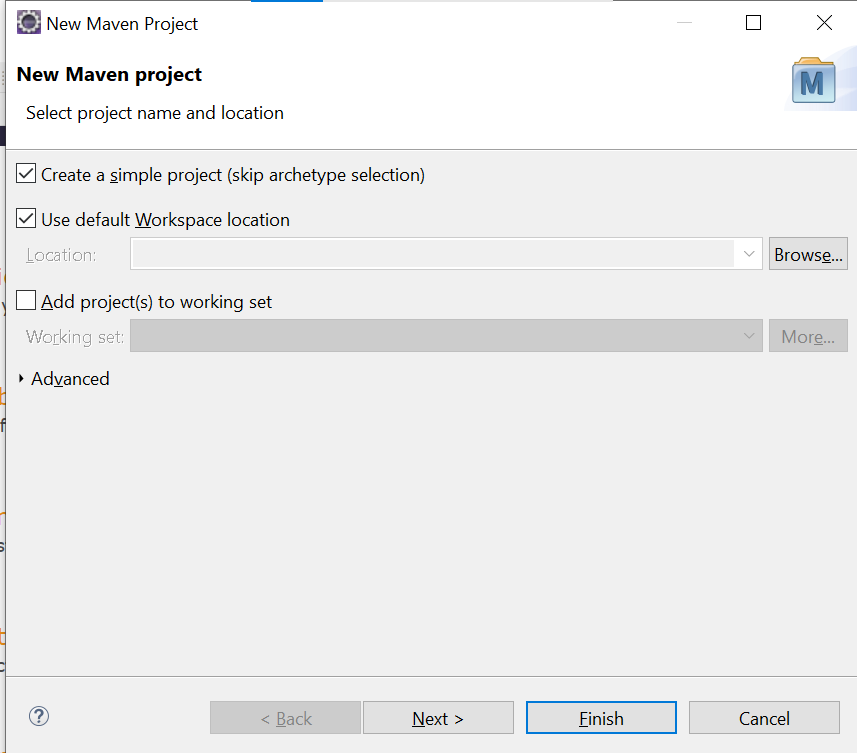
1. **What are different bean scope(s) (lifetime)?**
2. Singleton (by default): Only a single object is created and its reference is provided whenever needed.
3. Prototype: A similar new reference of the object is created, whenever requested.
4. Session
5. GlobalSession
6. Application
7. **What are Spring Containers?**

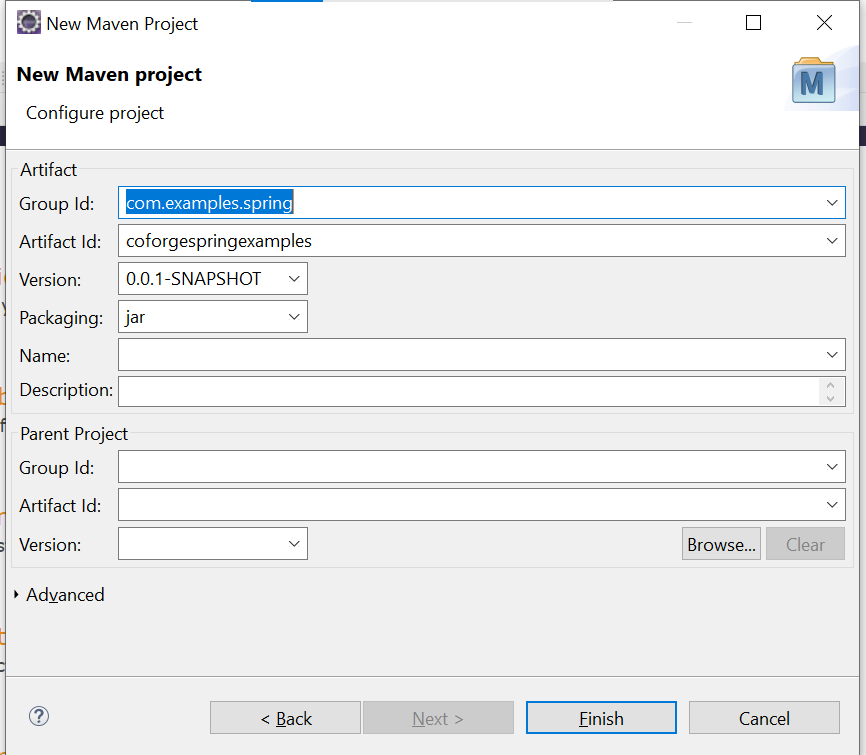
It’s the heart of the Spring framework. It is a component which creates, manages and destroys the objects.

Spring containers are basically our application context, but it is an interface having abstract methods for the bean implementation. Now we need implementation classes for creating the Spring container.

**Practical based:**

1. Create a new maven project, by File -> New -> Maven Project

****

****

1. Now we need to add 2 dependencies (in pom.xml file):
2. spring-core
3. spring-context
4. After adding these, they can be reflected under **Maven Dependencies** folder.
5. Add a new class Address.java
6. Add a new class TestSpring.java. This will be our Spring container.
7. Configuration to the IoC containers can be provided using XML/ Java annotations/ Java class.

**Providing the configuration using XML.**

1. Create a new file **beans.xml** under **src/main/java,** it’ll contain our configuration metadata for our spring container.
2. Beans.xml contains collection of beans, represented by <bean>
3. Every bean tag will have following attributes:
4. a unique **id=””,**
5. and **class*=”Fully qualified name of the class for which bean is being created”.*** E.g., if we have to create a bean for a class named Address belonging to same package, class=”packagename.Address”.
6. Every bean tag will have as many properties with <property> tag as there are data members in the class.
7. Every property tag will have following attributes:
8. name=”data member name”,
9. value=”value as per the data type of the member”,
10. scope=”” //lifetime scope for the bean, by default singleton.
11. Here, the process of setting values by defining in the beans.xml is called ***Setter Method Injection.*** Following steps are done by the spring in the backend:
12. Our IoC container context is created with the metadata defined in beans.xml
13. From this context we create a bean with specified id.
14. This goes to beans.xml and finds a bean with the specified id.
15. It further looks for the class with the specified value of the class.
16. It calls the setter methods of the class and provides the values specified in the property tag for the data members.

***Note: It’s mandatory to have setters defined, else we get error.***

1. It then returns the object as bean.
2. We need to type cast this object into our class.
3. For an object data member in the class, in the bean we’ll make use of existing bean using the ***ref=”bean id”*** attribute, instead of value attribute.
4. **autowired-constructor:** We can also set constructor values of the class through bean itself.
5. There will be <constructor-arg> tag with **name**=”data member name” and **value=”**value as per data type” attributes.
6. It’ll call the parameterized constructor of the class for which bean is created.

**Providing the configuration using Java Annotations.**

1. Here we’ll create a new class ***TestJavaAnnotationConfig,*** which will be our IoC container.
2. Create a new class springs.xml, it’ll contain the info about which component should be scanned in which package.
3. Create a new class AppConfig, which will be our configuration for IoC container.
4. Annotate the class with @Configuration annotation.
5. In AppConfig class, declare member functions annotated with @Bean, to indicate this is a bean object for each class.
6. When a bean for a class is requested using Java annotations, following is done at the backend:
7. The spring starts finding for the class annotated with @Configuration.
8. Then in the class, it looks for any annotated with @Bean and having return type matching with the specified name of the bean.
9. It creates the object and returns.

**Providing the configuration using Java class.**

1. In our IoC container, we’ll now provide the AppConfig.class as the class for our configuration.
2. In AppConfig, assign names for Beans, i.e., @Bean(“beanname”).
3. In our container we can call get bean method by providing the bean name.
4. **@Component annotation:**

When a class is having @Component annotation, it’s treated as a bean object, with name in camelCasing.

1. We can also provide the name to the class annotated with @Component, like @Component(“name”). This will be the name of the bean.
2. **Autowiring:**

The process of automatically referencing the missing objects/parameters. We add an attribute to the bean, i.e., autowire=”type”. When we’re using autowire, we need not specify the property, if there exists a bean with the same type/name.

It can be done in following ways:

1. byType: Spring will check if there exists a bean with the type(class of the bean) matching with the required class, and creates it’s object.
2. byName: Spring will check if there exists a bean with the name(id of the bean) matching with the required class, and creates it’s object.
3. Constructor:

**Reference:**

[coforge-june-microservices-training-git-repo](https://github.com/praveeniiht/coforge-june-microservices.git)

[spring-framework-reference](https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html)

