**\*\*\*\*\*Spring Questions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

1. **What is Spring Framework?**

Spring is one of the most widely used Java EE framework. Spring framework core concepts are “Dependency Injection” and “Aspect Oriented Programming”.

Spring framework can be used in normal java applications also to achieve loose coupling between different components by implementing dependency injection and we can perform cross cutting tasks such as logging and authentication using spring support for aspect oriented programming.

I like spring because it provides a lot of features and different modules for specific tasks such as Spring MVC and Spring JDBC. Since it’s an open source framework with a lot of online resources and active community members, working with Spring framework is easy and fun at same time.

1. **What are some of the important features and advantages of Spring Framework?**

Spring Framework is built on top of two design concepts – Dependency Injection and Aspect Oriented Programming.

Some of the features of spring framework are:

* + Lightweight and very little overhead of using framework for our 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 IoC container manages Spring Bean life cycle and project specific configurations such as JNDI lookup.
  + Spring MVC framework can be used to create web applications as well as restful web services capable of returning XML as well as JSON response.
  + Support for transaction management, JDBC operations, File uploading, Exception Handling etc with very little configurations, either by using annotations or by spring bean configuration file.

Some of the advantages of using Spring Framework are:

* + Reducing direct dependencies between different components of the application, usually Spring IoC container is responsible for initializing resources or beans and inject them as dependencies.
  + Writing unit test cases are easy in Spring framework because our business logic doesn’t have direct dependencies with actual resource implementation classes. We can easily write a test configuration and inject our mock beans for testing purposes.
  + Reduces the amount of boiler-plate code, such as initializing objects, open/close resources. I like JdbcTemplate class a lot because it helps us in removing all the boiler-plate code that comes with JDBC programming.
  + Spring framework is divided into several modules, it helps us in keeping our application lightweight. For example, if we don’t need Spring transaction management features, we don’t need to add that dependency in our project.
  + Spring framework support most of the Java EE features and even much more. It’s always on top of the new technologies, for example there is a Spring project for Android to help us write better code for native android applications. This makes spring framework a complete package and we don’t need to look after different framework for different requirements.

1. **What do you understand by Dependency Injection?**

Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection pattern to move the dependency resolution from compile-time to runtime.

Some of the benefits of using Dependency Injection are: Separation of Concerns, Boilerplate Code reduction, Configurable components and easy unit testing.

Read more at [Dependency Injection Tutorial](http://www.journaldev.com/2394/dependency-injection-design-pattern-in-java-example-tutorial). We can also use [Google Guice for Dependency Injection](http://www.journaldev.com/2403/google-guice-dependency-injection-example-tutorial) to automate the process of dependency injection. But in most of the cases we are looking for more than just dependency injection and that’s why Spring is the top choice for this.

1. **How do we implement DI in Spring Framework?**

We can use Spring XML based as well as Annotation based configuration to implement DI in spring applications. For better understanding, please read [Spring Dependency Injection](http://www.journaldev.com/2410/spring-dependency-injection-example-with-annotations-and-xml-configuration) example where you can learn both the ways with JUnit test case. The post also contains sample project zip file, that you can download and play around to learn more.

1. **What are the benefits of using Spring Tool Suite?**

We can install plugins into Eclipse to get all the features of Spring Tool Suite. However STS comes with Eclipse with some other important stuffs such as Maven support, Templates for creating different types of Spring projects and tc server for better performance with Spring applications.

I like STS because it highlights the Spring components and if you are using AOP pointcuts and advices, then it clearly shows which methods will come under the specific pointcut. So rather than installing everything on our own, I prefer using STS when developing Spring based applications.

1. **Name some of the important Spring Modules?**

Some of the important Spring Framework modules are:

* + **Spring Context** – for dependency injection.
  + **Spring AOP** – for aspect oriented programming.
  + **Spring DAO** – for database operations using DAO pattern
  + **Spring JDBC** – for JDBC and DataSource support.
  + **Spring ORM** – for ORM tools support such as Hibernate
  + **Spring Web Module** – for creating web applications.
  + **Spring MVC** – Model-View-Controller implementation for creating web applications, web services etc.

1. **What do you understand by Aspect Oriented Programming?**

Enterprise applications have some common cross-cutting concerns that is applicable for different types of Objects and application modules, such as logging, transaction management, data validation, authentication etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in AOP application modularity is achieved by Aspects and they are configured to cut across different classes methods.

AOP takes out the direct dependency of cross-cutting tasks from classes that is not possible in normal object oriented programming. For example, we can have a separate class for logging but again the classes will have to call these methods for logging the data. Read more about Spring AOP support at [Spring AOP Example](http://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations-xml-configuration).

1. **What is Aspect, Advice, Pointcut, JointPoint and Advice Arguments in AOP?**

**Aspect**: Aspect is a class that implements cross-cutting concerns, such as transaction management. Aspects can be a normal class configured and then configured in Spring Bean configuration file or we can use Spring AspectJ support to declare a class as Aspect using @Aspect annotation.

**Advice**: Advice is the action taken for a particular join point. In terms of programming, they are methods that gets executed when a specific join point with matching pointcut is reached in the application. You can think of Advices as [Spring interceptors](http://www.journaldev.com/2676/spring-mvc-interceptors-example-handlerinterceptor-and-handlerinterceptoradapter) or [Servlet Filters](http://www.journaldev.com/1933/java-servlet-filter-example-tutorial).

**Pointcut**: Pointcut are regular expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points. Spring framework uses the AspectJ pointcut expression language for determining the join points where advice methods will be applied.

**Join Point**: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

**Advice Arguments**: We can pass arguments in the advice methods. We can use args() expression in the pointcut to be applied to any method that matches the argument pattern. If we use this, then we need to use the same name in the advice method from where argument type is determined.

These concepts seems confusing at first, but if you go through [Spring Aspect, Advice Example](http://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations-xml-configuration) then you can easily relate to them.

1. **What is the difference between Spring AOP and AspectJ AOP?**

AspectJ is the industry-standard implementation for Aspect Oriented Programming whereas Spring implements AOP for some cases. Main differences between Spring AOP and AspectJ are:

* + Spring AOP is simpler to use than AspectJ because we don’t need to worry about the weaving process.
  + Spring AOP supports AspectJ annotations, so if you are familiar with AspectJ then working with Spring AOP is easier.
  + Spring AOP supports only proxy-based AOP, so it can be applied only to method execution join points. AspectJ support all kinds of pointcuts.
  + One of the shortcoming of Spring AOP is that it can be applied only to the beans created through Spring Context.

1. **What is Spring IoC Container?**

**Inversion of Control** (IoC) is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and make it ready for our use.

Spring Framework IoC container classes are part of org.springframework.beans and org.springframework.context packages and provides us different ways to decouple the object dependencies.

Some of the useful ApplicationContext implementations that we use are;

* + AnnotationConfigApplicationContext: For standalone java applications using annotations based configuration.
  + ClassPathXmlApplicationContext: For standalone java applications using XML based configuration.
  + FileSystemXmlApplicationContext: Similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
  + AnnotationConfigWebApplicationContext and XmlWebApplicationContext for web applications.

1. **What is a Spring Bean?**

Any normal java class that is initialized by Spring IoC container is called Spring Bean. We use Spring ApplicationContext to get the Spring Bean instance.

Spring IoC container manages the life cycle of Spring Bean, bean scopes and injecting any required dependencies in the bean.

1. **What is the importance of Spring bean configuration file?**

We use Spring Bean configuration file to define all the beans that will be initialized by Spring Context. When we create the instance of Spring ApplicationContext, it reads the spring bean xml file and initialize all of them. Once the context is initialized, we can use it to get different bean instances.

Apart from Spring Bean configuration, this file also contains spring MVC interceptors, view resolvers and other elements to support annotations based configurations.

1. **What are different ways to configure a class as Spring Bean?**

There are three different ways to configure Spring Bean.

* + **XML Configuration**: This is the most popular configuration and we can use bean element in context file to configure a Spring Bean. For example:

|  |  |
| --- | --- |
| 1 | <bean name="myBean" class="com.journaldev.spring.beans.MyBean"></bean> |

* + **Java Based Configuration**: If you are using only annotations, you can configure a Spring bean using @Bean annotation. This annotation is used with @Configuration classes to configure a spring bean. Sample configuration is:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Configuration  @ComponentScan(value="com.journaldev.spring.main")  public class MyConfiguration {        @Bean      public MyService getService(){          return new MyService();      }  } |

* + To get this bean from spring context, we need to use following code snippet:

|  |  |
| --- | --- |
| 1  2  3 | AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(          MyConfiguration.class);  MyService service = ctx.getBean(MyService.class); |

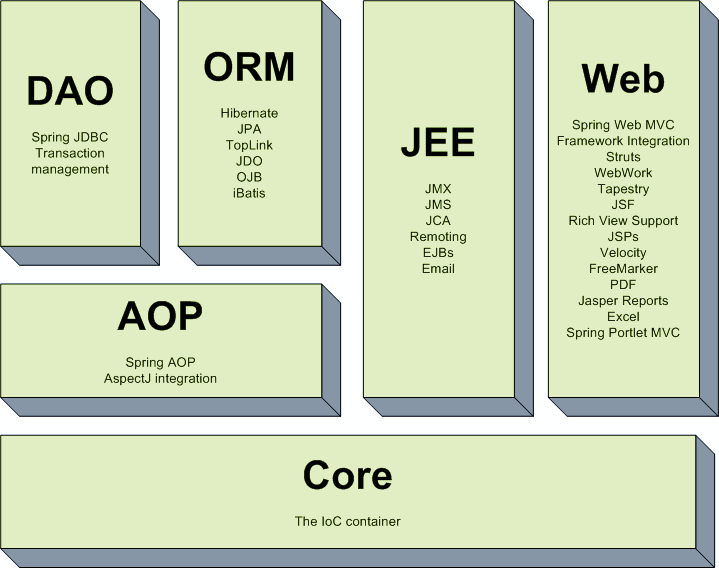
* + **Annotation Based Configuration**: We can also use @Component, @Service, @Repository and @Controller annotations with classes to configure them to be as spring bean. For these, we would need to provide base package location to scan for these classes. For example:

|  |  |
| --- | --- |
| 1 | <context:component-scan base-package="com.journaldev.spring" /> |

## 3) What are the different modules in Spring framework?

* The Core container module
* Application context module
* AOP module (Aspect Oriented Programming)
* JDBC abstraction and DAO module
* O/R mapping integration module (Object/Relational)
* Web module
* **MVC framework** module

## 4) What is the structure of Spring framework?

[](http://cdn.javabeat.net/wp-content/uploads/2009/02/spring-framework.png)

## 5) What is the Core container module?

This module is provides the fundamental functionality of the spring framework. In this module **BeanFactory** is the heart of any spring-based application. The entire framework was built on the top of this module. This module makes the **Spring container**.

## 6) What is Application context module?

The Application context module makes spring a framework. This module extends the concept of **BeanFactory**, providing support for internationalization (I18N) messages, application lifecycle events, and validation. This module also supplies many enterprise services such JNDI access, **EJB integration**, remoting, and scheduling. It also provides support to other framework.

## 7) What is AOP module?

The **AOP** module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between **Spring** and other **AOP** frameworks. This module also introduces metadata programming to **Spring**. Using Spring’s metadata support, we will be able to add **annotations** to our source code that instruct **Spring** on where and how to apply aspects.

## 8)What is JDBC abstraction and DAO module?

Using this module we can keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. A new layer of meaningful exceptions on top of the error messages given by several database servers is bought in this module. In addition, this module uses **Spring’s AOP module** to provide transaction management services for objects in a Spring application.

## 9) What are object/relational mapping integration module?

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provide support to tie into several popular **ORM frameworks**, including **Hibernate**, **JDO**, and **iBATIS SQL Maps**. Spring’s transaction management supports each of these **ORM frameworks** as well as **JDBC**.

## 10) What is web module?

This module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with **Jakarta Struts**.

## 11) What is web module?

Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring’s MVC framework uses IoC to provide for a clean separation of controller logic from business objects. It also allows you to decoratively bind request parameters to your business objects. It also can take advantage of any of Spring’s other services, such as I18N messaging and validation.

## 12) What is a BeanFactory?

A BeanFactory is an implementation of the factory pattern that applies Inversion of Control to separate the application’s configuration and dependencies from the actual application code.

## 13) What is AOP Alliance?

AOP Alliance is an open-source project whose goal is to promote adoption of AOP and interoperability among different AOP implementations by defining a common set of interfaces and components.

## 14) What is Spring configuration file?

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

## 15) What does a simple spring application contain?

These applications are like any Java application. They are made up of several classes, each performing a specific purpose within the application. But these classes are configured and introduced to each other through an XML file. This XML file describes how to configure the classes, known as the Spring configuration file.

## 16) What is XMLBeanFactory?

**BeanFactory** has many implementations in Spring. But one of the most useful one is **org.springframework.beans.factory.xml.XmlBeanFactory**, which loads its beans based on the definitions contained in an XML file. To create an **XmlBeanFactory**, pass a java.io.InputStream to the constructor. The **InputStream** will provide the XML to the factory. For example, the following code snippet uses a java.io.**FileInputStream** to provide a bean definition XML file to **XmlBeanFactory**.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | BeanFactory factory = new XmlBeanFactory( |
| 2 | new FileInputStream('beans.xml')); |

To retrieve the bean from a BeanFactory, call the getBean() method by passing the name of the bean you want to retrieve.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | MyBean myBean = (MyBean) factory.getBean('myBean'); |

## 17) What are important ApplicationContext implementations in spring framework?

* **ClassPathXmlApplicationContext –** This context loads a context definition from an XML file located in the class path, treating context definition files as class path resources.
* **FileSystemXmlApplicationContext –** This context loads a context definition from an XML file in the filesystem.
* **XmlWebApplicationContext –** This context loads the context definitions from an XML file contained within a web application.

## 18) Explain Bean lifecycle in Spring framework?

1. The spring container finds the bean’s definition from the XML file and instantiates the bean.
2. Using the dependency injection, spring populates all of the properties as specified in the bean definition.
3. If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName()** passing the bean’s ID.
4. If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory()**, passing an instance of itself.
5. If there are any **BeanPostProcessors** associated with the bean, their **post- ProcessBeforeInitialization()** methods will be called.
6. If an init-method is specified for the bean, it will be called.
7. Finally, if there are any **BeanPostProcessors** associated with the bean, their **postProcessAfterInitialization()** methods will be called.

## 19) What is bean wiring?

Combining together beans within the Spring container is known as bean wiring or wiring. When wiring beans, you should tell the container what beans are needed and how the container should use dependency injection to tie them together.

## 20) How do add a bean in spring application?

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <?xml version='1.0' encoding='UTF-8'?> |
| 2 | <!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN' | |

|  |  |  |
| --- | --- | --- |
| 3 | '<http://www.springframework.org/dtd/spring-beans.dtd>'> | |
| 4 | <beans> |

|  |  |
| --- | --- |
| 5 | <bean id='foo' class='com.act.Foo'/> |
| 6 | <bean id='bar' class='com.act.Bar'/ | |

|  |  |
| --- | --- |
| 7 | </beans> |

In the bean tag the id attribute specifies the bean name and the class attribute specifies the fully qualified class name.

## 21) What are singleton beans and how can you create prototype beans?

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named ‘singleton’ if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <beans> |
| 2 | <bean id='bar' class='com.act.Foo' | |

|  |  |  |
| --- | --- | --- |
| 3 | singleton='false'/> | |
| 4 | </beans> |

## 22) What are the important beans lifecycle methods?

There are two important bean lifecycle methods. The first one is setup which is called when the bean is loaded in to the container. The second method is the teardown method which is called when the bean is unloaded from the container.

## 23) How can you override beans default lifecycle methods?

The bean tag has two more important attributes with which you can define your own custom initialization and destroy methods. Here I have shown a small demonstration. Two new methods fooSetup and fooTeardown are to be added to your Foo class.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <beans> |
| 2 | <bean id='bar' class='com.act.Foo' | |

|  |  |  |
| --- | --- | --- |
| 3 | init-method='fooSetup' destroy='fooTeardown'/> | |
| 4 | </beans> |

## 24) What are Inner Beans?

When wiring beans, if a bean element is embedded to a property tag directly, then that bean is said to the Inner Bean. The drawback of this bean is that it cannot be reused anywhere else.

## 25) What are the different types of bean injections?

There are two types of bean injections.

1. By setter
2. By constructor

## 26) What is Auto wiring?

You can wire the beans as you wish. But spring framework also does this work for you. It can auto wire the related beans together. All you have to do is just set the autowire attribute of bean tag to an autowire type.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <beans> |
| 2 | <bean id='bar' class='com.act.Foo' Autowire='autowire type'/> | |

|  |  |
| --- | --- |
| 3 | </beans> |

## 27) What are different types of Autowire types?

There are four different types by which autowiring can be done.

* + byName
  + byType
  + constructor
  + autodetect

## 28) What are the different types of events related to Listeners?

There are a lot of events related to **ApplicationContext** of spring framework. All the events are subclasses of **org.springframework.context.Application-Event**. They are

* ContextClosedEvent – This is fired when the context is closed.
* ContextRefreshedEvent – This is fired when the context is initialized or refreshed.
* RequestHandledEvent – This is fired when the web context handles any request.

## 29) What is an Aspect?

An aspect is the cross-cutting functionality that you are implementing. It is the aspect of your application you are modularizing. An example of an aspect is logging. Logging is something that is required throughout an application. However, because applications tend to be broken down into layers based on functionality, reusing a logging module through inheritance does not make sense. However, you can create a logging aspect and apply it throughout your application using AOP.

## 30) What is a Jointpoint?

A joinpoint is a point in the execution of the application where an aspect can be plugged in. This point could be a method being called, an exception being thrown, or even a field being modified. These are the points where your aspect’s code can be inserted into the normal flow of your application to add new behavior.

## 31) What is an Advice?

Advice is the implementation of an aspect. It is something like telling your application of a new behavior. Generally, and advice is inserted into an application at joinpoints.

## 32) What is a Pointcut?

A pointcut is something that defines at what joinpoints an advice should be applied. Advices can be applied at any joinpoint that is supported by the AOP framework. These Pointcuts allow you to specify where the advice can be applied.

## 33) What is an Introduction in AOP?

An introduction allows the user to add new methods or attributes to an existing class. This can then be introduced to an existing class without having to change the structure of the class, but give them the new behavior and state.

## 34) What is a Target?

A target is the class that is being advised. The class can be a third party class or your own class to which you want to add your own custom behavior. By using the concepts of AOP, the target class is free to center on its major concern, unaware to any advice that is being applied.

## 35) What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

## 36) What is meant by Weaving?

The process of applying aspects to a target object to create a new proxy object is called as Weaving. The aspects are woven into the target object at the specified joinpoints.

## 37) What are the different points where weaving can be applied?

* Compile Time
* Classload Time
* Runtime

## 38) What are the different advice types in spring?

* + **Around :** Intercepts the calls to the target method
  + **Before :** This is called before the target method is invoked
  + **After :** This is called after the target method is returned
  + **Throws :** This is called when the target method throws and exception
* Around : org.aopalliance.intercept.MethodInterceptor
* Before : org.springframework.aop.BeforeAdvice
* After : org.springframework.aop.AfterReturningAdvice
* Throws : org.springframework.aop.ThrowsAdvice

## 39) What are the different types of AutoProxying?

* BeanNameAutoProxyCreator
* DefaultAdvisorAutoProxyCreator
* Metadata autoproxying

## 40) What is the Exception class related to all the exceptions that are thrown in spring applications?

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | DataAccessException - |
| 2 | org.springframework.dao.DataAccessException | |

## 41) What kind of exceptions those spring DAO classes throw?

The spring’s DAO class does not throw any technology related exceptions such as SQLException. They throw exceptions which are subclasses of DataAccessException.

## 42) What is DataAccessException?

DataAccessException is a RuntimeException. This is an Unchecked Exception. The user is not forced to handle these kinds of exceptions.

## 43) How can you configure a bean to get DataSource from JNDI?

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <bean id='dataSource' |
| 2 | class='org.springframework.jndi.JndiObjectFactoryBean'> | |

|  |  |
| --- | --- |
| 3 | <property name='jndiName'> |
| 4 | <value>java:comp/env/jdbc/myDatasource</value> | |

|  |  |  |
| --- | --- | --- |
| 5 | </property> | |
| 6 | </bean> |

## 44) How can you create a DataSource connection pool?

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <bean id='dataSource' |
| 2 | class='org.apache.commons.dbcp.BasicDataSource'> | |

|  |  |
| --- | --- |
| 3 | <property name='driver'> |
| 4 | <value>${db.driver}</value> | |

|  |  |
| --- | --- |
| 5 | </property> |
| 6 | <property name='url'> | |

|  |  |  |
| --- | --- | --- |
| 7 | <value>${db.url}</value> | |
| 8 | </property> |

|  |  |  |
| --- | --- | --- |
| 9 | <property name='username'> | |
| 10 | | <value>${db.username}</value> | |

|  |  |
| --- | --- |
| 11 | </property> |
| 12 | <property name='password'> | |

|  |  |  |
| --- | --- | --- |
| 13 | <value>${db.password}</value> | |
| 14 | </property> |

|  |  |
| --- | --- |
| 15 | </bean> |

## 45) How JDBC can be used more efficiently in spring framework?

JDBC can be used more efficiently with the help of a template class provided by spring framework called as **JdbcTemplate**.

## 46) How JdbcTemplate can be used?

With use of Spring JDBC framework the burden of resource management and error handling is reduced a lot. So it leaves developers to write the statements and queries to get the data to and from the database.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |
| --- | --- |
| 1 | <strong>JdbcTemplate</strong> template = new <strong>JdbcTemplate</strong>(myDataSource); |

A simple DAO class looks like this.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |  |
| --- | --- | --- |
| 1 | public class StudentDaoJdbc implements StudentDao { | |
| 2 | private JdbcTemplate jdbcTemplate; |

|  |  |  |
| --- | --- | --- |
| 3 | public void setJdbcTemplate(JdbcTemplate jdbcTemplate) { | |
| 4 | this.jdbcTemplate = jdbcTemplate; |

|  |  |  |
| --- | --- | --- |
| 5 | } more.. | |
| 6 | } |

The configuration is shown below.

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |  |
| --- | --- | --- |
| 1 | <bean id='jdbcTemplate' class='org.springframework.jdbc.core.JdbcTemplate'> | |
| 2 | <property name='dataSource'> |

|  |  |  |
| --- | --- | --- |
| 3 | <ref bean='dataSource'/> | |
| 4 | </property> |

|  |  |
| --- | --- |
| 5 | </bean> |
| 6 | <bean id='studentDao' class='StudentDaoJdbc'> | |

|  |  |  |
| --- | --- | --- |
| 7 | <property name='jdbcTemplate'> | |
| 8 | <ref bean='jdbcTemplate'/> |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | </property> | | |
| 10 | | </bean> |

|  |  |  |
| --- | --- | --- |
| 11 | <bean id='courseDao' class='CourseDaoJdbc'> | |
| 12 | <property name='jdbcTemplate'> |

|  |  |  |
| --- | --- | --- |
| 13 | <ref bean='jdbcTemplate'/> | |
| 14 | </property> |

|  |  |
| --- | --- |
| 15 | </bean> |

## 47) How do you write data to backend in spring using JdbcTemplate?

The JdbcTemplate uses several of these callbacks when writing data to the database. The usefulness you will find in each of these interfaces will vary. There are two simple interfaces. One is **PreparedStatementCreator** and the other interface is **BatchPreparedStatementSetter**.

## 48) Explain about PreparedStatementCreator?

PreparedStatementCreator is one of the most common used interfaces for writing data to database. The interface has one method createPreparedStatement().

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |  |
| --- | --- | --- |
| 1 | PreparedStatement <strong>createPreparedStatement</strong> | |
| 2 | (Connection conn) throws SQLException; |

When this interface is implemented, we should create and return a PreparedStatement from the Connection argument, and the exception handling is automatically taken care off. When this interface is implemented, another interface **SqlProvider** is also implemented which has a method called **getSql()** which is used to provide sql strings to JdbcTemplate.

## 49) Explain about BatchPreparedStatementSetter?

If the user what to update more than one row at a shot then he can go for **BatchPreparedStatementSetter**. This interface provides two methods

[view source](http://www.javabeat.net/spring-framework-interview-questions/#viewSource)

[print](http://www.javabeat.net/spring-framework-interview-questions/#printSource)[?](http://www.javabeat.net/spring-framework-interview-questions/#about)

|  |  |  |
| --- | --- | --- |
| 1 | setValues(PreparedStatement ps, int i) throws SQLException; | |
| 2 | int getBatchSize(); |

The getBatchSize() tells the JdbcTemplate class how many statements to create. And this also determines how many times setValues() will be called.

## 50) Explain about RowCallbackHandler and why it is used?

In order to navigate through the records we generally go for ResultSet. But spring provides an interface that handles this entire burden and leaves the user to decide what to do with each row. The interface provided by spring is **RowCallbackHandler**. There is a method processRow() which needs to be implemented so that it is applicable for each and everyrow.