**1.  What is IOC (or Dependency Injection)?**

**1. What is Inversion of Control (IoC)?**

**IoC** is a **design principle** in which the control over program flow is **inverted** — instead of your code controlling dependencies directly, the **framework or container** does it for you.

**🔸 Traditional Code (No IoC):**

Car car = new Car(); // You create the object

**🔸 IoC Style:**

@Autowired

Car car; // The framework injects the dependency

✅ **Key Idea**: Your classes **delegate the creation and management** of their dependencies to a framework (like Spring), instead of doing it themselves.

**🔌 2. What is Dependency Injection (DI)?**

**Dependency Injection** is a **design pattern** (and technique) to implement IoC, where an object receives its dependencies from an **external source** rather than creating them itself.

🔄 **Dependency Injection is the most common way to achieve IoC.**

**2. What are the different types of IOC (dependency injection) ?**

There are three types of dependency injection:

* **Constructor Injection** (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
* **Setter Injection** (e.g. Spring): Dependencies are assigned through JavaBeans properties (ex: setter methods).
* **Interface Injection**(e.g. Avalon): Injection is done through an interface.

*Note: Spring supports only Constructor and Setter Injection*

**3. What are the benefits of IOC (Dependency Injection)?**

Benefits of IOC (Dependency Injection) are as follows:

* Minimizes the amount of code in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method with little or no extra configuration.
* Make your application more testable by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.
* Loose coupling is promoted with minimal effort and least intrusive mechanism. The factory design pattern is more intrusive because components or services need to be requested explicitly whereas in IOC the dependency is injected into requesting piece of code. Also some containers promote the design to interfaces not to implementations design concept by encouraging managed objects to implement a well-defined service interface of your own.
* IOC containers support eager instantiation and lazy loading of services. Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management, and dependency resolution between managed objects etc.

|  |
| --- |
|  |

**4.  What is Spring ?**

Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

**5. What are the advantages of Spring framework?**

The advantages of Spring are as follows:

* Spring has layered architecture. Use what you need and leave you don't need now.
* Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables continuous integration and testability.
* Dependency Injection and Inversion of Control Simplifies JDBC
* Open source and no vendor lock-in.

**6. What are features of spring?**

* **Lightweight:**

spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 1MB. And the processing overhead is also very negligible.

* **Inversion of control (IOC):**

Loose coupling is achieved in spring using the technique Inversion of Control. The objects give their dependencies instead of creating or looking for dependent objects.

* **Aspect oriented (AOP):**

Spring supports Aspect oriented programming and enables cohesive development by separating application business logic from system services.

* **Container:**

Spring contains and manages the life cycle and configuration of application objects.

* **MVC Framework:**

Spring comes with MVC web application framework, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

* **Transaction Management:**

Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.

* **JDBC Exception Handling:**

The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling strategy. Integration with Hibernate, JDO, and iBATIS: Spring provides best Integration services with Hibernate, JDO and iBATIS

**7. How many modules are there in Spring? What are they?**

       Spring comprises of seven modules. They are..

* **The core container:**

The core container provides the essential functionality of the Spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the *Inversion of Control* (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.

* **Spring context:**

The Spring context is a configuration file that provides context information to the Spring framework. The Spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation, and scheduling functionality.

* **Spring AOP:**

The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the Spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

* **Spring DAO:**

The Spring JDBC DAO abstraction layer offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply to its generic DAO exception hierarchy.

* **Spring ORM:**

The Spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply to Spring's generic transaction and DAO exception hierarchies.

* **Spring Web module:**

The Web context module builds on top of the application context module, providing contexts for Web-based applications. As a result, the Spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

* **Spring MVC framework:**

The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

**8. What are the types of Dependency Injection Spring supports?>**

* **Setter Injection:**

Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

* **Constructor Injection:**

Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.

**9. What is Bean Factory?**

A BeanFactory is like a factory class that contains a collection of beans. The BeanFactory holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

* BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden of configuration from bean itself and the beans client.
* BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods

**10. What is Application Context?**

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context. On the surface, an application context is same as a bean factory.Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

* A means for resolving text messages, including support for internationalization.
* A generic way to load file resources.
* Events to beans that are registered as listeners.

**11. What is the difference between Bean Factory and Application Context?**

On the surface, an application context is same as a bean factory. But application context offers much more.

* Application contexts provide a means for resolving text messages, including support for i18n of those messages.
* Application contexts provide a generic way to load file resources, such as images.
* Application contexts can publish events to beans that are registered as listeners.
* Certain operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context.
* ResourceLoader support: Spring’s Resource interface us a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, Hence provides an application with access to deployment-specific Resource instances.
* MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable

**12. What are the common implementations of the Application Context?**

   The three commonly used implementation of 'Application Context' are

* **ClassPathXmlApplicationContext :** It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code .  
  ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
* **FileSystemXmlApplicationContext :** It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code .  
  ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
* **XmlWebApplicationContext :**It loads context definition from an XML file contained within a web application.

**13. How is a typical spring implementation look like ?**

   For a typical Spring Application, we need the following files:

* An interface that defines the functions.
* An Implementation that contains properties, its setter and getter methods, functions etc.,
* Spring AOP (Aspect Oriented Programming)
* A XML file called Spring configuration file.
* Client program that uses the function.

**14.  What is the typical Bean life cycle in Spring Bean Factory Container ?**

   Bean life cycle in Spring Bean Factory Container is as follows:

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

**15. What do you mean by Bean wiring?**

The act of creating associations between application components (beans) within the Spring container is referred to as Bean wiring.

**16. What do you mean by Auto Wiring?**

   The Spring container is able to auto wire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. The autowiring functionality has *five modes*.

* no
* byName
* byType
* constructor
* autodirect

**17. What is DelegatingVariableResolver?**

       Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard Java Server Faces managed beans mechanism which lets you use JSF and Spring together. This variable resolver is called as*DelegatingVariableResolver*

**18. How to integrate Java Server Faces (JSF) with Spring?**

JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By declaring JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties. We can integrate JSF and Spring in two ways:

* **DelegatingVariableResolver:** Spring comes with a JSF variable resolver that lets you use JSF and Spring together.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"

"http://www.springframework.org/dtd/spring-beans.dtd">

<faces-config>

<application>

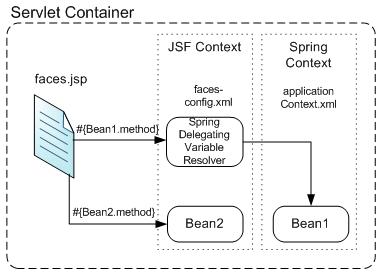
<variable-resolver>

org.springframework.web.jsf.DelegatingVariableResolver

</variable-resolver>

</application>

</faces-config>



The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-managed beans.

* FacesContextUtils:custom VariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.

ApplicationContext ctx = FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance());

[**More about Spring-JSF integration » »**](http://www.developersbook.com/articles/jsf/spring2-jsf-integration.php)

**19. What is Java Server Faces (JSF) - Spring integration mechanism?**

Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard JavaServer Faces managed beans mechanism. When asked to resolve a variable name, the following algorithm is performed:

* Does a bean with the specified name already exist in some scope (request, session, application)? If so, return it
* Is there a standard JavaServer Faces managed bean definition for this variable name? If so, invoke it in the usual way, and return the bean that was created.
* Is there configuration information for this variable name in the Spring WebApplicationContext for this application? If so, use it to create and configure an instance, and return that instance to the caller.
* If there is no managed bean or Spring definition for this variable name, return null instead.
* BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

As a result of this algorithm, you can transparently use either JavaServer Faces or Spring facilities to create beans on demand.

**20. What is Significance of JSF- Spring integration ?**

Spring - JSF integration is useful when an event handler wishes to explicitly invoke the bean factory to create beans on demand, such as a bean that encapsulates the business logic to be performed when a submit button is pressed.

**21. How to integrate your Struts application with Spring?**

To integrate your Struts application with Spring, we have two options:

* Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.
* Subclass Spring's ActionSupport classes and grab your Spring-managed beans explicitly using a getWebApplicationContext() method.

**22. What are ORM’s Spring supports ?**

**Spring supports the following ORM’s** :

* Hibernate
* iBatis
* JPA (Java Persistence API)
* TopLink
* JDO (Java Data Objects)
* OJB

**23. What are the ways to access Hibernate using Spring ?**

   There are two approaches to Spring’s Hibernate integration:

* Inversion of Control with a HibernateTemplate and Callback
* Extending HibernateDaoSupport and Applying an AOP Interceptor

**24. How to integrate Spring and Hibernate using HibernateDaoSupport?**

   Spring and Hibernate can integrate using Spring’s SessionFactory called LocalSessionFactory. The integration process is of 3 steps.

* Configure the Hibernate SessionFactory
* Extend your DAO Implementation from HibernateDaoSupport
* Wire in Transaction Support with AOP

**25. What are Bean scopes in Spring Framework ?**

| **Scope** | **Description** |
| --- | --- |
| singleton | Scopes a single bean definition to a single object instance per Spring IoC container. |
| Prototype | Scopes a single bean definition to any number of object instances. |
| request | Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext. |
| session | Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext. |
| global session | Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext. |

   The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext). The scopes supported are listed below:

|  |
| --- |
|  |

**26. What is AOP?**

   Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules.

**27. How the AOP used in Spring?**

   AOP is used in the Spring Framework: To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on the Spring Framework's transaction abstraction. To allow users to implement custom aspects, complementing their use of OOP with AOP.

**28. What do you mean by Aspect ?**

 A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).

**29. What do you mean by JointPoint ?**

A point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

**30. What do you mean by Advice?**

Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

**31. What are the types of Advice?**

Types of advice:

* Before advice: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
* After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
* After throwing advice: Advice to be executed if a method exits by throwing an exception.
* After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
* Around advice: Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception

# 

**32. What are the types of the transaction management Spring supports ?**

   Spring Framework supports:

* Programmatic transaction management.
* Declarative transaction management.

**33. What are the benefits of the Spring Framework transaction management ?**

   The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:

* Provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
* Supports declarative transaction management.
* Provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
* Integrates very well with Spring's various data access abstractions.

**34.  Why most users of the Spring Framework choose declarative transaction management ?**

   Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container.

**35. Explain the similarities and differences between EJB CMT and the Spring Framework's declarative transaction  
       management ?**

   The basic approach is similar: it is possible to specify transaction behavior (or lack of it) down to individual method level. It is  
    possible to make a setRollbackOnly() call within a transaction context if necessary. The differences are:

* Unlike EJB CMT, which is tied to JTA, the Spring Framework's declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.
* The Spring Framework enables declarative transaction management to be applied to any class, not merely special classes such as EJBs.
* The Spring Framework offers declarative rollback rules: this is a feature with no EJB equivalent. Both programmatic and declarative support for rollback rules is provided.
* The Spring Framework gives you an opportunity to customize transactional behavior, using AOP. With EJB CMT, you have no way to influence the container's transaction management other than setRollbackOnly().
* The Spring Framework does not support propagation of transaction contexts across remote calls, as do high-end application servers.

**37. When to use programmatic and declarative transaction management ?**

   Programmatic transaction management is usually a good idea only if you have a small number of transactional operations.   
On the other hand, if your application has numerous transactional operations, declarative transaction management is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

**38. Explain about the Spring DAO support ?**

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

**39. What are the exceptions thrown by the Spring DAO classes ?**

Spring DAO classes throw exceptions which are subclasses of DataAccessException(org.springframework.dao.DataAccessException).Spring provides a convenient translation from technology-specific exceptions likeSQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

**40. What is SQLExceptionTranslator ?**

SQLExceptionTranslator, is an interface to be implemented by classes that can translate between SQLExceptions and Spring's own data-access-strategy-agnosticorg.springframework.dao.DataAccessException.

**41. What is Spring's JdbcTemplate ?**

Spring's *JdbcTemplate* is central class to interact with a database through JDBC. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

JdbcTemplate template = new JdbcTemplate(myDataSource);

**42. What is PreparedStatementCreator ?**

   PreparedStatementCreator:

* Is one of the most common used interfaces for writing data to database.
* Has one method – createPreparedStatement(Connection)
* Responsible for creating a PreparedStatement.
* Does not need to handle SQLExceptions.

**43. What is SQLProvider ?**

   SQLProvider:

* Has one method – getSql()
* Typically implemented byPreparedStatementCreator implementers.
* Useful for debugging.

**44. What is RowCallbackHandler ?**

   The RowCallbackHandler interface extracts values from each row of a ResultSet.

* Has one method – processRow(ResultSet)
* Called for each row in ResultSet.
* Typically stateful.

**45. What are the differences between EJB and Spring ?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **EJB** | **Spring** |
| Transaction management | * Must use a JTA transaction manager. * Supports transactions that span remote method calls. | * Supports multiple transaction environments through itsPlatformTransactionManager interface, including JTA, Hibernate, JDO, and JDBC. * Does not natively support distributed transactions—it must be used with a JTA transaction manager. |
| Declarative transaction support | * Can define transactions declaratively through the deployment descriptor. * Can define transaction behavior per method or per class by using the wildcard character \*. * Cannot declaratively define rollback behavior—this must be done programmatically. | * Can define transactions declaratively through the Spring configuration file or through class metadata. * Can define which methods to apply transaction behavior explicitly or by using regular expressions. * Can declaratively define rollback behavior per method and per exception type. |
| Persistence | Supports programmatic bean-managed persistence and declarative container managed persistence. | Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS. |
| Declarative security | * Supports declarative security through users and roles. The management and implementation of users and roles is container specific. * Declarative security is configured in the deployment descriptor. | * No security implementation out-of-the box. * Acegi, an open source security framework built on top of Spring, provides declarative security through the Spring configuration file or class metadata. |
| Distributed computing | Provides container-managed remote method calls. | Provides proxying for remote calls via RMI, JAX-RPC, and web services. |

   Spring and EJB feature comparison.

===========================================================================

### 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### 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).

### 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.

### 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.

### 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 andorg.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.

### What is a Spring Bean?

Any normal java class that is initialized by Spring IoC container is called Spring Bean. We use SpringApplicationContext 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.

### 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.

### 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" /> |

[sociallocker id=”2713″]

### What are different scopes of Spring Bean?

There are five scopes defined for Spring Beans.

* + **singleton**: Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
  + **prototype**: A new instance will be created every time the bean is requested.
  + **request**: This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
  + **session**: A new bean will be created for each HTTP session by the container.
  + **global-session**: This is used to create global session beans for Portlet applications.

Spring Framework is extendable and we can create our own scopes too, however most of the times we are good with the scopes provided by the framework.

To set spring bean scopes we can use “scope” attribute in bean element or @Scope annotation for annotation based configurations.

### What is Spring Bean life cycle?

Spring Beans are initialized by Spring Container and all the dependencies are also injected. When context is destroyed, it also destroys all the initialized beans. This works well in most of the cases but sometimes we want to initialize other resources or do some validation before making our beans ready to use. Spring framework provides support for post-initialization and pre-destroy methods in spring beans.

We can do this by two ways – by implementing InitializingBean and DisposableBean interfaces or using**init-method** and **destroy-method** attribute in spring bean configurations. For more details, please read [Spring Bean Life Cycle Methods](http://www.journaldev.com/2637/spring-bean-life-cycle-methods-initializingbean-disposablebean-postconstruct-predestroy-aware-interfaces).

### How to get ServletContext and ServletConfig object in a Spring Bean?

There are two ways to get Container specific objects in the spring bean.

* + Implementing Spring \*Aware interfaces, for these ServletContextAware and ServletConfigAware interfaces, for complete example of these aware interfaces, please read [Spring Aware Interfaces](http://www.journaldev.com/2637/spring-bean-life-cycle-methods-initializingbean-disposablebean-postconstruct-predestroy-aware-interfaces)
  + Using @Autowired annotation with bean variable of type ServletContext and ServletConfig. They will work only in servlet container specific environment only though.

|  |  |
| --- | --- |
| 1  2 | @Autowired  ServletContext servletContext; |

### What is Bean wiring and @Autowired annotation?

The process of injection spring bean dependencies while initializing it called Spring Bean Wiring.

Usually it’s best practice to do the explicit wiring of all the bean dependencies, but spring framework also supports autowiring. We can use @Autowired annotation with fields or methods for **autowiring byType**. For this annotation to work, we also need to enable annotation based configuration in spring bean configuration file. This can be done by **context:annotation-config** element.

For more details about @Autowired annotation, please read [Spring Autowire Example](http://www.journaldev.com/2623/spring-bean-autowire-by-name-type-constructor-autowired-and-qualifier-annotations-example).

### What are different types of Spring Bean autowiring?

There are four types of autowiring in Spring framework.

* + **autowire byName**
  + **autowire byType**
  + **autowire by constructor**
  + autowiring by **@Autowired** and **@Qualifier** annotations

Prior to Spring 3.1, **autowire by autodetect** was also supported that was similar to autowire by constructor or byType. For more details about these options, please read [Spring Bean Autowiring](http://www.journaldev.com/2623/spring-bean-autowire-by-name-type-constructor-autowired-and-qualifier-annotations-example).

### Does Spring Bean provide thread safety?

The default scope of Spring bean is singleton, so there will be only one instance per context. That means that all the having a class level variable that any thread can update will lead to inconsistent data. Hence in default mode spring beans are not thread-safe.

However we can change spring bean scope to request, prototype or session to achieve thread-safety at the cost of performance. It’s a design decision and based on the project requirements.

### What is a Controller in Spring MVC?

Just like MVC design pattern, Controller is the class that takes care of all the client requests and send them to the configured resources to handle it. In Spring MVC,org.springframework.web.servlet.DispatcherServlet is the front controller class that initializes the context based on the spring beans configurations.

A Controller class is responsible to handle different kind of client requests based on the request mappings. We can create a controller class by using @Controller annotation. Usually it’s used with@RequestMapping annotation to define handler methods for specific URI mapping.

### What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

**@Component** is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used.

**@Controller** is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation.

**@Repository** annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service** is used to indicate that a class is a Service. Usually the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that you can easily distinguish the purpose of the annotated classes.

### What is DispatcherServlet and ContextLoaderListener?

DispatcherServlet is the front controller in the Spring MVC application and it loads the spring bean configuration file and initialize all the beans that are configured. If annotations are enabled, it also scans the packages and configure any bean annotated with @Component, @Controller, @Repository or@Service annotations.

1. ContextLoaderListener is the listener to start up and shut down Spring’s root WebApplicationContext. It’s important functions are to tie up the lifecycle of ApplicationContext to the lifecycle of the ServletContextand to automate the creation of ApplicationContext. We can use it to define shared beans that can be used across different spring contexts.

### What is ViewResolver in Spring?

ViewResolver implementations are used to resolve the view pages by name. Usually we configure it in the spring bean configuration file. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->  <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">      <beans:property name="prefix" value="/WEB-INF/views/" />      <beans:property name="suffix" value=".jsp" />  </beans:bean> |

InternalResourceViewResolver is one of the implementation of ViewResolver interface and we are providing the view pages directory and suffix location through the bean properties. So if a controller handler method returns “home”, view resolver will use view page located at /WEB-INF/views/home.jsp.

### What is a MultipartResolver and when its used?

MultipartResolver interface is used for uploading files – CommonsMultipartResolver andStandardServletMultipartResolver are two implementations provided by spring framework for file uploading. By default there are no multipart resolvers configured but to use them for uploading files, all we need to define a bean named “multipartResolver” with type as MultipartResolver in spring bean configurations.

Once configured, any multipart request will be resolved by the configured MultipartResolver and pass on a wrapped HttpServletRequest. Then it’s used in the controller class to get the file and process it. For a complete example, please read [Spring MVC File Upload Example](http://www.journaldev.com/2573/spring-mvc-file-upload-example-tutorial-single-and-multiple-files).

### How to handle exceptions in Spring MVC Framework?

Spring MVC Framework provides following ways to help us achieving robust exception handling.

* 1. **Controller Based** – We can define exception handler methods in our controller classes. All we need is to annotate these methods with @ExceptionHandler annotation.
  2. **Global Exception Handler** – Exception Handling is a cross-cutting concern and Spring provides @ControllerAdvice annotation that we can use with any class to define our global exception handler.
  3. **HandlerExceptionResolver implementation** – For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

For a complete example, please read [Spring Exception Handling Example](http://www.journaldev.com/2651/spring-mvc-exception-handling-exceptionhandler-controlleradvice-handlerexceptionresolver-json-response-example).

### How to create ApplicationContext in a Java Program?

There are following ways to create spring context in a standalone java program.

* 1. **AnnotationConfigApplicationContext**: If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.
  2. **ClassPathXmlApplicationContext**: If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.
  3. **FileSystemXmlApplicationContext**: This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.

### Can we have multiple Spring configuration files?

For Spring MVC applications, we can define multiple spring context configuration files through contextConfigLocation. This location string can consist of multiple locations separated by any number of commas and spaces. For example;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <servlet>      <servlet-name>appServlet</servlet-name>      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>      <init-param>          <param-name>contextConfigLocation</param-name>          <param-value>/WEB-INF/spring/appServlet/servlet-context.xml,/WEB-INF/spring/appServlet/servlet-jdbc.xml</param-value>      </init-param>      <load-on-startup>1</load-on-startup>  </servlet> |

We can also define multiple root level spring configurations and load it through context-param. For example;

|  |  |
| --- | --- |
| 1  2  3  4 | <context-param>      <param-name>contextConfigLocation</param-name>      <param-value>/WEB-INF/spring/root-context.xml /WEB-INF/spring/root-security.xml</param-value>  </context-param> |

Another option is to use import element in the context configuration file to import other configurations, for example:

|  |  |
| --- | --- |
| 1 | <beans:import resource="spring-jdbc.xml"/> |

### What is ContextLoaderListener?

ContextLoaderListener is the listener class used to load root context and define spring bean configurations that will be visible to all other contexts. It’s configured in web.xml file as:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <context-param>      <param-name>contextConfigLocation</param-name>      <param-value>/WEB-INF/spring/root-context.xml</param-value>  </context-param>    <listener>      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

### What are the minimum configurations needed to create Spring MVC application?

For creating a simple Spring MVC application, we would need to do following tasks.

* 1. Add spring-context and spring-webmvc dependencies in the project.
  2. Configure DispatcherServlet in the web.xml file to handle requests through spring container.
  3. Spring bean configuration file to define beans, if using annotations then it has to be configured here. Also we need to configure view resolver for view pages.
  4. Controller class with request mappings defined to handle the client requests.

Above steps should be enough to create a simple Spring MVC Hello World application.

### How would you relate Spring MVC Framework to MVC architecture?

As the name suggests Spring MVC is built on top of **Model-View-Controller** architecture.DispatcherServlet is the Front Controller in the Spring MVC application that takes care of all the incoming requests and delegate it to different controller handler methods.

Model can be any Java Bean in the Spring Framework, just like any other MVC framework Spring provides automatic binding of form data to java beans. We can set model beans as attributes to be used in the view pages.

View Pages can be JSP, static HTMLs etc. and view resolvers are responsible for finding the correct view page. Once the view page is identified, control is given back to the DispatcherServlet controller. DispatcherServlet is responsible for rendering the view and returning the final response to the client.

### How to achieve localization in Spring MVC applications?

Spring provides excellent support for localization or i18n through resource bundles. Basis steps needed to make our application localized are:

* 1. Creating message resource bundles for different locales, such as messages\_en.properties, messages\_fr.properties etc.
  2. Defining messageSource bean in the spring bean configuration file of typeResourceBundleMessageSource or ReloadableResourceBundleMessageSource.
  3. For change of locale support, define localeResolver bean of type CookieLocaleResolver and configure LocaleChangeInterceptor interceptor. Example configuration can be like below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <beans:bean id="messageSource"      class="org.springframework.context.support.ReloadableResourceBundleMessageSource">      <beans:property name="basename" value="classpath:messages" />      <beans:property name="defaultEncoding" value="UTF-8" />  </beans:bean>    <beans:bean id="localeResolver"      class="org.springframework.web.servlet.i18n.CookieLocaleResolver">      <beans:property name="defaultLocale" value="en" />      <beans:property name="cookieName" value="myAppLocaleCookie"></beans:property>      <beans:property name="cookieMaxAge" value="3600"></beans:property>  </beans:bean>    <interceptors>      <beans:bean          class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">          <beans:property name="paramName" value="locale" />      </beans:bean>  </interceptors> |

* 1. Use spring:message element in the view pages with key names, DispatcherServlet picks the corresponding value and renders the page in corresponding locale and return as response.

For a complete example, please read [Spring Localization Example](http://www.journaldev.com/2610/spring-mvc-internationalization-i18n-and-localization-l10n-example).

### How can we use Spring to create Restful Web Service returning JSON response?

We can use Spring Framework to create Restful web services that returns JSON data. Spring provides integration with [Jackson JSON API](http://www.journaldev.com/2324/jackson-json-processing-api-in-java-example-tutorial) that we can use to send JSON response in restful web service.

We would need to do following steps to configure our Spring MVC application to send JSON response:

* 1. Adding Jackson JSON dependencies, if you are using Maven it can be done with following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <!-- Jackson -->  <dependency>      <groupId>com.fasterxml.jackson.core</groupId>      <artifactId>jackson-databind</artifactId>      <version>${jackson.databind-version}</version>  </dependency> |

* 1. Configure RequestMappingHandlerAdapter bean in the spring bean configuration file and set the messageConverters property to MappingJackson2HttpMessageConverter bean. Sample configuration will be:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <!-- Configure to plugin JSON as request and response in method handler -->  <beans:bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">      <beans:property name="messageConverters">          <beans:list>              <beans:ref bean="jsonMessageConverter"/>          </beans:list>      </beans:property>  </beans:bean>    <!-- Configure bean to convert JSON to POJO and vice versa -->  <beans:bean id="jsonMessageConverter" class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">  </beans:bean> |

* 1. In the controller handler methods, return the Object as response using @ResponseBody annotation. Sample code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @RequestMapping(value = EmpRestURIConstants.GET\_EMP, method = RequestMethod.GET)  public @ResponseBody Employee getEmployee(@PathVariable("id") int empId) {      logger.info("Start getEmployee. ID="+empId);        return empData.get(empId);  } |

* 1. You can invoke the rest service through any API, but if you want to use Spring then we can easily do it using RestTemplate class.

For a complete example, please read [Spring Restful Webservice Example](http://www.journaldev.com/2552/spring-restful-web-service-example-with-json-jackson-and-client-program).

### What are some of the important Spring annotations you have used?

Some of the Spring annotations that I have used in my project are:

* 1. **@Controller** – for controller classes in Spring MVC project.
  2. **@RequestMapping** – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through [Spring MVC RequestMapping Annotation Examples](http://www.journaldev.com/3358/spring-mvc-requestmapping-annotation-example-with-controller-methods-headers-params-requestparam-pathvariable)
  3. **@ResponseBody** – for sending Object as response, usually for sending XML or JSON data as response.
  4. **@PathVariable** – for mapping dynamic values from the URI to handler method arguments.
  5. **@Autowired** – for autowiring dependencies in spring beans.
  6. **@Qualifier** – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.
  7. **@Service** – for service classes.
  8. **@Scope** – for configuring scope of the spring bean.
  9. **@Configuration**, **@ComponentScan** and **@Bean** – for java based configurations.
  10. AspectJ annotations for configuring aspects and advices, **@Aspect**, **@Before**, **@After**, **@Around**,**@Pointcut** etc.

### Can we send an Object as the response of Controller handler method?

Yes we can, using **@ResponseBody** annotation. This is how we send JSON or XML based response in restful web services.

### How to upload file in Spring MVC Application?

Spring provides built-in support for uploading files through **MultipartResolver** interface implementations. It’s very easy to use and requires only configuration changes to get it working. Obviously we would need to write controller handler method to handle the incoming file and process it. For a complete example, please refer [Spring File Upload Example](http://www.journaldev.com/2573/spring-mvc-file-upload-example-tutorial-single-and-multiple-files).

### How to validate form data in Spring Web MVC Framework?

Spring supports JSR-303 annotation based validations as well as provide Validator interface that we can implement to create our own custom validator. For using JSR-303 based validation, we need to annotate bean variables with the required validations.

For custom validator implementation, we need to configure it in the controller class. For a complete example, please read [Spring MVC Form Validation Example](http://www.journaldev.com/2668/spring-mvc-form-validation-example-using-annotation-and-custom-validator-implementation).

### What is Spring MVC Interceptor and how to use it?

Spring MVC Interceptors are like Servlet Filters and allow us to intercept client request and process it. We can intercept client request at three places – **preHandle**, **postHandle** and **afterCompletion**.

We can create spring interceptor by implementing HandlerInterceptor interface or by extending abstract class **HandlerInterceptorAdapter**.

We need to configure interceptors in the spring bean configuration file. We can define an interceptor to intercept all the client requests or we can configure it for specific URI mapping too. For a detailed example, please refer [Spring MVC Interceptor Example](http://www.journaldev.com/2676/spring-mvc-interceptors-example-handlerinterceptor-and-handlerinterceptoradapter).

### What is Spring JdbcTemplate class and how to use it?

Spring Framework provides excellent integration with JDBC API and provides JdbcTemplate utility class that we can use to avoid bolier-plate code from our database operations logic such as Opening/Closing Connection, ResultSet, PreparedStatement etc.

For JdbcTemplate example, please refer [Spring JDBC Example](http://www.journaldev.com/2593/spring-jdbc-and-jdbctemplate-crud-with-datasource-example-tutorial).

### How to use Tomcat JNDI DataSource in Spring Web Application?

For using servlet container configured JNDI DataSource, we need to configure it in the spring bean configuration file and then inject it to spring beans as dependencies. Then we can use it withJdbcTemplate to perform database operations.

Sample configuration would be:

|  |  |
| --- | --- |
| 1  2  3 | <beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">      <beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>  </beans:bean> |

For complete example, please refer [Spring Tomcat JNDI Example](http://www.journaldev.com/2597/spring-datasource-jndi-with-tomcat-example).

### How would you achieve Transaction Management in Spring?

Spring framework provides transaction management support through Declarative Transaction Management as well as programmatic transaction management. Declarative transaction management is most widely used because it’s easy to use and works in most of the cases.

We use annotate a method with @Transactional annotation for Declarative transaction management. We need to configure transaction manager for the DataSource in the spring bean configuration file.

|  |  |
| --- | --- |
| 1  2  3  4 | <bean id="transactionManager"      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">      <property name="dataSource" ref="dataSource" />  </bean> |

### What is Spring DAO?

Spring DAO support is provided to work with data access technologies like JDBC, Hibernate in a consistent and easy way. For example we have JdbcDaoSupport, HibernateDaoSupport, JdoDaoSupport andJpaDaoSupport for respective technologies.

Spring DAO also provides consistency in exception hierarchy and we don’t need to catch specific exceptions.

### How to integrate Spring and Hibernate Frameworks?

We can use Spring ORM module to integrate Spring and Hibernate frameworks, if you are using Hibernate 3+ where SessionFactory provides current session, then you should avoid usingHibernateTemplate or HibernateDaoSupport classes and better to use DAO pattern with dependency injection for the integration.

Also Spring ORM provides support for using Spring declarative transaction management, so you should utilize that rather than going for hibernate boiler-plate code for transaction management.

For better understanding you should go through following tutorials:

* 1. [Spring Hibernate Integration Example](http://www.journaldev.com/3524/spring-hibernate-integration-example-tutorial-spring-4-hibernate-3-and-hibernate-4)
  2. [Spring MVC Hibernate Integration Example](http://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial)

### What is Spring Security?

Spring security framework focuses on providing both authentication and authorization in java applications. It also takes care of most of the common security vulnerabilities such as CSRF attack.

It’s very beneficial and easy to use Spring security in web applications, through the use of annotations such as @EnableWebSecurity. You should go through following posts to learn how to use Spring Security framework.

* 1. [Spring Security in Servlet Web Application](http://www.journaldev.com/2715/spring-security-in-servlet-web-application-using-dao-jdbc-in-memory-authentication)
  2. [Spring MVC and Spring Security Integration Example](http://www.journaldev.com/2736/spring-mvc-security-example-using-in-memory-userdetailsservice-and-jdbc-authentication)

### How to inject a java.util.Properties into a Spring Bean?

We need to define propertyConfigurer bean that will load the properties from the given property file. Then we can use Spring EL support to inject properties into other bean dependencies. For example;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <bean id="propertyConfigurer"    class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">      <property name="location" value="/WEB-INF/application.properties" />  </bean>    <bean class="com.journaldev.spring.EmployeeDaoImpl">      <property name="maxReadResults" value="${results.read.max}"/>  </bean> |

If you are using annotation to configure the spring bean, then you can inject property like below.

|  |  |
| --- | --- |
| 1  2 | @Value("${maxReadResults}")  private int maxReadResults; |

### Name some of the design patterns used in Spring Framework?

Spring Framework is using a lot of design patterns, some of the common ones are:

* 1. [Singleton Pattern](http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-with-examples): Creating beans with default scope.
  2. [Factory Pattern](http://www.journaldev.com/1392/factory-design-pattern-in-java): Bean Factory classes
  3. [Prototype Pattern](http://www.journaldev.com/1440/prototype-pattern-in-java): Bean scopes
  4. [Adapter Pattern](http://www.journaldev.com/1487/adapter-design-pattern-in-java-example-tutorial): Spring Web and Spring MVC
  5. [Proxy Pattern](http://www.journaldev.com/1572/proxy-design-pattern-in-java-example-tutorial): Spring Aspect Oriented Programming support
  6. [Template Method Pattern](http://www.journaldev.com/1763/template-method-design-pattern-in-java): JdbcTemplate, HibernateTemplate etc
  7. Front Controller: Spring MVC DispatcherServlet
  8. Data Access Object: Spring DAO support
  9. Dependency Injection and Aspect Oriented Programming

### What are some of the best practices for Spring Framework?

Some of the best practices for Spring Framework are:

* 1. Avoid version numbers in schema reference, to make sure we have the latest configs.
  2. Divide spring bean configurations based on their concerns such as spring-jdbc.xml, spring-security.xml.
  3. For spring beans that are used in multiple contexts in Spring MVC, create them in the root context and initialize with listener.
  4. Configure bean dependencies as much as possible, try to avoid autowiring as much as possible.
  5. For application level properties, best approach is to create a property file and read it in the spring bean configuration file.
  6. For smaller applications, annotations are useful but for larger applications annotations can become a pain. If we have all the configuration in xml files, maintaining it will be easier.
  7. Use correct annotations for components for understanding the purpose easily. For services use @Service and for DAO beans use @Repository.
  8. Spring framework has a lot of modules, use what you need. Remove all the extra dependencies that gets usually added when you create projects through Spring Tool Suite templates.
  9. If you are using Aspects, make sure to keep the join pint as narrow as possible to avoid advice on unwanted methods. Consider custom annotations that are easier to use and avoid any issues.
  10. Use dependency injection when there is actual benefit, just for the sake of loose-coupling don’t use it because it’s harder to maintain.

[/sociallocker]