**Q. What is spring? What are the various parts of spring framework? What are the different persistence frameworks which could be used with spring?**A. 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. The Spring modules are built on top of the core container, which defines how beans are created, configured, and managed, as shown in the following figure. Each of the modules (or components) that comprise the Spring framework can stand on its own or be implemented jointly with one or more of the others. The functionality of each component is as follows:  
  
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.

**Q. Explain DI or IOC pattern?**A:  Dependency injection (DI) is a programming design pattern and architectural model, sometimes also referred to as inversion of control or IOC, although technically speaking, dependency injection specifically refers to an implementation of a particular form of IOC. Dependancy Injection describes the situation where one object uses a second object to provide a particular capacity. For example, being passed a database connection as an argument to the constructor instead of creating one internally. The term "Dependency injection" is a misnomer, since it is not a dependency that is injected, rather it is a provider of some capability or resource that is injected. There are three common forms of dependency injection: setter-, constructor- and interface-based injection. Dependency injection is a way to achieve loose coupling. Inversion of control (IOC) relates to the way in which an object obtains references to its dependencies. This is often done by a lookup method. The advantage of inversion of control is that it decouples objects from specific lookup mechanisms and implementations of the objects it depends on. As a result, more flexibility is obtained for production applications as well as for testing.  
  
**Q. What are the different IOC containers available?**A. Spring is an IOC container. Other IOC containers are HiveMind, Avalon, PicoContainer.  
  
**Q. What are the different types of dependency injection. Explain with examples?**A: There are two types of dependency injection: setter injection and constructor injection.  
Setter Injection:  Normally in all the java beans, we will use setter and getter method to set and get the value of property as follows: 

    public class namebean {   
     String      name;     
     public void setName(String a) {   
        name = a; }   
     public String getName() {   
        return name; }   
    } 

We will create an instance of the bean 'namebean' (say bean1) and set property as bean1.setName("tom"); Here in setter injection, we will set the property 'name'  in spring configuration file as showm below:  
<bean id="bean1"   class="namebean">   
   <property   name="name" >   
       <value>tom</value>   
   </property>   
</bean>   
The subelement <value> sets the 'name' property by calling the set method as setName("tom"); This process is called setter injection.   
To set properties that reference other beans <ref>, subelement of <property> is used as shown below,   
 <bean id="bean1"   class="bean1impl">   
   <property name="game">   
       <ref bean="bean2"/>   
   </property>   
</bean>   
<bean id="bean2"   class="bean2impl" />

Constructor injection:  For constructor injection, we use constructor with parameters as shown below, 

 public class namebean {   
     String name;   
     public namebean(String a) {   
        name = a;   
     }      
} 

We will set the property 'name' while creating an instance of the bean 'namebean' as namebean bean1 = new namebean("tom"); 

Here we use the <constructor-arg> element to set the the property by constructor injection as   
 <bean id="bean1"  class="namebean">   
    <constructor-arg>   
       <value>My Bean Value</value>   
   </constructor-arg>   
</bean>

**Q. What is AOP? How does it relate with IOC? What are different tools to utilize AOP?**A:  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. AOP and IOC are complementary technologies in that both apply a modular approach to complex problems in enterprise application development. In a typical object-oriented development approach you might implement logging functionality by putting logger statements in all your methods and Java classes. In an AOP approach you would instead modularize the logging services and apply them declaratively to the components that required logging. The advantage, of course, is that the Java class doesn't need to know about the existence of the logging service or concern itself with any related code. As a result, application code written using Spring AOP is loosely coupled. The best tool to utilize AOP to its capability is AspectJ. However AspectJ works at he byte code level and you need to use AspectJ compiler to get the aop features built into your compiled code. Nevertheless AOP functionality is fully integrated into the Spring context for transaction management, logging, and various other features.  In general any AOP framework control aspects in three possible ways:

Joinpoints: Points in a program's execution. For example, joinpoints could define calls to specific methods in a class   
Pointcuts: Program constructs to designate joinpoints and collect specific context at those points   
Advices: Code that runs upon meeting certain conditions. For example, an advice could log a message before executing a joinpoint 

**Q. What are the advantages of spring framework?**A.

1. Spring has layed architecture. Use what you need and leave you don't need now.
2. Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables continous integration and testability.
3. Dependency Injection and Inversion of Control Simplifies JDBC (Read the first question.)
4. Open source and no vendor lock-in.

**Q. Can you name a tool which could provide the initial ant files and directory structure for a new spring project?**A: Appfuse or equinox.  
  
**Q. Explain BeanFactory in spring?**A: Bean factory is an implementation of the factory design pattern and its function is to create and dispense beans. As the bean factory knows about many objects within an application, it is able to create association between collaborating objects as they are instantiated. This removes the burden of configuration from the bean and the client. There are several implementation of BeanFactory. The most useful one is "org.springframework.beans.factory.xml.XmlBeanFactory" It loads its beans based on the definition contained in an XML file. To create an XmlBeanFactory, pass a InputStream to the constructor. The resource will provide the XML to the factory. BeanFactory  factory = new XmlBeanFactory(new FileInputStream("myBean.xml"));   
This line tells the bean factory to read the bean definition from the XML file. The bean definition includes the description of beans and their properties. But the bean factory doesn't instantiate the bean yet. To retrieve a bean from a 'BeanFactory', the getBean() method is called. When getBean() method is called, factory will instantiate the bean and begin setting the bean's properties using dependency injection. myBean bean1 = (myBean)factory.getBean("myBean");   
  
**Q. Explain the role of ApplicationContext in spring?**A. While Bean Factory is used for simple applications, the Application Context is spring's more advanced container. Like 'BeanFactory' it can be used to load bean definitions, wire beans together and dispense beans upon request. It also provide 

1) a means for resolving text messages, including support for internationalization.   
2) a generic way to load file resources.   
3) events to beans that are registered as listeners. 

Because of additional functionality, 'Application Context' is preferred over a BeanFactory. Only when the resource is scarce like mobile devices, 'BeanFactory' is used. The three commonly used implementation of 'Application Context' are 

1. 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");   
2. 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");   
3. XmlWebApplicationContext: It loads context definition from an XML file contained within a web application. 

**Q. How does Spring supports DAO in hibernate?**A. Spring’s HibernateDaoSupport class is a convenient super class for Hibernate DAOs. It has handy methods you can call to get a Hibernate Session, or a SessionFactory. The most convenient method is getHibernateTemplate(), which returns a HibernateTemplate. This template wraps Hibernate checked exceptions with runtime exceptions, allowing your DAO interfaces to be Hibernate exception-free.  
Example:

public class UserDAOHibernate extends HibernateDaoSupport {

public User getUser(Long id) {  
return (User) getHibernateTemplate().get(User.class, id);  
}  
public void saveUser(User user) {  
getHibernateTemplate().saveOrUpdate(user);  
if (log.isDebugEnabled()) {  
log.debug(“userId set to: “ + user.getID());  
}  
}  
public void removeUser(Long id) {  
Object user = getHibernateTemplate().load(User.class, id);  
getHibernateTemplate().delete(user);  
}

}

**Q. How is a typical spring implementation look like?**A. For a typical Spring Application we need the following files 

1. An interface that defines the functions.   
2. An Implementation that contains properties, its setter and getter methods, functions etc.,   
3. A XML file called Spring configuration file.   
4. Client program that uses the function. 

**Q. How do you define hibernate mapping file in spring?**A. Add the hibernate mapping file entry in mapping resource inside Spring’s applicationContext.xml file in the web/WEB-INF directory. 

<property name="mappingResources">  
    <list>  
        <value>org/appfuse/model/User.hbm.xml</value>  
    </list>  
</property>

**Q. How do you configure spring in a web application?**A. It is very easy to configure any J2EE-based web application to use Spring. At the very least, you can simply add Spring’s ContextLoaderListener to your web.xml file:

<listener>  
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>

**Q. Can you have xyz.xml file instead of applicationcontext.xml?**A. ContextLoaderListener is a ServletContextListener that initializes when your webapp starts up. By default, it looks for Spring’s configuration file at WEB-INF/applicationContext.xml. You can change this default value by specifying a <context-param> element named “contextConfigLocation.” Example:

<listener>  
    <listener-class>org.springframework.web.context.ContextLoaderListener

    <context-param>  
        <param-name>contextConfigLocation</param-name>  
        <param-value>/WEB-INF/xyz.xml</param-value>  
    </context-param>

    </listener-class>  
</listener>

**Q. How do you configure your database driver in spring?**  
A. Using datasource "org.springframework.jdbc.datasource.DriverManagerDataSource". Example:

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName">  
        <value>org.hsqldb.jdbcDriver</value>  
    </property>  
    <property name="url">  
        <value>jdbc:hsqldb:db/appfuse</value>  
    </property>  
    <property name="username"><value>sa</value></property>  
    <property name="password"><value></value></property>  
</bean>

**Q. How can you configure JNDI instead of datasource in spring applicationcontext.xml?**A. Using "org.springframework.jndi.JndiObjectFactoryBean". Example:

<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">  
    <property name="jndiName">  
        <value>java:comp/env/jdbc/appfuse</value>  
    </property>  
</bean>

**Q. How do you configured Spring DAO?**

**A.** In contextApplication.xml

**Q. How to integrate HB with Spring?**

**A.** The **DAO implementation class** of spring must extends **HibernateDAOSupport** and

implements **DAO interface.**

**Q. How to integrate Struts with Spring?**

**A.** ContextLoaderServlet **and**

applicationContext.xml configured in **struts-config.xml** that is there in **web.xml**

**Q. What is Setter and Constructor injection?**

**A.**

**Setter Injection:**

It is a **process** of **injecting**/assigning the **dependent values** to the **Bean properties**

by the **spring container** using **Bean class setXXX(-) methods**.

**Note:** In **Spring Config.xml:**

<**property** name=”age ”>

<**value**> 20</value>

</property>

**Constructor Injection:**

It is a process of assigning/injecting the dependent values to the Bean properties

by the spring container **using “parameterized Constructors”.**

**Note:** In **Spring config.xml**:

**<constructor-arg>**

**<value>**ramu</value>

</constructor-arg>

**Q. Can I develop a spring application without hbm file?**

**A.** Yes, by using “Annotation”

Q. **Can I develop a spring application without spring config.xml?**

**A.** Yes, by using Properties file.

**Q. What are the differences b/w struts, spring and hibernate?**

**A. Struts:** allows to develop **only** **web applications** and

it **can’t** support **POJO/POJI** model programming.

**Spring:** is useful in developing **all types of Java applications** and

**support POJO/P**OJI model programming.

**Hibernate:** is used to develop **DB independent persistence logic**

It also **supports POJO/POJI** model programming.

**Q. What classes are there in Spring Environment?**

**A.** Spring config.xml, spring bean class, spring bean interface.

**Q. How do you configured Spring DAO?**

**A.** In contextApplication.xml