**Spring**

**What is an abstract bean**

**What is the difference between BeanFactory and ApplicationContext**

Bean Factory

* Bean instantiation/wiring

Application Context

* Bean instantiation/wiring
* Automatic BeanPostProcessor registration
* Automatic BeanFactoryPostProcessor registration
* Convenient MessageSource access (for i18n)
* ApplicationEvent publication

**How to write java based configuration**

Use @Configuration annotated class to create bean definitions – annotated with @Bean.

Annotations used along with @Bean - @Qualifier, @Scope

@Component, @Controller, @Service, @Repository – no need to register these stereotype annotated classes separately.

**Explain templates – database access**

1. Less boilerplate code
2. More cohesive exceptions handling (e.g. JDBC drivers checked exceptions are translated to meaningful runtime exceptions)
3. Easier and uniform configuration (almost everything can be configured in Spring context files)
4. Automatized resource management (like, for instance, closing DB connections)

**Significance of @Service, @Repository**

@Repository qualifies annotated class as data access layer class and performs automatic translation of exceptions to DataAccessException. These stereotype annotations make ideal targets for pointcuts. It is also possible that @Repository, @Service, and @Controller may carry additional semantics in future releases of the Spring Framework.

**How to Rollback spring transactions**

In its default configuration, the Spring Framework's transaction infrastructure code *only* marks a transaction for rollback in the case of runtime, unchecked exceptions; that is, when the thrown exception is an instance or subclass of RuntimeException. (Errors will also - by default - result in a rollback). Checked exceptions that are thrown from a transactional method do *not* result in rollback in the default configuration.

**What do you know about AOP?**

* Aspect: a modularization of a concern that cuts across multiple classes. Transaction management is a good example of a crosscutting concern in enterprise Java applications.
* Join point: a point during the execution of a program, such as the execution of a method or the handling of an exception.
* Advice: action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. (Advice types are discussed below.) Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors around the join point.
* Pointcut: a predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.
* Introduction: declaring additional methods or fields on behalf of a type.

Types of advice:

* *Before advice*
* *After returning advice*
* *After throwing advice*
* *After (finally) advice*
* *Around advice*

**Explain AOP Support in Spring**

* Spring AOP is implemented in pure Java. There is no need for a special compilation process. Spring AOP does not need to control the class loader hierarchy, and is thus suitable for use in a Servlet container or application server.
* In Spring AOP, aspects are implemented using regular classes (the [schema-based approach](http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/aop.html#aop-schema)) or regular classes annotated with the @Aspect annotation (the [@AspectJ style](http://static.springsource.org/spring/docs/3.2.x/spring-framework-reference/html/aop.html#aop-ataspectj)).
* In Spring AOP, a join point always represents a method execution.
* Spring AOP allows you to introduce new interfaces (and a corresponding implementation) to any advised object. For example, you could use an introduction to make a bean implement an IsModified interface, to simplify caching. (An introduction is known as an inter-type declaration in the AspectJ community.)
* Spring AOP defaults to using standard J2SE dynamic proxies for AOP proxies. CGLIB is used by default if a business object does not implement an interface.

**Bean Lifecycle**

1. **Instantiate** – First Spring instantiate the bean.
2. **Populate properties**- Spring Inject the bean’s properties.
3. **Set Bean Name**- Spring set bean name. if the bean implements *BeanNameAware*, spring passes .The  bean’s id to *setBeanName()* method.
4. **Set Bean factory**-If Bean implements *BeanFactoryAware*, spring passes the beanfactory to *setBeanFactory()*.
5. **Pre Initialization**-  It also called postprocess of bean . if there are any bean BeanPostProcessors,  Spring calls *postProcesserBeforeInitialization ()* method.
6. **Initialize beans**-   If the bean implements*IntializingBean*,its *afterPropertySet()*method is called. If the bean has init method declaration, the specified initialization method is  Called.
7. **Post Initialization**-  If there is *BeanPostProcessors*, is implements , spring calls their *postProcessAfterinitalization()* method.
8. **Ready to use**- Now the bean is ready to use by the application.
9. **Destroy**- If the bean implement *DisposableBean* , it will call the *destroy()* method . If custom  *destroy ()* method is defined . the specified method is called.