What bean scopes are supported by Spring and what do they mean? Which is used by default?

The Spring Framework supports following scopes:

* **singleton** (used by default): This means a single instance per Spring container; not thread-safe
* **prototype**: This means any number of object instances.
* **request**: This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
* **session**: This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
* **global-session**: This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

What is dependency injection (DI)? What are the types of DI?

Dependency injection is the concept where you do not create your objects but describe how they should be created, and then expect pre-created objects to be passed in. Likewise, you don’t directly connect your components together but describe which components are needed with either a configuration file or an annotation. The Spring container is responsible for the rest.

DI can be either constructor based or setter based. Constructor based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other classes. Setter based dependency injection is accomplished when the container calls setter methods on a bean after instantiating it.

Describe the Spring bean lifecycle

The lifecycle of a Spring bean consists the following steps:

1. Instantiation
2. Properties population
3. Call of setBeanName() method of BeanNameAware
4. Call of setBeanFactory() method of BeanFactoryAware
5. Call of setApplicationContext() of ApplicationContextAware
6. Pre-initialization with BeanPostProcessor
7. Call of afterPropertiesSet() method of InitializingBean
8. Custom init method
9. Post-initialization with BeanPostProcessor
10. Bean is ready to use
11. Call of destroy() method of DisposableBean
12. Custom destroy method

Numbers 11-12 are actual for all scopes except prototype, since Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, and otherwise assembles a prototype object and hands it to the client with no further record of that prototype instance.

Which steps of beans lifecycle can be overridden or controlled?

In the context of Spring, what is a “stereotype”? What are the existing stereotypes and what is the difference between them?

Stereotype is a class-level annotation denoting the roles of types or methods in the overall architecture (at a conceptual level, rather than implementation). In Spring, these annotations live in the package org.springframework.stereotype.

Currently, this package has the following annotations:

* @Component indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
* @Controller indicates that an annotated class is a “Controller” (e.g. a web controller).
* @Repository indicates that an annotated class is a “Repository”, originally defined by Domain-Driven Design (Evans, 2003) as “a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects”.
* @Service indicates that an annotated class is a “Service”, originally defined by Domain-Driven Design (Evans, 2003) as “an operation offered as an interface that stands alone in the model, with no encapsulated state.” May also indicate that a class is a Business Service Facade (in the Core J2EE patterns sense) or something similar.

These different types primarily allow a developer easily distinguish the purpose of the annotated classes. Starting with Spring 2.5, @Controller, @Repository and @Service serve as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning.

Which dependency injection method is better: Constructor-based or setter-based?

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

1. What is the difference between get and load in Hibernate?

get vs load is one of the most frequently asked Hibernate Interview questions, since the correct understanding of both get() and load() is required to effectively using Hibernate. Main difference between get and load is that, get will hit the database if object is not found in the cache and returned completely initialized object, which may involve several database call while load() method can return proxy if the object is not found in the cache and only hit database if any method other than getId() is called. This can save a lot of performance in some cases. You can also see a difference between get and load in Hibernate for more differences and detailed discussion on this question.

Hibernate interview questions with answers

2. What is the difference between save, persist and saveOrUpdate methods in Hibernate?

After get vs load, this is another Hibernate Interview question which appears quite often. All three methods i.e. save(), saveOrUpdate() and persist() is used to save objects into database, but has subtle differences e.g. save() can only INSERT records but saveOrUpdate() can either INSERT or UPDATE records.

Also, the return type of save() is a Serializable object, while return type of persist() method is void. You can also check save vs persist vs saveOrUpdate for complete differences between them in hibernate.

3. What is named SQL query in Hibernate?

This Hibernate Interview question is related to query functionality provided by Hibernate. Named queries are SQL queries which are defined in mapping document using <sql-query> tag and called using Session.getNamedQuery() method.

Named query allows you to refer a particular query by the name you provided, by the way, you can define named query in hibernate either by using annotations or XML mapping file, as I said above. @NameQuery is used to define a single named query and @NameQueries is used to define multiple named query in hibernate. See Java Persistence with Hibernate for more details.

4. What is SessionFactory in Hibernate? is SessionFactory thread-safe?

Another common Interview question related to the Hibernate framework. SessionFactory, as the name suggests, is a factory to hibernate Session objects. SessionFactory is often built during start-up and used by application code to get the session object. It acts as a single data store and it's also thread-safe so that multiple threads can use the same SessionFactory.

Usually, a Java JEE application has just one SessionFactory, and individual threads, which are servicing client’s requests obtain hibernate Session instances from this factory, that’s why any implementation of SessionFactory interface must be thread-safe.

Also, the internal state of SessionFactory, which contains all metadata about Object/Relational mapping is Immutable and can not be changed once created.

5. What is Session in Hibernate? Can we share a single Session among multiple threads in Hibernate?

This is usually asked as a follow-up question of the previous Hibernate Interview question. After SessionFactory its time for Session. Session represents a small unit of work in Hibernate, they maintain a connection with the database and they are not thread-safe, it means you can not share Hibernate Session between multiple threads. Though Session obtains database connection lazily it's good to close the session as soon as you are done with it.

6. What is the difference between sorted and ordered collection in hibernate?

This is one of the easy Hibernate interview questions you ever face. A sorted collection is sorted in memory by using Java Comparator while an ordered collection uses the database's order by clause for ordering. For large data set it's better to use ordered collection to avoid any OutOfMemoryError in Java, by trying to sort them in memory.

7. What is the difference between a transient, persistent, and detached object in Hibernate?

In Hibernate, Objects can remain in three state transient, persistent, or detached. An object which is associated with the Hibernate session is called a persistent object.

Any change in this object will reflect in the database based on your flush strategy i.e. automatic flush whenever any property of object change or explicit flushing by calling Session.flush() method.

On the other hand, if an object which is earlier associated with Session, but currently not associated with it are called detached object.

Hibernate Interview Questions and Answers

You can reattach detached objects to any other session by calling either update() or saveOrUpdate() method on that session. Transient objects have newly created an instance of persistence class, which is never associated with any Hibernate Session.

Similarly, you can call persist() or save() methods to make a transient object persistent. Just remember, here transient doesn’t represent the transient keyword in Java, which is an altogether different thing.

8. What does Session lock() method do in Hibernate?

This one is one of the tricky Hibernate Interview questions because Session's lock() method reattach objects without synchronizing or updating with the database. So you need to be very careful while using the lock() method. By the way, you can always use the Session's update() method to sync with the database during attachment.

Sometimes this Hibernate question is also asked as what is the difference between Session's lock() and update() method. You can use this key point to answer that question as well. See Java Persistence with Hibernate for more details.

Hibernate Interview Questions for Java JEE Programmers

9. What is Second-level Cache in Hibernate?

This is one of the first interview question related to caching in Hibernate, you can expect a few more. Second-level Cache is maintained at SessionFactory level and can improve performance by saving a few database round trips. Another worth noting point is that second-level cache is available to the whole application rather than any particular session.

10. What is the query cache in Hibernate?

This question Sometimes asked as a follow-up of last Hibernate Interview question, QueryCache actually stores the result of SQL query for future calls. Query cache can be used along with second-level cache for improved performance. Hibernate support various open-source caching solution to implement Query cache e.g. EhCache.

11. Why it's important to provide no-argument constructor in Hibernate Entities?

Every Hibernate Entity class must contain a no-argument constructor, because Hibernate framework creates an instance of them using Reflection API, by calling Class.newInstance() method. This method will throw InstantiationException if it doesn't found any argument constructor inside Entity class.

12. Can we make a Hibernate Entity Class final?

Yes, you can make a Hibernate Entity class final, but that's not a good practice. Since Hibernate uses a proxy pattern for performance improvement in the case of the lazy association, by making an entity final, Hibernate will no longer be able to use a proxy, because Java doesn't allow extension of the final class, thus limiting your performance improvement options.

Though, you can avoid this penalty if your persistent class is an implementation of an interface, which declares all public methods defined in the Entity class.

That's all on this list of Hibernate Interview questions and answer for Java developers. No one can doubt the popularity of Hibernate as an ORM solution and if you are going for a Java J2EE position, you can expect questions from Hibernate. Especially Spring and Hibernate are the two most popular Java framework in JEE space. Don't forget to share any other Hibernate Interview Question, which you have been asked and good enough to share with the Java community.

) Explain what is Java Design Pattern?

A design pattern is a language independent strategy for solving common object-oriented design problem. It describes how to structure classes to meet a given requirement.

2) Explain what is creational design patterns and Factory pattern?

Creational design pattern: This pattern is used to define and describe how objects are created at class instantiation time.

Factory pattern: The factory pattern is used to create an object without exposing the creation logic to the client and refer to a newly created object using a common interface.

3) Which design pattern is used to get a way to access the elements of a collection object in sequential manner?

Iterator pattern is used to get a way to access the elements of a collection object in sequential manner.

4) When service locator pattern is used?

Primis Player Placeholder

When we want to locate various services using JNDI we use service locator pattern.

5) Mention in how many ways can you create singleton pattern?

To create single objects there are two famous ways

Lazy loading

Eager loading

Java Design Patterns Interview Questions

6) Mention which pattern is used when we need to decouple an abstraction from its implementation?

When we want to decouple an abstraction from its implementation in order that two can vary independently we use bridge pattern.

7) Mention which design pattern will be helpful to add new functionality to an existing object?

A decorator pattern allows a user to add new functionality to an existing object without changing its structure.

8) Explain how can you create a Singleton class in Java?

It is two step process,

First make the constructor private so that new operator cannot be used to instantiate the class

Return an object of the object if not null otherwise create the object and return the same via a method.

9) Is it possible to write thread safe singleton in Java?

To write thread safe singleton in Java there are multiple ways for example by using static singleton instance initialized during class loading, by writing singleton using double checked locking. Java Enum is the simplest way to create thread safe singleton.

10) Mention how one should describe a design pattern?

To describe a design pattern, following things need to be taken care of

Pattern name and classification

Problem and solution

Consequences : Variation and language dependent alternatives should also be addressed

Know Uses: Identify the uses in the real systems and its efficiency

11) Mention why access to the non-static variable is not allowed from static method in Java?

You cannot access non-static data from static context because non-static variable are associated with a specific instance of an object while static is not associated with any instance.

12) Mention which pattern is useful when one has to pass data with multiple attributes in one shot from client to server?

Transfer Object Pattern is useful when one has to pass data with multiple attributes in one shot from client to the server.

13) Name some of the entities of DAO pattern?

Some of the entities of DAO include,

Data access object concrete class

Data access object interface

Model object or value object

14) Mention when can you use the Intercepting pattern?

Intercepting pattern is used when you have to do some pre-processing or post processing with request or response of the application.

15) Mention when to use a Factory Pattern?

Factory pattern can be used,

When a class does not know which class of objects needs to create

When class specifies its sub-classes to specify which objects to create

In programming language, you can use factory pattern where you have to create an object of any one of sub-classes depending on the given data

16) Explain in singleton pattern whether it is better to make the whole getinstance() method synchronized or just critical section is enough? Which one is preferable?

Synchronization of whole getinstance() method is costly and is only needed during the initialization on singleton instance, to stop creating another instance of Singleton. Therefore it is better to only synchronize critical section and not the whole method.

17) Mention in how many ways can you write singleton class in Java?

One can write singleton class in Java in four ways

Singleton with public static final field initialized during class loading

Singleton generated by static nested class, also referred as singleton holder pattern

Singleton by synchronizing get instance () method

From Java 5 on-wards using Enums

18) Explain how can you prevent creating another instance of singleton using clone() method?

The preferred way to prevent creating another instance of a singleton is by not implementing Cloneable interface and if you do just throw an exception from clone() method “ not to create a clone of singleton class”.

19) Mention what is the difference between “throw” and “throws”?

Keyword “Throw” is used to explicitly throw as an exception, while “Throws” is utilized to handle checked exceptions for re-intimating the compiler that exceptions are being handled. The throws need to be used in the method’s definition and also while invoking the method that raises checked exceptions.

20) Mention which classes in JDK uses singleton pattern?

Java.lang.Runtime classes uses singleton pattern in JDK.

21) Mention what is the limitation of using singleton pattern?

The singleton pattern ensures that a class has only one instance and to provide a global point of access to it. But at the same time this becomes its limitation as most classes in an application you will need to create multiple instances.

22) Mention what is the difference between VO and JDO?

The difference between JDO and VO is that the JDO is a persistent technology that compete against entity beans in enterprise application development. It enables you to create POJO (plain old java objects) and persist them to the database.

While VO stands for value objects represents an abstract design pattern used in conjuction with entity beans, jdbc and possibly even JDO to overcome commonly found isolation and transactional problems in enterprise apps.