**What is spring ? why it is required ? its modules ?**

* Spring is an open source development framework for Enterprise Java.
* The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform.
* Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

**What is spring container ?**

* The Spring container is at the core of the Spring Framework.
* The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.
* The Spring container uses DI to manage the components that make up an application.
* These objects are called Spring Beans.

**Spring bean scopes , what is by default scope ?**

**1. singleton(default\*)**

Scopes a single bean definition to a single object instance per Spring IoC container.

**2. prototype**

Scopes a single bean definition to any number of object instances.

**3. 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.

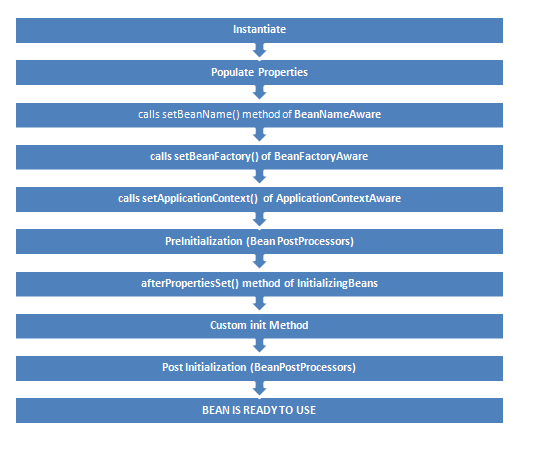
**4. session**

Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.

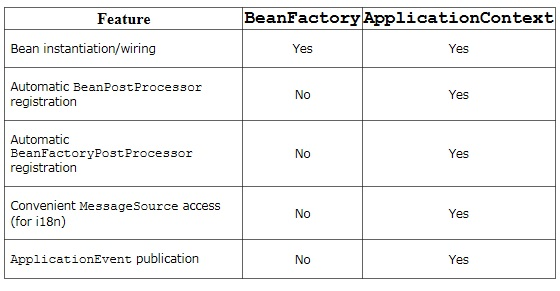
**5. global session**

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.

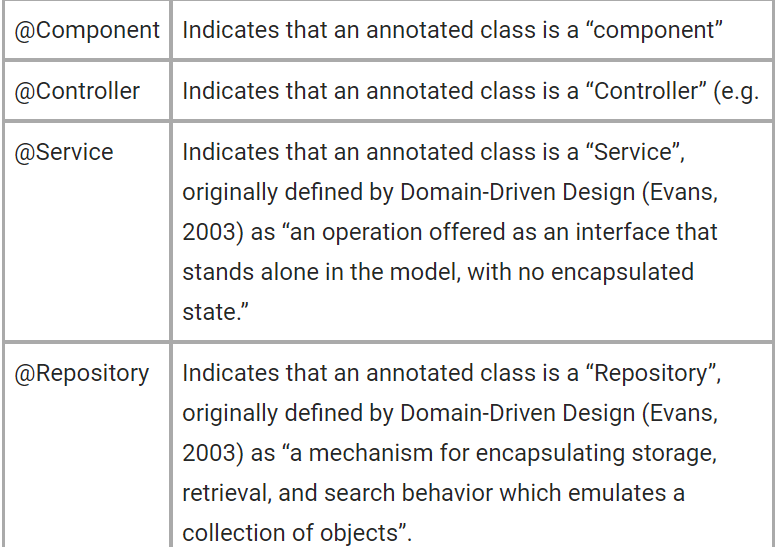
**how to make configuration of spring?**

**Spring bean lifecycle ?**

**Spring bean factory vs application context**

****

**Spring stereotypes**

****

**Spring AOP**

* Modularity :- Spring AOP (Aspect Oriented Programming) compliments OOPs in the sense that it also provides modularity.
* Aspect :- In OOPs, key unit is Objects, but in AOP key unit is aspects or concerns (simply assume stand-alone modules in your application).
* Some aspects have centralized code but other aspects may be scattered or tangled e.g. logging or transactions.
* These scattered aspects are called cross-cutting concern.

**Spring AOP core interfaces - Advice,Aspect,joinpoint & pointcut**

**Aspect**: An aspect is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management. Aspects can be a normal class configured through Spring XML configuration or we can use Spring AspectJ integration to define a class as Aspect using @Aspectannotation.

**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**: Advices are actions taken for a particular join point. In terms of programming, they are methods that gets executed when a certain join point with matching pointcut is reached in the application. You can think of Advices as [Struts2 interceptors](https://www.journaldev.com/2210/struts-2-interceptor-example) or [Servlet Filters](https://www.journaldev.com/1933/java-servlet-filter-example-tutorial).

**Pointcut**: Pointcut are 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 and Spring framework uses the AspectJ pointcut expression language.

**Target Object**: They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.

**AOP proxy**: Spring AOP implementation uses JDK dynamic proxy to create the Proxy classes with target classes and advice invocations, these are called AOP proxy classes. We can also use CGLIB proxy by adding it as the dependency in the Spring AOP project.

**Weaving**: It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime.

**Spring JDBC Template**

**Central Class :-** Spring's *JdbcTemplate* is central class to interact with a database through JDBC.

**Provide Methods :-** 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);**

### **Advantages of Spring JdbcTemplate over JDBC API**

* Spring JdbcTemplate is a powerful mechanism to connect to the database and execute SQL queries.
* It internally uses JDBC API, but eliminate the lot of problems of JDBC API.

Problems While using JDBC API :-

* **Lot of Code:-** We need to write a lot of code before and after executing the query such as creating connection,statement and resultset and closing the resultset, statement and connection etc.
* **Exception Handling:-** We need to perform exception handling code on the database logic.
* **Transaction :-** We need to handle transaction.
* **Repetition of these codes:-** i.e connection, statement and resultset codes for every transaction, so it's time consuming task.

**Advantages of Spring JdbcTemplate :**

* **Simplify DB access :-** Spring provides simplification in handling database access with the Spring JdbcTemplate
* **Clean up resource automatically :-** The Spring JdbcTemplate allows to clean-up the resources automatically, no need to write the extra code  **The Spring JdbcTemplate converts the standard JDBC SqlExceptions into RuntimeExceptions.**
* **specific error messages to the better understandable error messages.**
* **3) The Spring JdbcTemplate offers several ways to query the database e.g queryForList() returns a list of HashMaps. key is the column name of database and value is the actual column data.**
* **4) More convenient is the usage of ResultSetExtractor or RowMapper which allows to translates the SQL result direct into an Object or a list of Objects .**

**Benefit of Dependency Injection**

* **Reduced Dependencies**
* **Reduced Dependency Carrying**
* **More Reusable Code**
* **More Testable Code**
* **More Readable Code**

**Which Injection is better Setter or Constructor?**

* **Setter Injection has upper hand over Constructor Injection in terms of readability.**
* **Once number of dependency crossed a threshold e.g. 5 or 6 its handy manageable to passing dependency via constructor.**
* **Setter Injection is preferred choice when number of dependency to be injected is lot more than normal, if some of those arguments is optional than using** [**Builder design pattern**](http://javarevisited.blogspot.in/2012/06/builder-design-pattern-in-java-example.html) **is also a good option.**
* **Use Setter injection when a number of dependencies are more or you need readability.**
* **Use Constructor Injection when Object must be created with all of its dependency.**

**Hibernate**

* **ORM :-** Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database. Tables.
* **Framework to map object with table :-** Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa.

**how to configure hibernate?**

**Hibernate Benefits**

* **Remove Boilerplate Code**
* **Opensource**
* **Object Oriented HQl**
* **Easy to Integrate with other J2EE application**
* **Lazy Initialization**
* **Better Performance through Caching**
* **Can execute native query**
* **DB Independent**
* **Named Query**

### 

### **Interfaces of Hibernate framework?**

* + **SessionFactory (org.hibernate.SessionFactory):**
    - SessionFactory is an [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) thread-safe cache of compiled mappings for a single database.
    - We need to initialize SessionFactory once and then we can cache and reuse it.
    - SessionFactory instance is used to get the Session objects for database operations.
  + **Session (org.hibernate.Session):**
    - Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store.
    - It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction.
    - We should open session only when it’s required and close it as soon as we are done using it.
    - Session object is the interface between java application code and hibernate framework and provide methods for CRUD operations.
  + **Transaction (org.hibernate.Transaction):**
    - Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work.
    - It abstracts the application from the underlying JDBC or JTA transaction.
    - A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases.

# [**Is Hibernate's session thread safe?**](https://stackoverflow.com/questions/3777794/is-hibernates-session-thread-safe)

Session is a lightweight and a non-thread safe object (No, you cannot share it between threads)

**Is Hibernate Sessionfactory threadsafe?**

SessionFactory is Hibernates concept of a single datastore and is thread safe

**When to create sessionfactory?**

A SessionFactory is usually only built once at startup.

**States of entity hibernat**



## **1.Transient State:**

A New instance of a persistent class which is not associated with a ***Session***, has no representation in the ***database*** and no identifier value is considered ***transient*** by Hibernate:

UserDetail user = new UserDetail();  
user.setUserName("Dinesh Rajput");  
//user is in a transient state

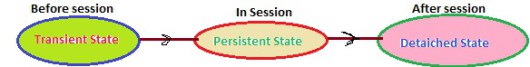
## **2. Persistent State:**

A persistent instance has a representation in the ***database*** , an identifier value and is associated with a ***Session***.

You can make a transient instance persistent by associating it with a ***Session***:

Long id = (Long) session.save(user);  
// user is now in a persistent state

In couple of previous chapters we have seen that every entity object are passed to three states of the object before saving and updating the row in the database table as per as given in the following picture.



## **1. Transient State:**

A New instance of a persistent class which is not associated with a ***Session***, has no representation in the ***database*** and no identifier value is considered ***transient*** by Hibernate:

UserDetail user = new UserDetail();  
user.setUserName("Dinesh Rajput");  
// user is in a transient state

## **2. Persistent State:**

A persistent instance has a representation in the ***database*** , an identifier value and is associated with a ***Session***. You can make a transient instance persistent by associating it with a ***Session***:

Long id = (Long) session.save(user);  
// user is now in a persistent state

## **3. Detached State:**

Now, if we close the ***Hibernate Session***, the ***persistent*** instance will become a ***detached*** instance: it isn’t attached to a ***Session*** anymore (but can still be modified and reattached to a new Session later though).

The first-level cache:

-The first level cache type is the session cache.

-The session cache caches object within the current session but this is not enough for long level i.e. session factory scope.

The second-level cache:

-The second-level cache is called ‘second-level’ because there is already a cache operating for you in Hibernate for the duration

-you have a session open. A Hibernate Session is a transaction-level cache of persistent data.

-It is possible to configure a SessionFactory-level cache on a class-by-class and collection-by-collection basis.

Second-level cache

— Across sessions in an Application

— Across applications (different applications on same servers with same database)

— Across clusters (different applications on different servers with same database)

save,saveorupdate ,update ,persist ,merge - when to use which one? where transaction is required and not?

### ***Persist***

* The persist method is intended for adding a new entity instance to the persistence context, i.e. transitioning an instance from transient to persistent state.
* We usually call it when we want to add a record to the database (persist an entity instance):

Person person = new Person();

person.setName("John");

session.persist(person)

What happens after the *persist* method is called?

* The *person* object has transitioned from *transient* to *persistent* state.
* The object is in the persistence context now, but not yet saved to the database
* The generation of *INSERT* statements will occur only upon committing the transaction, flushing or closing the session.
* the *persist* method has *void* return type

**Save**

* The method is guaranteed to return the *Serializable* value of this identifier.

### ***Merge***

**The main intention of the *merge* method is to update a *persistent* entity instance with new field values from a *detached* entity instance.**

* **what happen if we use save twice ?**

# **Hibernate Error: org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session**

**how to convert entity object from detached to persist state?**

**Use update() if you are certain that the session does not contain an already persistent instance with the same identifier. Use merge() if you want to merge your modifications at any time without consideration of the state of the session.**

**what is @transient annotation ?**

* **Java's transient keyword is used to denote that a field is not to be serialized,**
* **JPA's @Transient annotation is used to indicate that a field is not to be persisted in the database, i.e. their semantics are different.**

**what is multiple classs per table**

***@MappedSuperclass***

***INHERITANCE***

***@Embedded***

**Java**

**how to create unmodified set from existing collection?**

*List readOnlyList = Collections.unmodifiableList(tempList);*

*Set readOnlySet = Collections.unmodifiableSet(readOnlySet);*

*Map readOnlyMap = Collections.unmodifiableMap(tempMap);*

**how to create unmodified set from existing collection?**

*List<String> syncList = Collections.synchronizedList(new ArrayList<String>());*

*Set<String> syncSet = Collections.synchronizedSet(new HashSet<String>());*

*Map<String, String> map = Collections.synchronizedMap(new HashMap<String, String>());*

*// when iterating over a synchronized list, we need to synchronize access to the synchronized list*

*synchronized (syncList) {*

*Iterator<String> iterator = syncList.iterator();*

*while (iterator.hasNext()) {*

*System.out.println("item: " + iterator.next());*

*}*

*}*

**Singleton example**

*public class StaticBlockSingleton {*

*private static StaticBlockSingleton instance;*

*private StaticBlockSingleton() {*

*}*

*static {*

*try { instance = new StaticBlockSingleton();*

*} catch (Exception e) {*

*}*

*}*

*public static StaticBlockSingleton getInstance() {*

*return instance;*

*}*

*}*

**List vs Set**

**Duplicate Value :- List allow Duplicate value.**

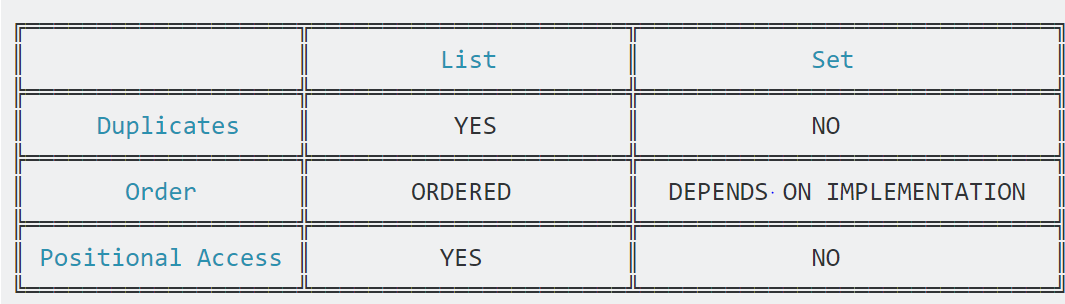
**Set do not allow duplicate value.**

**Order :- List maintain insertion order**

**Set do not make order**

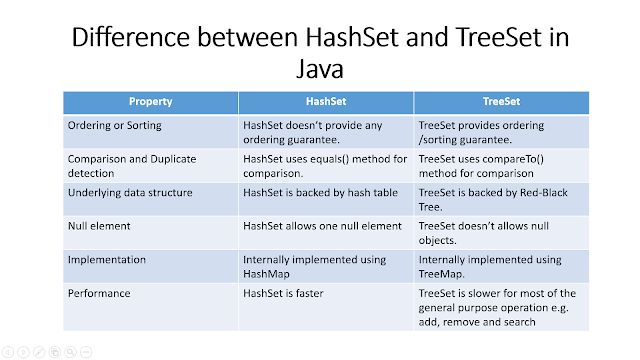
**Null Value :- Allow multiple null**

**Allow only one null.**

****

**HashSet Vs TreeSet Vs LinkedHashSet**

* ***HashSet*** *is Implemented using a hash table.*
* *Elements are not ordered.*
* *The add, remove, and contains methods have constant time complexity O(1).*
* ***TreeSet*** *is implemented using a tree structure(red-black tree in algorithm book).*
* *The elements in a set are sorted, but the add, remove, and contains methods has time complexity of O(log (n)).*
* *It offers several methods to deal with the ordered set like first(), last(), headSet(), tailSet(), etc.*
* ***LinkedHashSet*** *is between HashSet and TreeSet.*
* *It is implemented as a hash table with a linked list running through it, so it provides the order of insertion.*
* *The time complexity of basic methods is O(1).*

****

**ArrayList vs Vector**

**Thread Safety**

* ArrayList class is not thread safety where as Vector class is thread safety. Vector class is a synchronized class.
* Only one thread can enter into Vector object at any moment of time during execution.
* Where as ArrayList class is not synchronized.
* Multiple threads can access ArrayList object simultaneously

## **Performance**

* Arraylist is fast because not thread safe so fast Vector slow.

**Capacity Increment**

* **Increase half in arraylist**
* **Can pass increase capacity in vector double**

**setSize()**

* **No such method list**
* **Method in arraylist**

**Traverse**

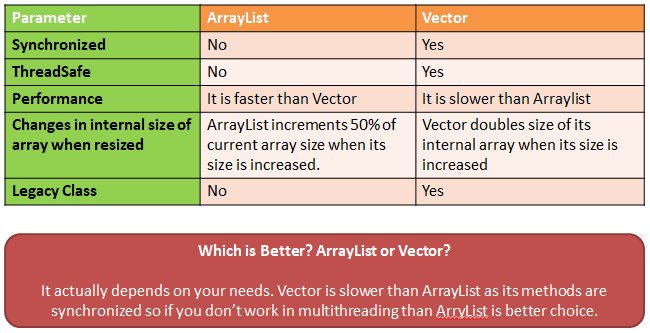
* **Iterator**
* **Enumeration**

**Search**

* ArrayList slow
* Vector fast

## **Legacy Code**

* Vector legacy code added later



**Why we should not use Vector?**

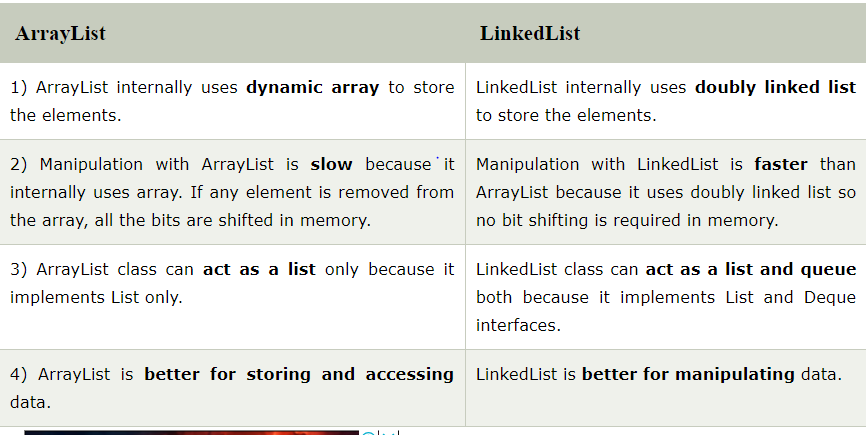
* All methods of Vector class are synchronized. This makes each and every operation on Vector object thread safe.

## Thread Safeness of Vector class is time consuming.

*Instead use*

*List<String> syncList = Collections.synchronizedList(new ArrayList<String>());*

**ArrayList Vs LinkedList**

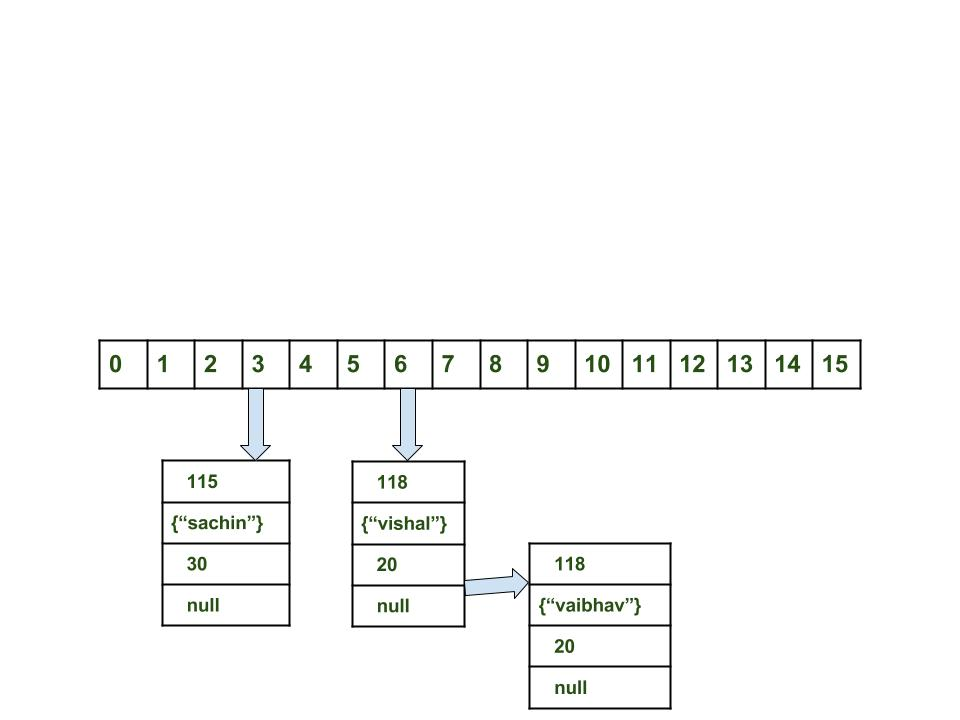
****

**Cheatshit**

****

**how linkedlist & arraylist internal work ?**

**how hashmap internal works ?**

****

**difference between iterator,enumeration and listiterator.its usecases**

**what is nested/inner class and its use case ?**

**ways of creating thead ? which one to use when?**

**There are actuall 3 ways to creating threads:**

**1->Extending thread class**

**ex:public class th extends Thread{**

**public static void main (String[] args){**

**th t=new t();**

**t.start();**

**}**

**}**

**2->By implementing Runnable Interface**

**ex:**

**public class th implements Runnable{**

**public void run ()**

**{**

**}**

**public static void main()**

**{**

**Thread t=new Thread();**

**th tx=new th(t);**

**}**

**}**

**3->Is by creating a unnamed class n implementing the interdace there itself**

**public class abcd{**

**public static void main()**

**{**

**Thread t =new Thread();**

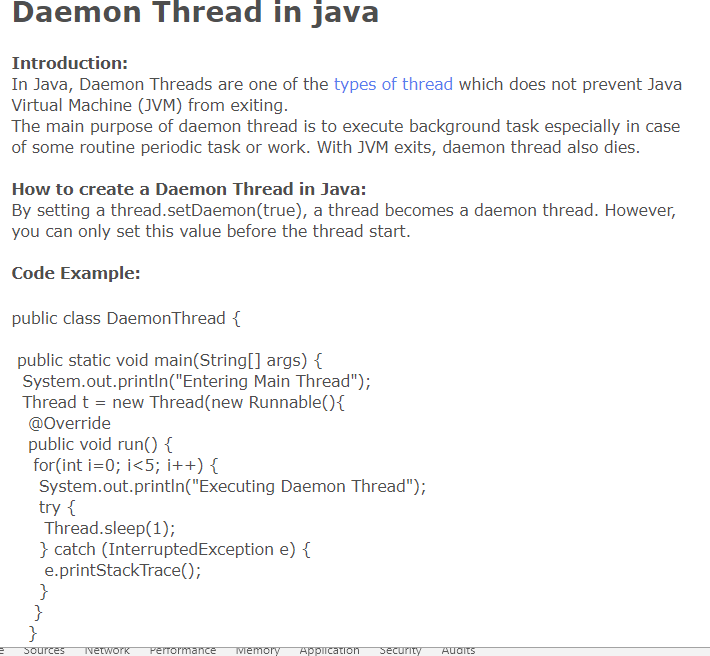
**Runnable x =new Runnable(t){**

**public void run(){}**

**};**

**I would prefer using interfaces when all you need is a run method.Since java does not support multiple inheritance through classes so you will not be able to extend other classes if u have already extended Thread class n besides why inherit all the methods in cases when you just need a run method.**

**what is daemon thread?**

****

**what is wait,notify and notify all ?**

### **wait**

**Object wait methods has three variance, one which waits indefinitely for any other thread to call notify or notifyAll method on the object to wake up the current thread. Other two variances puts the current thread in wait for specific amount of time before they wake up.**

### **notify**

**notify method wakes up only one thread waiting on the object and that thread starts execution. So if there are multiple threads waiting for an object, this method will wake up only one of them. The choice of the thread to wake depends on the OS implementation of thread management.**

### **notifyAll**

**notifyAll method wakes up all the threads waiting on the object, although which one will process first depends on the OS implementation.**

**what is synchronized ? object level/class level synchronized ?**

**what is static block ? when it gets called ?**

**static block (also called static clause) which can be used for static initializations of a class. This code inside static block is executed only once: the first time you make an object of that class or the first time you access a static member of that class (even if you never make an object of that class).**

**what is init block ? when it gets called ?**

**Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.**

**class Bike7{**

**int speed;**

**Bike7(){System.out.println("speed is "+speed);}**

**{speed=100;}**

**what is clone interface ? is it marker interface ? what is its usecases? what is deep and shallow cloning ?**

**what is compile time and dynamic polymorphism ?**

**what is dynamic method dispatch ??**

**what is super class of all class ? what methods it contains ?**

**what is generic ? what are its usecases ?what is templating with generic ?**

* Generics allow you to customize a "generic" method or class to whatever type you're working with
* public T Add<T>(T a, T b)

**what is final ? and where can we apply it ?its use cases ?**

final expresses intent. It tells the user of a class, method or variable "This element is not supposed to change, and if you want to c

hange it, you haven't understood the existing design."

This is important because program architecture would be really, really hard if you had to anticipate that *every class and every method* you ever write might be changed to do something completely different by a subclass.

It is much better to decide up-front which elements are supposed to be changeable and which aren't, and to enforce the unchangeablility via final.You could also do this via comments and architecture documents, but it is always better to let the compiler enforce things that it can than to hope that future users will read and obey the documentation.

**how to create immutable class?**

**Steps to create immutable class**

* Make your class *final*
* *Make all your fields final,*
* *Don’t expose setter methods.*
* *When exposing methods which modify the state of the class, you must always return a new instance of the class.*
* *If the class holds a mutable object:*
* *Inside the constructor, make sure to use a clone copy of the passed argument and never set your mutable field to the real instance passed through constructor. this is to prevent the clients who pass the object from modifying it afterwards.*
* *Make sure to always return a clone copy of the field and never return the real object instance.*

package com.programmer.gate.beans;

public final class ImmutableStudent {

private final int id;

private final String name;

public ImmutableStudent(int id, String name) {

this.name = name;

this.id = id;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

}

//Mainly Used for the caching and thread safe

public class Age {

private int day;

private int month;

private int year;

public int getDay() {

return day;

}

public void setDay(int day) {

this.day = day;

}

public int getMonth() {

return month;

}

public void setMonth(int month) {

this.month = month;

}

public int getYear() {

return year;

}

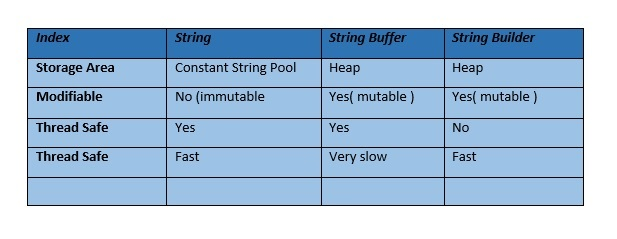
public void setYear(int year) {

this.year = year;

}

}

**what is string ,string builder and string buffer ?**

****

**Difference between comparable and comparator .Use cases of it.**

* Comparable interface is used for single sequence sorting i.e.sorting the objects based on single data member.
* comparator interface is used to sort the object based on multiple data members.

**Can we create private constructor ? what its use cases?**

* The use of private constructor is to serve singleton classes.
* A singleton class is one which limits the number of objects creation to one.
* Using private constructor we can ensure that no more than one object can be created at a time.
* By providing a private constructor you prevent class instances from being created in any place other than this very class.

**Can we create constructor in abstract class?**

* Yes, an abstract class can have a constructor.
* Since an abstract class can have variables of all access modifiers, they have to be initialized to default values, so constructor is necessary.

**what is abstract class ? when is itsreal life example?**

* An abstract class is a class that you cannot create an instance of.
* It can provide basic functionality, but in order for that functionality to be used, one or more other classes must derive from the abstract class.
* One of the major benefits of abstract classes is that you can reuse code without having to retype it.
* That has a plethora of benefits, such as reducing bugs and making coding faster.
* A concrete example of an abstract class would be a class called Animal.

**what is interfece? when to use it? can it contain instance variable ?**

* An interface in java is a blueprint of a class.
* It has static constants and abstract methods.
* The interface in java is a mechanism to achieve abstraction.
* There can be only abstract methods in the java interface not method body.
* It is used to achieve abstraction and multiple inheritance in Java.
* We can declare variables in Java interfaces. By default, these are public, final and static.
* variables declared in interface are by default public, static and final. Since it is static you cannot call it instance variable.

**can i crate constructor inside it?**

* No interface cant have construtor
* Abstract class can have constructor

**what is marker inteface ? example . can i create custom marker interface?**

* **Marker Interfaces in Java have special significance because of the fact that they have no methods declared in them which means that the classes implementing these interfaces don't have to override any of the methods.**
* **A few of the marker interfaces already exist in the JDK like Serializable and Cloneable.**
* **One can also create their own custom interfaces which doesn't have any method.**
* **The purpose of these interfaces is to force some kind of functionality in the classes by providing some functionality to a class if it implements the marker interface.**
* **A common question asked very frequently is about Runnable interface being marker or not.**
* **Runnable interface is not marker because Runnable interface has the public void run() method declared inside it.**
* **A very good example of marker interface is Serializable where the class implements can be used with ObjectOutputStream and ObjectInputStream classes.**
* **The Java language specification doesn't itself define the term marker interface and the term has been coined by authors, developers and designers.**
* **One common question asked is if we can create a marker interface or not and the answer is yes because of following reason:**
* **We can't create marker interface similar to Serializable or Cloneable but we can simulate the functionality by writing extra code around the custom marker interface.**

**what is serizable ? when to use it ? what is serialversionid?**

**what is the use of transient variable inside serializable class and its use cases?**

**The serialization of objects in Java allows you to make a byte sequence from any object that has implemented the Serializable interface; it also allows you to turn that byte sequence back into an object. The mechanism does not depend on the operating system, which means you can transfer objects via your network and restore them at the other side of the wire.**

**what is externilizable ? when it's required ?**

As name suggest it is externalilizing your serialization.

If you want to customize your serialization mechanism then you can use it.

It uses custom written mechanism to perform marshalling and unmarshalling of objects.

Externalizable interface extends Serializable interface.

Serializable serialize everything upto OBJect class.

**what is criteria ? and its benefits**

**4. Criteria**

* Hibernate Criteria query is only used to fetch the results from the database using object oriented approach.

1. Hibernate Criteria API provides Projection that we can use for aggregate functions such as sum(), min(), max() etc.
2. Hibernate Criteria API can be used with ProjectionList to fetch selected columns only.
3. Criteria in Hibernate can be used for join queries by joining multiple tables, useful methods for Hibernate criteria join are createAlias(), setFetchMode() and setProjection()

**what is HQL ? and why its required?**

Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).

1. You don't need to learn SQL
2. Database independent
3. Simple to write query

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

**what is named query? and its benefits**

**Advantages**

* **compiled and validated at app start-up time**
* **easier to maintain than string literals embedded in your code**
* **HQL and native SQL queries can be used and replaced without code changes (no need to re-compile your code)**

**Disadvantages**

* **static**
* **result-set mapping with native SQL queries sometimes cumbersome**

**how to do pagination in hibernate3?**

**he simplest and most common way to do pagination in Hibernate is using HQL:**

**Session session = sessionFactory.openSession();**

**Query query = sess.createQuery("From Foo");**

**query.setFirstResult(0);**

**query.setMaxResults(10);**

**List<Foo> fooList = fooList = query.list();**

**what is hibernate dialect?**

**To connect to any database with hibernate, we need to specify the SQL dialect class in hibernate.cfg.xml**

Dialect class is java class, which contains code to map between java language data type database data type.

All Dialect classes extend the Dialect abstract class.

Dialect is used to convert HQL statements to data base specific statements.

**difference between load and get? which one to use when?**

## Different between session.get() and session.load()

Actually, both functions are use to retrieve an object with different mechanism, of course.

## 1. session.load()

* It will always return a “proxy” (Hibernate term) without hitting the database. In Hibernate, proxy is an object with the given identifier value, its properties are not initialized yet, it just look like a temporary fake object.
* If no row found , it will throws an ObjectNotFoundException.

## 2. session.get()

* It always hit the database and return the real object, an object that represent the database row, not proxy.
* If no row found , it return null.

## It’s about performance

Hibernate create anything for some reasons, when you do the association, it’s normal to obtain retrieve an object (persistent instance) from database and assign it as a reference to another object, just to maintain the relationship. Let’s go through some examples to understand in what situation you should use session.load().

## 1. session.get()

For example, in a Stock application , Stock and StockTransactions should have a “one-to-many” relationship, when you want to save a stock transaction, it’s common to declared something like below