1. **A thread live cycle** ?

New, Runnable, Running, Wait, Dead

1. **Maven live cycle ?**

Validate, Compile, Test, Package, Integration Test, Verify, Install, Déploy.

1. **An applet live cycle** **?**

Init, Start, Paint Stop, Destroy.

1. **Can we override protected method as private?**

No. While overriding, visibility of a method can be increased but can not be reduced.

1. **In what order statements are catched:**

. Exception's subclasses have to be caught first. e.g the FileNoFoundException will be catched before the IOException.

1. **Difference between Map, Set, List, HashMap, TreeMap, Hashtable and LinkedHashMap**

**Map** store maps of key - value. Does not have duplicate keys but duplicate values are allowed.

**Set** stores objects (not key – value). Does not alow duplicates. It is an unordered collection

**List** provides ordered and indexed collection which may contain duplicates. It maintains insertion order of elements

**HashMap** is not synchronized, and there is no ordering on keys or values. Null values are allowed. Only one null key are accepted.

**LinkedHashMap** is like HashMap and preserves the insertion order.

**Hashtable** is implemented as hashmap but is synchronized. Does not accept null value or null key.

**TreeMap** is implemented based on red-black tree structure, and it is ordered by the key. it allows to store comparable objects in a tree in an orderly way.

1. **Difference between ‘volatile’ and ‘transient’ keyword in Java**

**Transient** keyword is used along with instance variables to exclude them from serialization process. If a field is transient its value will not be persisted.

**Volatile** keyword can be used in variables to indicate compiler and JVM that always read its value from main memory. Volatile variable will have only one main copy which will be updated by different threads and updating by one thread to the volatile variable will immediately reflect to the other Thread.

Transient keyword can not be used along with static keyword but volatile can be used along with static

1. **Difference between ‘volatile’ and ‘static’ keyword in multi-threads context?**

**Static Variable:** If two Threads (suppose Thread 1 and Thread 2) are accessing the same object and updating a variable which is declared as static then it means Thread 1 and Thread 2 can make their own local copy of the same object (including static variables) in their respective cache, so updating by Thread 1 to the static variable in its local cache wont reflect in the static variable for Thread 2 cache. Static variables are used in the Object Context where updating by one object would reflect in all the other objects of the same class but not in the Thread context where updating of one thread to the static variable will reflect the changes to all the threads **(in their local cache).**

**Volatile variable:** If two Threads(suppose Thread 1 and Thread 2) are accessing the same object and updating a variable which is declared as static then it means Thread 1 and Thread 2 can make their own local cache of the Object **except** the variable which is declared as a **volatile** . So the volatile variable will have only one main copy which will be updated by different threads and updating by one thread to the volatile variable will immediately reflect to the other Thread. So the volatile variable is used in the Thread context.

1. **The difference between hashcode and equals and their relation?**

**Hashcode** (prints the reference number) is a referencing equality while **equals** is a logical equality. If two objects are equal then they should have the same hashcode and if two objects are not equal then they may or may not have same hash code.

You need to override equals() and hashcode() methods of a class whose objects you want to use as Key in a hashmap. This is required because hashmap uses these 2 methods to retrieve the stored values.

1. **Can I call Finalize() much time?**

you can call finalize() as many times as you wish. But the garbage collector will call ( but not guaranteed) it once only.

1. **How would you decide between different collection classes for storing a keyed data set?**

I will choose according to the following criteria:

Are the data ordered? Data related? Data sorted? Are keys that are null? Are values that are null? is a multi-threads application?

1. **Is possible to insert Null values in treeset and hashset ?**

For an empty **TreeSet**, a first element null value can be inserted but after inserting that null value if we are trying to insert any other objects we will get NullPointerException.

For a non empty **TreeSet** if we are trying to inserte null value, at run time you will get nullPointerException. This is because when some elements are already in the tree, before inserting any object it compares the new object to the existing ones by the compareTo() method and decides where to put the new object. So when inserting null the compareTo() method internally throws NullPointerException.

For **HashSet** only one null value insertion is possible. When we try to add more null values the add method returns false and null is not added**.**

1. **What is immutable object and his benefits?**

Immutable objects have no fields that can be changed after the object is created. To made and object immutable : The class must be declared **final,** must not have setters, all fields must be initialized at the beginning and declared **private** and **final**.

## *Benefits of Immutable Classes in Java :*

* Immutable objects are by default [thread safe](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html), can be shared without synchronization in concurrent environment.
* Immutable object simplifies development, because its easier to share between multiple threads without external synchronization.
* Immutable object boost performance of Java application by reducing [synchronization](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) in code.
* They are excellent keys for Map and Set.
* Their values can be cached by the client without risk of desynchronization.
* The immutable classes are adapted to the representation of abstract data types.
* Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers.  You can use [static factory methods](http://javarevisited.blogspot.it/2011/12/factory-design-pattern-java-example.html) to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

1. **How to get Immutable Object with ArrayList member variable ?**
   * return a copy of the ArrayList

public List<String> get() { return **new** ArrayList<String>(this.list); }

* + Or you can return a unmodifiable list:

public List<String> get() { return Collections.unmodifiableList(this.list); }

# [**Why is char[] preferred over String for passwords?**](https://stackoverflow.com/questions/8881291/why-is-char-preferred-over-string-for-passwords)

**Strings are immutable**. That means once you've created the String, if another process can dump memory, there's no way (aside from reflection) you can get rid of the data before garbage collection kicks in.

With **an array of chars**, you can explicitly wipe the data after you're done with it. You can overwrite the array with anything you like, and the password won't be present anywhere in the system, even before garbage collection.

1. **Differents access modifiers ?**
   * **public :** ‘Any member declared as ‘public’ accessible in any of the classes. Any class declared as public can have its public members accessed anywhere inside Java.
   * **prorected:** Any member declared as ‘**protected‘** can be accessed within any class in the same package and also within any subclass of the class where the member is declared. The subclass can be inside another package as well.
   * **Private : Any member declared as ‘private‘ can only be accessed within the class where it is declared.**
   * **default: The class or its members that do have any of the three access specifiers mentioned in its declaration falls under the default access modifier or the package access specifier. These members are accessible within any class present under the same package.**

Below Table summarizes the access modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **class** | **constructor** | **method** | **Data/variables** |
| public | Yes | Yes | Yes | Yes |
| protected |  | Yes | Yes | Yes |
| default | Yes | Yes | Yes | Yes |
| private |  | Yes | Yes | Yes |
| static |  |  | Yes |  |
| final | Yes |  | Yes |  |

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

**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 object is not found in cache and only hit database if any method other than getId()is called. This can save a lot of performance in some cases.

1. **What will happen if two different HashMap  key objects have the same hashcode?**

They will be stored in the same bucket but  another node will be created in this bucket to store the second object. And keys equals () method will be used to identify correct key value pair in HashMap.

1. **How does HashSet is implemented in Java, How does it use Hashing?**

In reality, HashSet is implemented like hashmap. It is a map where values added are considered as Keys and for each key a boolean value (PRESENT) is associated with this key.

Set.add(e) means :

private static final Object PRESENT = new Object();

public boolean add(E e) {

**return** map.put(e, **PRESENT**);

}

1. **How to find middle element of linked list in one pass?**

you need to maintain two-pointer, one increment at each node while other increments after two nodes at a time, by having this arrangement, when first pointer reaches to the end, second pointer will point to middle element of linked list.

1. **How to find 3rd element from end in a linked list in one pass ?**

you need to maintaining two pointers and start incrementing the second pointer when first pointer has moved up to 3rd element. Than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

1. **How do you find duplicates in an array if there is more than one duplicate?**

One way of solving this problem is using a Hashtable or HashMap data structure. You can traverse through array, and store each number as key and number of occurrence as value. At the end of traversal you can find all duplicate numbers, for which occurrence is more than one.

A way to delete duplicates is to passe these elements to a set.

In java 8, you can use Stream.distinct().

1. **Can we make array volatile in Java?**

Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. That means, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee,

1. **What are practical uses of volatile modifier?**

One of the practical use of the volatile variable is to make reading **double** and **long** atomic. Another use of the volatile variable is to provide a memory barrier.

Volatile variables provide the guarantee about ordering and visibility.

1. **What is false sharing in the context of multi-threading?**

False sharing occurs when threads on different processor modify variables that reside on same cache line.

1. **What is busy spin? Why should you use it?** Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core.
2. **Can we create an Immutable object, which contains a mutable object?**

Yes, you just need to be a little bit careful not to share the reference of the mutable component, instead you should return a copy of it if you have to.

1. **Difference between WeakReference and SoftReference in Java ?**

**WeakReference** becomes eligible for garbage collection as soon as last strong reference is lost but **SoftReference** even thought it can not prevent GC, it can delay it until JVM absolutely need memory.

1. **What is Java Heap space?** When a Java process is started using java command, memory is allocated to it. Part of this memory is used to create heap space, which is used to allocate memory to objects whenever they are created in the program.
2. **What is the difference between stack and heap in Java?**

Stack and heap are different memory areas in the JVM and they are used for different purposes. The **stack** is used to hold method frames and local variables while objects are always allocated memory from the **heap**. The stack is usually much smaller than heap memory and also didn't shared between multiple threads, but heap is shared among all threads in JVM.

1. **What is Composition in Java?**

Creating an object of other class in our class and calling other class variables and methods is known as composition. The rule says, in a single source code, if a number of classes exist, only one class can be public. The file name must be of that class which is public**.**

1. **The difference between Inheritance and Composition?**

Though both allows code reuse, Composition is more flexible than Inheritance because it allows you to switch to another implementation at run-time. Code written using Composition is also easier to test than code involving inheritance hierarchies.

1. **Difference between Composition and Aggregation ?**

The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.

1. **What is difference between dependency injection and factory design pattern?**

Use of dependency injection results in cleaner code than factory pattern. By using dependency injection, your classes are nothing but POJO which only knows about dependency but doesn't care how they are acquired. In the case of factory pattern, the class also needs to know about factory to acquire dependency. hence, DI results in more testable classes than factory pattern.

1. **Adapter pattern ?**

It is used for interface conversion To bridge the gap between two interfaces. If your client is using some interface but you have something else, you can write an Adapter to bridge them together.

1. **Decorator pattern ?**

It is used to add new functionality into the class without the modifying existing code.

1. **Proxy pattern ?**

It is used to add an extra level of indirection to support distribute, controlled or intelligent access.

1. **Template pattern ?**

It provides an outline of an algorithm and lets you configure or customize its steps. For examples, it defines steps for sorting but let you configure how to compare them using Comparable or something similar.

1. **Visitor pattern ?**

It is a solution of problem where you need to add operation on a class hierarchy but without touching them. This pattern uses double dispatch to add another level of indirection.

1. **Composite pattern ?**

It arranges objects into tree structures to represent part-whole hierarchies. It allows clients treat individual objects and container of objects uniformly. Use Composite pattern when you want to represent part-whole hierarchies of objects.

1. **Abstract factory pattern ?**

It provide a way / interface to encapsulate a group / families of related or dependent objects without specifying their concrete classes.

1. **Prototype pattern ?**

It is a creational design pattern. It is used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects. If the creation of an instance is complex or time consuming. Rather than creating multiple instances of the class, copy the first instance and modify the copy appropriately.

1. **Open closed design principle ?**

It asserts that your code should be open for extension but closed for modification. .Which means if you want to add new functionality, you can add it easily using the new code but without touching already tried and tested code. e.g Strategy pattern.

1. **Flyweight pattern ?**

It allows you to share object to support large numbers without actually creating too many objects. In order to use Flyweight pattern, you need to make your object Immutable so that they can be safely shared.

1. **Difference between Serializable and Externalizable ?**

Serializable interface is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it leverages default serialization built-in JVM, which is expensive, fragile and not secure. Externalizable allows you to fully control the Serialization process, specify a custom binary format and add more security measure.

1. **Difference between DOM and SAX parser in Java?**

DOM parser loads the whole XML into memory to create a tree based DOM model which helps it quickly locate nodes and make a change in the structure of XML while SAX parser is an event based parser and doesn't load the whole XML into memory. Due to this reason DOM is faster than SAX but require more memory and not suitable to parse large XML files.

1. **Circular dependency in spring ?**

It happens when a bean A depends on another bean B, and the bean B depends on the bean A as well:

Of course, we could have more beans implied. It can happen in Spring when using constructor injection.

Ways to deal with this problem:

Redesign, use @Lazy, use Setter/Field Injection, use @PostConstruct, implement ApplicationContextAware and InitializingBean.

1. **What's the difference between the methods sleep() and wait():**

Sleep(1000); puts thread aside for exactly one second. Wait(1000), causes a wait of up to one second

The major difference is sleep() is meant for specfied time pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock.

Wait() is used for inter-thread communication while sleep() is used to introduce pause on execution, generally.

• **wait():** until call notify(), notifyAll() from object

• **sleep():** until at least time expire or call interrupt().

1. **The difference between overloading and overridding (redefinition):**

• The **overloading** of a method or constructor is static, so done at compile time. It allows to define several times the same method / constructor with different arguments. The compiler chooses the method that should be called based on the number and type of arguments.

• The **overriding** of an inherited method is dynamic, so done in runtime. It must imperatively keep the declaration of the parent method (type and number of parameters and the return value).

The return value must be compatible (from the same class or a sub-class) and the propagated exceptions must be identical..

1. **What is encapsulation and why is it important in Object Oriented programming?**

Encapsulation is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. that means binding object state(fields) and behaviour(methods) together

**importances:**

* It improves maintainability and flexibility and re-usability.
* The fields of a class can be made read-only or write-only
* A class can have total control over what is stored in its fields.
* The users of a class do not know how the class stores its data.