**Real time java interview questions:**

<http://howtodoinjava.com/java-interview-questions/>

<http://howtodoinjava.com/2014/04/18/real-java-interview-questions-asked-for-oracle-enterprise-manager-project/>

 ------------------------------------------   Core Java ----------------------------------------------------

Java program simple to and important:

1.      To reverse a string.

2.       To reverse a number.

3.       To find Fibonacci series.

4.       To print prime number.

5.       Decimal to binary conversion.

6.       To find the occurrence of elements in a Sting

7.       To find the occurrence of a character in string by HashMap.

8. Write a program to check if two String are Anagram in Java - Program Example

**Q: Abstraction?**

[Abstraction](http://www.javacodegeeks.com/2014/04/why-abstraction-is-really-important.html) is the process of separating ideas from specific instances and thus, develop classes in terms of their own functionality, instead of their implementation details. Java supports the creation and existence of abstract classes that expose interfaces, without including the actual implementation of all methods. The abstraction technique aims to separate the implementation details of a class from its behavior.

**Q) What is Association?**

Ans. Association is a relationship between two classes. In this relationship the object of one instance perform an action on behalf of the other class. The typical behavior can be invoking the method of other class and using the member of the other class.

 Public class MyMainClass{

public void init() {

new OtherClass.init();

   }

}

**Q) What is Aggregation?**

 Ans) Aggregation has a relationship between two classes. In this relationship the object of one class is a member of the other class. Aggregation always insists for a direction.

public class MyMainClass {

OtherClass otherClassObj = new OtherClass();

}

**Q) What is Composition?**

Ans. Composition is a special type of aggregation relationship with a difference that its the compulsion for the OtherClass object (in previous example) to exist for the existence of MyMainClass.

**Q: Differences between Abstraction and Encapsulation?**

Abstraction and encapsulation are complementary concepts. On the one hand, abstraction focuses on the behavior of an object. On the other hand, encapsulation focuses on the implementation of an object’s behavior. Encapsulation is usually achieved by hiding information about the internal state of an object and thus, can be seen as a strategy used in order to provide abstraction.

**Q: What is difference between fail-fast and fail-safe?**

The [Iterator's](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) fail-safe property works with the clone of the underlying collection and thus, it is not affected by any modification in the collection. All the collection classes in java.util package are fail-fast, while the collection classes in **java.util.concurrent** are fail-safe. Fail-fast iterators throw a [ConcurrentModificationException](http://examples.javacodegeeks.com/java-basics/exceptions/java-util-concurrentmodificationexception-how-to-handle-concurrent-modification-exception/), while fail-safe iterator never throws such an exception.

**Q: How do you ensure that N threads can access N resources without deadlock?**

A very simple way to avoid deadlock while using N threads is to impose an ordering on the locks and force each thread to follow that ordering. Thus, if all threads lock and unlock the mutexes in the same order, no deadlocks can arise.

**Q : Why Collection doesn’t extend Cloneable and Serializable interfaces ?**

 The [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) interface specifies groups of objects known as elements. Each concrete implementation of a [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) can choose its own way of how to maintain and order its elements. Some collections allow duplicate keys, while some other collections don’t. The semantics and the implications of either cloning or serialization come into play when dealing with actual implementations. Thus, the concrete implementations of collections should decide how they can be cloned or serialized.

**Q :**[**For-each vs Iterator. Which will be the better option**](http://stackoverflow.com/questions/18508786/for-each-vs-iterator-which-will-be-the-better-option)

for-each is an advanced looping construct. Internally it creates an Iterator and iterates over the Collection. Only possible advantage of using actual Iterator object over for-each construct is that you can modify your collection using Iterator's methods like .remove(). Modifying the collection without using Iterator's methods while iterating will led to ConcurrentModificationException

**Q: Covariant Return Type**

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the **return type.** But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type

 class A {

A get(){return this;}

}

class B1 extends A {

B1 get(){return this;}

void message(){System.out.println("welcome to covariant return type");}

 public static void main(String args[])  {

new B1().get().message();

}

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=B1)

Output: welcome to covariant return type.

**Fail Fast vs Fail Safe Iterator in java**

<http://javahungry.blogspot.com/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html>

**23. How HashMap works in Java?**

HashMap uses set to collect elements indirectly.

A [HashMap in Java stores key-value pairs](http://www.javacodegeeks.com/2014/03/how-hashmap-works-in-java.html). The [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) requires a hash function and uses [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29) and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) **calculates the** **hash value of the key** **and stores the pair in the appropriate index inside the collection**. If the key exists, its value is updated with the new value. Some important characteristics of a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) are its capacity, its load factor and the threshold resizing.

**Q: What are some of the best practices relating to the Java Collection framework?**

**39. What is structure of Java Heap? What is Perm Gen space in Heap ?**

The [JVM has a heap](http://www.javacodegeeks.com/2012/07/5-tips-for-proper-java-heap-size.html) that is the runtime data area from which memory for all class instances and arrays is allocated. It is created at the JVM start-up. Heap memory for objects is reclaimed by an automatic memory management system which is known as a garbage collector. Heap memory consists of live and dead objects. Live objects are accessible by the application and will not be a subject of garbage collection. Dead objects are those which will never be accessible by the application, but have not been collected by the garbage collector yet. Such objects occupy the heap memory space until they are eventually collected by the garbage collector.

**What is an Exception?**

An Exception is an unwanted situation or event that occurs during the execution of a program, exceptions may lead to termination of the program if not handled properly.

**Unchecked Exceptions in Java**

 ArithmeticException

ArrayIndexOutOfBoundsException

ClassCastException

IllegalArgumentException

IllegalArgumentException

NullPointerException

ConcurrentModificationException

**Checked Exceptions in Java**

[IOException](http://docs.oracle.com/javase/8/docs/api/java/io/IOException.html)

FileNotFoundException

ClassNotFoundException

[NoSuchMethodException](http://docs.oracle.com/javase/8/docs/api/java/lang/NoSuchMethodException.html)

ParseException

Q. **What is the difference when the synchronized keyword is applied to a static method or to a non static method?**

Ans. When a synch non static method is called a lock is obtained on the object. When a **synch static method is called a lock is obtained on the class and not on the object**. The lock on the object and the lock on the class donâ interfere with each other. It means, a thread accessing a synch non static method, then the other thread can access the synch static method at the same time but canâ€™t access the synch non static method.

**Q. What is a volatile keyword?**

Ans. In general each thread has its own copy of variable, such that one thread is not concerned with the value of same variable in the other thread. But sometime this may not be the case. Consider a scenario in which the count variable is holding the number of times a method is called for a given class irrespective of any thread calling, in this case irrespective of thread access the count has to be increased so the count variable is declared as volatile. The copy of volatile variable is stored in the main memory, so every time a thread access the variable even for reading purpose the local copy is updated each time from the main memory. The volatile variable also have performance issues.

**Q. If code running is a thread creates a new thread what will be the initial priority of the newly created thread?**

Ans. When a code running in a thread creates a new thread object , the priority of the new thread is set equal to the priority of the thread which has created it.

**Q. What all constructors are present in the Thread class?**

Thread()  
Thread(Runnable target)  
Thread(Runnable target, String name)  
Thread(String name)

**Q. What is difference between Arrays and ArrayList?**

 Ans)

**1. Arrays**are created of fix size whereas ArrayList is of not fix size. It means that once array is declared as :

[1.int](http://1.int/) [] intArray= new int[6];

2.intArray[7]   // will give ArraysOutOfBoundException.

Also the size of array cannot be incremented or decremented. But with arrayList the size is variable.

**2.Once**the array is created elements cannot be added or deleted from it. But with ArrayList the elements can be added and deleted at runtime.

List list = new ArrayList();

list.add(1);

list.add(3);

list.remove(0) // will remove the element from the 1st location.

**3. ArrayList** is one dimensional but array can be multidimensional.

int [][][] intArray= new int[3][2][1];   // 3 dimensional array

**4.To create** an array the size should be known or initialized to some value. If not initialized carefully there could me memory wastage. But arrayList is all about dynamic creation and there is no wastage of memory.

**Q12) which design pattern Iterator follows?**

Ans) It follows **Iterator design pattern**. Iterator Pattern is a type of behavioral pattern. The Iterator pattern is one, which allows you to navigate through a collection of data using a common interface without knowing about the underlying implementation. Iterator should be implemented as an interface. This allows the user to implement it anyway its easier for him/her to return data. The benefits of Iterator are about their strength to provide a common interface for iterating through collections without bothering about underlying implementation.

Example of Iteration design pattern - Enumeration The class java.util.Enumeration is an example of the Iterator pattern. It represents and abstract means of iterating over a collection of elements in some sequential order without the client having to know the representation of the collection being iterated over. It can be used to provide a uniform interface for traversing collections of all kinds.

**What is the difference between preemptive scheduling and time slicing?** –

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

**Q. Boxing and Unboxing Conversions.**

             int i = 10;

             Integer iObj = new Integer(100);

             iObj = i;                              //boxing conversion

             i = iObj;                             //unboxing conversion

[**widening / narrowing conversion?**](http://stackoverflow.com/questions/16781649/real-world-application-of-widening-narrowing-conversion)

        Integer integer = new Integer(10);

        Float floatt = new Float(20F);

**//upcast - widening conversion**

        Object obj = integer; //no explicit cast required

        System.out.println(obj);

**//downcast - narrowing conversion**

        Integer in = (Integer)obj;//only subtype

        System.out.println(in);

**-----------------------------------------------    JSP   ---------------------------------------------------------**

**What are the different types of JSP tags?**

The different types of JSP tags are as follows:

**21. Differentiate between pageContext.include and jsp:include?**

The <jsp:include> standard action and the pageContext.include() method are both used to include resources at runtime. However, thepageContext.include() method always flushes the output of the current page before including the other components, whereas <jsp:include> flushes the output of the current page only if the value of flush is explicitly set to true as follows:

 <jsp:include page="/index.jsp" flush="true"/>

**30. How is scripting disabled?**

Scripting is disabled by setting the scripting-invalid element of the deployment descriptor to true. It is a subelement of jsp-property-group. Its valid values are true and false. The syntax for disabling scripting is as follows:

*<jsp-property-group>*

*<url-pattern>\*.jsp</url-pattern>*

*<scripting-invalid>true</scripting-invalid>*

*</jsp-property-group>*

**---------------------------------------------    Servlet ------------------------------------------------------**

<http://www.developersbook.com/struts/interview-questions/struts-interview-questions-faqs.php>

**Q. Explain the architecture of a Servlet.**

The core abstraction that must be implemented by all servlets is the javax.servlet.Servlet interface. Each servlet must implement it either directly or indirectly, either by extending javax.servlet.GenericServlet or javax.servlet.http.HTTPServlet. Finally, each servlet is able to serve multiple requests in parallel using multithreading.

**Q. What’s the difference between sendRedirect and forward methods ?**

The sendRedirect method creates a new request, while the forward method just forwards a request to a new target. The previous request scope objects are not available after a redirect, because it results in a new request. On the other hand, the previous request scope objects are available after forwarding. Finally, in general, the sendRedirect method is considered to be slower compare to the forward method.

**Q. What is URL Encoding and URL Decoding?**

The URL encoding procedure is responsible for replacing all the spaces and every other extra special character of a URL, into their corresponding Hex representation. In correspondence, URL decoding is the exact opposite procedure.

**---------------------------------------    Struts all about ---------------------------------------**

Q : What design patterns are used in Struts?

|  |
| --- |
|  |

Struts is based on model 2 MVC (Model-View-Controller) architecture. Struts controller uses the command design pattern and the action classes use the adapter design pattern. The process() method of the RequestProcessor uses the template method design pattern. Struts also implement the following J2EE design patterns.

* Service to Worker
* Dispatcher View
* Composite View (Struts Tiles)
* Front Controller
* View Helper
* Synchronizer Token

21.What is DispatchAction?

The DispatchAction class is used to group related actions into one class. Using this class, you can have a method for each logical action compared than a single execute method. The DispatchAction dispatches to one of the logical actions represented by the methods. It picks a method to invoke based on an incoming request parameter. The value of the incoming parameter is the name of the method that the DispatchAction will invoke.

**23. What is the use of ForwardAction?**

The ForwardAction class is useful when you’re trying to integrate Struts into an existing application that uses Servlets to perform business logic functions. You can use this class to take advantage of the Struts controller and its functionality, without having to rewrite the existing Servlets. Use ForwardAction to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page. By using this predefined action, you don’t have to write your own Action class. You just have to set up the struts-config file properly to use ForwardAction.

**Q. What is IncludeAction?**

The IncludeAction class is useful when you want to integrate Struts into an application that uses Servlets. Use the IncludeAction class to include another resource in the response to the request being processed.

**Q: What is difference between LookupDispatchAction and DispatchAction?**

The difference between LookupDispatchAction and DispatchAction is that the actual method that gets called in LookupDispatchAction is based on a lookup of a key value instead of specifying the method name directly.

Q: **Different types of design patterns used in struts?**

 Struts are based on model 2 MVC (Model-View-Controller) architecture. Struts controller uses the command design pattern and the action classes use the adapter design pattern. The process() method of the RequestProcessor uses the template method design pattern. Struts also implement the following J2EE design patterns.   
  
Service to Worker   
Dispatcher View   
Composite View (Struts Tiles)   
Front Controller   
View Helper   
Synchronizer Token

**Q: How to Setup validator framework in Struts?**

<http://www.techfaq360.com/struts_interview_questions.jsp?qid=420>

**---------------------    Hibernate all about   -----------------------**

<http://www.developersbook.com/hibernate/interview-questions/hibernate-interview-questions-faqs-2.php>

[**What is the use of cascade in hbm file?**](http://www.geekinterview.com/question_details/46108)

[**What is the difference between and merge and update**](http://www.geekinterview.com/question_details/51897)**.**

**What’s the difference between load() and get()?**

load() vs. get() :-

|  |  |
| --- | --- |
| **load()** | **get()** |
| Only use the load() method if you are sure that the object exists. | If you are not sure that the object exists, then use one of the get() methods. |
| load() method will throw an exception if the unique id is not found in the database. | get() method will return null if the unique id is not found in the database. |
| load() just returns a proxy by default and database won’t be hit until the proxy is first invoked. | get() will hit the database immediately. |

**What is the difference between and merge and update?**

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | ------  -----  SessionFactory factory = cfg.buildSessionFactory();  Session session1 = factory.openSession();    Student s1 = null;  Object o = session1.get(Student.class, new Integer(101));  s1 = (Student)o;  session1.close();    s1.setMarks(97);    Session session2 = factory.openSession();  Student s2 = null;  Object o1 = session2.get(Student.class, new Integer(101));  s2 = (Student)o1;  Transaction tx=session2.beginTransaction();    session2.merge(s1); |

 Q: **Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a **helper class which provides different methods for querying/retrieving data from the database**. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**Advantages of hibernates:**

         Hibernate supports Inheritance, Associations, Collections

         In hibernate if we save the derived class object, then its base class object will also be stored into the database, it means hibernate supporting inheritance

         Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One

         This will also supports collections like List, Set, Map (Only new collections)

         In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws. Actually in hibernate we have the translator which converts checked to Un-checked

         Hibernate has capability to generate primary keys automatically while we are storing the records into database

         Hibernate has its own query language, i.e hibernate query language which is database independent

         So if we change the database, then also our application will works as HQL is database independent

         HQL contains database independent commands

         While we are inserting any record, if we don’t have any particular table in the database, JDBC will rises an error like “View not exist”, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us

         Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.

         Hibernate supports annotations, apart from XML

         Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.

         Getting pagination in hibernate is quite simple.

**Q: What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set: ->

**dynamic-update** (defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed

**dynamic-insert** (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**-------------------------------------    Spring  --------------------------------------------**

**Bean Factory and Application Context?**

Similarities:

         They like a factory class that contains a collection of beans.

         Both load bean definitions, wire beans together, and dispense beans upon request. But it also provides:

         They holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients

**Advantage of Application Context over Bean Factory –**

A bean factory is fine to simple applications, but to take advantage of the full power of the Spring framework, you may want to move up to Springs more advanced container, the application context.

         Application contexts provide a means for resolving text messages, including support for i18n of those messages.

         Application contexts provide a generic way to load file resources, such as images.

         Application contexts can publish events to beans that are registered as listeners.

         Certain operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context.

         ResourceLoader support: Spring’s Resource interface us a flexible generic abstraction for handling low-level resources. An application context itself is a ResourceLoader, Hence provides an application with access to deployment-specific Resource instances.

         MessageSource support: The application context implements MessageSource, an interface used to obtain localized messages, with the actual implementation being pluggable

**How to integrate spring and Hibernate:-**

1.    <bean id="template" **class**="org.springframework.orm.hibernate3.HibernateTemplate">

2.    <property name="sessionFactory" ref="**mysessionFactory**"></property>

3.    </bean>

4.

5.    <bean id="d" **class**="com.javatpoint.EmployeeDao">

6.    <property name="template" ref="template"></property>

7.    </bean>

**And in mysession Factory :**

1.    <bean id="**mysessionFactory**" **class**="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

2.    <property name="mappingResources">

3.    <property name="dataSource" ref="dataSource"></property>

**---------------------------------------   ENUMS in Java --------------------------------------------------------------**

**Collecting key notes about enum**

         enums are implicitly final subclasses of java.lang.Enum

         if an enum is a member of a class, it’s implicitly static

         new can never be used with an enum, even within the enum type itself

         name and valueOf simply use the text of the enum constants, while toString may be overridden to provide any content, if desired

         for enum constants, equals and == amount to the same thing, and can be used interchangeably

         enum constants are implicitly public static final

         the order of appearance of enum constants is called their “natural order”, and defines the order used by other items as well : compareTo, iteration order of values , EnumSet, EnumSet.range.

         Constructors for an enum type should be declared as private. The compiler allows non private declares for constructors, but this seems misleading to the reader, since new can never be used with enum types.

         Since these enumeration instances are all effectively singletons, they can be compared for equality using identity (“==”).

         you can use Enum in Java inside Switch statement like int or char primitive data type

**Q: What is CountDownLatch,  CyclicBarrier, [ConcurrentHashMap](http://howtodoinjava.com/2013/05/27/best-practices-for-using-concurrenthashmap/" \t "_blank" \o "Best practices for using ConcurrentHashMap) and [BlockingQueue](http://howtodoinjava.com/2012/10/20/how-to-use-blockingqueue-and-threadpoolexecutor-in-java/" \t "_blank" \o "How to use BlockingQueue and ThreadPoolExecutor in java) ?**

**CountDownLatch** was introduced with JDK 1.5 along with other concurrent utilities like CyclicBarrier, Semaphore, [ConcurrentHashMap](http://howtodoinjava.com/2013/05/27/best-practices-for-using-concurrenthashmap/" \t "_blank" \o "Best practices for using ConcurrentHashMap) and[BlockingQueue](http://howtodoinjava.com/2012/10/20/how-to-use-blockingqueue-and-threadpoolexecutor-in-java/) in java.util.concurrent package. This class enables a java thread to wait until other set of threads completes their tasks. e.g. Application’s main thread want to wait, till other service threads which are responsible for starting framework services have completed started all services.

CountDownLatch works by having a counter initialized with number of threads, which is decremented each time a thread complete its execution. When count reaches to zero, it means all threads have completed their execution, and thread waiting on latch resume the execution.

More here  :

<http://howtodoinjava.com/2013/07/18/when-to-use-countdownlatch-java-concurrency-example-tutorial/>

**CyclicBarrier**

The java.util.concurrent.CyclicBarrier class is a synchronization mechanism that can synchronize threads progressing through some algorithm. In other words, it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue. Here is a diagram illustrating that:

The waiting threads waits at the CyclicBarrier until either:

* The last thread arrives (calls await() )
* The thread is interrupted by another thread (another thread calls its interrupt() method)
* Another waiting thread is interrupted
* Another waiting thread times out while waiting at the CyclicBarrier
* The CyclicBarrier.reset() method is called by some external thread.

**BlockingQueue :**

o   java.util.concurrent.BlockingQueue is a Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

o   BlockingQueue doesn’t accept null values and throw NullPointerException if you try to store null value in the queue.

o   BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.

o   BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue as it’s handled by implementation classes of BlockingQueue.

---------------------------------------   Data base interview question --------------------------

**Best top 100 interview questions and answers:**

[**http://a4academics.com/interview-questions/53-database-and-sql/397-top-100-database-sql-interview-questions-and-answers-examples-queries?showall=&limitstart**](http://a4academics.com/interview-questions/53-database-and-sql/397-top-100-database-sql-interview-questions-and-answers-examples-queries?showall=&limitstart)**=**

**update to flip-flop the GENDER values:**

update emp

  set gender = case

                     when gender = 'F' then 'M'

                     when gender = 'M' then 'F'

             end ;

------------------

**--------------------------------------   Real time interview questions  --------------------------------**

**2) Have you faced out of memory error? If yes how you fixed ? Tell different scenarios why it comes?**

<http://stackoverflow.com/questions/37335/how-to-deal-with-java-lang-outofmemoryerror-java-heap-space-error-64mb-heap>

**Shallow cloning and deep cloning in java:**

## Shallow Copy

Generally clone method of an object, creates a new instance of the same class and copies all the fields to the new instance and returns it. This is nothing but shallow copy. Object class provides a clone method and provides support for the shallow copy. It returns ‘Object’ as type and you need to explicitly cast back to your original object.

Since the Object class has the clone method ([protected](http://javapapers.com/core-java/access-modifiers-in-java-explain/)) you cannot use it in all your classes. The class which you want to be cloned should implement clone method and [overwrite](http://javapapers.com/core-java/overloading-and-overriding/)it. It should provide its own meaning for copy or to the least it should invoke the super.clone(). Also you have to implement Cloneable[marker interface](http://javapapers.com/core-java/abstract-and-interface-core-java-2/what-is-a-java-marker-interface/) or else you will get CloneNotSupportedException. When you invoke the super.clone() then you are dependent on the Object class’s implementation and what you get is a shallow copy.

## Deep Copy

When you need a deep copy then you need to implement it yourself. When the copied object contains some other object its references are copied recursively in deep copy. When you implement deep copy be careful as you might fall for cyclic dependencies. If you don’t want to implement deep copy yourselves then you can go for [serialization](http://javapapers.com/core-java/java-serialization/). It does implements deep copy implicitly and gracefully handling cyclic dependencies.  
One more disadvantage with this clone system is that, most of the [interface / abstract](http://javapapers.com/core-java/abstract-and-interface-core-java-2/difference-between-a-java-interface-and-a-java-abstract-class/" \o "Difference Between Interface and Abstract Class) class writers in java forget to put a public clone method. For example you can take List. So when you want to clone their implementations you have to ignore the abstract type and use actual implementations like ArrayList by name. This completely removes the advantage and goodness of abstractness.

When implementing a [singleton pattern](http://javapapers.com/design-patterns/singleton-pattern/" \o "Singleton Pattern), if its superclass implements a public clone() method, to prevent your subclass from using this class’s clone() method to obtain a copy overwrite it and throw an [exception](http://javapapers.com/core-java/java-exception/explain-type-of-exceptions-or-checked-vs-unchecked-exceptions-in-java/" \o "Checked vs Unchecked Exceptions) of type  CloneNotSupportedException.

Note that clone is not for instantiation and initialization. It should not be synonymously used as creating a new object. Because the constructor of the cloned objects may never get invoked in the process. It is about copying the object in discussion and not creating new. It completely depends on the clone implementation. One more disadvantage (what to do there are so many), clone prevents the use of [final](http://javapapers.com/core-java/explain-the-final-keyword-in-java/" \o "Java Final Keyword) fields. We have to find roundabout ways to copy the final fields into the copied object.

Clone is an agreement between you, compiler and implementer. If you are confident that you all three have good knowledge of java, then go ahead and use clone. If you have a slightest of doubt better copy the object manually.

**Ques:** Java has the ability to create a copy of object with similar attributes and state form original object. This concept in java is called cloning. To implement cloning, we have to implement**java.lang.Cloneable** interface and override **clone()** method from Object class. It is a good idea to prevent cloning in a singleton class. To prevent cloning on singleton object, let us explicitly throw **CloneNotSupportedException** exception in clone() method.