**Core JAVA Concept**

**Encapsulation:**

Encapsulation is the process to binding the data and function in single unit. So a class is actually an encapsulation which holds both member variable and method as single unit.

It also provides maintainability, flexibility, extensibility but also depends upon your implementation.

**IS-A vs. HAS-A**

IS-A relationship is based on class inheritance or interface implementation. IS-A is a way of saying this thing is type of that thing.

HAS-A relationship is based on uses rather than inheritance.

The **Cloneable** interface defines no members. It is used to indicate that a class allows a bitwise copy of an object (that is, a clone) to be made. If you try to call clone( ) on a class that does not implement Cloneable, a CloneNotSupportedException is thrown. When a clone is made, the constructor for the object being cloned is not called. A clone is simply an exact copy of the original.

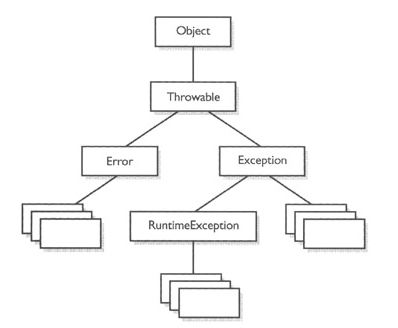
Cloning is a potentially dangerous action, because it can cause unintended side effects. For example, if the object being cloned contains a reference variable called obRef, then when the clone is made, obRef in the clone will refer to the same object as does obRef in the original. If the clone makes a change to the contents of the object referred to by obRef, then it will be changed for the original object, too. Here is another example. If an object opens an I/O stream and is then cloned, two objects will be capable of operating on the same stream. Further, if one of these objects closes the stream, the other object might still attempt to write to it, causing an error.

Because cloning can cause problems, clone( ) is declared as protected inside Object. This means that it must either be called from within a method defined by the class that implements Cloneable, or it must be explicitly overridden by that class so that it is public. Let's look at an example of each approach.

The following program implements Cloneable and defines the method cloneTest( ), which calls clone( ) in Object:

**Exception:**

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

**Exception Hierarchy:**  


There are basically three kinds of exceptions,

1. **Checked Exception:**

Checked exceptions *are subject* to the Catch or Specify Requirement. All exceptions are checked exceptions, except for those indicated by Error, RuntimeException, and their subclasses. They are also treated as compile time exceptions.

1. **Error**

These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from.

1. **RuntimeException**

These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from. These usually indicate programming bugs, such as logic errors or improper use of an API.

Errors and runtime exceptions are collectively known as ***unchecked*** *exceptions*.

Java 7 <https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>

Tutorial link

<http://tutorials.jenkov.com/java-exception-handling/try-with-resources.html>

**What is exception matching?**

Ans) Exception matching is the process by which the the jvm finds out the matching catch block for the exception thrown from the list of catch blocks. When an exception is thrown, Java will try to find by looking at the available catch clauses in the top down manner. If it doesn't find one, it will search for a handler for a supertype of the exception. If it does not find a catch clause that matches a supertype for the exception, then the exception is propagated down the call stack. This process is called exception matching.

**Overloading Methods**

Overloading means methods in same class with same name but different signature.

* Overloaded methods must change the argument list.
* Overloaded methods can change the return type.
* Overloaded methods can change the access modifier.
* Overloaded methods can declare new or broader checked exceptions.
* A method can be overloaded in the same class or in a subclass.

**Overridden Method:**

An instance method in subclass with same signature and return type, as an instance method in the super class is call overridden method.

The overriding method has the same name, number and type of parameters, and return type as the method it overrides. An overriding method can also return a subtype of the type returned by the overridden method. This is called a covariant return type.

If you will define the class (static) method in the Sub class with same signature and return type as a class method (static) in super class then it will call as method hiding.

If a method has Private access modifier in a super class and its subclass has the method with same signature then it will be called as redefinition.

**Overridden method Summary:**

|  |  |  |
| --- | --- | --- |
|  | **Superclass Instance Method** | **Superclass Static Method** |
| **Subclass Instance Method** | Overrides | Generates a compile-time error |
| **Subclass Static Method** | Generates a compile-time error | Hides |

**Difference between Overloaded and Overridden Methods**

|  |  |  |
| --- | --- | --- |
|  | **Overloaded Methods** | **Overridden Methods** |
| argument list | Must change | Must not change |
| return type | Can change | Must not change |
| Exceptions | Can change | Can reduce or eliminate.  Must not throw new or  broader checked exceptions |
| Access | Can change | Must not make more  restrictive (can be less  restrictive) |
| Invocation | Determine At Compile time(Compile time polymorphism ) | Determined at run (Run Time Polymorphism) |

**JAVA Generics**

Introduced in J2SE 5.0, this long-awaited enhancement to the type system allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety to the Collections Framework and eliminates the drudgery of casting.

**What is the difference between Enumeration and Iterator?**

The functionality of Enumeration interface is duplicated by the Iterator interface. Iterator has a remove() method while Enumeration doesn't. Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects, where as using Iterator we can manipulate the objects also like adding and removing the objects.

So Enumeration is used whenever we want to make Collection objects as Read-only.

**What is the difference b/w Iterator and ListIterator?**

**Iterator :** Enables you to cycle through a collection in the forward direction only, for obtaining or removing elements.

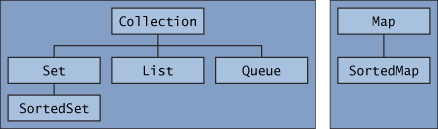
**ListIterator :** It extends Iterator, allow bidirectional traversal of list and the modification of elements.

**What Is a Collections Framework?**

A *collection*—sometimes called a container—is simply an object that groups multiple

elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data.

A *collections framework* is a unified architecture for representing and manipulating collections. All collections frameworks contain the following:

* **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. 
* **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
* **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

There are two "groups" of interfaces: Collection's and Map's.

**Iterable Interface**

The Iterable interface (java.lang.Iterable) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface. Iterable interface has only one method i.e. iterator().

A class that implements Iterable interface can be used with the new for loop.

List list = new ArrayList();

for(Object o : list){

//do something o;

}

# Collections Interview Questions

**Q1) What is difference between ArrayList and vector?**

Ans: )

1) Synchronization - ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block and thus making Vector class thread-safe.

2) Data growth - Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.

**Q2) How can Arraylist be synchronized without using Vector?**

Ans) Arraylist can be synchronized using:

Collection.synchronizedList(List list)

Other collections can be synchronized:

Collection.synchronizedMap(Map map)

Collection.synchronizedCollection(Collection c)

**Q3) If an Employee class is present and its objects are added in an arrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?**

Ans) 1) Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID

2) Now call Collections.sort() method and pass list as an argument.

Now consider that Employee class is a jar file.

1) Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method .

2) Call Collections.sort() on the list and pass comparator as an argument.

**Q4)What is difference between HashMap and HashTable?**

Ans) Both collections implements Map. Both collections store value as key-value pairs. The key differences between the two are

1. Hashmap is not synchronized in nature but hshtable is.

2. Another difference is that iterator in the HashMap is fail-safe while the enumerator for the Hashtable isn't.  
**Fail-safe** - â€œif the Hashtable is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationExceptionâ€?

3. HashMap permits null values and only one null key, while Hashtable doesn't allow key or value as null.

5. In HashTable you can change the iteration but in the case of HashMap you can change the iteration but you will get a java.util.ConcurrentModificationException.

**Q5) What are the classes implementing List interface?**

Ans)  
There are three classes that implement List interface:  
1) **ArrayList** : It is a resizable array implementation. The size of the ArrayList can be increased dynamically also operations like add,remove and get can be formed once the object is created. It also ensures that the data is retrieved in the manner it was stored. The ArrayList is not thread-safe.  
  
2) **Vector**: It is thread-safe implementation of ArrayList. The methods are wrapped around a synchronized block.  
  
3) **LinkedList**: the LinkedList also implements Queue interface and provide FIFO(First In First Out) operation for add operation. It is faster if than ArrayList if it performs insertion and deletion of elements from the middle of a list.

**Q6) Which all classes implement Set interface?**

Ans) A Set is a collection that contains no duplicate elements. More formally, sets contain no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. **HashSet,SortedSet and TreeSet** are the commnly used class which implements Set interface.

**SortedSet** - It is an interface which extends Set. A the name suggest , the interface allows the data to be iterated in the ascending order or sorted on the basis of Comparator or Comparable interface. All elements inserted into the interface must implement Comparable or Comparator interface.

**TreeSet** - It is the implementation of SortedSet interface.This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains). The class is not synchronized.

**HashSet:** This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets

**Q7) What is difference between List and a Set?**

Ans)  
1) List can contain duplicate values but Set doesnt allow. Set allows only to unique elements.   
2) List allows retrieval of data to be in same order in the way it is inserted but Set doesnt ensures the sequence in which data can be retrieved.(Except HashSet)

**Q8) What is difference between Arrays and ArrayList ?**

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

* 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.

1. 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.

1. ArrayList is one dimensional but array can be multidimensional.

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

1. To create an array the size should be known or initalized 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.

**Q9) When to use ArrayList or LinkedList ?**

Ans)  Adding new elements is pretty fast for either type of list. For the ArrayList, doing  random lookup using "get" is fast, but for LinkedList, it's slow. It's slow because there's no efficient way to index into the middle of a linked list. When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But here LinkedList is fast, because deletion can be done simply by changing a couple of links. So an ArrayList works best for cases where you're doing random access on the list, and a LinkedList works better if you're doing a lot of editing in the middle of the list.

Source : [Read More - from java.sun](http://java.sun.com/developer/TechTips/1999/tt0809.html)

**Q10) Consider a scenario. If an ArrayList has to be iterate to read data only, what are the possible ways and which is the fastest?**

Ans) It can be done in two ways, using for loop or using iterator of ArrayList. The first option is faster than using iterator. Because value stored in arraylist is indexed access. So while accessing the value is accessed directly as per the index.

**Q11) Now another question with respect to above question is if accessing through iterator is slow then why do we need it and when to use it.**

Ans) For loop does not allow the updation in the array(add or remove operation) inside the loop whereas Iterator does. Also Iterator can be used where there is no clue what type of collections will be used because all collections have iterator.

// Concurrent modification example.

List<Integer> list = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < 10; i++) {

list.add(i \* 10);

}

**for** (Iterator iterator = list.iterator(); iterator.hasNext();) {

Integer integer = (Integer) iterator.next();

System.*out*.println(integer);

**if** (integer == 90) {

list.remove(integer);// wrong can throw // ConcurrentModificationException

}

iterator.remove(); //Right

}

**Q14) What is difference between iterator access and index access**?

Ans) Index based access allow access of the element directly on the basis of index. The cursor of the datastructure can directly goto the 'n' location and get the element. It doesnot traverse through n-1 elements.

In Iterator based access, the cursor has to traverse through each element to get the desired element.So to reach the 'n'th element it need to traverse through n-1 elements.

Insertion,updation or deletion will be faster for iterator based access if the operations are performed on elements present in between the datastructure.

Insertion,updation or deletion will be faster for index based access if the operations are performed on elements present at last of the datastructure.

Traversal or search in index based datastructure is faster.

ArrayList is index access and LinkedList is iterator access

**Q15) How to sort list in reverse order?**

Ans) To sort the elements of the List in the reverse natural order of the strings, get a reverse Comparator from the Collections class with reverseOrder(). Then, pass the reverse Comparator to the sort() method.

List list = new ArrayList();

Comparator comp = Collections.reverseOrder();

Collections.sort(list, comp)

**Q16) Can a null element added to a Treeset or HashSet?**

Ans) A null element can be added only if the set contains one element because when a second element is added then as per set defination a check is made to check duplicate value and comparison with null element will throw NullPointerException.  
HashSet is based on hashMap and can contain null element.

**Q17) How to sort list of strings - case insensitive?**

Ans) using Collections.sort(list, String.CASE\_INSENSITIVE\_ORDER);

**Q18) How to make a List (ArrayList,Vector,LinkedList) read only?**

Ans) A list implemenation can be made read only using **Collections.unmodifiableList(list)**. This method returns a new list. If a user tries to perform add operation on the new list; UnSupportedOperationException is thrown.

**Q21) Which data structure HashSet implements**

Ans) HashSet implements hashmap internally to store the data. The data passed to hashset is stored as key in hashmap with null as value.

**Q24) What is WeakHashMap?**

Ans) A hashtable-based Map implementation with weak keys. An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely, the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector, that is, made finalizable, finalized, and then reclaimed. When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently than other Map implementations.

About Singleton:

If you have a singleton class and you run two webapps that use this class in Tomcat both webapps will get 2 different instances of this singleton in JVM running the Tomcat.

But if your webapp will use a singleton from JRE or Tomcat shared libs, eg Runtime.getRuntime webapps will get the same instance of Runtime.

This is because Tomcat uses individual class loaders for webapps. When a webapp class loader loads a class it first tries to find it on webapp class path, if the class is not found it asks parent class loader to load the class.

Inner Classes: (IC)

Firstly you cannot make top-level-class static. you can only make a nested class static. By making an nested class static you basically are saying that you don't need an instance of the nested class to use it from your outer class/top-level-class.

Top level class in java can have only two access modifiers public or default. Where is an inner class can have private, public protected or default.

Example:

class Outer {

static class nestedStaticClass {

//its member variables and methods (don't nessarily need to be static)

//but cannot access members of the enclosing class

}

public void OuterMethod(){

//can access members of nestedStaticClass w/o an instance

}

}

Also to add, it is illegal to declare static fields inside an inner class unless they are constants (in other words, static final). As a static nested class isn't an inner class you *can* declare static members here.

Can class be nested in nested class?

In a word, yes. Look at the below Test, both nested inner classes and nested static class can have nested classes in 'em. But remember you can only declare a static class inside a top-level class, it is illegal to declare it inside an inner class.

public class Test {

public class Inner1 {

public class Inner2 {

public class Inner3 {

}

}

}

public static class nested1 {

public static class nested2 {

public static class nested3 {

}

}

}

}

**Benefits of Java Inner Class**

1. If a class is useful to only one class, it makes sense to keep it nested and together. It helps in packaging of the classes.
2. Java inner classes implements encapsulation. Note that inner classes can access outer class private members and at the same time we can hide inner class from outer world.
3. Keeping the small class within top-level classes places the code closer to where it is used and makes code more readable and maintainable.