**Collections** : A collection allows a group of objects to be treated as a single unit. collection define a set of core Interfaces as follows.

**Question:** what is a collection ?

**Answer:** Collection is a group of objects. java.util package provides important types of collections. There are two fundamental types of collections they are Collection and Map. Collection types hold a group of objects, Eg. Lists and Sets where as Map types hold group of objects as key, value pairs Eg. HashMap and Hashtable.

**Collection** **Map** *Hash Map class*

*Hash Table class*

**Set**  *Hash set* **List**

*Array List*

**Sorted set** *Tree set Vector List*

*Linked List* **Sorted map** *Tree Map class*

* + **Collection Interface** :
    - * The CI is the root of collection hierarchy and is used for common functionality across all collections. There is no direct implementation of Collection Interface.
  + **Set Interface**: extends *Collection* Interface. The Class Hash set implements Set Interface.
    - * Is used to represent the group of *unique* elements.
      * Set stores elements in an unordered way but does *not* contain duplicate elements.
  + **Sorted set** : extends *Set* Interface. The class *Tree Set* implements Sorted set Interface.
    - * It provides the extra functionality of keeping the elements sorted.
      * It represents the collection consisting of *Unique*, sorted elements in *ascending order*.
  + **List** : extends Collection Interface. The classes *Array List, Vector List & Linked List* implements List Interface.
    - * Represents the sequence of numbers in a fixed order.
      * But may contain duplicate elements.
      * Elements can be inserted or retrieved by their position in the List using Zero based index.
      * List stores elements in an ordered way.
  + **Map Interface**:basic Interface. The classes *Hash Map & Hash Table* implements Map interface.
    - * Used to represent the mapping of unique keys to values.
      * By using the key value we can retrive the values. Two basic operations are get( ) & put( ) .
  + **Sorted Map** : extends *Map* Interface. The Class *Tree Map* implements Sorted Map Interface.
    - * Maintain the values of key order.
      * The entries are maintained in ascending order.
* **Collection classes:**

**Abstract Collection**

**Abstract List Abstract Set Abstract Map**

**Abstract Array List Hash Set Tree Set Hash Map Tree Map**

**Sequential**

**List**

## **Linked List**

**List Map**

| |

## **Abstract List Dictonary**

| |

**Vector HashTable**

| |

**Stack Properities**

* + **HashSet** : Implements Set Interface. HashSet hs=new HashSet( );
* The elements are not stored in *sorted* order. hs.add(“m”);
  + **TreeSet** : Implements Sorted set Interface. TreeSet ts=new TreeSet( );
* The elements are stored in sorted *ascending* order. ts.add(“H”);
* Access and retrieval times are quit fast, when storing a large amount of data.
  + **Vector** : Implements List Interface.
* Vector implements *dynamic* array. Vector v = new vector( );
* Vector is a *growable* object. V1.addElement(new Integer(1));
* Vector is *Synchronized*, it can’t allow special *characters* and *null values*.
* All vector starts with intial capacity, after it is reached next time if we want to store object in vector, the vector automatically allocates space for that Object plus extra room for additional Objects.
  + **ArrayList** : Implements List Interface.
* Array can dynamically increase or decrease size. ArrayList a1=new ArrayList( );
* Array List are ment for Random ascessing. A1.add(“a”);
* Array List are created with intial size, when the size is increased, the collection is automatically enlarged. When an Objects are removed, the array may be shrunk.
  + **Linked List** : Implements List Interface.
    - Inserting or removing elements in the *middle of the array*. LinkedList l1=new LinkedList( );
    - Linked list are meant for *Sequential accessing*. L1.add(“R”);
    - Stores Objects in a separate link.

 **Map Classes**: Abstract Map; Hash Map ; Tree Map

* + **Hash Map** : Implements Map Interface. Hashmap() , Hashmap(Map m), Hashmap(int capacity)
    - The Elements may not in Order.
    - Hash Map is *not synchronized* and permits null values
    - Hash Map is *not serialized*. Hashmap hm = new HashMap( );
    - Hash Map supports *Iterators*. hm.put(“Hari”,new Double(11.9));
      * **Hash Table** : Implements Map Interface.
* Hash Table is *synchronized* and does *not* permit *null* values.
* Hash Table is *Serialized*. Hashtable ht = new Hashtable( );
* Stores key/value pairs in Hash Table. ht.put(“Prasadi”,new Double(74.6));

 A Hash Table stores information by using a mechanism called hashing. In hashing the informational content of a key is used to determine a unique value, called its Hash Code. The Hash Code is then used as the index at which the data associated with the key is stored. The Transformation of the key into its Hash Code is performed automatically- we never see the Hash Code. Also the code can’t directly index into h c.

* + **Tree Map** : Implements Sorted Set Interface. TreeMap tm=new TreeMap( );
* The elements are stored in *sorted ascending order*. tm.put( “Prasad”,new Double(74.6));
* Using *key value* we can retrieve the data.
* Provides an efficient means of storing *key/value pairs* in sorted order and allows *rapid retrivals*.
  + **Iterator**: Each of collection class provided an iterator( ).

By using this iterator Object, we can access each element in the collection – one at a time.

We can remove() ; Hashnext( ) – go next; if it returns false –end of list.

**Iterarator Enumerator**

Iterator itr = a1.iterator( ); Enumerator vEnum = v.element( );

While(itr.hashNext( )) System.out.println(“Elements in Vector :”);

{ while(vEnum.hasMoreElements( ) )

Object element = itr.next( ); System.out.println(vEnum.nextElement( ) + “ “);

System.out.println(element + “ “);

}

**Collections**

**1.Introduction**

**2.Legacy Collections**

**1. The Enumeration Interface**

**2. Vector**

**3. Stack**

**4. Hashtable**

**5. Properties**

**3.Java 2 Collections**

**1. The Interfaces of the collections framework**

**2. Classes in the collections framework**

**3. ArrayList & HashSet**

**4. TreeSet & Maps**

*Introduction :*

•Does your class need a way to easily search through thousands of items quickly?

• Does it need an ordered sequence of elements and the ability to rapidly insert and remove elements in the middle of the sequence?• Does it need an array like structure with random-access ability that can grow at runtime?

**List Map**

| |

## **Abstract List Dictonary**

| |

**Vector HashTable**

| |

## **Stack Properities**

#### **The Enumeration Interface :**

•enumerate (obtain one at a time) the elements in a collection of objects.

specifies two methods :

**boolean hasMoreElements() :** Returns true when there are still more elements to extract, and false when all of the elements have been enumerated.

**Object nextElement() :** Returns the next object in the enumeration as a generic Object reference.

**VECTOR :**

* Vector implements *dynamic* array. Vector v = new vector( );
* Vector is a *growable* object. V1.addElement(new Integer(1));
* Vector is *Synchronized*, it can’t allow special *characters* and *null values*.
* Vector is a variable-length array of object references.
* Vectors are created with an initial size.
* When this size is exceeded, the vector is automatically enlarged.
* When objects are removed, the vector may be shrunk.

Constructors : **Vector() :** Default constructor with initial size 10.

**Vector(int size) :** Vector whose initial capacity is specified by size.

**Vector(int size,int incr) :**Vector whose initialize capacity is specified by size and whose increment is specified by incr.

Methods :

**final void addElement(Object element) :** The object specified by element is added to the vector.

**final Object elementAt(int index) :** Returns the element at the location specified by index.

**final boolean removeElement(Object element) :** Removes element from the vector

**final boolean isEmpty() :** Returns true if the vector is empty, false otherwise.

**final int size() :** Returns the number of elements currently in the vector.

**final boolean contains(Object element) :** Returns true if element is contained by the vector and false if it is not.

**STACK :**

•Stack is a subclass of Vector that implements a standard last-in, first-out stack

Constructor : **Stack()** Creates an empty stack.

**Methods :**

Object push(Object item) : Pushes an item onto the top of this stack.

Object pop() : Removes the object at the top of this stack and returns that object as the value of this function. An EmptyStackException is thrown if it is called on empty stack.

boolean empty() : Tests if this stack is empty.

Object peek() : Looks at the object at the top of this stack without removing it from the stack.

int search(Object o) **:** Determine if an object exists on the stack and returns the number of pops that would be required to bring it to the top of the stack.

**HashTable :**

* Hash Table is *synchronized* and does *not* permit *null* values.
* Hash Table is *Serialized*. Hashtable ht = new Hashtable( );
* Stores key/value pairs in Hash Table. ht.put(“Prasadi”,new Double(74.6));
* Hashtable is a concrete implementation of a Dictionary.
* Dictionary is an abstract class that represents a key/value storage repository.
* A Hashtable instance can be used store arbitrary objects which are indexed by any other arbitrary object.
* A Hashtable stores information using a mechanism called hashing.
* When using a Hashtable, you specify an object that is used as a key and the value (data) that you want linked to that key.

Constructors : **Hashtable() Hashtable(int size)**

Methods :

**Object put(Object key,Object value) :** Inserts a key and a value into the hashtable.

**Object get(Object key) :** Returns the object that contains the value associated with key.

**boolean contains(Object value) :** Returns true if the given value is available in the hashtable. If not, returns false.

**boolean containsKey(Object key) :** Returns true if the given key is available in the hashtable. If not, returns false.

**Enumeration elements() :** Returns an enumeration of the values contained in the hashtable.

**int size() :** Returns the number of entries in the hashtable.

#### **Properties**

•Properties is a subclass of Hashtable

• Used to maintain lists of values in which the key is a String and the value is also a String

• Constructors

**Properties()**

**Properties(Properties propDefault) :** Creates an object that uses propDefault for its default value.

Methods :

**String getProperty(String key) :** Returns the value associated with key.

**Strng getProperty(String key, String defaultProperty) :** Returns the value associated with key. defaultProperty is returned if key is neither in the list nor in the default property list .

**Enumeration propertyNames() :** Returns an enumeration of the keys. This includes those keys found in the default property list.

*The Interfaces in Collections Framework*

# **Collection Map Iterator**

**Set List SortedMap ListIterator**

**|**

**SortedSet**

**Collection :**

* A collection allows a group of objects to be treated as a single unit.
* The Java collections library forms a framework for collection classes.
* The CI is the root of collection hierarchy and is used for common functionality across all collections.
* There is no direct implementation of Collection Interface.
* Two fundamental interfaces for containers:

• Collection

**boolean add(Object element) :** Inserts element into a collection

**Set Interface**: extends *Collection* Interface. The Class Hash set implements Set Interface.

* Is used to represent the group of *unique* elements.
* Set stores elements in an unordered way but does *not* contain duplicate elements.
* identical to Collection interface, but doesn’t accept duplicates.

**Sorted set** : extends *Set* Interface. The class *Tree Set* implements Sorted set Interface.

* It provides the extra functionality of keeping the elements sorted.
* It represents the collection consisting of *Unique*, sorted elements in *ascending order*.
* expose the comparison object for sorting.

### **List Interface** :

* ordered collection – Elements are added into a particular position.
* Represents the sequence of numbers in a fixed order.
* But may contain duplicate elements.
* Elements can be inserted or retrieved by their position in the List using Zero based index.
* List stores elements in an ordered way.

**Map Interface**: Basic Interface.The classes *Hash Map & HashTable* implements Map interface.

* Used to represent the mapping of unique keys to values.
* By using the key value we can retrive the values.
* Two basic operations are get( ) & put( ) .

**boolean put(Object key, Object value) :** Inserts given value into map with key

**Object get(Object key) :** Reads value for the given key.

**Tree Map Class**: Implements Sorted Set Interface.

* The elements are stored in *sorted ascending order*.
* Using *key value* we can retrieve the data.
* Provides an efficient means of storing *key/value pairs* in sorted order and allows *rapid retrivals*.

TreeMap tm=new TreeMap( );

tm.put( “Prasad”,new Double(74.6));

*The* ***Classes*** *in Collections Framework*

**Abstract Collection**

**Abstract List Abstract Set Abstract Map**

**Abstract Array List Hash Set Tree Set Hash Map Tree Map**

**Sequential**

**List**

## **Linked List**

##### ***ArrayList***

• Similar to Vector: it encapsulates a dynamically reallocated Object[] array

• Why use an ArrayList instead of a Vector?

• All methods of the Vector class are synchronized, It is safe to access a Vector object from two threads.

• ArrayList methods are not synchronized, use ArrayList in case of no synchronization

• Use *get* and *set* methods instead of *elementAt* and *setElementAt* methods of vector

##### ***HashSet***

• Implements a set based on a hashtable

• The default constructor constructs a hashtable with 101 buckets and a load factor of 0.75

**HashSet(int initialCapacity)**

**HashSet(int initialCapacity,float loadFactor)**

loadFactor is a measure of how full the hashtable is allowed to get before its capacity is automatically increased

• Use Hashset if you don’t care about the ordering of the elements in the collection

##### ***TreeSet***

• Similar to hash set, with one added improvement

• A tree set is a *sorted collection*

•Insert elements into the collection in any order, when it is iterated, the values are automatically presented in sorted order

• **Maps :**  Two implementations for maps:

##### ***HashMap***

* hashes the keys
* The Elements may not in Order.
* Hash Map is *not synchronized* and permits null values
* Hash Map is *not serialized*.
* Hash Map supports *Iterators*.

##### ***TreeMap***

• uses a total ordering on the keys to organize them in a search tree

• The hash or comparison function is applied *only to the keys*

• The values associated with the keys are not hashed or compared.

**How are memory leaks possible in Java**

If any object variable is still pointing to some object which is of no use, then JVM will not garbage collect that object and object will remain in memory creating memory leak

**What are the differences between EJB and Java beans**

the main difference is Ejb componenets are distributed which means develop once and run anywhere. java beans are not distributed. which means the beans cannot be shared .

**What would happen if you say this = null**

this will give a compilation error as follows

cannot assign value to final variable this

**Will there be a performance penalty if you make a method synchronized? If so, can you make any design changes to improve the performance**

yes.the performance will be down if we use synchronization.

one can minimise the penalty by including garbage collection algorithm, which reduces the cost of collecting large numbers of short- lived objects. and also by using Improved thread synchronization for invoking the synchronized methods.the invoking will be faster.

**How would you implement a thread pool**

public class ThreadPool extends java.lang.Object implements ThreadPoolInt

This class is an generic implementation of a thread pool, which takes the following input

a) Size of the pool to be constructed

b) Name of the class which implements Runnable (which has a visible default constructor)

and constructs a thread pool with active threads that are waiting for activation. once the threads have finished processing they come back and wait once again in the pool.

This thread pool engine can be locked i.e. if some internal operation is performed on the pool then it is preferable that the thread engine be locked. Locking ensures that no new threads are issued by the engine. However, the currently executing threads are allowed to continue till they come back to the passivePool

**How does serialization work**

Its like FIFO method (first in first out)

**How does garbage collection work**

There are several basic strategies for garbage collection: reference counting, mark-sweep, mark-compact, and copying. In addition, some algorithms can do their job incrementally (the entire heap need not be collected at once, resulting in shorter collection pauses), and some can run while the user program runs (concurrent collectors). Others must perform an entire collection at once while the user program is suspended (so-called stop-the-world collectors). Finally, there are hybrid collectors, such as the generational collector employed by the 1.2 and later JDKs, which use different collection algorithms on different areas of the heap

**How would you pass a java integer by reference to another function**

Passing by reference is impossible in JAVA but Java support the object reference so.

Object is the only way to pass the integer by refrence.

**What is the sweep and paint algorithm**

The painting algorithm takes as input a source image and a list of brush sizes. sweep algo is that it computes the arrangement of n lines in the plane ... a correct algorithm,

**Can a method be static and synchronized**

no a static mettod can't be synchronised

**Do multiple inheritance in Java**

Its not possible directly. That means this feature is not provided by Java, but it can be achieved with the help of Interface. By implementing more than one interface.

**What is data encapsulation? What does it buy you**

The most common example I can think of is a javabean. Encapsulation may be used by creating 'get' and 'set' methods in a class which are used to access the fields of the object. Typically the fields are made private while the get and set methods are public.

dEncapsulation can be used to validate the data that is to be stored, to do calculations on data that is stored in a field or fields, or for use in introspection (often the case when using javabeans in Struts, for instance).

**What is reflection API? How are they implemented**

Reflection package is used mainlyfor the purpose of getting the class name. by using the getName method we can get name of the class for particular application .

Reflection is a feature of the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program.

**What are the primitive types in Java**

According to Java in a Nutshell, 5th ed

boolean, byte, char, short, long float, double, int

**Is there a separate stack for each thread in Java**

No

**What is heap in Java**

JAVA is fully Object oriented language. It has two phases first one is Compilation phase and second one is interpratation phase. The Compilation phase convert the java file to class file (byte code is only readable format of JVM) than Intepratation phase interorate the class file line by line and give the proper result.

**In Java, how are objects / values passed around**

In Java Object are passed by reference and Primitive data is always pass by value

**Do primitive types have a class representation**

Primitive data type has a wrapper class to present.

Like for int - Integer , for byte Byte, for long Long etc ...

**How all can you free memory**

With the help of finalize() method.

If a programmer really wants to explicitly request a garbage collection at some point, System.gc() or Runtime.gc() can be invoked, which will fire off a garbage collection at that time.

**Does java do reference counting**

It is more likely that the JVMs you encounter in the real world will use a tracing algorithm in their garbage-collected heaps

**What does a static inner class mean? How is it different from any other static member**

A static inner class behaves like any ``outer'' class. It may contain methods and fields.

It is not necessarily the case that an instance of the outer class exists even when we have created an instance of the inner class. Similarly, instantiating the outer class does not create any instances of the inner class.

The methods of a static inner class may access all the members (fields or methods) of the inner class but they can access only static members (fields or methods) of the outer class. Thus, f can access the field x, but it cannot access the field y.

**How do you declare constant values in java**

Using Final keyword we can declare the constant values How all can you instantiate final members Final member can be instantiate only at the time of declaration. null

**How is serialization implemented in Java**

A particular class has to implement an Interface java.io.Serializable for implementing serialization. When you have an object passed to a method and when the object is reassigned to a different one, then is the original reference lost No Reference is not lost. Java always passes the object by reference, now two references is pointing to the same object.

**What are the different kinds of exceptions? How do you catch a Runtime exception**

There are 2 types of exceptions.

1. Checked exception

2. Unchecked exception.

***Checked exception*** is catched at the compile time while *unchecked exception* is checked at run time.

1.Checked Exceptions : Environmental error that cannot necessarily be detected by testing; e.g. disk full, broken socket, database unavailable, etc.

**2. Unchecked exception**.

Errors : Virtual machine error: class not found, out of memory, no such method, illegal access to private field, etc.

Runtime Exceptions :Programming errors that should be detected in testing: index out of bounds, null pointer, illegal argument, etc.

Checked exceptions must be handled at compile time. Runtime exceptions do not need to be. Errors often cannot be

**What are the differences between JIT and HotSpot**

The Hotspot VM is a collection of techniques, the most significant of which is called "adaptive optimization.

The original JVMs interpreted bytecodes one at a time. Second-generation JVMs added a JIT compiler, which compiles each method to native code upon first execution, then executes the native code. Thereafter, whenever the method is called, the native code is executed. The adaptive optimization technique used by Hotspot is a hybrid approach, one that combines bytecode interpretation and run-time compilation to native code.

Hotspot, unlike a regular JIT compiling VM, doesn't do "premature optimization"

**What is a memory footprint? How can you specify the lower and upper limits of the RAM used by the JVM? What happens when the JVM needs more memory?**

when JVM needs more memory then it does the garbage collection, and sweeps all the memory which is not being used.

**What are the disadvantages of reference counting in garbage collection?**

An advantage of this scheme is that it can run in small chunks of time closely interwoven with the execution of the program. This characteristic makes it particularly suitable for real-time environments where the program can't be interrupted for very long. A disadvantage of reference counting is that it does not detect cycles. A cycle is two or more objects that refer to one another, for example, a parent object that has a reference to its child object, which has a reference back to its parent. These objects will never have a reference count of zero even though they may be unreachable by the roots of the executing program. Another disadvantage is the overhead of incrementing and decrementing the reference count each time. Because of these disadvantages, reference counting currently is out of favor.

**Is it advisable to depend on finalize for all cleanups**

The purpose of finalization is to give an opportunity to an unreachable object to perform any clean up before the object is garbage collected, and it is advisable.

**can we declare multiple main() methods in multiple classes. ie can we have each main method in its class in our program?**

YES

**Question: What is Collection API**

**Answer:** The Collection [API](http://www.roseindia.net/interviewquestions/corejava/java-util-packages.shtml) is a set of classes and interfaces that support [operation](http://www.roseindia.net/interviewquestions/corejava/java-util-packages.shtml) on collections of objects. These classes and [interfaces](http://www.roseindia.net/interviewquestions/corejava/java-util-packages.shtml) are more flexible, more powerful, and more regular than the vectors, arrays, and hashtables if effectively replaces.   
Example of classes: HashSet, HashMap, ArrayList, LinkedList, TreeSet and TreeMap.   
Example of interfaces: Collection, Set, List and Map.   
  
**Question: What are different types of collections**

**Answer:** A collection has no special order and does not reject duplicates   
A list is ordered and does not reject duplicates   
A set has no special order but rejects duplicates   
A map supports [searching](http://www.roseindia.net/interviewquestions/corejava/java-util-packages.shtml) on a key field, values of which must be unique   
  
**Question: Tell me something about Arrays**

**Answer:** Arrays are fast to access, but are inefficient if the number of elements grow and if you have to insert or delete an element   
  
**Question: Difference between ArrayList and Vector**

**Answer:** Vector methods are synchronized while ArrayList methods are not   
  
**Question: Iterator a Class or Interface? What is its use?**

**Answer:** Iterator is an interface which is used to step through the elements of a Collection   
  
**Question: Difference between Hashtable and HashMap**

**Answer:** Hashtable does not store null value, while HashMap does   
Hashtable is synchronized, while HashMap is not

**Collections**

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the Collections API?** |  |
| **A:** | The Collections API is a set of classes and interfaces that support operations on collections of objects. |  |
| **Q:** | **What is the List interface?** |  |
| **A:** | The List interface provides support for ordered collections of objects. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the Vector class?** |  |
| **A:** | The Vector class provides the capability to implement a growable array of objects. |  |
| **Q:** | **What is an Iterator interface?** |  |
| **A:** | The Iterator interface is used to step through the elements of a Collection . |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **Which java.util classes and interfaces support event handling?** |  |
| **A:** | The EventObject class and the EventListener interface support event processing. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the GregorianCalendar class?** |  |
| **A:** | The GregorianCalendar provides support for traditional Western calendars |  |
| **Q:** | **What is the Locale class?** |  |
| **A:** | The Locale class is used to tailor program output to the conventions of a particular geographic, political, or cultural region . |  |
|  | [ Received from Prasanna Inamanamelluri] | **TOP** |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the SimpleTimeZone class?** |  |
| **A:** | The SimpleTimeZone class provides support for a Gregorian calendar . |  |
|  | [ Received from Prasanna Inamanamelluri] | **TOP** |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the Map interface?** |  |
| **A:** | The Map interface replaces the JDK 1.1 Dictionary class and is used associate keys with values. |  |
| **Q:** | **What is the highest-level event class of the event-delegation model?** |  |
| **A:** | The java.util.EventObject class is the highest-level class in the event-delegation class hierarchy. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the Collection interface?** |  |
| **A:** | The Collection interface provides support for the implementation of a mathematical bag - an unordered collection of objects that may contain duplicates. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the Set interface?** |  |
| **A:** | The Set interface provides methods for accessing the elements of a finite mathematical set. Sets do not allow duplicate elements. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the typical use of Hashtable?** |  |
| **A:** | Whenever a program wants to store a key value pair, one can use Hashtable. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **I am trying to store an object using a key in a Hashtable. And some other object already exists in that location, then what will happen? The existing object will be overwritten? Or the new object will be stored elsewhere?** |  |
| **A:** | The existing object will be overwritten and thus it will be lost. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is the difference between the size and capacity of a Vector?** |  |
| **A:** | The size is the number of elements actually stored in the vector, while capacity is the maximum number of elements it can store at a given instance of time. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **Can a vector contain heterogenous objects?** |  |
| **A:** | Yes a Vector can contain heterogenous objects. Because a Vector stores everything in terms of Object. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **Can a ArrayList contain heterogenous objects?** |  |
| **A:** | Yes a ArrayList can contain heterogenous objects. Because a ArrayList stores everything in terms of Object. |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Q:** | **What is an enumeration?** |  |
| **A:** | An enumeration is an interface containing methods for accessing the underlying data structure from which the enumeration is obtained. It is a construct which collection classes return when you request a collection of all the objects stored in the collection. It allows sequential access to all the elements stored in the collection. |  |
|  |  | **TOP** |

|  |  |  |
| --- | --- | --- |
| **Q:** | **Considering the basic properties of Vector and ArrayList, where will you use Vector and where will you use ArrayList?** |  |
| **A:** | The basic difference between a Vector and an ArrayList is that, vector is synchronized while ArrayList is not. Thus whenever there is a possibility of multiple threads accessing the same instance, one should use Vector. While if not multiple threads are going to access the same instance then use ArrayList. Non synchronized data structure will give better performance than the synchronized one. |  |
|  |  |  |

|  |  |
| --- | --- |
| **Q:** | **Can a vector contain heterogenous objects?** |
| **A:** | Yes a Vector can contain heterogenous objects. Because a Vector stores everything in terms of Object. |

Q.Difference between HashMap and HashTable?

a)The key difference between the two is that access to the Hashtable is synchronized while access to the HashMap is not synchronized.   
b)HashMap permits null values as Key, while Hashtable doesn't.   
c)iterator in a HashMap is fail-safe(Fail-safe is relevant from the context of iterators.   
If an iterator has been created on a collection object and some other thread tries to modify the collection object "structurally",   
a concurrent modification exception will be thrown) while the enumerator of HashTable is not

Q.Difference between Vector and ArrayList?

Vectors are synchronized. Any method that touches the Vector's contents is thread safe.   
ArrayList, on the other hand, is unsynchronized, making them, therefore, not thread safe.   
With that difference in mind, using synchronization will incur a performance hit.   
So if you don't need a thread-safe collection, use the ArrayList don't use Vector.   
  
A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.   
Depending on how you use these classes, you could end up taking a large performance hit while adding new elements.   
If you don't know how much data you'll have, but you do know the rate at which it grows, Vector does possess a slight advantage since you can set the increment value.   
  
Both the ArrayList and Vector are good for retrieving elements from a specific position in the container or for adding and removing elements from the end of the container.   
All of these operations can be performed in constant time -- O(1).   
However, adding and removing elements from any other position proves more expensive -- linear to be exact: O(n-i), where n is the number of elements and i is the index of the element added or removed.   
For Example : ArrayList Has 5 elements and you want to add an element on 2nd position.   
Then arrayList add the element on 2nd position and shifted existing 2nd position element to 3rd , 3rd position element to 4th etc..   
For the case of remove just opposite.   
These operations are more expensive because you have to shift all elements at index i and higher over by one element

### **Q.Difference between ArrayList and LinkedList?**

ArrayList is good for adding and removing elements from the end of the container. All of these operations can be performed in constant time -- O(1).   
adding and removing elements from any other position proves more expensive -- linear to be exact: O(n-i), where n is the number of elements and i is the index of the element added or removed.   
For Example : ArrayList Has 5 elements and you want to add an element on 2nd position.   
Then arrayList add the element on 2nd position and shifted existing 2nd position element to 3rd , 3rd position element to 4th etc..   
  
LinkedList can add or remove an element at any position in constant time -- O(1). Indexing an element is a bit slower -- O(i) where i is the index of the element in case of LinkedList.   
  
Traversing an ArrayList is also easier since you can simply use an index instead of having to create an iterator.   
The LinkedList creates an internal object for each element inserted. So you have to be aware of the extra garbage being created.   
  
An ArrayList is a List implementation backed by a Java array.   
With a LinkedList, the List implementation is backed by a doubly linked list data structure

**Question: You are planning to do an indexed search in a list of objects. Which of the two Java collections should you use: ArrayList or LinkedList?**

**Answer:** ArrayList