1. [What are Collection related features in Java 8?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "java8-collections)

Ans: Stream Collection Types ([java.util.stream](http://download.java.net/jdk8/docs/api/java/util/stream/package-summary.html))

A stream is a iterator that allows a single run over the collection it is called on. Along with Lambdas, this is another noteworthy feature to watch out for. You can use streams to perform functional operations like filer or map/reduce over collections which can be streamed as individual elements using Stream objects. Streams can run sequentially or parallely as desired. The parallel mode makes use of fork/join framework and can leverage power of multiple cores.

1. [What is Java Collections Framework? List out some benefits of Collections framework?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "java-collections-framework)

Ans: The **Java collections framework** (JCF) is a set of classes and interfaces that implement commonly reusable **collection** data structures. Although referred to as a **framework**, it works in a manner of a library. The JCF provides both interfaces that define various **collections** and classes that implement them.

Some of the benefits of collections framework are:

* Reduced development effort by using core collection classes rather than implementing our own collection classes.
* Code quality is enhanced with the use of well tested collections framework classes.
* Reduced effort for code maintenance by using collection classes shipped with JDK.
* Reusability and Interoperability

1. [What is the benefit of Generics in Collections Framework?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "generics-in-collections)

Ans: Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don’t need to use casting and **instanceof** operator. It also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

1. [What are the basic interfaces of Java Collections Framework?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collections-framework-interfaces)

Ans: Set,List,Map,Oueue,Tree.

1. [Why Collection doesn’t extend Cloneable and Serializable interfaces?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collection-cloneable-serializable)

Ans: Because if it did, that would require all Collection implementations to be Cloneable and Serializable, which is more restrictive than needed. *Implementations* frequently also implement those interfaces, but it's not for the Collection interface to require it.

1. [Why Map interface doesn’t extend Collection interface?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "map-vs-collection)

Ans: Because **Map** is three **collections**: Keys, values and key-value pairs. It's **part** of the**collection** framework but it doesn't implement the java.util.**Collection interface**. It's a different branch of the hierarchy. If you want, you can view it on the same level of the hierarchy as the **Collection interface**

1. [What is an Iterator?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterator-interface)

Ans:Public interface **Iterator**<E> An **iterator** over a collection. **Iterator** takes the place of Enumeration in the **Java** Collections Framework. **Iterators** differ from enumerations in two ways: **Iterators** allow the caller to remove elements from the underlying collection during the **iteration** with well-defined semantics.

1. [What is difference between Enumeration and Iterator interface?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterator-vs-enumeration)

Ans:**Enumeration** and **Iterator** are two interfaces in java.util package which are used to traverse over the elements of a Collection object. ... But using **Iterator**, you can also remove an element while traversing the Collection. This is the one major **difference between Enumeration** and **Iterator** in java.

1. [Why there is not method like Iterator.add() to add elements to the collection?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterator-add)

Ans:The sole purpose of an Iterator is to enumerate through a collection. All collections contain the add()method to serve your purpose. There would be no point in adding to an Iterator because the collection may or may not be ordered (in the case of a HashSet).

1. [Why Iterator don’t have a method to get next element directly without moving the cursor?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterator-next)

Ans:Because object store in the memory has reference address and to get value of any object you need to provide information to java. getNext() could have written but according to java API cursor always point to middle and calling next() will move it to the current position.

1. [What is different between Iterator and ListIterator?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterator-vs-listiterator)

Ans:1) **Iterator** is used for traversing List and Set both. We can use **ListIterator** to traverse List only, we cannot traverse Set using **ListIterator**. 2) We can traverse in only forward direction using **Iterator**. Using **ListIterator**, we can traverse a List in both the directions (forward and Backward).

1. [What are different ways to iterate over a list?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "iterate-a-list)

Ans: 6 ways:

For Loop

1. Enhanced For Loop
2. While Loop
3. Iterator
4. Collections stream() util (Java8 feature)
5. ListIterator
6. [What do you understand by iterator fail-fast property?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "fail-fast-iterator)

When a problem occurs, a **fail**-**fast** system **fails** immediately. In Java, we can find this behavior with **iterators**. Incase, you have called **iterator** on a collection object, and another thread tries to modify the collection object, then concurrent modification exception will be thrown. This is called **fail**-**fast**.

1. [What is difference between fail-fast and fail-safe?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "fail-fast-vs-fail-safe)

**Fail**-**Safe iterators** don't throw any exceptions if the collection is modified while **iterating** over it. Because, they iterate on the clone of the collection not on the actual collection. ... **Iterator** returned by ConcurrentHashMap is a **fail**-**safe iterator**.

1. **[How to avoid ConcurrentModificationException while iterating a collection?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "avoid-concurrentmodificationexception)**

To Avoid ConcurrentModificationException in multi-threaded environment

1. You can convert the list to an array and then iterate on the array. ...
2. You can lock the list while iterating by putting it in a synchronized block. ...
3. If you are using JDK1.5 or higher then you can use ConcurrentHashMap and CopyOnWriteArrayList classes.
4. **Why there are no concrete** **implementations of Iterator interface?**

**Answer**: Implementation could have done for iterator interface but by design Collection interface extends Iterable interface which give one method interator() and return Iterator interface. Designer given flexibility to all of its concrete classes (Child classes) to implement rule base on their need.

For example: All collection classes in java.util package are fail fast and java.util.concurrent package are fail safe.

1. **[What is UnsupportedOperationException?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "unsupportedoperationexception)**

[public class](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashmap-working) **UnsupportedOperationException**

extends [RuntimeException](https://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html)

Thrown to indicate that the requested operation is not supported.

This class is a member of the [Java Collections Framework](https://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html).

1. **[How HashMap works in Java?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashmap-working)**

[HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashcode-equals)O(1)provided we know the key. In hashing, hash functions are used to link key and value in HashMap. Objects are stored by calling put(key, value)method of HashMap and retrieved by calling get(key)method.  
  
Read more: [http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz4YaaU82Zm](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html" \l "ixzz4YaaU82Zm)

1. **[What is the importance of hashCode() and equals() methods?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashcode-equals)**

[Java.lang.Object has methods called hasCode() and equals(). These methods play a significant role in the real time application. However its use is not always common to all applications.](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "map-key)

**hashCode()**

As you know this method provides the has code of an object. Basically the default implementation of hashCode() provided by Object is derived by mapping the memory address to an integer value.

**equals()**

This particular method is used to make equal comparison between two objects. There are two types of comparisons in Java. One is using “= =” operator and another is “equals()”.

1. **[Can we use any class as Map key?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "map-key)**

**[Answer](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "map-collection-views)**: Yes we can use class as key of HashMap. HashMap store data based on hashing algorithm so before using class as key below point should be consider:

* Class should be immutable and if don’t keep it then hash code value can be change and when you try to pull value from hash map then possibility is you won’t get same value which you will see in below java class example
* Class must override hashcode() and equals() method

1. **[What are different Collection views provided by Map interface?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "map-collection-views)**

**[Answer](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashmap-vs-hashtable)**: Map is object which maps keys to its values. Map can’t contain duplicate keys. Each key will map to one value. Map interface includes methods which can be used for general operations i.e.get, put, containsKey, containsValue, remove, size, and empty for bulk operations i.e. putAll and clear.

It provides **three collection views** methods which allow Map to be viewed as Collection in below ways:

* **Keyset**: This method provides Set of the keys contained in Map.
* **entrySet**: This method provides Collection of values contained in Map. Please note basically this Collection is not Set because we can have multiple keys that is map to same value.
* **Values**: This method provides Set of key value pairs contained in Map. Map interface provides small nested interface that is called Map.Entry and type of the elements in this Set.

1. **[What is difference between HashMap and Hashtable?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashmap-vs-hashtable)**

[Hashtable](http://java.sun.com/javase/7/docs/api/java/util/Hashtable.html) is [synchronized](http://stackoverflow.com/questions/1085709/what-does-synchronized-mean), whereas [HashMap](http://java.sun.com/javase/7/docs/api/java/util/HashMap.html) is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.

Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.

One of HashMap's subclasses is [LinkedHashMap](http://java.sun.com/javase/7/docs/api/java/util/LinkedHashMap.html), so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

1. **[How to decide between HashMap and TreeMap?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "hashmap-vs-treemap)**

**[When to Prefer TreeMap over HashMap](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "arraylist-vs-vector)**1. Sorted elements are required instead of unordered  elements. The sorted list return by TreeMap is always in ascending order.2. [TreeMap uses Red-Black algorithm](http://javahungry.blogspot.com/2014/06/how-treemap-works-ten-treemap-java-interview-questions.html)underneath  to sort out the elements . When one need to perform read/write operations frequently , then TreeMap is a good choice.

1. **[What are similarities and difference between ArrayList and Vector?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "arraylist-vs-vector)**

**Similarities between ArrayList and Vector**

* Both ArrayList and Vector are dynamic data structures.
* Both ArrayList and Vector maintain elements in the order they are inserted.
* Both ArrayList and Vector allows null values and duplicate values.
* When items are deleted from ArrayList and Vector, their size gets shrinked automatically.
* When an overflow of insertion occurs in ArrayList and Vector, their size will grow automatically.
* Both ArrayList and Vector are present in java.util package.
* Both use array as the internal data structure.

**Difference between ArrayList and Vector**

|  |  |  |
| --- | --- | --- |
| **Feature** | **ArrayList** | **Vector** |
| **Legacy** | ArrayList was introduced as part of collection framework in Java 1.2. | Vector was introduced in JDK 1.0. Vector was later added to collection framework in Java 1.2. |
| **Synchronization** | ArrayList is not synchronized. ArrayList can be accessed by multiple threads during the same time. | Vector supports synchronization and multithreading. A Vector can be accessed by only one thread at a point of time. |
| **Resize while growing** | When a new item has to be added to the ArrayList whose size limit is reached, the ArrayList size will be increased by half of its original size. | When a new item has to be added to the Vector whose size limit is reached, the Vector size will be doubled. |
| **Increment size** | In ArrayList, the increment size cannot be set. | In Vector, the increment size can be defined using setSize() method. |
| **Performance** | ArrayList provides high performance as it is not synchronized. | Vectors are poor in performance as they are synchronized. When multiple threads are accessing the Vector at the same point of time, Vector creates a lock for the first thread and other threads are made to wait until the lock is released. This leads to poor performance. |
| **Enumeration and fail fast** | ArrayList returns an iterator or listIterator that are fail fast. ArrayList does not return any enumeration. | Vector returns an enumeration that is not fail fast. However, Vector can also return an iterator that is fail fast. |
| **Method to retrieve an element** | An element can be retrieved from the ArrayList using get(index) method. | An element can be retrieved from the Vector using elementAt(index) method. |
| **Method to replace an element** | An element can be replaced in an ArrayList using add(index, object) method. | An element can be replaced in a Vector usin |

1. **[What is difference between Array and ArrayList? When will you use Array over ArrayList?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "array-vs-arraylist)**

*[Difference between array and arraylist in java](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "arraylist-vs-linkedlist)*include eight points namely Resizable, Performance, Traversal ,Primitives , Length , Type-Safety, Adding elements , Multi-dimensional.

**Only when the size of elements are know.**

1. **[What is difference between ArrayList and LinkedList?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "arraylist-vs-linkedlist)**

[Use LinkedList in Java if:](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collection-random-access)

1) Your application can live without Random access. Because if you need nth element in LinkedList you need to first traverse up to nth element O(n) and than you get data from that node.

2) Your application is more insertion and deletion driver and you insert or remove more than retrieval. Since insertion or

removal doesn't involve resizing its much faster than ArrayList.

That’s all on **difference between ArrayList and LinkedList in Java**. Use ArrayList in Java for all there situation where you need a **non-synchronized index based access**. ArrayList is fast and easy to use, just try to minimize array resizing by constructing arraylist with proper initial size.

Read more: [http://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html#ixzz4Yae8zcgk](http://javarevisited.blogspot.com/2012/02/difference-between-linkedlist-vs.html" \l "ixzz4Yae8zcgk)

1. **[Which collection classes provide random access of it’s elements?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collection-random-access)**

**[ArrayList](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "enumset)**, **HashMap**, TreeMap, **Hashtable**classes

1. **[What is EnumSet?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "enumset)**

[The java.util.](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "thread-safe-collections)**EnumSet**class is a specialized Set implementation for use with enum types.Following are the important points about **EnumSet**: All of the elements in an**enum set**must come from a single enum type that is specified, explicitly or implicitly, when the set is created.

1. **[Which collection classes are thread-safe?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "thread-safe-collections)**

Do you notice that all the basic collection classes - **ArrayList**, **LinkedList**,**HashMap**, HashSet, TreeMap, TreeSet, etc - all are not synchronized? In fact, all collection classes (except Vector and **Hashtable**) in the java.util package are not thread-safe

1. **[What are concurrent Collection Classes?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "concurrent-collection-classes)**

**[java.util.concurrent collections.](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "blocking-queue)**

* ConcurrentHashMap. ...
* CopyOnWriteArrayList and CopyOnWriteArraySet. ...
* BlockingQueue. ...
* Deque and BlockingDeque. ...
* ConcurrentSkipListMap and ConcurrentSkipListSet.

1. **[What is BlockingQueue?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "blocking-queue)**

[A](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "queue-vs-stack)**blocking queue**is a **queue**that blocks when you try to dequeue from it and the**queue**is empty, or if you try to enqueue items to it and the **queue**is already full. A thread trying to dequeue from an empty **queue**is **blocked**until some other thread inserts an item into the **queue**.

1. **[What is Queue and Stack, list their differences?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "queue-vs-stack)**

**STACK:**

1. Stack is defined as a list of element in which we can insert or delete elements only at the top of the stack.
2. The behaviour of a stack is like a Last-In First-Out(LIFO) system.
3. Stack is used to pass parameters between function. On a call to a function, the parameters and local variables are stored on a stack.
4. High-level programming languages such as Pascal, c, etc. that provide support for recursion use the stack for bookkeeping. Remember in each recursive call, there is a need to save the current value of parameters, local variables, and the return address (the address to which the control has to return after the call).

**QUEUE:**

1. Queue is a collection of the same type of element. It is a linear list in which insertions can take place at one end of the list,called ***rear*** of the list, and deletions can take place only at other end, called the ***front*** of the list
2. The behaviour of a queue is like a First-In-First-Out (FIFO) system.
3. **[What is Collections Class?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collections-class)**

**[Java](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "compable-comparator)**Collection simply means a single unit of objects. **Java**Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (**ArrayList**, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

1. **[What is Comparable and Comparator interface?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "compable-comparator)**

**[Comparable](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "comparable-vs-comparator)**

A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface in order to be able to compare its instances.

**Comparator**

A comparator object is capable of comparing two different objects. The class is not comparing its instances, but some other class’s instances. This comparator class must implement the java.util.Comparator interface.

1. **[What is difference between Comparable and Comparator interface?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "comparable-vs-comparator)**

|  |  |
| --- | --- |
| Comparable | Comparator |
| 1) Comparable provides single sorting sequence. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides multiple sorting sequence. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| 2) Comparable affects the original class i.e. actual class is modified. | Comparator doesn't affect the original class i.e. actual class is not modified. |
| 3) Comparable provides compareTo() method to sort elements. | Comparator provides compare() method to sort elements. |
| 4) Comparable is found in java.lang package. | Comparator is found in java.util package. |
| 5) We can sort the list elements of Comparable type by Collections.sort(List) method. | We can sort the list elements of Comparator type by Collections.sort(List,Comparator) method. |

1. **[How can we sort a list of Objects?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "sort-objects)**

[To sort an Object by its property, you have to make the Object implement the](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "unmodifiable-collections)**Comparable**interface and override the **compareTo()**method.

1. **[While passing a Collection as argument to a function, how can we make sure the function will not be able to modify it?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "unmodifiable-collections)**

[We can create a read-only collection using](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "synchronized-collections)Collections.unmodifiableCollection(Collection c)method before passing it as argument, this will make sure that any operation to change the collection will throw UnsupportedOperationException.

1. **[How can we create a synchronized collection from given collection?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "synchronized-collections)**

We can use Collections.synchronizedCollection(Collection c)to get a synchronized (thread-safe) collection backed by the specified collection.

1. **[What are common algorithms implemented in Collections Framework?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collection-algorithms)**

Java Collections Framework provides algorithm implementations that are commonly used such as sorting and searching. Collections class contain these method implementations. Most of these algorithms work on List but some of them are applicable for all kinds of collections.Some of them are sorting, searching, shuffling, min-max values.

1. **[What is Big-O notation? Give some examples?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "big-o-notation-performance)**

The Big-O notation describes the performance of an algorithm in terms of number of elements in a data structure. Since Collection classes are actually data structures, we usually tend to use Big-O notation to chose the collection implementation to use based on time, memory and performance.

Example 1: ArrayList get(index i) is a constant-time operation and doesn’t depend on the number of elements in the list. So it’s performance in Big-O notation is O(1).  
Example 2: A linear search on array or list performance is O(n) because we need to search through entire list of elements to find the element.

1. **[What are best practices related to Java Collections Framework?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "collection-best-practices)**

[Chosing the right type of collection based on the need, for example if size is fixed, we might want to use Array over ArrayList. If we have to iterate over the Map in order of insertion, we need to use TreeMap. If we don’t want duplicates, we should use Set.](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "java-priority-queue)

* Some collection classes allows to specify the initial capacity, so if we have an estimate of number of elements we will store, we can use it to avoid rehashing or resizing.
* Write program in terms of interfaces not implementations, it allows us to change the implementation easily at later point of time.
* Always use Generics for type-safety and avoid ClassCastException at runtime.
* Use immutable classes provided by JDK as key in Map to avoid implementation of hashCode() and equals() for our custom class.
* Use Collections utility class as much as possible for algorithms or to get read-only, synchronized or empty collections rather than writing own implementation. It will enhance code-reuse with greater stability and low maintainability.

1. **[What is Java Priority Queue?](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "java-priority-queue)**

[PriorityQueue is an unbounded queue based on a priority heap and the elements are ordered in their natural order or we can provide](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "generics-sub-typing)[Comparator](http://www.journaldev.com/780/comparable-and-comparator-in-java-example)for ordering at the time of creation. PriorityQueue doesn’t allow null values and we can’t add any object that doesn’t provide natural ordering or we don’t have any comparator for them for ordering. Java PriorityQueue is not [thread-safe](http://www.journaldev.com/1061/thread-safety-in-java)and provided O(log(n)) time for enqueing and dequeing operations. Check this post for [java priority queue example](http://www.journaldev.com/1642/java-priority-queue-priorityqueue-example).

1. **[Why can’t we write code as](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "generics-sub-typing)List<Number> numbers = new ArrayList<Integer>();?**

[Generics doesn’t support sub-typing because it will cause issues in achieving type safety. That’s why List<T> is not considered as a subtype of List<S> where S is the super-type of T.](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "generics-array)

1. **[Why can’t we create generic array? or write code as](http://www.journaldev.com/1330/java-collections-interview-questions-and-answers" \l "generics-array)List<Integer>[] array = new ArrayList<Integer>[10];**

We are not allowed to create generic arrays because array carry type information of it’s elements at runtime. This information is used at runtime to throw ArrayStoreException if elements type doesn’t match to the defined type. Since generics type information gets erased at runtime by Type Erasure, the array store check would have been passed where it should have failed.