**1. Difference between HashMap and HashTable.**

* HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.
* HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.
* HashMap is generally preferred over HashTable if thread synchronization is not needed
* HashMap is a new class introduced in JDK1.2, whereas HashTable is a legacy class.
* HashMap is fast and HashTable is slow.
* HashMap is traversed by Iterator whereas HashTable can be traversed by Enumerator and Iterator.
* HashMap inherits AbstractMap class and HashTable inherits Dictionary class.

**2. Intern method:-**

The intern() method of the string method returns a canonical representation of the string object.

For any two strings s and t, s.intern()==t.intern() is true if and only if s.equals(t) is true. This method is used to return the string from the string pool.

**3. Why Checked Expections are Compile Time Exceptions?**

Checked Exceptions are compile time exceptions since these exceptions are checked at compile time. It should handle the exception

Using try-catch block or it should declare the exception using throws keyword. Otherwise the program will give compilation error.

**4. Difference between Linked List and Vector?**

* Linked list is implemented as a double linked list. Vector is similar with arraylist.
* Its Performance on add and remove is better than arraylist but worse on get and set methods. Vector each time doubles its arraysize.
* Linked list can act as a list cost. Vector is slow because it is synchronized.
* Linked list is better for manipulating data. Vector can use Iterator interface or Enumerator Interface to transfer elements.

**5. What are Vector, Queue and Dequeue ?**

**Vector class:-**

The Vector class implements a growable array of objects.

Vectors is fully compatible with collections.

* Vector implements a dynamic array that means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index
* They are very similar to ArrayList but Vector is synchronised and have some legacy method which collection framework does not contain.
* It extends AbstractList and implements List interfaces.

**Queue:-**

* The Queue interface is available in java.util package and extends the Collection interface.
* The queue collection is used to hold the elements about to be processed
* Provides various operations like the insertion, removal etc.
* It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of list (FIFO principle).
* Being an interface the queue needs a concrete class for the declaration and the most common classes are the [PriorityQueue](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/" \t "_blank) and [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/" \t "_blank) in Java.
* Both the implementations are not thread safe.
* PriorityBlockingQueue is one alternative implementation if thread safe implementation is needed.

Methods in queue:

* add()- This method is used to add elements at the tail of queue.
* peek()- This method is used to view the head of queue without removing it. It returns Null if the queue is empty.
* element()- This method is similar to peek(). It throws NoSuchElementException when the queue is empty.
* remove()- This method removes and returns the head of the queue. It throws NoSuchElementException when the queue is empty.
* poll()- This method removes and returns the head of the queue. It returns null if the queue is empty.
* size()- This method return the no. of elements in the queue.

**Dequeue :-**

* The java.util.Deque interface is a subtype of the [java.util.Queue](https://www.geeksforgeeks.org/queue-interface-java/) interface.
* The Deque is related to the double-ended queue that supports addition or removal of elements from either end of the data structure.
* It can be used as a [queue (first-in-first-out/FIFO)](https://www.geeksforgeeks.org/queue/) or as a [stack (last-in-first-out/LIFO)](https://www.geeksforgeeks.org/stack/).
* These are faster than Stack and LinkedList.

Methods of Dequeue

* a[dd(element)](https://www.geeksforgeeks.org/deque-add-method-in-java/" \t "_blank): Adds an element to the tail.
* [addFirst(element)](https://www.geeksforgeeks.org/deque-addfirst-method-in-java-with-examples/): Adds an element to the head.
* [addLast(element)](https://www.geeksforgeeks.org/deque-addlast-method-in-java/): Adds an element to the tail.
* [offer(element)](https://www.geeksforgeeks.org/deque-offer-method-in-java/): Adds an element to the tail and returns a boolean to explain if the insertion was successful.
* [offerFirst(element)](https://www.geeksforgeeks.org/deque-offerfirst-method-in-java/): Adds an element to the head and returns a boolean to explain if the insertion was successful.
* [offerLast(element)](https://www.geeksforgeeks.org/deque-offerlast-method-in-java/): Adds an element to the tail and returns a boolean to explain if the insertion was successful.
* [iterator()](https://www.geeksforgeeks.org/deque-iterator-method-in-java/): Returns an iterator for this deque.
* [descendingIterator()](https://www.geeksforgeeks.org/deque-descendingiterator-method-in-java/): Returns an iterator that has the reverse order for this deque.