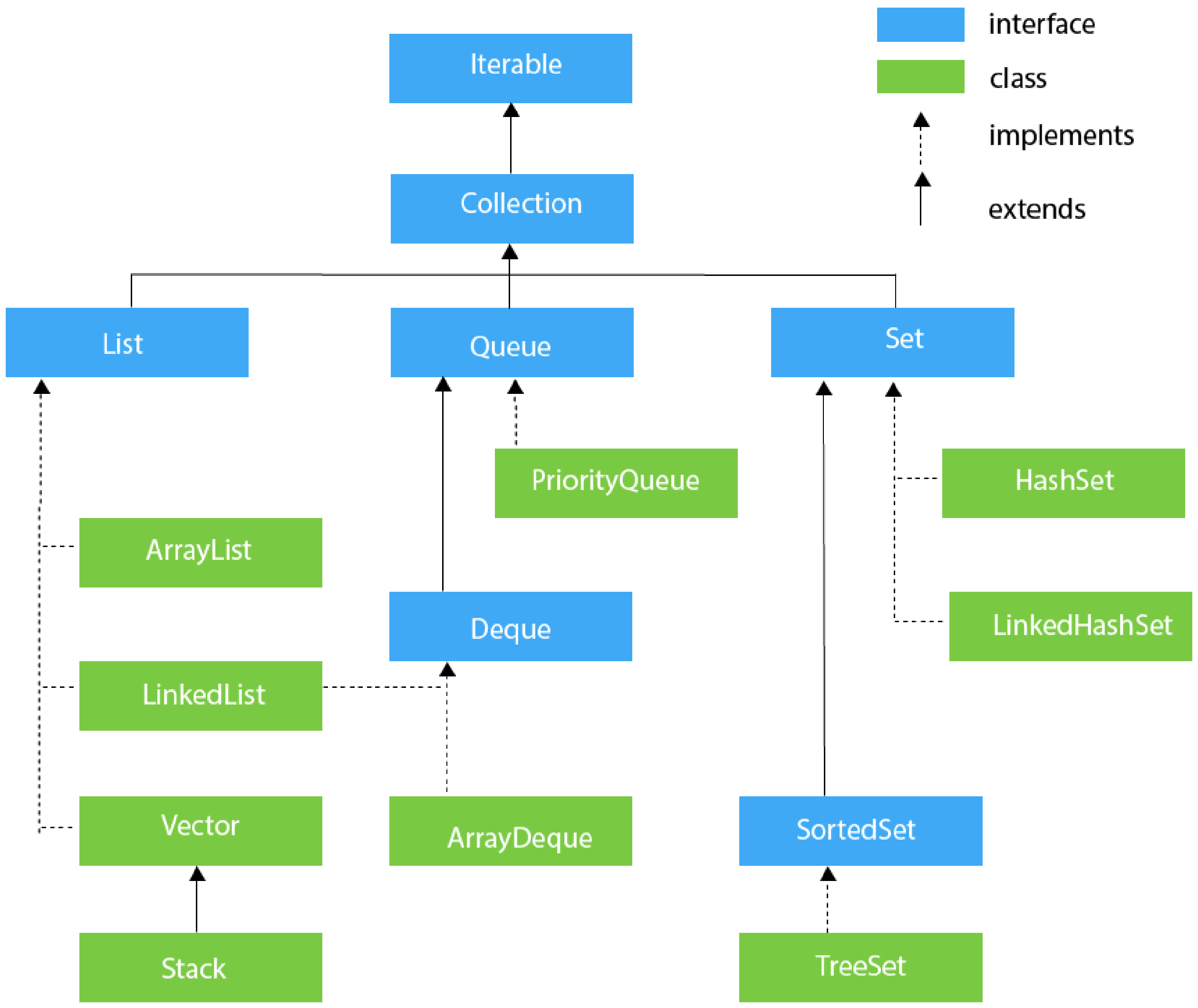
1. Draw Collections Framework Class Diagram
Ans:



2. What is HashMap and Map?
Ans:
Hashmap :
HashMap<K, V> is a part of Java's collection since Java 1.2. This class is found in java.util package. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer). One object is used as a key (index) to another object (value). If you try to insert the duplicate key, it will replace the element of the corresponding key.
HashMap is similar to HashTable, but it is unsynchronized. It allows to store the null keys as well, but there should be only one null key object and there can be any number of null values. This class makes no guarantees as to the order of the map. To use this class and its methods, you need to import java.util.HashMap package or its superclass.

Map : The map interface is present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys. Some common scenarios are as follows:

- A map of error codes and their descriptions.
- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value)

3. Difference between HashMap and HashTable? Can we make hashmap synchronized?
Ans:

S. No.	Hashmap	Hashtable
1.	No method is synchronized.	Every method is synchronized.
2.	Multiple threads can operate simultaneously and hence hashmap's object is not thread-safe.	At a time only one thread is allowed to operate the Hashtable's object. Hence it is thread-safe.
3.	Threads are not required to wait and hence relatively performance is high.	It increases the waiting time of the thread and hence performance is low.
4.	Null is allowed for both key and value.	Null is not allowed for both key and value. Otherwise, we will get a null pointer exception.
5.	It is introduced in the 1.2 version.	It is introduced in the 1.0 version.
6.	It is non-legacy.	It is a legacy.

Can we make hashmap synchronized?
Ans: HashMap can be synchronized using the Collections.synchronizedMap() method.

4. Difference between Vector and ArrayList?

Array List	Vector
1) ArrayList is not synchronized.	Vector is synchronized.
2) ArrayList increments 50% of current array size if the number of elements exceeds from its capacity.	Vector increments 100% means doubles the array size if the total number of elements exceeds than its capacity.
3) ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a legacy class.
4) ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until current thread releases the lock of the object.
5) ArrayList uses the Iterator interface to traverse the elements.	A Vector can use the Iterator interface or Enumeration interface to traverse the elements.

5. What is an Iterator?
Ans: An Iterator is an object that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

- To use an Iterator, you must import it from the java.util package.

6. List vs Set vs Map. Purposes and definitions.
Ans:
List :
The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java 5.
Purposes : List implementation classes are Array List, LinkedList

Set : The set interface is present in java.util package and extends the Collection interface is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the

mathematical set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.
Purposes : Set implementation classes are HashSet, LinkedHashSet, and TreeSet.

Map : The map interface is present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.Maps are perfect to use for key-value association mapping such as dictionaries.
Purposes : Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap.

7. Pros and cons of ArrayList and LinkedList

ArrayList	**LinkedList**
1) ArrayList internally uses a dynamic array to store the elements	LinkedList internally uses a doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4) ArrayList is better for storing and accessing data	LinkedList is better for manipulating data.
5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.

6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	**LinkedList implements the doubly linked list of the list interface.**

8. TreeSet vs LinkedHashSet

LinkedHashSet	**TreeSet**
LinkedHashSet uses LinkedHashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.
LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted.	TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order.
The performance of LinkedHashSet is between HashSet and TreeSet. It's performance is almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements.	TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations.
LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations.	TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations.

LinkedHashSet also uses equals() and hashCode() methods to compare the elements.	TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn’t use equals() and hashCode() methods for comparision of elements.
LinkedHashSet also allows maximum one null element.	TreeSet doesn’t allow even a single null element. If you try to insert null element into TreeSet, it throws NullPointerException.

9. What are relationships between equals and hash codes?
Ans: If two objects are equal(according to equals() method) then the hashCode() method should return the same integer value for both the objects. But, it is not necessary that the hashCode() method will return the distinct result for the objects that are not equal (according to equals() method).

10. What are the advantages of ArrayList over arrays?
Ans:
1. Implementation
The array is a native programming component or data structure but ArrayList is a class from Java Collections framework, an API. In fact, ArrayList is internally implemented using an array. Since ArrayList is a class, it holds all properties of a class e.g. you can create objects and call methods but even though the array is an object in Java it doesn’t provide any method. It just exposes a length attribute to give you the length of the array, which is constant.

2. Performance
Since ArrayList is based upon array, you would assume that it provides the same performance as an array. This is true at some extent but because of extra functionality ArrayList provides there is some difference in performance between ArrayList and array, mainly in terms of memory usage and CPU time.

For index-based access, both ArrayList and array provides O(1) performance but add can beO(logN) in ArrayList if adding a new element triggers resize, as it involves creating a new array in background and copying elements from the old array to new array. The memory requirement for ArrayList is also more than an array for storing the same number of objects e.g. an int[] will take less memory to store 20 int variables than an ArrayList because of object metadata overhead on both ArrayList and wrapper class**.**

3. Type Safety
ArrayList is type safe because it supports generics which allows the compiler to check if all objects stored in ArrayList are of the correct type. On the other hand, the array doesn’t support Generics. Which means

compile time checking is not possible but array provides runtime type checking by throwing ArrayStoreException if you try to store an incorrect object into array e.g. storing a String into an int array.

4. Flexibility
Flexibility is the single most important thing that separates array and ArrayList. In short, ArrayList is more flexible than a plain native array because it's dynamic. It can grow itself when needed, which is not possible with the native array. ArrayList also allows you to remove elements which are not possible with native arrays. By remove, we mean not just assigning null to the corresponding index but also copying rest of elements one index down, which ArrayList automatically does for you. You can learn more about removing objects from ArayList in my article difference between clear() and removeAll().

5. Primitives
If you first start using ArrayList then you will realize that you cannot store primitives on ArrayList. This is a key difference between array and ArrayList because array allows storing both primitives and object. For example int[] numbers are valid but ArrayList of int is not valid. how do you deal with this problem?

Suppose you want to store int primitives into ArrayList than how do you that? Well, you can use the wrapper class. This is one of the reasons why wrapper class was introduced in Java. So if you want to store int 2 into ArrayList just put it, autoboxing will do the rest. Btw, this difference is not so obvious from Java 5 onwards because of auto-boxing as you will see that ArrayList.add(21) is perfectly valid and works.

6. Generics
One more significant difference between an ArrayList and an array is that the former supports Generic but the latter doesn't. Since an array is of covariant type, you can use Generics with them. This means it's not possible for a compiler to check the type-safety of an array at compile time but they can verify type-safety of Array. So how do you deal with this problem while writing a type-safe class in Java? Well, you can use the technique shown in Effective Java, where you can declare an array like E[] and later use type casting.

7. Iteration
ArrayList provides more ways for iteration i.e. accessing all elements one by one than an array. You can only use loop e.g. for, while, enhanced for loop and do-while to iterate over an array but you can also use Iterator and ListIterator class to iterate over ArrayList. See here to learn different ways to iterate over ArrayList in Java.

8. Supported Operations
Since ArrayList is backed by an array internally, it exposes the operation which is possible with an array but given its dynamic nature it also added operation which is not possible with native array e.g. you can store elements in both array and ArrayList, but only ArrayList allows you to remove an element. Though you can simulate that with an array by assigning null to respective index, it won't be like remove unless you also move all element above that index in the array to one level down.

Both ArrayList and array provide ways to retrieve an element e.g. get() method of ArrayList uses an index to get an element from array e.g. version[0] will return the first element.

ArrayList also provides an operation to clear and reuse e.g. clear() and removeAll(), the array doesn't provide that but you can loop over Array and assign each index null to simulate that.

9. Size() vs length
Array only provides a length attribute which tells you the number of slots in the array i.e. how many elements it can store, it doesn't provide you any method to find out how many are filled and how many slots are empty i.e. the current number of elements. While ArrayList does provides a size() method which tells a number of objects stored in ArrayList at a given point of time. The size() is always different than length, which is also the capacity of ArrayList. If you want to know more, I suggest you read the difference between size() and length in ArrayList article.

10. Dimension
Another significant difference between an array and an ArrayList is that array can be multi-dimensional e.g. you can have a two-dimensional array or a three-dimensional array, which makes it a really special data structure to represent matrices and 2D terrains. On the other hand, ArrayList doesn't allow you to specify dimension. See this tutorial learn more about how to use a multi-dimensional array in Java.

Here is the nice slide highlighting all important difference between Array and ArrayList in Java:

1) You can define ArrayList as re-sizable array. Size of the ArrayList is not fixed. ArrayList can grow and shrink dynamically.
2) Elements can be inserted at or deleted from a particular position
3) ArrayList class has many methods to manipulate the stored objects
4) If generics are not used, ArrayList can hold any type of objects
5) You can traverse an ArrayList in both the directions – forward and backward using ListIterator.

11. Principle of storing data in a hashtable
Ans: Hash table intrinsically contains a slot/bucket in which the storage of key and value pair. It uses the key's hash code to discover which bucket the key/value of a set should map. To find an item in a list you do the first approach i.e. linear search this involves checking each item, it will take more time. Suppose you have retrieved a value and its index number of an array, then you will look up a value very quickly. In fact, a time will be taken less if you know the index number is independent of the size & position of the array. But how do you know which element of the array contains the value you are looking for. The answer is by calculating the value itself the index number is somewhat related to data.

12. Differences between Hashtable, ConcurrentHashMap and Collections.synchronizedMap()
Ans:

ConcurrentHashMap	**SynchronizedMap**	**HashTable**
We will get thread safety without locking the total map object just with a bucket level lock.	We will get thread safety by locking the whole map object.	We will get thread safety by locking the whole map object
At a time multiple threads are allowed to operate on map objects safely.	At a time only one thread is allowed to perform any operation on a map object.	At a time one thread is allowed to operate on a map object.
Read operation can be performed without lock but write operation can be performed with bucket level lock.	Every read and write operations required total map object	Every read and write operations required total map object
While one thread iterating map objects the other thread is allowed to modify the map and won't get ConcurrentModificationException.	While one thread iterating map object the other threads are not allowed to modify the map otherwise we will get ConcurrentModificationException	While one thread iterating map object the other threads are not allowed to modify the map otherwise we will get ConcurrentModificationException
Iterator of ConcurrentHashMap is fail-safe and won't raise ConcurrentModificationException	Iterator of SynchronizedMap is fail-fast and it will raise ConcurrentModificationException	Iterator of HashTable is fail-fast and it will raise ConcurrentModificationException
Null is not allowed for both keys and values.	Null is allowed for both keys and values	Null is not allowed for both keys and values.
Introduce in java 1.5version	Introduce in java 1.2 version	Introduce in java 1.0version

13. How are hash codes computed?
Ans: hashcode() is computed via jvm argument -XX:hashCode=N where N can be a number from [0-5]

14. Is it possible that hash code is not unique?
Ans: Obviously, **hashCode() can't produce unique values**, since int is limited to 2^32 distinct values, and there are an infinity of possible String values. In conclusion, the result of hashCode() is not a good fit for a key of a Map .

15. Can we put two elements with equal hash code to one hash map?
Ans: It's perfectly legal for two unequal objects to have the same hash code. It's used by HashMap as a "first pass filter" so that the map can quickly find possible entries with the specified key. The keys with the same hash code are then tested for equality with the specified key.

16. Iterator and modification of a List. ConcurentModificationException.
Ans : Java Iterator Core Methods

Method	**Description**
hasNext()	Returns true if the Iterator has more elements, and false if not.
next()	Return the next element from the Iterator
remove()	Removes the latest element returned from next() from the Collection the Iterator is iterating over.
forEachRemaining()	Iterates over all remaining elements in the Iterator and calls a **Java Lambda Expression** passing each remaining element as parameter to the lambda expression.

Iterating an Iterator

```
List<String> list = new ArrayList<>();
list.add("one");
list.add("two");
list.add("three");

Iterator<String> iterator = list.iterator();

Set<String> set = new HashSet<>();
set.add("one");
set.add("two");
set.add("three");

Iterator<String> iterator2 = set.iterator();

Iterator iterator = list.iterator();

while(iterator.hasNext()) {
    Object nextObject = iterator.next(); .}
```

Modification During Iteration

```
List<String> list = new ArrayList<>();

list.add("123");
list.add("456");
list.add("789");

Iterator<String> iterator = list.iterator();

while(iterator.hasNext()) {
    String value = iterator.next();

    if(value.equals("456")){
        list.add("999");
    }
}
```

ConcurrentModificationException in Java with Examples

```
// Java program to show
// ConcurrentModificationException

import java.util.Iterator;
import java.util.ArrayList;
```

```
public class modificationError {
	public static void main(String args[])
	{

		// Creating an object of ArrayList Object
		ArrayList<String> arr
			= new ArrayList<String>();

		arr.add("One");
		arr.add("Two");
		arr.add("Three");
		arr.add("Four");

		try {
			// Printing the elements
			System.out.println(
				"ArrayList: ");
			Iterator<String> iter
				= arr.iterator();

			while (iter.hasNext()) {

				System.out.print(iter.next()
									+ ", ");

				// ConcurrentModificationException
				// is raised here as an element
				// is added during the iteration
				System.out.println(
					"\n\nTrying to add"
					+ " an element in "
					+ "between iteration\n");
				arr.add("Five");
			}
		}
		catch (Exception e) {
			System.out.println(e);
		}
	}
}
```

17. What is the significance of ListIterator? What is the difference b/w Iterator and ListIterator?

Sr.	Iterator	ListIterator
1	Can traverse elements present in Collection only in the forward direction.	Can traverse elements present in Collection both in forward and backward directions.
2	Helps to traverse Map, List and Set.	Can only traverse List and not the other two.
3	Indexes cannot be obtained by using Iterator.	It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing List.
4	Cannot modify or replace elements present in Collection	We can modify or replace elements with the help of set(E e)
5	Cannot add elements and it throws ConcurrentModificationException .	Can easily add elements to a collection at any time.
6	Certain methods of Iterator are next(), remove() and hasNext().	Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e).

18. What is the Collections API?
Ans:
A collections API is a unified framework for representing and manipulating collections, allowing them to be manipulated independent of the details of their representation. The primary advantages of a Collections framework are: It provides interoperability between unrelated APIs.

19. How can we access elements of a collection?
Ans:

1) Using Iterators
    - Java provides Iterator and ListIterator classes to retrieve the elements of the collection object.

    - The hasNext() method of these interfaces returns true if the collection object has next

2) Using Iterators
    - Java provides Iterator and ListIterator classes to retrieve the elements of the collection object.

    - The hasNext() method of these interfaces returns true if the collection object has next element else it returns false.

    - The next() methods of the Iterator and ListIterator returns the next element of the collection.

3) Using Enumeration
    - The Enumeration class contains a method named hasMoreElements() which returns true if the current object contains more elements after the current position (else it returns false).

    - If you call the nextElement() method of the Enumeration class returns the next element in the current enumeration object.

20. What is the difference between a queue and a stack?

Stack Data Structure	**Queue Data Structure**
It is a linear data structure. The objects are removed or inserted at the same end.	It is also a linear data structure. The objects are removed and inserted from two different ends.
It follows the Last In, First Out (LIFO) principle. It means that the last inserted element gets deleted at first.	It follows the First In, First Out (FIFO) principle. It means that the first added element gets removed first from the list.

It has only one pointer- the **top**. This pointer indicates the address of the topmost element or the last inserted one of the stack.	It uses two pointers (in a simple queue) for reading and writing data from both the ends- the **front** and the **rear**. The rear one indicates the address of the last inserted element, whereas the front pointer indicates the address of the first inserted element in a queue.
Stack uses **push** and **pop** as two of its operations. The pop operation functions to remove the element from the list, while the push operation functions to insert the element in a list.	Queue uses **enqueue** and **dequeue** as two of its operations. The dequeue operation deletes the elements from the queue, and the enqueue operation inserts the elements in a queue.
Insertion and deletion of elements take place from one end only. It is called the top.	It uses two ends- front and rear. Insertion uses the rear end, and deletion uses the front end.
When top== max-1, it means that the stack is full.	When rear==max-1, it means that the queue is full.
When top==-1, it indicates that the stack is empty.	When front = rear+1 or front== -1, it indicates that the queue is empty.
A Stack data structure does not have any types.	A Queue data structure has three types- circular queue, priority queue, and double-ended queue.

You can visualize the Stack as a vertical collection.	You can visualize a Queue as a horizontal collection.
The implementation is simpler in a Stack.	The implementation is comparatively more complex in a Queue than a stack

21. What is the Properties class?
Ans:
The Properties class represents a persistent set of properties. The Properties can be saved to a stream or loaded from a stream. It belongs to java.util package. Properties define the following instance variable. This variable holds a default property list associated with a Properties object.

Features of Properties class:

- Properties is a subclass of Hashtable.
- It is used to maintain a list of values in which the key is a string and the value is also a string i.e; it can be used to store and retrieve string type data from the properties file.
- Properties class can specify other properties list as it's the default. If a particular key property is not present in the original Properties list, the default properties will be searched.
- Properties object does not require external synchronization and Multiple threads can share a single Properties object.
- Also, it can be used to retrieve the properties of the system.

22. Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?
Ans: Linked list

23. How can we use hashset in collection interface?
Ans:
- HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage.

- 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.

25. Can you limit the initial capacity of vector in java?

Ans: limit the initial capacity
we can construct an empty vector with specified initial capacity
public vector(int initialcapacity)

26. What method should the key class of Hashmap override?

Ans:: hashCode() and equals() method should the key class of hashmap override.

27. What is the difference between Enumeration and Iterator?

Sr. No.	Key	Iterator	Enumeration
1	Basic	In Iterator, we can read and remove elements while traversing elements in the collections.	Using Enumeration, we can only read elements during traversing elements in the collections.
2.	Access	It can be used with any class of the collection framework.	It can be used only with legacy classes of the collection framework such as a Vector and HashTable.
3.	Fail-Fast and Fail - Safe	Any changes in the collection, such as removing element from the collection during a thread is iterating collection then it throw concurrent modification exception.	Enumeration is Fail safe in nature. It doesn’t throw concurrent modification exception

4.	Limitation	Only forward direction iterating is possible	Remove operations can not be performed using Enumeration.
5.	Methods	It has following methods –*hasNext()*next()*remove()	It has following methods –*hasMoreElements()*nextElement()

28. Collections class and Arrays class
Ans:
Collections class is a member of the Java Collections Framework. The java.util.Collections package is the package that contains the Collections class. Collections class is basically used with the static methods that operate on the collections or return the collection. All the methods of this class throw the NullPointerException if the collection or object passed to the methods is null.

Array Class : The Arrays class in java.util package is a part of the Java Collection Framework. This class provides static methods to dynamically create and access Java arrays. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself