----------------------------------

Collections Q & A

----------------------------------

1. Draw Collections Framework Class Diagram

Graphical user interface

Description automatically generated

1. What is HashMap and Map?

Map: A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap.

HashMap:

Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

Points to remember

o Java HashMap contains values based on the key.

o Java HashMap contains only unique keys.

o Java HashMap may have one null key and multiple null values.

o Java HashMap is non synchronized.

o Java HashMap maintains no order.

o The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Hierarchy of HashMap class

Text, application

Description automatically generated

As shown in the above figure, HashMap class extends AbstractMap class and implements Map interface.

HashMap class declaration

Let's see the declaration for java.util.HashMap class.

1. public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

o K: It is the type of keys maintained by this map.

o V: It is the type of mapped values.

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

|  |  |
| --- | --- |
| HashMap | Hashtable |
| 1) HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is synchronized. It is thread-safe and can be shared with many threads. |
| 2) HashMap allows one null key and multiple null values. | Hashtable doesn't allow any null key or value. |
| 3) HashMap is a new class introduced in JDK 1.2. | Hashtable is a **legacy class**. |
| 4) HashMap is fast. | Hashtable is slow. |
| 5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is traversed by Iterator. | Hashtable is traversed by Enumerator and Iterator. |
| 7) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits AbstractMap class. | Hashtable inherits Dictionary class. |

1. Difference between Vector and ArrayList?

|  |  |
| --- | --- |
| ArrayList | 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?

Java Iterator is an interface that is practiced in order to iterate over a collection of Java object components entirety one by one. It is free to use in the Java programming language since the Java 1.2 Collection framework. It belongs to java.util package.

Points to Remember

The Java Iterator is an interface added in the Java Programming language in the Java 1.2 Collection framework. It belongs to java.util package.

It is one of the Java Cursors that are practiced to traverse the objects of the collection framework.

The Java Iterator is used to iterate the components of the collection object one by one.

The Java Iterator is also known as the Universal cursor of Java as it is appropriate for all the classes of the Collection framework.

The Java Iterator also supports the operations like READ and REMOVE.

The methods names of the Iterator class are very simple and easy to use compared to the method names of Enumeration Iterator.

6. List vs Set vs Map. Purposes and definitions.

A List in java extends the collection interface and represent an sequenced or ordered group of elements. It can contain duplicate elements. It also defines some additional methods which it inherits from Collection interface.

Note: Elements in List can be inserted, updated, or retrieved by their position or index. Index or position value starts from 0.

A set represents a group or collection of items. Set has a special property that is unique items, it can not contain a duplicate item or element. It extends the collection interface.

Note: Set interface does not have any additional method other than methods inherited from Collection interface. With all collection interface methods it adds the restriction that it can not contain a duplicate elements.

A map in java, not extends the Collection interface. It represents a group of special elements or objects. Every map element or object contains key and value pair. A map can’t contain duplicate keys and one key can refer to at most one value.

|  |  |  |
| --- | --- | --- |
| LIST | SET | MAP |
| Allow duplicate elements. | Does not allow duplicate elements. | Does not allow duplicate key but values can be duplicate. |
| Allow multiple null values | Allow single null value. | Allow single null as key and multiple null as values. |
| Maintains insertions order. | Set represents an unordered collection but some of its implementation classes maintains some order. LinkedHashSet maintains order, TreeSet maintains ascending order. | Like Set, Map also represents an unordered collection. Again same like Set, some of its implementation classes maintains some order. TreeMap maintains ascending order of keys. |

7. Pros and cons of ArrayList and LinkedList

Advantages and disadvantages of ArrayList and LinkedList

List is an interface, ArrayList and LinkedList are the implementation class of List.

ArrayList is implemented based on arrays , while LinkedList is implemented based on linked lists ;

Arrays store elements continuously in memory. Since each element occupies the same memory, you can quickly access any element in the array through subscripts. But if you want to add an element to the array, you need to move a large number of elements, free up one element of space in memory, and then put the element to be added into it. In the same way, if you want to delete an element, you also need to move a large number of elements to fill out the moved element.

The linked list is just the opposite. The elements in the linked list are not stored sequentially in memory, but are linked together by pointers stored in the elements. For example: the previous element has a pointer to the next element, and so on until the last element. If you want to access an element in the linked list, you need to start from the first element and always find the position of the element you need. But adding and deleting an element is very simple for the linked list data structure, as long as you modify the pointer in the element.

8. TreeSet vs LinkedHashSet

|  |  |  |
| --- | --- | --- |
|  | LinkedHashSet | TreeSet |
| How they work internally? | LinkedHashSet uses LinkedHashMap internally to store it’s elements. | TreeSet uses TreeMap internally to store it’s elements. |
| Order Of 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. |
| Performance | 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. |
| Insertion, Removal And Retrieval 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. |
| How they compare the elements? | 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. |
| Null 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. |
| Memory Occupation | LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements. | TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap. |
| When To Use? | Use LinkedHashSet if you want to maintain insertion order of elements. | Use TreeSet if you want to sort the elements according to some Comparator. |

**9. What are relationships between equals and hash codes?**

hashCode() Method

public int hashCode()

This method returns an integer value, which is referred to as the hash code value of an object. Every Object, at the time of creation assigned with a unique 32-bit, signed int value. This value is the hash code value of that object.

General contract associated with hashCode() method

The hashCode() method should return the same integer value for the same object for each calling of this method unless the value stored in the object is modified.

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).

equals() Method

public boolean equals(Object obj)

The equals() method of Object class checks the equality of the objects and accordingly it returns true or false. The default implementation, as provided by Object class, checks the equality of the objects on the basis if both references refer to the same object. It does not check the value or state of the objects. But we can override this method to provide own implementation to compare the state or value of the objects.

General contract associated with equals() method

For any non-null reference variables a, b and c

a.equals(a) should always return true.

a.equals(b) should return true if and only if b.equals(a) returns true.

If a.equals(b) returns true and b.equals(c) returns true then a.equals(c) should return true.

Multiple calling of a.equals(b) should consistently return true or consistently return false If the value of the object is not modified for either object.

a.equals(null) should return false.

So it is necessary to override the hashCode() method of Object class if we are overriding the equals() method

**10. What are the advantages of ArrayList over arrays ?**

Disadvantages of arrays:

Check the disadvantages of arrays in below article.

Advantages and disadvantages of arrays in java

Advantages / Benefits of arraylist in java:

We have some disadvantages of arrays like arrays are fixed in length. and we need to mention size of the array while creation itself.

So we have some advantages of arraylist when compared to arrays in java.

Here the major advantages of arraylist over arrays.

1.ArrayList is variable length

One of the major benefit of arraylist is it is dynamic in size. we can increase as well as decrease size of the arraylist dynamically.

Resizable.

2.Default initial capacity is 10.

One of the major benefit of arraylist is by default it will assign default size as 10.

Whenever we create object of ArrayList constructor of ArrayList assign default capacity as 10.

3.Insert and remove elements also at particular position of ArrayList

Another advantage of ArrayList is it can add and remove elements at particular position.

4.Add any type of data into ArrayList.

We can add different type of objects in to the ArrayList.

In this scenario avoid mentioning generics while declaring ArrayList.

5.Traverse in both directions.

We can traverse both direction using List Iterator.

6.ArrayList allows Multiple null values

We can add multiple null elements to ArrayList

7.ArrayList allows to add duplicate elements

We can add duplicate elements into arrayList

8.ArrayList has many methods to manipulate stored objects.

ArrayList has many methods to manipulate stored objects.

addAll(), isEmpty(). lastIndexOf() etc.

**11. Principle of storing data in a hashtable**

What is a hash table?

Hash table (Hash table, also called hash table) is a data structure that is directly accessed based on the key value. In other words, it accesses the record by mapping the key code value to a position in the table to speed up the search. This mapping function is called a hash function, and the array storing records is called a hash table.

Record storage location = f (keyword)

The corresponding relationship f here is called a hash function, also called a hash (Hash function). The record is stored in a continuous storage space using hashing technology. This continuous storage space is called a hash table or hash table ( Hash table).

The hashtable (key, value) is to convert the Key into an integer number through a fixed algorithm function, the so-called hash function, and then take the remainder of the number to the length of the array, and the result of the remainder will be treated as an array The subscript of the value is stored in the array space with the number as the subscript. (Or: Transform an input of any length (also called pre-image, pre-image) into a fixed-length output through a hashing algorithm. The output is the hash value. This conversion is a compression mapping, that is, The hash value space is usually much smaller than the input space. Different inputs may be hashed into the same output, and it is impossible to uniquely determine the input value from the hash value. Simply put, it is a way of compressing messages of any length Function to a fixed-length message digest)

When using the hash table to query, it is to use the hash function again to convert the key to the corresponding array subscript, and locate the space to obtain the value. In this way, the positioning performance of the array can be fully utilized for data position.

The characteristics of arrays are: easy to address, difficult to insert and delete;

The characteristics of the linked list are: difficult to address, easy to insert and delete.

So can we combine the characteristics of the two to create a data structure that is easy to address and easy to insert and delete? The answer is yes. This is the hash table we are going to mention. There are many different implementation methods for the hash table. What I will explain next is the most commonly used method-the zipper method. We can understand it as "linked list Array", as shown in the figure:

Diagram, engineering drawing

Description automatically generated

The left side is obviously an array. Each member of the array includes a pointer to the head of a linked list. Of course, the linked list may be empty or there may be many elements. We assign the elements to different linked lists according to some characteristics of the elements, and also find the correct linked list based on these characteristics, and then find the element from the linked list.

Hash application

1. Hash is mainly used in encryption algorithms in the field of information security. It converts information of different lengths into messy 128-bit codes. These code values ​​are called hash values. It can also be said that Hash is to find a kind of data content and data storage The mapping relationship between addresses.

2. Search: Hash table, also known as hash, is a faster search technique. Our previous search was based on this kind of thinking: take out an element from the set to see if it is equal to what we are looking for, if not, narrow the scope and continue searching. The hash table is a completely different way of thinking: when I know the key value, I can directly calculate the position of the element in the set, without having to look up again and again !

For example, if the key in the i-th element in my array A is i, then the number 3 must be in the third position, and the number 10 must be in the tenth position. The hash table is to use this basic idea to establish a function from key to position, and then perform direct calculation and search.

3. Hash table is widely used in massive data processing.

Hashing method: the method of transforming element characteristics into array subscripts

I think everyone is thinking about a very serious question: "What if two strings correspond to the same position in the hash table?" After all, the capacity of an array is limited, which is very likely. There are many ways to solve this problem. The first thing I thought of was to use a "linked list". Many algorithms I have encountered can be converted into linked lists to solve them. Just hang a linked list at each entry of the hash table and save all the corresponding strings.

Steps to find the hash table:

When storing the record, the hash address of the record is calculated through the hash function

When looking up a record, we use the same hash function to calculate the hash address of the record, and access the record according to this hash address

Keyword-hash function (hash function)-hash address

Advantages: One-to-one search efficiency is very high;

Disadvantages: A keyword may correspond to multiple hash addresses; when you need to find a range, the effect is not good.

Hash conflict: Different keywords get the same hash address through the calculation of the hash function.

Good hash function = simple calculation + uniform distribution (the calculated hash address is evenly distributed)

Hash table is a kind of data structure, it can provide fast insert operation and search operation

1. Divided hashing method

The most intuitive one, the above figure uses this hashing method,

Formula: index = value% 16

Anyone who has studied assembly knows that finding the modulus is actually obtained through a division operation, so it is called "division hashing".

2. Square hash method

Finding the index is a very frequent operation, and the operation of multiplication is more time-saving than the operation of division (for current CPUs, it is estimated that we can't feel it), so we consider replacing the division with a multiplication and a shift operation.

Formula: index = (value \* value) >> 28

(Shift to the right, divide by 2^28. Notation: shift to the left to become larger, which is multiplication. Shift to the right to become smaller, to divide.)

If the value distribution is relatively even, this method can get good results, but the value of each element of the graph I drew above is calculated as 0-very failed. Maybe you still have a question, if the value is very large, will the value \* value not overflow? The answer is yes, but our multiplication does not care about overflow, because we are not to get the result of the multiplication at all, but to get the index.

3. Fibonacci (Fibonacci) hashing method

The shortcomings of the square hash method are obvious, so can we find an ideal multiplier instead of using the value itself as a multiplier? The answer is yes.

1. For a 16-bit integer, the multiplier is 40503

2. For a 32-bit integer, this multiplier is 2654435769

3. For 64-bit integers, this multiplier is 11400714819323198485

How are these "ideal multipliers" derived? This is related to a rule called the golden section rule, and the most classic expression describing the golden section rule is undoubtedly the famous Fibonacci sequence, which is a sequence of this form: 0, 1, 1, 2, 3, 5, 8. , 13, 21, 34, 55, 89, 144, 233,377, 610, 987, 1597, 2584, 4181, 6765, 10946,... In addition, the value of the Fibonacci sequence is surprisingly consistent with the ratio of the orbital radius of the eight planets in the solar system.

For our common 32-bit integers,

Formula: index = (value \* 2654435769) >> 28

If you use this Fibonacci hashing method, then the above diagram becomes like this:

Diagram

Description automatically generated

Solution to hash conflict:

1. Create a buffer zone, and put all the people whose pinyin is repeated into the buffer zone. When I searched for a person by name, I found that it was not right, so I looked for it in the buffer.

2. Perform reprobing. Just look elsewhere. There can also be many detection methods.

(1) Search at index-1, index+1, index-2, index+2, and so on. This method is called linear reprobing.

(2) Random search around the search position index. It is called random detection.

(3) Hash again. That is, when there is a conflict, another mapping method is used to find it.

In this program, the process of finding repeated elements is simulated by taking the modulus. The way to deal with duplicate elements is to hash again: +7 to the current key position. Finally, you can use global variables to determine how many times you need to search. Here I calculated the total number of searches by searching the lowercase of 26 English letters in turn. Obviously, when the total number of searches/the total number of elements searched is closer to 1, the hash table is closer to a one-to-one mapping function, and the search efficiency is higher

Expand

The d in d-left hashing means multiple. Let's simplify this problem first and take a look at 2-left hashing. 2-left hashing refers to dividing a hash table into two halves of equal length, called T1 and T2, respectively. T1 and T2 are respectively equipped with a hash function, h1 and h2. When storing a new key, two hash functions are used for calculation at the same time, and two addresses h1[key] and h2[key] are obtained. At this time, you need to check the position of h1[key] in T1 and the position of h2[key] in T2, which position has stored more (collision) keys, and then store the new key in a position with less load. If there are equal numbers on both sides, for example, both positions are empty or a key is stored in both, the new key is stored in the T1 sub-table on the left, and 2-left is also derived from this. When looking up a key, you must hash twice and look up two locations at the same time.

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

|  |  |  |
| --- | --- | --- |
| 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 1.2version | Introduce in java 1.0version |

13. How are hash codes computed?

The implementation of hashCode() may differ from class to class (if overriden) but the contract for hashCode() is very specific and stated clearly and explicitly in the Javadocs:

Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable.

The general contract of hashCode is:

Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

As much as is reasonably practical, the hashCode method defined by class Object does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the JavaTM programming language).

A hashcode is an integer value that represents the state of the object upon which it was called. That is why an Integer that is set to 1 will return a hashcode of "1" because an Integer's hashcode and its value are the same thing. A character's hashcode is equal to it's ASCII character code. If you write a custom type you are responsible for creating a good hashCode implementation that will best represent the state of the current instance.

14. Is it possible that hash code is not unique?

Unique, no. By nature, hash values are not guaranteed to be unique.

Any system with an arbitrarily large number of possible inputs and a limited number of outputs will have collisions.

So, you won't be able to use a unique database key to store them if it's based only on the hash code. You can, however, use a non-unique key to store them.

In reply to your second question about whether different versions of Java will generate different hash codes for the same string, no.

Provided a Java implementation follows the Oracle documentation (otherwise it's not really a Java implementation), it will be consistent across all implementations. The Oracle docs for String.hashCode specify a fixed formula for calculation the hash:

s[0]\*31^(n-1) + s[1]\*31^(n-2) + ... + s[n-1]

You may want to check this is still the case if you're using wildly disparate versions of Java (such as 1.2 vs 8) but it's been like that for a long time, at least since 1.5.

15. Can we put two elements with equal hash code to one hash map?

A hashmap works like this (this is a little bit simplified, but it illustrates the basic mechanism):

It has a number of "buckets" which it uses to store key-value pairs in. Each bucket has a unique number - that's what identifies the bucket. When you put a key-value pair into the map, the hashmap will look at the hash code of the key, and store the pair in the bucket of which the identifier is the hash code of the key. For example: The hash code of the key is 235 -> the pair is stored in bucket number 235. (Note that one bucket can store more then one key-value pair).

When you lookup a value in the hashmap, by giving it a key, it will first look at the hash code of the key that you gave. The hashmap will then look into the corresponding bucket, and then it will compare the key that you gave with the keys of all pairs in the bucket, by comparing them with equals().

Now you can see how this is very efficient for looking up key-value pairs in a map: by the hash code of the key the hashmap immediately knows in which bucket to look, so that it only has to test against what's in that bucket.

Looking at the above mechanism, you can also see what requirements are necessary on the hashCode() and equals() methods of keys:

If two keys are the same (equals() returns true when you compare them), their hashCode() method must return the same number. If keys violate this, then keys that are equal might be stored in different buckets, and the hashmap would not be able to find key-value pairs (because it's going to look in the same bucket).

If two keys are different, then it doesn't matter if their hash codes are the same or not. They will be stored in the same bucket if their hash codes are the same, and in this case, the hashmap will use equals() to tell them apart.

16. Iterator and modification of a List. ConcurentModificationException.

The ConcurrentModificationException occurs when an object is tried to be modified concurrently when it is not permissible. This exception usually comes when one is working with Java Collection classes.

For Example - It is not permissible for a thread to modify a Collection when some other thread is iterating over it. This is because the result of the iteration becomes undefined with it. Some implementation of the Iterator class throws this exception, including all those general-purpose implementations of Iterator which are provided by the JRE. Iterators which do this are called fail-fast as they throw the exception quickly as soon as they encounter such situation rather than facing undetermined behavior of the collection any time in the future.

Note: It is not mandatory that this exception will be thrown only when some other thread tries to modify a Collection object. It can also happen if a single thread has some methods called which are trying to violate the contract of the object. This may happen when a thread is trying to modify the Collection object while it is being iterated by some fail-fast iterator, the iterator will throw the exception.

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

It is only applicable for List collection implemented classes like arraylist, linkedlist etc. It provides bi-directional iteration. ListIterator must be used when we want to enumerate elements of List. This cursor has more functionality(methods) than iterator.

ListIterator object can be created by calling listIterator() method present in List interface.

// Here "l" is any List object, ltr is of type

// ListIterator interface and refers to "l"

ListIterator ltr = l.listIterator();

|  |  |
| --- | --- |
| Iterator | ListIterator |
| Can traverse elements present in Collection only in the forward direction. | Can traverse elements present in Collection both in forward and backward directions. |
| Helps to traverse Map, List and Set. | Can only traverse List and not the other two. |
| 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. |
| Cannot modify or replace elements present in Collection | We can modify or replace elements with the help of set(E e) |
| Cannot add elements and it throws ConcurrentModificationException. | Can easily add elements to a collection at any time |
| 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?

The Collection API is a set of classes and interfaces that support operation on collections of objects. These classes and interfaces 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.

19. How can we access elements of a collection?

Following are the 4 ways to retrieve any elements from a collection object:

1) For-each

For each loop is meant for traversing items in a collection.

// Iterating over collection 'c' using for-each

for (Element e: c)

System.out.println(e);

We read the ‘:’ used in for-each loop as “in”. So loop reads as “for each element e in elements”, here elements are the collection which stores Element type items.

Note:In Java 8 using lambda expressions we can simply replace for-each loop with

elements.forEach (e -> System.out.println(e) );

Using Cursors

Cursor is an interface and it is used to retrieve data from collection object, one by one. Cursor has 3 types, which are given below:

2)Iterator Interface: Iterator is an interface provided by collection framework to traverse a collection and for a sequential access of items in the collection.

// Iterating over collection 'c' using iterator

for (Iterator i = c.iterator(); i.hasNext(); )

System.out.println(i.next());

It has 3 methods:

boolean hasNext(): This method returns true if the iterator has more elements.

elements next(): This method returns the next elements in the iterator.

void remove(): This method removes from the collection the last elements returned by the iterator.

3)ListIterator Interface: It is an interface that contains methods to retrieve the elements from a collection object, both in forward and reverse directions. This iterator is for list based collections.

It has following important methods:

booleanhasNext(): This returns true if the ListIterator has more elements when traversing the list in the forward direction.

booleanhasPrevious(): This returns true if the ListIterator has more elements when traversing the list in the reverse direction.

element next(): This returns the next element in the list.

element previous():This returns the previous element in the list.

void remove(): This removes from the list the last elements that was returned by the next() or previous() methods.

int nextIndex() Returns the index of the element that would be returned by a subsequent call to next(). (Returns list size if the list iterator is at the end of the list.)

int previousIndex() Returns the index of the element that would be returned by a subsequent call to previous(). (Returns -1 if the list iterator is at the beginning of the list.)

4)EnumerationIterator Interface: The interface is useful to retrieve one by one the element. This iterator is based on data from Enumeration and has methods:

booleanhasMoreElements(): This method tests if the Enumeration has any more elements or not .

element nextElement(): This returns the next element that is available in elements that is available in Enumeration

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

|  |  |
| --- | --- |
| stack | queue |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |

21. What is the Properties class?

The properties object contains key and value pair both as a string. The java.util.Properties class is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.

An Advantage of the properties file

Recompilation is not required if the information is changed from a properties file: If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.

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

ArrayList and Vector both use an array to store the elements of the list. When an element is inserted into the middle of the list the elements that follow the insertion point must be shifted to make room for the new element. The LinkedList is implemented using a doubly linked list; an insertion requires only the updating of the links at the point of insertion. Therefore, the LinkedList allows for fast insertions and deletions.

23. How can we use hashset in collection interface?

The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance. No guarantee is made as to the iteration order of the set which means that the class does not guarantee the constant order of elements over time. This class permits the null element. The class also offers constant time performance for the basic operations like add, remove, contains, and size assuming the hash function disperses the elements properly among the buckets

Implements Set Interface.

The underlying data structure for HashSet is Hashtable.

As it implements the Set Interface, duplicate values are not allowed.

Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.

NULL elements are allowed in HashSet.

HashSet also implements Serializable and Cloneable interfaces.

Declaration of HashSet:

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable

where E is the type of elements stored in a HashSet.

1. HashSet(): This constructor is used to build an empty HashSet object in which the default initial capacity is 16 and the default load factor is 0.75. If we wish to create an empty HashSet with the name hs, then, it can be created as:

HashSet<E> hs = new HashSet<E>();

2. HashSet(int initialCapacity): This constructor is used to build an empty HashSet object in which the initialCapacity is specified at the time of object creation. Here, the default loadFactor remains 0.75.

HashSet<E> hs = new HashSet<E>(int initialCapacity);

3. HashSet(int initialCapacity, float loadFactor): This constructor is used to build an empty HashSet object in which the initialCapacity and loadFactor are specified at the time of object creation.

HashSet<E> hs = new HashSet<E>(int initialCapacity, float loadFactor);

4. HashSet(Collection): This constructor is used to build a HashSet object containing all the elements from the given collection. In short, this constructor is used when any conversion is needed from any Collection object to the HashSet object. If we wish to create a HashSet with the name hs, it can be created as:

HashSet<E> hs = new HashSet<E>(Collection C);

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

Yes you can 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?

HashMap and HashSet use the hashcode value of an object to find out how the object would be stored in the collection, and subsequently hashcode is used to help locate the object in the collection. Hashing retrieval involves:

First, find out the right bucket using hashCode().

Secondly, search the bucket for the right element using equals()

Let us consider all the cases of Overriding in these methods

You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable. (-Joshua Bloch)

Here is the contract, from the java.lang.Object specialization:

Whenever it(hashcode) is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.

If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.

It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

27. What is the difference between Enumeration and Iterator?

|  |  |  |
| --- | --- | --- |
| Key | Iterator | Enumeration |
| Basic | In Iterator, we can read and remove element while traversing element in the collections. | Using Enumeration, we can only read element during traversing element in the collections. |
| Access | It can be used with any class of the collection framework. | It can be used only with legacy class of the collection framework such as a Vector and HashTable. |
| 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 |
| Limitation | Only forward direction iterating is possible | Remove operations can not be performed using Enumeration. |
| Methods | It has following methods −  \*hasNext()  \*next()  \*remove() | It has following methods −  \*hasMoreElements()  \*nextElement() |

28. Collections class and Arrays class

|  |  |
| --- | --- |
| Arrays | Collections |
| Arrays are fixed in size that is once we create an array we can not increased or decreased based on our requirement. | Collection are growable in nature that is based on our requirement. We can increase or decrease of size. |
| With respect to memory Arrays are not recommended to use. | With respect to memory collection are recommended to use. |
| With respect to performance Arrays are recommended to use. | With respect to performance collection are not recommended to use. |
| Arrays can hold only homogeneous data types elements. | Collection can hold both homogeneous and and heterogeneous elements. |
| There is no underlying data structure for arrays and hence ready made method support is not available. | Every collection class is implemented based on some standard data structure and hence for every requirement ready made method support is available being a performance. we can use these method directly and We are not responsible to implement these methods. |
| Arrays can hold both object and primitive. | Collection can hold only object types but not primitive datatypes such as int, long, short, etc. |