1. What is collection classes?

Ans: The standard classes that implements collection interface is called collection classes.

2.List and give description of different collection classes?

Class	Description
AbstractCollection	Implements most of the **Collection** interface.
AbstractList	Extends **AbstractCollection** and implements most of the **List** interface.
AbstractQueue	Extends **AbstractCollection** and implements parts of the **Queue** interface.
AbstractSequentialList	Extends **AbstractList** for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending **AbstractSequentialList**.
ArrayList	Implements a dynamic array by extending **AbstractList**.
ArrayDeque	Implements a dynamic double-ended queue by extending **AbstractCollection** and implementing the **Deque** interface. (Added by Java SE 6.)
AbstractSet	Extends **AbstractCollection** and implements most of the **Set** interface.
EnumSet	Extends **AbstractSet** for use with **enum** elements.
HashSet	Extends **AbstractSet** for use with a hash table.
LinkedHashSet	Extends **HashSet** to allow insertion-order iterations.
PriorityQueue	Extends **AbstractQueue** to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends **AbstractSet**.

3.Explain ArrayList Class with example.

- ArrayList class extends AbstractList
- It implements the List interface
- ArrayList is a generic class
- The capacity of an ArrayList object increases automatically as objects are stored in it.
- The capacity of an ArrayList object manually by calling ensureCapacity().

Declaration: class ArrayList<E>

- E specifies the type of objects that the list will hold

Syntax of ensureCapacity(): void ensureCapacity(int cap)

- cap is the new capacity.

- By calling trimToSize() can reduce the size of ArrayList

Example(continuation Qno 3):

```
1. // Demonstrate ArrayList.
2. import java.util.*;
3. class ArrayListDemo {
4.  public static void main(String args[]) {
5.  // Create an array list.
6. ArrayList<String> al = new
   ArrayList<String>();
7. System.out.println("Initial size of al: " +
   al.size());
8. // Add elements to the array list.
9. al.add("C");
```

```
10. al.add("A");
11. al.add("E");
12. al.add("B");
13. al.add("D");
14. al.add("F");
15. al.add(1, "A2");
16. System.out.println("Size of al after additions+al.size());
17. // Display the array list.
18.  System.out.println("Contents of al: " + al);
19. // Remove elements from the array list.
20. al.remove("F");
21. al.remove(2);
22.System.out.println("Size of al after deletions: " +
al.size());
23. System.out.println("Contents of al: " + al);
```

Output:

❖The output from this program is shown here:

- Initial size of al: 0
- Size of al after additions: 7
- Contents of al: [C, A2, A, E, B, D, F]
- Size of al after deletions: 5
- Contents of al: [C, A2, E, B, D]

4.Explain the three types of constructors of ArrayList.

1. ArrayList():

- It builds an empty array list.

2. ArrayList(Collection<? Extends E>c):

- This constructor builds an array list that is initialized with the elements of the collection c.

3. ArrayList(int capacity):

- The third constructor builds an array list that has the specified initial capacity.
- The capacity is the size of the underlying array that is used to store the elements.
- The capacity grows automatically as elements are added to an array list.

5.What is the necessity of ArrayList in case of Java?

- To support dynamic arrays that can grow as we needed.
- Since it is a variable-length array Collections Framework.
- So ArrayList can dynamically increase or decrease in size

6. Explain how to obtain an array from ArrayList , with example?

- It can be done by calling toArraymethod()
- This method obtain array of integers
- There are two versions of toArray()

1.Object[ ] toArray( )

2. <T> T[ ] toArray(T array[ ])

- The first returns an array of Object
- The second returns an array of elements that have the same type as T.
- Second form is more convenient because it returns the proper type of array.

Example:

```
1.  // Convert an ArrayList into an array.
2.  import java.util.*;
3.  class ArrayListToArray {
4.  public static void main(String args[]) {
5.  // Create an array list.
6.   ArrayList<Integer> al = new ArrayList<Integer>();
7.  // Add elements to the array list.
8.  al.add(1);
9.  al.add(2);
10.  al.add(3);
11. al.add(4);
12. System.out.println("Contents of al: " + al);
13. // Get the array.
14. Integer ia[] = new Integer[al.size()];
15. ia = al.toArray(ia);
16. int sum = 0;
17. // Sum the array.
18. for(int i : ia) sum += i;
19. System.out.println("Sum is: " + sum);
```

Output:

- Contents of al: [1, 2, 3, 4]
- Sum is: 10

7.What is the reason to convert ArrayList(collection) into array?

- To obtain faster processing times for certain operations
- To pass an array to a method that is not overloaded to accept a collection
- To integrate collection-based code with legacy code that does not understand collections.

8. **Explain LinkedList class with example.**

- LinkedList class extends AbstractSequentialList
- Implements the List, Deque, and Queue interfaces
- It provides a linked-list data structure
- LinkedList is a generic class

Declaration: class LinkedList<E>

- E specifies the type of objects that the list will hold.

Constructors:

1. LinkedList( )
2. LinkedList(Collection<? extends E> c)

- The first constructor builds an empty linked list
- The second constructor builds a linked list that is initialized with the elements of the collection c.

Example:

```
1. // Demonstrate LinkedList.
2. import java.util.*;
3. class LinkedListDemo {
4. public static void main(String args[]) {
5. // Create a linked list.
6.  LinkedList<String> ll = new LinkedList<String>();
7. // Add elements to the linked list.
8. ll.add("F");
9. ll.add("B");
10. ll.add("D");
11. ll.add("E");
12. ll.add("C");
13. ll.addLast("Z");
14. ll.addFirst("A");
15. ll.add(1, "A2");
```

```
16. System.out.println("Original contents of ll: " + ll);
17. // Remove elements from the linked list.
18.ll.remove("F");
19.  ll.remove(3);
20. System.out.println("Contents of ll after deletion: " + ll);
21. // Remove first and last elements.
22.  ll.removeFirst();
23. ll.removeLast();
24. System.out.println("ll after deleting first and last: " + ll);
25. // Get and set a value.
26. String val = ll.get(2);
27.  ll.set(2, val + " Changed");
28.System.out.println("ll after change: " + ll);
```

Output:

- Original contents of ll: [A, A2, F, B, D, E, C, Z]
- Contents of ll after deletion: [A, A2, D, E, C, Z]
- ll after deleting first and last: [A2, D, E, C]
- ll after change: [A2, D, E Changed, C]

17.List and explain the methods defined by deque with example.

1. addFirst( ) or offerFirst( ): To add elements to the start of a list
2. addLast( ) or offerLast( ): To add elements to the end of the list
3. getFirst( ) or peekFirst( ): To obtain the first element
4. getLast( ) or peekLast( ): To obtain the last element
5. removeFirst( ) or pollFirst( ): To remove the first element
6. removeLast( ) or pollLast(): To remove the last element,
7. Example: Refer previous program

18.What is hashing?

- A hashtable 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 hashcode
- The hash code is then used as the index at which the data associated with the key is stored

19.Explain Hashset class with example

- HashSet extends AbstractSet
- It implements the Set interface
- It creates a collection that uses a hash table for storage

Declaration: class HashSet<E>

- E specifies the type of objects that the set will hold.
- The elements are not stored in sorted order, and the precise output may vary
- HashSet does not define any additional methods beyond those provided by its super classes and interfaces.

Example:

```
1.  // Demonstrate HashSet.
2.  import java.util.*;
3.  class HashSetDemo {
4.   public static void main(String args[]) {
5.  // Create a hash set.
6.   HashSet<String> hs = new HashSet<String>();
7.  // Add elements to the hash set.
8.   hs.add("B");
9.   hs.add("A");
10. hs.add("D");
11.  hs.add("E");
12. hs.add("C");
13. hs.add("F");
14. System.out.println(hs);
```

Output:

- [B, A, D, E, C, F]

20.List and explain the constructors of HashSet Class

1.HashSet( ):

- The first form constructs a default hash set

2. HashSet(Collection<? extends E> c) :

- It initializes the hash set by using the elements of c

3. HashSet(int capacity):

- It initializes the capacity of the hash set to capacity.
- The default capacity is 16

4. HashSet(int capacity, float fillRatio):

- It initializes both the capacity and the fill ratio of the hash set from its arguments
- Fill ratio is also called as load capacity,
- The fill ratio must be between 0.0 and 1.0
- It determines how full the hash set can be before it is resized upward
- Constructors that do not take a fill ratio, 0.75 is used

21.Explain LinkedHashSet with example.

- The LinkedHashSet class extends HashSet
- Adds no members of its own
- It is a generic class that has this

Declaration: class LinkedHashSet<E>

- E specifies the type of objects that the set will hold

Constructors: Refer constructors of HashSet

- Linked HashSet maintains a linked list of the entries in the set,in the order in which they were inserted.
- This allows insertion-order iteration over the set
- When cycling through a LinkedHash Set using an iterator, the elements will be returned in the order in which they were inserted

Example:

```
1.  // Demonstrate HashSet.
2.  import java.util.*;
3.  class LinkedHashSetDemo {
4.   public static void main(String args[]) {
5.  // Create a Linkedhash set.
6.   LinkedHashSet<String> hs = new  LinkedHashSet<String>();
7.  // Add elements to the Linkedhash set.
8.   hs.add("B");
9.   hs.add("A");
10. hs.add("D");
11.  hs.add("E");
12. hs.add("C");
13. hs.add("F");
14. System.out.println(hs);
```

Output: [B, A, D, E, C, F]

22.Explain TreeSetClass with example.

- TreeSet extends AbstractSet
- It implements the NavigableSet interface
- It creates a collection that uses a tree for storage
- Objects are stored in sorted, ascending order
- Access and retrieval times are quite fast
- So TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly.
- The methods defined by NavigableSet is used to retrieve elements of a TreeSet

Declaration: class TreeSet<E>

- E specifies the type of objects that the set will hold

Example:

```
1.  // Demonstrate TreeSet.
2.   import java.util.*;
3.  class TreeSetDemo {
4.   public static void main(String args[]) {
5.  // Create a tree set.
6.  TreeSet<String> ts = new TreeSet<String>();
7.  // Add elements to the tree set.
8.  ts.add("C");
9.  ts.add("A");
10.  ts.add("B");
11.  ts.add("E");
12.  ts.add("F");
13.  ts.add("D");
14. System.out.println(ts);
```

Output: [A, B, C, D, E, F]

23.Explain the constructors of TreeSetClass.

1.TreeSet( ):

- The first form constructs an empty tree set that will be sorted in ascending order according to the natural order of its elements

2.TreeSet(Collection<? extends E> c):

- The second form builds a tree set that contains the elements of c.

3. TreeSet(Comparator<? super E> comp):

- The third form constructs an empty treeset that will be sorted according to the comparator specified by comp

4. TreeSet(SortedSet<E> ss):

- The fourth form builds a tree set that contains the elements of ss.

24.Explain six constructors of PriorityQueue.

1. PriorityQueue( ):

- The first constructor builds an empty queue.
- Its starting capacity is 11.

2. PriorityQueue(int capacity) :

- The second constructor builds a queue that has the specified initial capacity.

3. PriorityQueue(int capacity, Comparator<? super E> comp) :

- The third constructor builds a queue with the specified capacity and comparator.

4. PriorityQueue(Collection<? extends E> c)

5. PriorityQueue(PriorityQueue<? extends E> c)

6.PriorityQueue(SortedSet<? extends E> c)

- The last three constructors create queues that are initialized with the elements of the collection passed in c.
- In all cases, the capacity grows automatically as elements are added
- If no comparator is specified when a PriorityQueue is constructed, then the default comparator for the type of data stored in the queue is used
- The default comparator will order the queue in ascending order
- Thus,the head of the queue will be the smallest value

25.Explain ArrayDeque with example

- ArrayDeque class, which extends AbstractCollection
- It implements the Deque interface.
- Array Deque creates a dynamic array and has no capacity restrictions.

Declaration: class ArrayDeque<E>

- E specifies the type of objects stored in the collection

Constructors:

1. ArrayDeque( ):

- The first constructor builds an empty deque
- Its starting capacity is 16

2. ArrayDeque(int size):

- The second constructor builds a deque that has the specified initial capacity.

3. ArrayDeque(Collection<? extends E> c):

- The third constructor creates a deque that is initialized with the elements of the collection passed in c.

Example:

```
1.  // Demonstrate ArrayDeque.
2.  import java.util.*;
3.  class ArrayDequeDemo {
4.  public static void main(String args[]) {
5.   // Create a array deque
6.  ArrayDeque<String> adq = new ArrayDeque<String>();
7.  // Use an ArrayDeque like a stack.
8.   adq.push("A");
9.  adq.push("B");
10. adq.push("D");
11. adq.push("E");
12. adq.push("F");
13. System.out.print("Popping the stack: ");
14. while(adq.peek() != null)
15. System.out.print(adq.pop() + " ");
16. System.out.println();
```

Output: Popping the stack: F E D B A

26.Explain the two ways of accessing a collection

1.By using an iterator method:

- It is an object that implements either the Iterator or the List Iterator interface.
- Iterator enables you to cycle through a collection, obtaining or removing elements
- ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.
- Steps to follow inorder to access collection

1. Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.
2. Set up a loop that makes a call to hasNext(). Have the loop iterate as long as hasNext() returns true.
3. Within the loop, obtain each element by calling next( ).

Example:

```
1.  // Demonstrate iterators.
2.   import java.util.*;
3.  class IteratorDemo {
4.  public static void main(String args[]) {
5.   // Create an array list.
6.   ArrayList<String> al = new ArrayList<String>();
7.  // Add elements to the array list.
8.   al.add("C");
9.   al.add("A");
10.  al.add("E");
11.  al.add("B");
12.  al.add("D");
13.  al.add("F");
14. // Use iterator to display contents of al.
15. System.out.print("Original contents of al: ");
16. Iterator<String> itr = al.iterator();
```

```
17. while(itr.hasNext()) {
18.  String element = itr.next();
19. System.out.print(element + " ");
 20. }
21. System.out.println();
22. // Modify objects being iterated.
 23. ListIterator<String> litr = al.listIterator();
24. while(litr.hasNext()) {
25. String element = litr.next();
26.litr.set(element + "+"); }
27.System.out.print("Modified contents of al: ");
28. itr = al.iterator();
29. while(itr.hasNext()) {
30. String element = itr.next();
```

```
31. System.out.print(element + " ");
32. }
33. System.out.println();
34. // Now, display the list backwards.
35. System.out.print("Modified list backwards: ");
36. while(litr.hasPrevious()) {
37.  String element = litr.previous();
38. System.out.print(element + " ");
39.  }
40.  System.out.println();
```

Output:

- Original contents of al: C A E B D F
- Modified contents of al: C+ A+ E+ B+ D+ F+
- Modified list backwards: F+ D+ B+ E+ A+ C+

Second approach to access collection:

2. For-Each Alternative to Iterators :

It can be used when:

- If you wont obtain the elements in reverse order
- If you wont modify the elements of collection

To cycling through a collection than is using an iterator

Example:

```
1. // Use the for-each for loop to cycle through a collection.
2. import java.util.*;
3. class ForEachDemo {
4.  public static void main(String args[]) {
5. // Create an array list for integers.
6.  ArrayList<Integer> vals = new ArrayList<Integer>();
```

```
7.// Add values to the array list.
8.vals.add(1);
9.vals.add(2);
10. vals.add(3);
11. vals.add(4);
12. vals.add(5);
13.// Use for loop to display the values.
14.System.out.print("Original contents of vals: ");
15. for(int v : vals)
16. System.out.print(v + " ");
17.System.out.println();
18.// Now, sum the values by using a for loop.
19. int sum = 0;
20.for(int v : vals)
21. sum += v;
22.System.out.println("Sum of values: " + sum);
```

Output:

```
Original contents of vals: 1 2 3 4 5
```

28.Expalin Storing User-Defined Classes in Collections

- It stores built-in objects , such as String or Integer, in a collection.
- They can also store any type of object, including objects of classes that you create

Example:

```
// A simple mailing list example.
2. import java.util.*;
3. class Address {
4. private String name;
5. private String street;
6. private String city;
7. private String state;
8. private String code;
9.Address(String n, String s, String c, String st, String cd) {
10. name = n;
11.  street = s;
12. city = c;
13. state = st;
14. code = cd;
15. }
```

```
16.public String toString() {
17. return name + "\n" + street + "\n" + city + " " + state + " " + code;
18.  }
19. }
20.class MailList {
21.  public static void main(String args[]) {
22. LinkedList<Address> ml = new LinkedList<Address>();
23. // Add elements to the linked list.
24. ml.add(new Address("J.W. West", "11 Oak Ave", "Urbana", "IL",
"61801"));
25.  ml.add(new Address("Ralph Baker", "1142 Maple Lane",
"Mahomet", "IL", "61853"));
26. // Display the mailing list.
27. for(Address element : ml)
28. System.out.println(element + "\n");
29. System.out.println();
```

Output:

J.W. West

11 Oak Ave

Urbana IL 61801

Ralph Baker

1142 Maple Lane

Mahomet IL 61853