THE COLLECTION FRAMEWORK

CHAPTER 23

How can we handle a group of elements? This is a very easy question. We use an array to store a group of elements and handle them easily. OK, then how can we handle a group of objects? Can we use an array to store a group of objects? Yes, It is possible to use an array to store a group of objects.

Using an Array to Store a Group of Objects

It is possible to store a group of objects into an array. Let us take an example where we want to store 100 objects of `Employee` class into an array. For this purpose, we need to create an array of `Employee` type as:

```
Employee arr[ ] = new Employee[100];
```

This array can store 100 Employee objects. This can be achieved using a single loop as:

```
for(int i=0; i<100; i++)
{
    arr[i] = new Employee(data);
}
```

In Program 1, we create an `Employee` class with id and name details. Next, in the `Group` class, we create an array of `Employee` type with size 5. We accept 5 objects data from the keyboard and store the data into the array using a loop. Then we display the data once again by reading it from the array. In this program, the array `arr[ ]` is storing the Employee object references. So `arr[0]` represents first object reference, `arr[1]` represents second object reference, and so on.

Program 1: Write a program to store a group of objects into an array and retrieve the object data and display.

```
//To store and a group of objects in an array
import java.io.*;
class Employee
{
    //instance vars
    int id;
    String name;

    //to store data
    Employee(int i, String n)
```

```
        {
                id = i;
                name = n;
        }

        //a method to display data
        void displayData()
        {
                System.out.println(id+"\t"+ name);
        }
}

class Group
{
        public static void main(String args[ ]) throws IOException
        {
                //to accept data from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));


                //create Employee type array with size 5
                Employee arr[ ] = new Employee[5];

                //store 5 employees' data into the array
                for(int i=0; i<5; i++)
                {
                        System.out.print("Enter id: ");
                        int id = Integer.parseInt(br.readLine());

                        System.out.print("Enter name: ");
                        String name = br.readLine();

                        arr[i] = new Employee(id, name);
                }

                System.out.println("\nThe employee data is: ");

                //display the Employee data from the array
                for(int i=0; i<arr.length; i++)
                {
                        arr[i].displayData();
                }

        }
}
```

Output:

```
C:\> javac Group. java
C:\> java Group
Enter id: 10
Enter name: Nagesh
Enter id: 11
Enter name: Vijaya
Enter id: 22
Enter name: Ganesh
Enter id: 33
Enter name: Kumar
Enter id: 20
Enter name: Chandu

The employee data is:
10              Nagesh
11              Vijaya
22              Ganesh
33              Kumar
```

20 Chandu

So, here we have seen how to store a group of objects into an array and retrieve them again easily. But there are also certain inconveniences in this mechanism. They are as follows:

- We cannot store different class objects into the same array. The reason is that an array can store only one data type of elements.
- Adding the objects at the end of an array is easy. But, inserting and deleting the elements in the middle of the array is difficult. In this case, we have to re-arrange all the elements of the array.
- Retrieving the elements from an array is easy but after retrieving the elements, if we want to process them, then there are no methods available to carry out this.

Due to these problems, programmers want a better mechanism to store a group of objects. The alternative is using an object to store a group of other objects. It means that we can use a class object as an array. Such an object is called 'collection object' or 'container object'.

Collection Objects

A collection object or a container object is an object which can store a group of other objects. In Figure 23.1, we are using a collection object to store 4 objects. A collection object has a class called as 'collection class' or 'container class'. All the collection classes are available in the package—`java.util`. (Util stands for utility). A group of collection classes is called a 'collection framework'.

Important Interview Question

what is a collection framework?

A collection framework is a class library to handle groups of objects. Collection framework is implemented in java.util package.

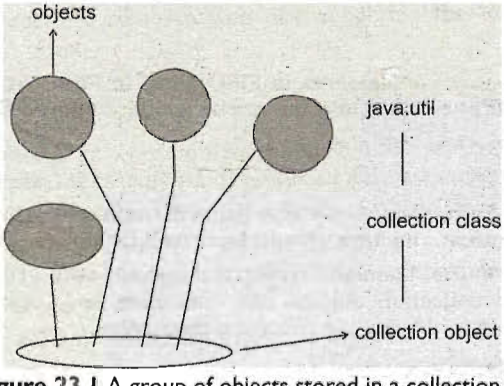

Figure 23.1 A group of objects stored in a collection object

In fact, collection object does not store the physical copies of other objects. Since the other objects are already available in memory, storing another copy of them into the collection object would be a mere wasting of memory. So, JVM does not store the copies of other objects, it simply stores the references of other objects into a collection object.

Important Interview Question

Does a collection object store copies of other objects or their references?

A collection object stores references of other objects.

All the collection classes in `java.util` package are the implementation classes of differe interfaces as shown in the Table 23.1.

Table 23.1

Interface type	Implementation classes
Set<T>	HashSet<T> LinkedHashSet<T>
List<T>	Stack<T> LinkedList<T> ArrayList<T> Vector<T>
Queue<T>	LinkedList<T>
Map<K,V>	HashMap<K,V> Hashtable<K,V>

Sets

A set represents a group of elements arranged just like an array. The set will grow dynamica when the elements are stored into it. A set will not allow duplicate elements. If we try to pass t same element that is already available in the set, then it is not stored into the set.

Lists

Lists are like sets. They store a group of elements. But lists allow duplicate values to be stored.

Queues

A Queue represents arrangement of elements in FIFO (First In First Out) order. This means that element that is stored as a first element into the queue will be removed first from the queue.

Maps

Maps store elements in the form of key and value pairs. If the key is provided then its correspond value can be obtained. Of course, the keys should have unique values.

Remember, in all the cases the 'elements' refer to 'objects' only. This means we cannot st primitive data types in the collection objects. We can store only objects since the main aim collections is to handle objects only, not the primitive data types.

Important Interview Question

Can you store a primitive data type into a collection?

No, Collections store only objects.

Retrieving Elements from Collections

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

- Using `for-each` loop.
- Using `Iterator` interface.
- Using `ListIterator` interface.
- Using `Enumeration` interface.

for-each Loop

`for-each` loop is like `for` loop which repeatedly executes a group of statements for each element of the collection. The format is:

```
for(variable: collection-object)
{
    Statements;
}
```

Here, the variable assumes each element of the collection-object and the loop is executed as many times as there are number of elements in the collection-object. If collection-object has `n` elements the loop is executed exactly `n` times and the variable stores each element in each step.

Iterator Interface

`Iterator` is an interface that contains methods to retrieve the elements one by one from a collection object. It has 3 methods:

- `boolean hasNext()`: This method returns true if the iterator has more elements.
- `element next()`: This method returns the next element in the iterator.
- `void remove()`: This method removes from the collection the last element returned by the iterator.

ListIterator Interface

`ListIterator` is an interface that contains methods to retrieve the elements from a collection object, both in forward and reverse directions. It has the following important methods:

- `boolean hasNext()`: This returns true if the ListIterator has more elements when traversing the list in the forward direction.
- `boolean hasPrevious()`: 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 element that was returned by the `next()` or `previous()` methods.

Important Interview Question

What is the difference between Iterator and ListIterator?

Both are useful to retrieve elements from a collection. Iterator can retrieve the elements only in forward direction. But ListIterator can retrieve the elements in forward and backward direction also. So ListIterator is preferred to Iterator.

Enumeration Interface

This interface is useful to retrieve one by one the eleme

- `boolean hasMoreElements()`: This method test or not.
- `element nextElement()`: This returns the next e

Important Interview Question

What is the difference between Iterator and Enumeration

Both are useful to retrieve elements from a collection. I follow and Enumeration methods are difficult to reme elements from the collection which is not available Enumeration.

HashSet Class

A `HashSet` represents a set of elements (objects). It do does not allow the duplicate elements to be stored.

We can write the `HashSet` class as:

```
class HashSet<T>
```

Here, `<T>` represents the generic type parameter. It stored into the `HashSet`. Suppose, we want to create a can create the object as:

```
HashSet<String> hs = new HashSet<String>
```

The following constructors are available in `HashSet`:

- `HashSet();`
- `HashSet(int capacity);`

Here, `capacity` represents how many elements ca capacity may increase automatically when more numl

- `HashSet(int capacity, float loadfactor);`

Here, `loadfactor` determines the point where th internally. For example, the product of `capacity` an after storing the 50th element into the `HashSet`, accommodate more elements. The default initial capa

HashSet Class Methods

`HashSet` class provides the following methods:

- `boolean add(obj)`: This method adds an ele the element is added to the `HashSet`, else it r available in the `HashSet`, then the present elem
- `boolean remove(obj)`: This method removes present. It returns true if the element is removed
- `void clear()`: This removes all the elements fr

- `boolean contains(obj)` : This returns true if the `HashSet` contains the specified element `obj`.
- `boolean isEmpty()` : This returns true if the `HashSet` contains no elements.
- `int size()` : This returns the number of elements present in the `HashSet`.

Let us take a program to understand how to construct a `HashSet` with String type elements and how to retrieve the elements using an Iterator. In this program, we store "America" and the same string is stored second time into the `HashSet`. The set will not store it second time, since a set will not allow the duplicate data. Also we can observe that the set will not maintain the same order of elements as in which they were entered.

Program 2: Write a program which shows the use of HashSet and Iterator.

```
//HashSet demo
import java.util.*;
class HS
{
      public static void main(String args[ ])
      {
            //create a HashSet to store strings
            HashSet<String> hs = new HashSet<String>();

            //store some string elements
            hs.add("India");
            hs.add("America");
            hs.add("Japan");
            hs.add("China");
            hs.add("America");

            //view the HashSet
            System.out.println("Hash set = "+ hs);

            //add an Iterator to hs.
            Iterator it = hs.iterator();

            //display element by element using Iterator
            System.out.println("Elements using Iterator: ");
            while(it.hasNext())
            {
                  String s = (String)it.next();
                  System.out.println(s);
            }
      }
}
```

Output:

```
C:\> javac HS.java
C:\> java HS
Hash set = [America, China, Japan, India]
Elements using Iterator:
America
China
Japan
India
```

In this program, we store a group of strings into a HashSet. Then we retrieve the elements one by one using an Iterator.

LinkedHashSet Class

This is a subclass of `HashSet` class and does not contain any additional members on its own. It is a generic class that has the declaration:

```
class LinkedHashSet<T>
```

Here, <T> represents the generic type parameter. It represents the data type of elements stored into the LinkedHashSet. LinkedHashSet internally uses a linked list to store the elements.

Stack Class

A stack represents a group of elements stored in LIFO (Last In First Out) order. This means the element which is stored as a last element into the stack will be the first element to be removed from the stack. Inserting elements (objects) into the stack is called 'push operation' and removing elements from stack is called 'pop operation'. Searching for an element in the stack is called 'peek operation'. Insertion and deletion of elements take place only from one side of the stack, called top of the stack, as shown in Figure 23.2.

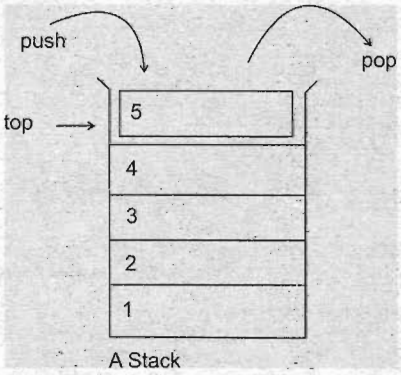

Figure 23.2 A stack with some elements

A pile of plates in a cafeteria where the lastly washed plate will be coming out first can be an example for a stack. For example, if we take the top plate (the 3rd plate) from the weight on the spring will be lessened and the next plate (2nd one) will come up. Similarly, a Disk holder where the CDs are arranged such that the last CD is available first is also an for a stack (Figure 23.3). This means if the objects are logically arranged in the form of the CDs in the memory, it becomes a stack. Generally, stacks are used to evaluate the expression ax^2+bx+c, and rarely for storing data into memory.

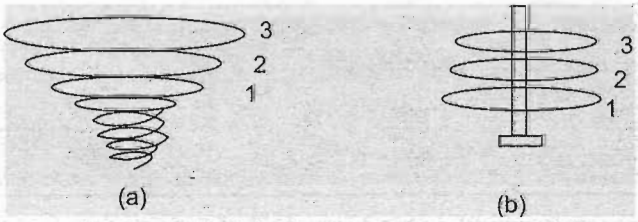

Figure 23.3 (a) A pile of plates (b) A group of CDs

We can write a Stack class as:

```
class Stack<E>
```

where E stands for element type. Suppose, we want to create a Stack object that contains objects, we can do so as shown here:

```
Stack<Integer> obj = new Stack<Integer>();
```

Stack Class Methods

`Stack class` includes the following methods:

- `boolean empty()`: This method tests whether the stack is empty or not. If the stack is empty then true is returned otherwise false.
- `element peek()`: This method returns the top-most object from the stack without removing it.
- `element pop()`: This method pops the top-most element from the stack and returns it.
- `element push(element obj)`: This method pushes an element `obj` onto the top of the stack and returns that element.
- `int search(Object obj)`: This method returns the position of an element `obj` from the top of the stack. If the element (object) is not found in the stack then it returns -1.

Let us write a program to perform different operations on a stack. First, we want to create a stack that can store Integer type objects as:

```
Stack<Integer> st = new Stack<Integer>();
```

Suppose, in this stack we want to store some integer elements. Since `int` is a primitive data type, we should convert the `int` values into Integer objects and then store into the stack. For this purpose, we can write a statement as:

```
st.push(element);
```

Here, we can pass `int` type element to `push()` method. But the method automatically converts it into Integer object and then stores it into the stack `st`. This is called 'auto boxing'.

Important Interview Question

What is auto boxing?

Converting a primitive data type into an object form automatically is called 'auto boxing'. Auto boxing is done in generic types.

To remove the top-most element (object) from the stack, we can use `pop()` method, as:

```
Integer obj = st.pop();
```

The code to search the position of an element in the stack is as:

```
position = st.search(element);
```

Program 3: Write a program to perform different operations on a stack through a menu.

```
//Pushing, popping, searching elements in a stack.
import java.io.*;
import java.util.*;

class StackDemo
{
      public static void main(String args[]) throws Exception
      {
            //create an empty stack to contain Integer objects
            Stack<Integer> st = new Stack<Integer>();
```

```
                //take vars
                int choice =0;
                int position,element;
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                //display the menu as long as user choice < 4
                while(choice<4)
                {
                        System.out.println("STACK OPERATIONS");
                        System.out.println("1 Push an element");
                        System.out.println("2 Pop an element");
                        System.out.println("3 Search an element");
                        System.out.println("4 Exit");
                        System.out.print("Your choice: ");

                        choice = Integer.parseInt(br.readLine());

                        //perform a task depending on user choice
                        switch(choice)
                        {
                                case 1: System.out.print("Enter element: ");
                                        element = Integer.parseInt(br.readLine());
                                        //int type element is converted into Intege
                                        object and
                                        //then pushed into the stack
                                        st.push(element);
                                        break;

                                case 2: //the top-most Integer object is popped
                                        Integer obj = st.pop();
                                        System.out.println("Popped= "+ obj);
                                        break;

                                case 3: System.out.print("Which element? ");
                                        element = Integer.parseInt(br.readLine());
                                        //int type element is converted into Intege
                                        object and
                                        //then searched in the stack
                                        position = st.search(element);
                                        if(position == -1)
                                        System.out.println("Element not found");
                                        else System.out.println("Position:
                                        "+position);
                                        break;

                                        default://come out if user choice is other
                                        than 1,2 or 3
                                        return;
                        }
                        //view the contents of stack
                        System.out.println("Stack contents: "+st);

                }
        }
}
```

Output:

```
C:\> javac StackDemo.java
C:\> java StackDemo
STACK OPERATIONS
1 Push an element
2 Pop an element
3 Search an element
4 Exit
```

```
Your choice: 1
Enter element: 11
Stack contents: [11]
STACK OPERATIONS
1 Push an element
2 Pop an element
3 Search an element
4 Exit
Your choice: 1
Enter element: 20
Stack contents: [11, 20]
STACK OPERATIONS
1 Push an element
2 Pop an element
3 Search an element
4 Exit
Your choice: 3
Which element? 20
Position: 1
Stack contents: [11, 20]
STACK OPERATIONS
```

LinkedList Class

A Linked list contains a group of elements in the form of nodes. Each node will have three fields—the data field contains data and the link fields contain references to previous and next nodes. Figure 23.4 displays that how the traversing in the linked list is done using these link fields.

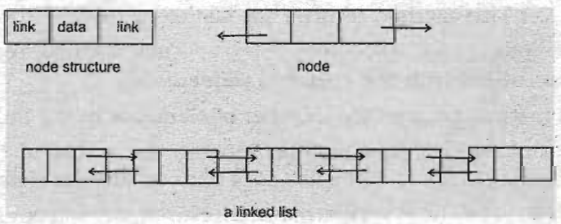

Figure 23.4 Node structure and a linked list with nodes

Linked list is very convenient to store data. Inserting the elements into the linked list and removing the elements from the linked list is done quickly and takes the same amount of time.

Important Interview Question

What is the difference between a Stack and LinkedList?

1. A Stack is generally used for the purpose of evaluation of expressions. A LinkedList is used to store and retrieve data.

2. Insertion and deletion of elements only from the top of the Stack is possible. Insertion and deletion of elements from any where is possible in case of a LinkedList.

A linked list is written in the form of:

```
class LinkedList<E>
```

We can create an empty linked list for storing String type elements (objects) as:

```
LinkedList<String> ll = new LinkedList<String>();
```

LinkedList Class Methods

`LinkedList` class includes the following methods:

- `boolean add(element obj)`: This method adds an element to the linked list. It returns tru the element is added successfully.
- `void add(int position, element obj)`: This method inserts an element `obj` into the lin list at a specified position.
- `void addFirst(element obj)`: This method adds the element `obj` at the first position of linked list.
- `void addLast(element obj)`: This method appends the specified element to the end of linked list.
- `element removeFirst()`: This method removes the first element from the linked list returns it.
- `element removeLast()`: This method removes the last element from the linked list returns it.
- `element remove(int position)`: This method removes an element at the specified posi in the linked list.
- `void clear()`: This method removes all the elements from the linked list.
- `element get(int position)`: This method returns the element at the specified positio the linked list.
- `element getFirst()`: This method returns the first element from the list.
- `element getLast()`: This method returns the last element from the list.
- `element set(int position, element obj)`: This method replaces the element at specified position in the list with the specified element `obj`.
- `int size()`: This method returns the number of elements in the linked list.
- `int indexOf(Object obj)`: This method returns the index of the first occurrence of specified element in the list, or -1 if the list does not contain the element.
- `int lastIndexOf(Object obj)`: This method returns the index of the last occurrence of specified element in the list, or -1 if the list does not contain the element.
- `Object[ ] toArray()`: This method converts the linked list into an array of Object class t All the elements of the linked list will be stored into the array in the same sequence.

Let us create a linked list to store a group of strings and perform some important operation on list. In Program 4, we create a linked list as:

```
LinkedList<String> ll= new LinkedList<String>();
```

Now, we can add any string type elements to the list using `add()` method, as:

```
ll.add(position-1,element);
```

In case of linked list, the counting will start from 0 and we start counting from 1. Hence deducted 1 from the position number to get the position number maintained by the list.

To remove a particular element, we can use `remove()` method, as:

```
ll.remove(position-1);
```

To replace an existing element with a new element, the `set()` method can be used as:

```
ll.set(position-1,element);
```

Program 4: Write a program that shows the use of `LinkedList` class

```
//A LinkedList with strings
import java.io.*;
import java.util.*;
class LLDemo
{
      public static void main(String args[ ]) throws IOException
      {
            //create an empty linked list to store strings
            LinkedList<String> ll= new LinkedList<String>();

            //add some names to linked list
            ll.add("America");
            ll.add("India");
            ll.add("Japan");

            //display the elements in the linked list
            System.out.println("List= "+ ll);

            //vars
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            String element;
            int position, choice=0;

            //menu

            while(choice<4)
            {
                  System.out.println("\nLINKEDLIST OPERATIONS");
                  System.out.println("1 Add an element");
                  System.out.println("2 Remove an element");
                  System.out.println("3 Change an element");
                  System.out.println("4 Exit");

                  System.out.print("Your choice: ");
                  choice = Integer.parseInt(br.readLine());

                  //perform a task depending on user choice
                  switch(choice)
                  {
                        case 1: System.out.print("Enter element: ");
                              element = br.readLine();
                              System.out.print("At what position? ");
                              position= Integer.parseInt(br.readLine());
                              ll.add(position-1,element);
                              break;

                        case 2: System.out.print("Enter position: ");
                              position= Integer.parseInt(br.readLine());
                              ll.remove(position-1);
                              break;

                        case 3: System.out.print("Enter position: ");
                              position= Integer.parseInt(br.readLine());
                              System.out.print("Enter new element: ");
                              element= br.readLine();
                              ll.set(position-1,element);
                              break;

                              default: return;
```

```
                        }

                        //Use Iterator to retrieve the elements
                        System.out.print("List= ");
                        Iterator it = ll.iterator();
                        while(it.hasNext())
                        System.out.print(it.next()+" ");

                }//end of while
        }
}
```

Output:

```
C:\> javac LLDemo.java
C:\> java LLDemo
List= [America, India, Japan]

LINKEDLIST OPERATIONS
1 Add an element
2 Remove an element
3 Change an element
4 Exit
Your choice: 1
Enter element: China
At what position? 1
List= China America India Japan
LINKEDLIST OPERATIONS
1 Add an element
2 Remove an element
3 Change an element
4 Exit
Your choice: 2
Enter position: 2
List= China India Japan
LINKEDLIST OPERATIONS
```

In this program, we create a linked list with some strings. Then we display a menu to the user to opt any option he wants and the specified operation is carried on the list.

ArrayList Class

An ArrayList is like an array, which can grow in memory dynamically. It means that when we store elements into the ArrayList, depending on the number of elements, the memory is dynamically allotted and re-allotted to accommodate all the elements. ArrayList is not synchronized. This means that when more than one thread acts simultaneously on the ArrayList object, the results may be incorrect in some cases. Please see the chapter on Threads for synchronization concepts.

The `ArrayList` class can be written as:

```
class ArrayList<E>
```

where `E` represents the type of elements to be stored into the ArrayList. For example, to store String type elements, we can create an object to ArrayList as:

```
ArrayList<String> arl = new ArrayList<String>();
```

The preceding statement constructs an ArrayList with a default initial capacity of 10. We can also mention the capacity at the time of creating ArrayList object as:

```
ArrayList<Double>arl = new ArrayList<Double>(101);
```

The preceding ArrayList can store Double type objects and initial capacity is declared to be 101.

ArrayList Class Methods

`ArrayList` class includes the following methods:

- `boolean add(element obj)`: This method appends the specified element to the end of the ArrayList. If the element is added successfully then the preceding method returns true.
- `void add(int position, element obj)`: This method inserts the specified element at the specified position in the ArrayList.
- `element remove(int position)`: This method removes the element at the specified position in the ArrayList. This method also returns the element which was removed from the ArrayList.
- `boolean remove(Object obj)`: This method removes the first occurrence of the specified element `obj` from the ArrayList, if it is present.
- `void clear()`: This method removes all the elements from the ArrayList.
- `element set(int position, element obj)`: This method replaces an element at the specified position in the ArrayList with the specified element `obj`.
- `boolean contains(Object obj)`: This method returns true if the ArrayList contains the specified element `obj`.
- `element get(int position)`: This method returns the element available at the specified position in the ArrayList.
- `int indexOf(Object obj)`: This method returns the position of the first occurrence of the specified element `obj` in the list, or -1 if the element is not found in the list.
- `int lastIndexOf(Object obj)`: This method returns the position of the last occurrence of the specified element `obj` in the list, or -1 if the element is not found in the list.
- `int size()`: This method returns the number of elements present in the ArrayList.
- `Object[ ] toArray()`: This method returns an Object class type array containing all the elements in the ArrayList in proper sequence.

Let us write a program to understand the way to create an ArrayList with some String type elements and perform various operations on it. For example, we can add the elements to ArrayList using:

```
arl.add("Apple");
```

We can remove 3rd element using `remove()` method as:

```
Arl.remove(3);
```

We can use an Iterator or ListIterator to retrieve elements from the ArrayList. These things are demonstrated in Program 5.

Program 5: Write a program to create an ArrayList with strings and perform various operations on it.

```
//ArrayList with String objects
import java.util.*;
class ArrayListDemo
{
      public static void main(String args[ ])
      {
            //create ArrayList
            ArrayList<String> arl = new ArrayList<String>();

            //add four objects
            arl.add("Apple");
            arl.add("Mango");
            arl.add("Grapes");
            arl.add("Guava");

            //display contents
            System.out.println("Contents: "+arl);

            //remove two objects
            arl.remove(3);
            arl.remove("Apple");

            //display again
            System.out.println("Contents after Removing: "+arl);

            //display its size
            System.out.println("Size of ArrayList: "+arl.size());

            //extract elements using Iterator
            System.out.println("Extracting using Iterator:");

            //add an  Iterator to ArrayList to retrieve elements
            Iterator it = arl.iterator();

            while(it.hasNext())
            {
                  System.out.println(it.next());
            }

      }
}
```

Output:

```
C:\> javac ArrayListDemo.java
C:\> java ArrayListDemo
Contents: [Apple, Mango, Grapes, Guava]
Contents after Removing: [Mango, Grapes]
Size of ArrayList: 2
Extracting using Iterator
Mango
Grapes
```

Vector Class

A Vector also stores elements (objects) similar to ArrayList, but Vector is synchronized. It means even if several threads act on Vector object simultaneously, the results will be reliable.

We can write a Vector class as:

```
class Vector<E>
```

Here, E represents the type of elements stored into the Vector. For example, if we want to create an empty Vector that can be used to store Float type objects, we can write:

```
Vector<Float> v = new Vector<Float>();
```

The preceding statement creates a Vector object v which can be used to store Float type objects. The default capacity will be 10.

```
Vector<Integer> v = new Vector<Integer>(101);
```

This vector v can store Integer objects. The capacity of Vector is given as 101. Another way of creating a Vector object is by specifying a 'capacity increment' which specifies how much the capacity should be incremented when the Vector is full with elements.

```
Vector<Integer> v = new Vector<Integer>(101, 20);
```

Here, 101 is the capacity of the Vector and capacity increment is 20. Should the Vector is full with elements; the capacity automatically goes to 121.

Vector Class Methods

Vector class includes the following methods:

- boolean add(element obj): This method appends the specified element to the end of the Vector. If the element is added successfully then the preceding method returns true.
- void add(int position, element obj): This method inserts the specified element at the specified position in the Vector.
- element remove(int position): This method removes the element at the specified position in the Vector. This method also returns the element which was removed from the Vector.
- boolean remove(Object obj): This method removes the first occurrence of the specified element obj from the Vector, if it is present.
- void clear(): This method removes all the elements from the Vector.
- element set(int position, element obj): This method replaces an element at the specified position in the Vector with the specified element obj.
- boolean contains(Object obj): This method returns true if the Vector contains the specified element obj.
- element get(int position): This method returns the element available at the specified position in the Vector.
- int indexOf(Object obj): This method returns the position of the first occurrence of the specified element obj in the Vector, or -1 if the element is not found in the list.
- int lastIndexOf(Object obj): This method returns the position of the last occurrence of the specified element obj in the Vector, or -1 if the element is not found in the list.
- int size(): This method returns the number of elements present in the Vector.
- Object[ ] toArray(): This method returns an Object class type array containing all the elements in the Vector in proper sequence.

- `int capacity()`: This method returns the current capacity of the Vector.

To understand the Vector concept, let us take an example. In Program 6, we take a Vector to s Integer objects as:

```
Vector<Integer> v = new Vector<Integer>();
```

Our idea is to take an `int` type array `x[ ]` and store the elements of this array into the vector this case, since `x[ ]` has `int` type elements, we should convert them all to Integer type objects then store them into the vector `v`. This is done by the statement:

```
v.add(x[i]);
```

In fact, here we are storing `x[i]` directly into `v`. The Java compiler converts internally the `int` elements of `x[i]` into Integer type objects. This is called auto boxing.

Program 6: Write a program that shows the use of Vector Class.

```
//Creating a vector with Integer elements
import java.util.*;
class VectorDemo
{
      public static void main(String args[ ])
      {
            //take a vector to store Integer objects
            Vector<Integer> v = new Vector<Integer>();

            //take an int type array
            int x[ ]={22,20,10,40,15,60};

            //When x[i] is stored into v below. x[i] values are converted i
            //Integer objects and stored into v. This is auto boxing.
            for(int i=0; i<x.length; i++)
            {
                  v.add(x[i]);
            }

            //retrieve the elements using get()
            System.out.println("Vector elements: ");
            for(int i=0; i<v.size(); i++)
            {
                  System.out.println(v.get(i));
            }

            //retrieve using ListIterator
            System.out.println("Elements using ListIterator:");
            ListIterator lit = v.listIterator();

            System.out.println("In forward direction:");
            while(lit.hasNext())
            System.out.print(lit.next()+"\t");

            System.out.println("\nIn backward direction:");
            while(lit.hasPrevious())
            System.out.print(lit.previous()+"\t");

      }
}
```

Output:

```
C:\> javac VectorDemo.java
C:\> java VectorDemo
Vector elements:
22
20
10
40
15
60
Elements using ListIterator:
In forward direction:
22   20     10     40     15     60
In backward direction:
60   15     40     10     20     22
```

In this program, we create a Vector and store the elements of `x[i]` into the Vector. Then we retrieve the elements from Vector using `get()`. ListIterator is used later to retrieve the elements in forward and reverse directions from Vector.

Important Interview Question

What is the difference between ArrayList and Vector?

See Table 23.2.

Table 23.2

ArrayList	Vector
ArrayList object is not synchronized by default.	Vector object is synchronized by default.
In case of a single thread, using ArrayList is faster than the Vector.	In case of multiple threads, using Vector is advisable. With a single thread, Vector becomes slow.
ArrayList increases its size every time by 50 percent (half).	Vector increases its size every time by doubling it.

Important Interview Question

Can you synchronize the ArrayList object?

Yes, we can use synchronizedList() method to synchronize the ArrayList, as:

Collections.synchronizedList(new ArrayList());

HashMap Class

HashMap is a collection that stores elements in the form of key-value pairs. If key is provided later, its corresponding value can be easily retrieved from the HashMap. Keys should be unique. This means we cannot use duplicate data for keys in the HashMap. However, HashMap is not synchronized and hence while using multiple threads on HashMap object, we get unreliable results.

```
                                    case 3: //use keySet() to display the names
                                            //create HashSet object to store names an
                                            refer it by Set reference
                                            Set<String> set = new HashSet<String>();
                                            set = hm.keySet();
                                            System.out.println(set);
                                            break;

                                            case 4: return;
                                }
                            }

                        }
                }
```

Output:

```
C:\> javac HashMapDemo.java
C:\> java HashMapDemo
1 Enter Phone entries
2 Lookup in the book
3 Display Names in book
4 Exit
Your choice: 1
Enter name: Laxmi
Enter phno: 9866633445
1 Enter Phone entries
2 Lookup in the book
3 Display Names in book
4 Exit
Your choice: 1
Enter name: Ganesh
Enter phno: 22340056
1 Enter Phone entries
2 Lookup in the book
3 Display Names in book
4 Exit
Your choice: 3
[Laxmi, Ganesh]
1 Enter Phone entries
2 Lookup in the book
3 Display Names in book
4 Exit
Your choice: 2
Enter name: Laxmi
Phno: 9866633445
1 Enter Phone entries
2 Lookup in the book
3 Display Names in book
4 Exit
Your choice: 4
```

This program creates a HashMap to store name and phone number. When name is given, we get back the corresponding phone number.

Hashtable Class

Hashtable is similar to HashMap which can store elements in the form of key-value pairs. Hashtable is synchronized assuring proper results even if multiple threads act on it simultaneo

We can write Hashtable class as:

```
class Hashtable<K,V>
```

where K represents the type of key element and V represents the type of value element. For example, to store a String as key and an Integer object as its value, we can create the Hashtable as,

```
Hashtable<String, Integer> hm = new Hashtable<String, Integer>();
```

Here, we did not mention any capacity for the Hashtable. The default initial capacity of this Hashtable will be taken as 11 and the load factor as 0.75. Load factor represents at what level the HashMap capacity should be doubled. For example, the product of capacity and load factor = 11 * 0.75 = 8.25. This represents that after storing 8th key-value pair into the Hashtable, its capacity will become 22.

We can also create Hashtable in the following ways:

- `Hashtable<String, Integer> ht = new Hashtable<String, Integer>(81);`

Here, 81 is initial capacity of Hashtable

- `Hashtable<String, Integer> hm = new Hashtable<String, Integer>(81, 0.5);`

Here, 81 is initial capacity and 0.5 is load factor.

Hashtable Class Methods

`Hashtable` class includes the following methods:

- `value put(key, value)` : This method stores key-value pair into the Hashtable.
- `value get(Object key)` : This method returns the corresponding value when key is given. If the key does not have a value associated with it, then it returns null.
- `Set<K> keySet()` : This method, when applied on a Hashtable converts it into a Set where only keys will be stored.
- `Collection<V> values()` : This method, when applied on a Hashtable object returns all the values of the Hashtable into a Collection object.
- `value remove(Object key)` : This method removes the key and corresponding value from the Hashtable.
- `void clear()` : This method removes all the key-value pairs from the Hashtable.
- `boolean isEmpty()` : This method returns true if there are no key-value pairs in the Hashtable.
- `int size()` : This method returns the number of key-value pairs in the Hashtable.

In Program 8, we create a Hashtable with the names of cricket players and their scores. For this purpose, the Hashtable can be created as:

```
Hashtable<String, Integer> ht = new Hashtable<String, Integer>();
```

Then using `put()` method, we can store the key-value pairs. Here, key is the player name and the value is his score.

```
ht.put("Ajay", 50);
```

Here, the primitive `int` value 50 will be converted into Integer object and then stored into the Hashtable. To retrieve all the keys (names), we can use `keys()` method which returns the keys of the Hashtable into an Enumeration object.

```
Enumeration e = ht.keys();
```

Then using `get()` method, we can retrieve the score of a player.

```
Integer score= ht.get(name);
```

Program 8: Write a program that shows the use of HashTable class.

```
//Hashtable with cricket player names and their scores
import java.io.*;
import java.util.*;
class HashtableDemo
{
      public static void main(String args[ ])
      throws IOException
      {
            //create Hashtable with names and scores
            Hashtable<String, Integer> ht = new Hashtable<String, Integer>(
            ht.put("Ajay", 50);
            ht.put("Sachin", 77);
            ht.put("Gavaskar", 44);
            ht.put("Kapil", 60);
            ht.put("Dhoni", 88);

            //display all player names using enumerator
            System.out.println("The player names: ");
            Enumeration  e = ht.keys();
            while (e.hasMoreElements())
            System.out.println(e.nextElement());

            //accept player name from keyboard
            BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));
            System.out.print("Enter player name: ");
            String name= br.readLine();
            name= name.trim();  //remove unnecessary spaces

            //get score of the player
            Integer score= ht.get(name);
            if(score != null)
            {
                  //convert score from Integer object to int value
                  int sc=score.intValue();
                  System.out.println(name +" Scored: "+sc);
            }
            else System.out.println("Player not found");

      }
}
```

Output:

```
C:\> javac HashtableDemo.java
C:\> java HashtableDemo
The player names:
Ajay
Dhoni
Sachin
Gavaskar
Kapil
Enter player name: Sachin
Sachin Scored: 77
```

In this program, we are creating Hashtable that stores the cricket players' names as keys and scores as values.

Important Interview Question

What is the difference between HashMap and Hashtable ?

The difference between a HashMap and a Hashtable is given in Table 23.3

Table 23.3

HashMap	Hashtable
HashMap object is not synchronized by default.	Hashtable object is synchronized by default.
In case of a single thread, using HashMap is faster than the Hashtable.	In case of multiple threads, using Hashtable is advisable. With a single thread, Hashtable becomes slow.
HashMap allows null keys and null values to be stored.	Hashtable does not allow null keys or values.
Iterator in the HashMap is fail-fast. This means Iterator will produce exception if concurrent updates are made to the HashMap.	Enumeration for the Hashtable is not fail-fast. This means even if concurrent updations are done to Hashtable, there will not be any incorrect results produced by the Enumeration.

Important Interview Question

Can you make HashMap synchronized ?

Yes, we can make HashMap object synchronized using synchronizedMap() method as shown here:

Collections.synchronizedMap(new HashMap());

What is the difference between a Set and a List?

The difference between a Set and a List is given in Table 23.4.

Table 23.4

Set	List
A set represents a collection of elements. Order of the elements may change in the set.	A List represents ordered collection of elements. List preserves the order of elements in which they are entered.
Set will not allow duplicate values to be stored.	List will allow duplicate values.
Accessing elements by their index (position number) is not possible in case of sets.	Accessing elements by index is possible in lists.
Sets will not allow null elements.	Lists allow null elements to be stored.

Using Comparator to Sort an Array

`java.util` package offers an interface, called Comparator that is useful to impose a total order on a collection of elements. It can be used to sort the elements (objects) of an array into ascending order or descending order. Comparator is written as:

```
interface Comparator<T>
```

where `T` represents the type of elements (objects) compared by the Comparator. For example, to compare Integer objects, we can write a class that implements the Comparator as:

```
class Ascend implements Comparator<Integer>
```

The Comparator interface contains a method `compare()` that should be implemented in such a way that the two objects should be compared using `compareTo()` method in its body as:

```
public int compare(Integer i1, Integer i2)
{
     return i1.compareTo(i2);
}
```

Here, the `compareTo()` method returns a positive number if `i1>i2` and a negative value if `i1<i2`. It returns `0` if `i1==i2`. This logic can be applied to a group of objects when we want to arrange them in ascending order. Similarly, the reverse order is needed to sort the elements in descending order. Now the Ascend class object should be passed to Arrays class `sort()` method as:

```
Arrays.sort(arr, new Ascend());
```

Then all the elements (objects) of the array `arr[ ]` will be sorted into ascending order. In Program 10, we want to sort a group of Integer objects. The objects are first stored into an array `arr[ ]`. Then using Comparator, we compare only two objects of the array and decide the order. In Ascend class we write the code of ascending order and in Descend class, we write the code for descending order. Then the objects of Ascend class and Descend classes are passed to the `sort()` method of Arrays class as:

```
Arrays.sort(arr, new Ascend());  //for sorting into ascending order
Arrays.sort(arr, new Descend());  // for sorting into descending order
```

Program 10: Write a program that shows sorting using comparator.

```
//Sorting an array with a group of Integer objects
import java.io.*;
import java.util.*;

//to sort into ascending order
 class Ascend implements Comparator<Integer>
{
     public int compare(Integer i1, Integer i2)
     {
          return i1.compareTo(i2);
     }
}

//to sort into descending order
 class Descend implements Comparator<Integer>
{
     public int compare(Integer i1, Integer i2)
```

```
        {
                return i2.compareTo(i1);
        }
}

class Arrays1
{
        public static void main(String args[ ])
        throws IOException
        {

                //to accept array elements from keyboard
                BufferedReader br = new BufferedReader(new
                InputStreamReader(System.in));

                System.out.print("How many elements? ");
                int size = Integer.parseInt(br.readLine());

                //create an array to store Integer type objects.
                Integer arr[ ] = new Integer[size];

                //Below, we pass int values to the array but they are
                //converted into Intege objects and then stored
                for(int i=0; i<size; i++)
                {
                        System.out.print("Enter int: ");
                        arr[i]= Integer.parseInt(br.readLine());
                }

                //sort the array in ascending order
                Arrays.sort(arr, new Ascend());

                //display the sorted array
                System.out.println("\nSorted in Ascending order: ");
                display(arr);

                //in descending order
                Arrays.sort(arr, new Descend());
                System.out.println("\nSorted in Descending order: ");
                display(arr);
        }

        static void display(Integer arr[ ])
        {
                for(Integer i: arr)
                System.out.print(i+"\t");

        }
}
```

Output:

```
C:\> javac Arrays1.java
C:\> java Arrays1
How many elements? 5
Enter int: 55
Enter int: 60
Enter int: 12
Enter int: 30
Enter int: 12

Sorted in Ascending order:
12      12      30      55      60
Sorted in Descending order:
60      55      30      12      12
```

In this program, we are demonstrating, how to sort objects into ascending and descending orde using Comparator.

StringTokenizer Class

This class is useful to break a string into pieces, called 'tokens'. These tokens are then stored in th StringTokenizer object from where they can be retrieved. The code to create an object StringTokenizer class is:

```
StringTokenizer st = new StringTokenizer(str, "delimiter");
```

In the preceding statement, the actual string `str` is broken into pieces at the positions marked by group of characters, called 'delimiters'. For example, to break the string wherever a comma is foun we can write:

```
StringTokenizer st = new StringTokenizer(str, ",");
```

Similarly, to break the string wherever a comma or colon or both are found, we can use:

```
StringTokenizer st = new StringTokenizer(str, ",:");
```

StringTokenizer Class Methods

`StringTokenizer` class includes the following methods:

- `int countTokens()`: This method counts and returns the number of tokens available in StringTokenizer object.
- `boolean hasMoreTokens()`: This method tests if there are more tokens available in th StringTokenizer object or not. If next token is there then it returns true.
- `String nextToken()`: This method returns the next token from the StringTokenizer.

Program 11: Write a program that shows the use of StringTokenizer object.

```
//A string broken into pieces at spaces
import java.util.*;
class STDemo
{
      public static void main(String args[ ])
      {
            //take a string
            String str = "He is a gentle man";

            //break into tokens at spaces. Here delimiter is a space
            StringTokenizer st = new StringTokenizer(str," ");

            //retrieve tokens from st and display
            System.out.println("The tokens are:");

            while(st.hasMoreTokens())
            {
                  String one=st.nextToken();
                  System.out.println(one);
            }
      }
}
```

Output:

```
C:\> javac STDemo.java
C:\> java STDemo
The tokens are:
He
is
a
gentle
man
```

In this program, we take a string and break it into tokens wherever a space is found in the string. The tokens are then retrieved from StringTokenizer object and then displayed.

Calendar Class

Calender class is useful in two ways:

- It helps in knowing the system date and time.
- It helps in storing a date and time value so that it can be transported to some other application. To create an object to Calendar class, we can write:

```
Calendar cl = Calendar.getInstance();
```

Then the object `cl` is created to Calendar object which stores the current system date and time by default.

Calendar Class Methods

`Calendar class` includes the following methods:

- `int get(int field)`: This method returns the value of the given Calendar field. For example,
- `Calendar.DATE` gives the date number from 1 to 31.
- `Calendar.MONTH` gives the month number from 0 to 11. (January is taken as 0)
- `Calendar.YEAR` gives the year number.
- `Calendar.HOUR` gives the hour number from 0 to 11.
- `Calendar.MINUTE` gives the minute number from 0 to 59.
- `Calendar.SECOND` gives the second number from 0 to 59.
- `Calendar. AM_PM` gives 0 if it is AM. Gives 1 if it is PM.
- `void set(int field, int value)`

This method sets the given field in Calendar object to the given value. For example, we can set a date like 15th March 2007 to Calendar object as:

```
cl.set(Calendar.DATE, 15);
cl.set(Calendar.MONTH, 2);
cl.set(Calendar.YEAR, 2007);
```

- `String toString()`: This method returns the String representation of the Calendar object.
- `boolean equals(Object obj)`: This method compares the Calendar object with another object `obj` and returns true if they are same, otherwise false.

Program 12: Write a program that shows the use of Calendar class.

```
//To display System date and time
import java.util.*;

class CalendarDemo
{
      public static void main(String args[ ])
      {
            //create Calendar class object. By default it
            //contains the system date and time
            Calendar cl = Calendar.getInstance();

            //display date separately
            System.out.print("Current date: ");
            int dd= cl.get(Calendar.DATE);
            int mm= cl.get(Calendar.MONTH);
            ++mm;
            int yy= cl.get(Calendar.YEAR);
            System.out.println(dd+ "-" + mm + "-" +yy);

            //display time alone
            System.out.print("Current time: ");
            int h= cl.get(Calendar.HOUR);
            int m= cl.get(Calendar.MINUTE);
            int s= cl.get(Calendar.SECOND);
            System.out.println(h+ ":" + m +":" +s);

            int x = cl.get(Calendar.AM_PM);
            if(x == 0) System.out.println("Good morning");
            else System.out.println("Good evening");


      }
}
```

Output:

```
C:\> javac CalendarDemo.java
C:\> java CalendarDemo
Current date: 19-9-2007
Current time: 11:9:10
Good morning
```

In this program, we create a Calendar class object and by default it contains the date and time shown by the local system. From the Calendar object, we get the date and time separately using `get()` method and display them.

Date Class

Date class is useful to display the date and time at a particular moment. When an object to Date class is created, it contains the system date and time by default. To create an object, we write:

```
Date d = new Date();
```

Now, `d` contains the system date and time. Once Date class object is created, it should be formatted using the following methods of DateFormat class of `java.text` package.

- `DateFormat fmt = DateFormat. getDateInstance(formatconst, region);`

This method is useful to store format information for date value into DateFormat object `fmt`.

- `DateFormat fmt = DateFormat. getTimeInstance(formatconst, region);`

This method is useful to store format information for time into DateFormat object `fmt`.

- DateFormat fmt = DateFormat.getDateTimeInstance(formatconst,formatconst,region);

This method stores the format information for date and time into the DateFormat class object `fmt`.

The DateFormat object `fmt` contains the formatting information which the programmer wants to format the date and time of Date class object. This can be done using `format()` method as:

```
String str = fmt.format(d);
```

The preceding `format()` method will apply the format which is in `fmt` object to the Date class object `d`. The formatted date and time will appear as a string in `str`. In the preceding DateFormat methods, we used 'formatconst' parameter which represents one of the following values:

```
DateFormat.FULL
DateFormat.LONG
DateFormat.MEDIUM
DateFormat.SHORT
```

And the 'region' parameter represents one of the following values for countries in the world: Locale.US, Locale.UK, Locale.US, Locale.UK, Locale.CHINA, Locale.CANADA, Locale.ITALY, Locale.JAPAN, Locale.KOREA, Locale.TAIWAN, Locale.FRANCE, Locale.GERMANY. All these are defined as constants in Locale class which is available in `java.util` package.

Please see Table 23.5 to understand how the 'formatconst' will modify the appearance of the Date class object for the 'region' Locale.UK.

Table 23.5

Formatconst	Example (region= Locale.UK)
DateFormat.FULL	03 September 2007 19:43:14 O'clock GMT + 05:30
DateFormat.LONG	03 September 2007 19:43:14 GMT + 05:30
DateFormat.MEDIUM	03-Sep-07 19:43:14
DateFormat.SHORT	03/09/07 19:43

Program 13: Write a program that shows the use of Date class.

```
//Display System date and time using Date class
import java.util.*;
import java.text.*;
class MyDate
{
      public static void main(String args[ ])
      {
            //Creat Date class object- this contains system date and time
            Date d = new Date();

            //Format the date to medium format and time to short format
            DateFormat fmt = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,
            DateFormat.SHORT, Locale.UK);

            //Apply the above format to the Date object
            String str = fmt.format(d);

            //Now display the formatted date and time
            System.out.println(str);
      }
```

```
}
```

Output:

```
C:\> javac MyDate.java
C:\> java MyDate
19-Sep-2007 23:46
```

In this program, first we create a Date class object and then format the object contents using DateFormat class method and finally display the formatted date and time in string form.

Let us see one application of collection objects to store and handle groups of objects. We take an ArrayList to hold a group of Employee class objects in Program 14.

Let us write create several Employee class objects and store them in an ArrayList. For this purpose, we can create the ArrayList object as:

```
ArrayList<Employee> arl = new ArrayList<Employee>();
```

Then store the Employee class object obj into the ArrayList as:

```
arl.add(obj);
```

Let us then search for a specific employee id in the ArrayList and pick up that employee details and display them. For this purpose, get() method of ArrayList can be used as:

```
Employee obj = arl.get(i);
```

In the preceding statement, by changing the i values, the objects from ArrayList can be retrieved one by one. After retrieving an Employee object, the id number is checked and if it is the one we are searching for, then the details of that employee will be displayed. This is seen in Program 14.

Program 14: An ArrayList handling a group of Employee class objects

```
//To create an ArrayList of Employee objects and search for a particular
Employee //object based on id number.

import java.io.*;
import java.util.*;
class Employee
{
      //take variables
      int id;
      String name;
      String address;

      //initialize them
      Employee(int i, String n, String a)
      {
            id = i;
            name = n;
            address = a;
      }

      //display employee details
      void display()
      {
            System.out.println("Id: "+ id);
            System.out.println("Name: "+ name);
            System.out.println("Address: "+ address);
      }
}
```

```
class EmpList
{
	public static void main(String args[ ]) throws IOException
	{
		//vars
		BufferedReader br = new BufferedReader(new
		InputStreamReader(System.in));
		int id;
		String name;
		String address;

		//create an ArrayList arl to store Employee objects
		ArrayList<Employee> arl = new ArrayList<Employee>();

		//accept 5 employee's details and store into arl
		for(int i=0; i<5; i++)
		{
			System.out.print("Enter id: ");
			id = Integer.parseInt(br.readLine());

			System.out.print("Enter name: ");
			name = br.readLine();

			System.out.print("Enter address: ");
			address = br.readLine();

			//create Employee object with accepted data
			Employee obj = new Employee(id, name, address);

			//store Employee object into arl
			arl.add(obj);
		}

		//Now search for an employee id
		System.out.print("Enter id to search: ");
		id = Integer.parseInt(br.readLine());

		//found becomes true if employee id is found in arl
		boolean found = false;

		//search all elements in arl
		for(int i=0; i<arl.size(); i++)
		{
			//get() method of ArrayList will return i-th Employee
			object
			Employee obj = arl.get(i);

			//check if given id is equal to id of Employee object
			if(id == obj.id)
			{
				obj.display();  //display that Employee data
				found = true;
			}
		}

		if(!found)
		System.out.println("Employee not found");

	}
}
```

Output:

```
C:\> javac EmpList.java
C:\> java EmpList
Enter id: 10
```

```
Enter name: Vijaya
Enter address: Hyderabad
Enter id: 11
Enter name: Kumar
Enter address: New Delhi
Enter id: 12
Enter name: Ganesh
Enter address: Mumbai
Enter id: 13
Enter name: Nehru
Enter address: Chennai
Enter id: 14
Enter name: Chandana
Enter address: Kolkata
Enter id to search: 13
Id: 13
Name: Nehru
Address: Chennai
```

Study Program 14 keenly and observe Figure 23.5 to understand the logic used in the program.

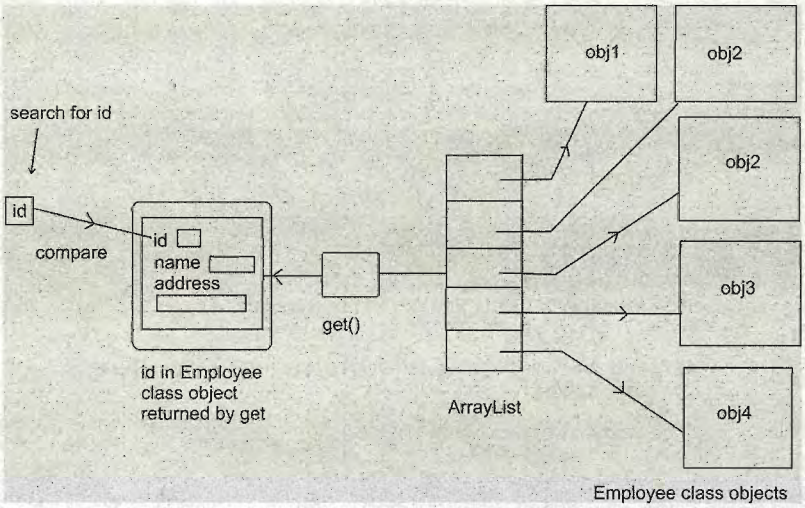

Figure 23.5 Retrieving objects using get() from an ArrayList

Conclusion

Collection framework contains classes exclusively to handle groups of objects. This is very essenti because all the applications which are designed in OOPS extensively interchange data in the form objects. Since generic classes are useful to handle any class type objects, all the collection classes `java.util` package are written as generic classes. So they can handle any class type objects. Ea collection class has a different mechanism of storing data in memory and has methods to perfo various operations on the data. The programmer has to exercise discretion as to which collection helpful to properly handle the data which he has planned in his software.