JAVA

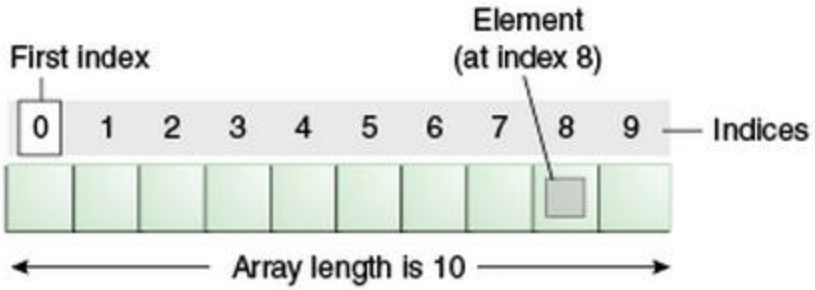
Agenda

- Array
- Collection

Array

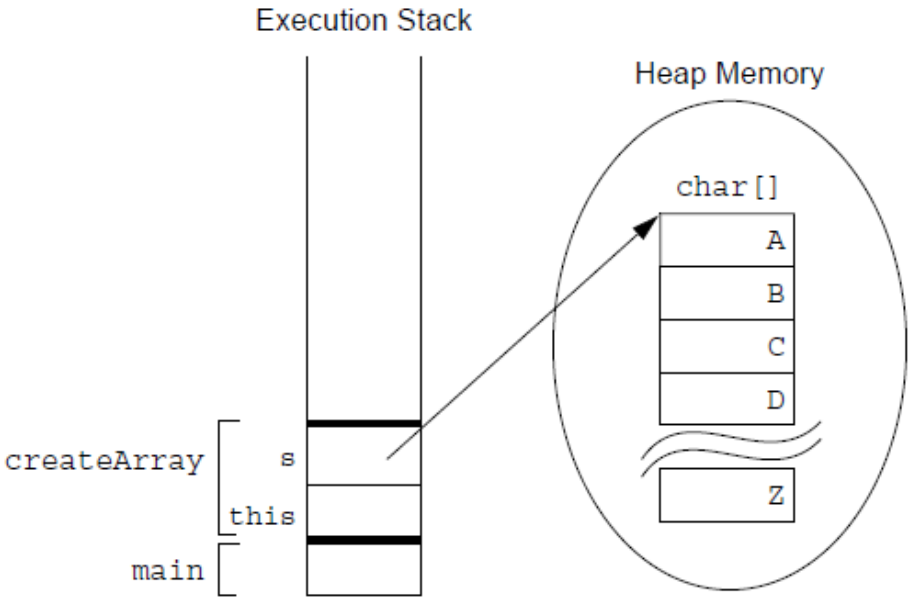
Array

- An *array* is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created.
- After creation, its length is fixed.



An array of 10 elements.

Memory Allocation




```
public char[] createArray() {
  char[] s;

  s = new char[26];
  for ( int i=0; i<26; i++ ) {
    s[i] = (char) ('A' + i);
  }

  return s;
}
```

Types

- One Dimension
- Multi Dimensional

Array

dataType[] arrayRefVar; // preferred way.

Or

dataType arrayRefVar[]; // works but not preferred way.

Eg:

double[] myList;

Creating Arrays

- dataType[] arrayRefVar = new dataType[arraySize];
- dataType[] arrayRefVar = {value0, value1, ..., valuek};

Example:

double[] myList = new double[10];

```
class MultiDimArrayDemo {
    public static void main(String[] args) {
        String[][] names = {
            {"Mr. ", "Mrs. ", "Ms. "},
            {"Smith", "Jones"}
        };
        // Mr. Smith
        System.out.println(names[0][0] + names[1][0]);
        // Ms. Jones
        System.out.println(names[0][2] + names[1][1]);
    }
}
```

MultiDimensional Arrays

Arrays of arrays:

```
int[][] twoDim = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];

int[][] twoDim = new int[][4]; // illegal
```

- Non-rectangular arrays of arrays:

```
int[][] twoDim = new int[4][];
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

- Array of four arrays of five integers each:
  - int[][] twoDim = new int[4][5];

Array Bounds

- All array subscripts begin at 0

- Example

```
public void printElements(int[] list)
 {
for (int i = 0; i < list.length; i++)
{
System.out.println(list[i]);
}
}
```

Processing Arrays

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

Enhanced For Loops

- The enhanced for loop has the following characteristics:
  - Simplified iteration over collections
  - Much shorter, clearer, and safer
  - Effective for arrays
  - Simpler when using nested loops
  - Iterator disadvantages removed

The foreach Loops

```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

Array Resizing

- You cannot resize an array.
- You can use the same reference variable to refer to an entirely new array, such as:
  - int[] myArray = new int[6];
  - myArray = new int[10];

Copying Arrays

- The System.arraycopy() method to copy arrays

```
 //original array
 int[] myArray = { 1, 2, 3, 4, 5, 6 };
// new larger array
int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
// copy all of the myArray array to the hold
// array, starting with the 0th index
System.arraycopy(myArray, 0, hold, 0,
   myArray.length);
```

Copying Arrays

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

```
class ArrayCopyDemo {
    public static void main(String[] args) {
        char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                            'i', 'n', 'a', 't', 'e', 'd' };
        char[] copyTo = new char[7];

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
    }
}
```

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length;
i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

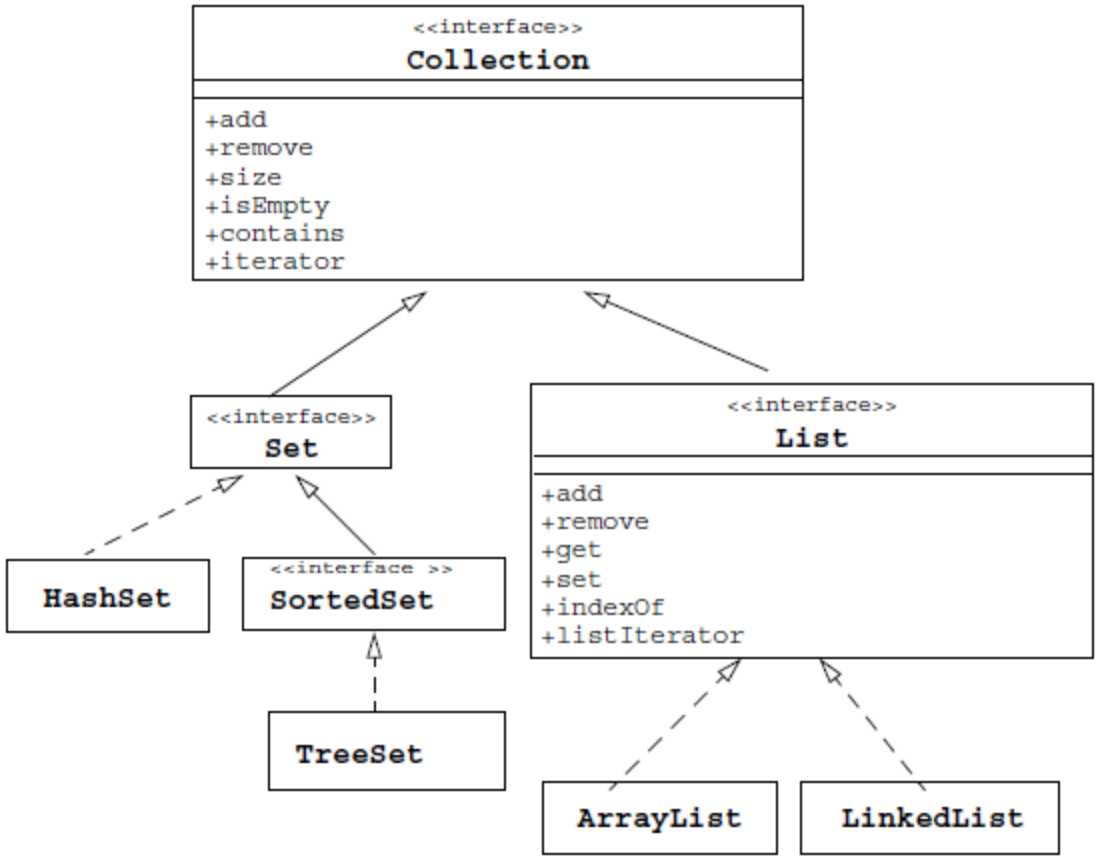
Collection

Collections

A *collection is a single object managing a group of* objects known as its elements.

- Collection – A group of objects called elements;
- Implementations determine whether there is specific ordering and whether duplicates are permitted.

The Collections API



There are several general purpose implementations of the core interfaces (`Set`, `List`, `Deque` and `Map`)

	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
`Set`	`HashSet`		`TreeSet`		`LinkedHashSet`
`List`		`ArrayList`		`LinkedList`	
`Deque`		`ArrayDeque`		`LinkedList`	
`Map`	`HashMap`		`TreeMap`		`LinkedHashMap`

- Set – An unordered collection; no duplicates are permitted.
- List – An ordered collection; duplicates are permitted.

Set Example

```
import java.util.*;
 public class SetExample {
 public static void main(String[] args) {
 Set set = new HashSet();
 set.add("one");
 set.add("second");
 set.add("3rd");
 set.add(new Integer(4));
set.add(new Float(5.0F));
 set.add("second"); // duplicate, not added
 set.add(new Integer(4)); // duplicate, not added
 System.out.println(set);
 }
 }
```

Output :
[one, second, 5.0, 3rd, 4]

List – Example

```
import java.util.*
 public class ListExample {
 public static void main(String[] args) {
 List list = new ArrayList();
 list.add("one");
 list.add("second");
 list.add("3rd");
 list.add(new Integer(4));
 list.add(new Float(5.0F));
 list.add("second"); // duplicate, is added
 list.add(new Integer(4)); // duplicate, is added
 System.out.println(list);
}
}
```

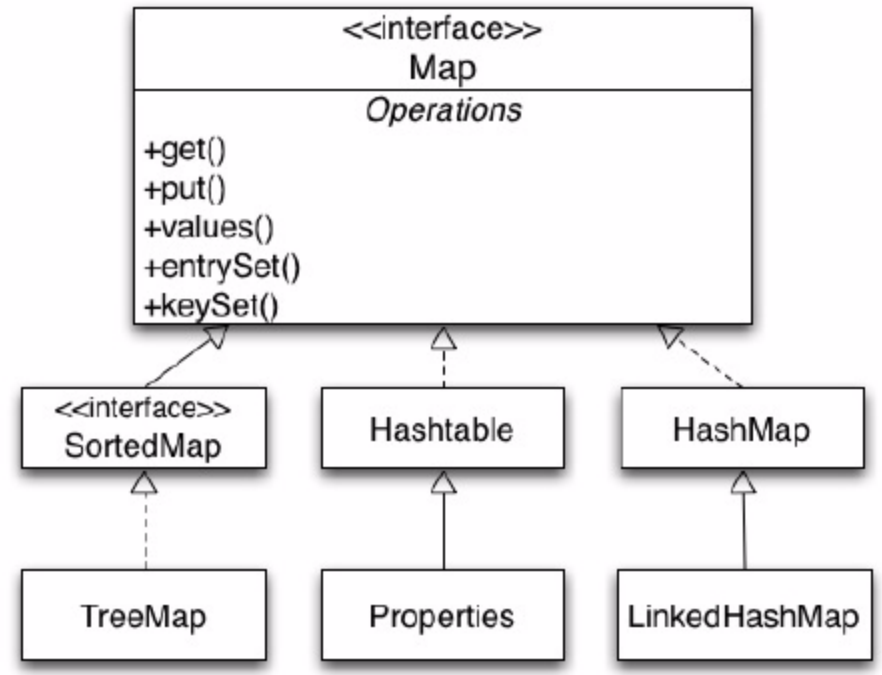
Output :

[one, second, 3rd, 4, 5.0, second, 4]

Map

- A Map object describes mappings from keys to values:
- Duplicate keys are not allowed
- One-to-many mappings from keys to values is not permitted
- The contents of the Map interface can be viewed and manipulated as collections
- entrySet – Returns a Set of all the key-value pairs.
- keySet – Returns a Set of all the keys in the map.
- values – Returns a Collection of all values in the map.

The Map Interface API



Map - Example

```
import java.util.*;
 public class MapExample {
 public static void main(String args[]) {
 Map map = new HashMap();
 map.put("one","1st");
 map.put("second", new Integer(2));
 map.put("third","3rd");
// Overwrites the previous assignment
 map.put("third“,"III");
 // Returns set view of keys
 Set set1 = map.keySet();
 // Returns Collection view of values
 Collection collection = map.values();
 // Returns set view of key value mappings
 Set set2 = map.entrySet();
System.out.println(set1 + "\n" + collection + "\n" + set2);
 }
}
```

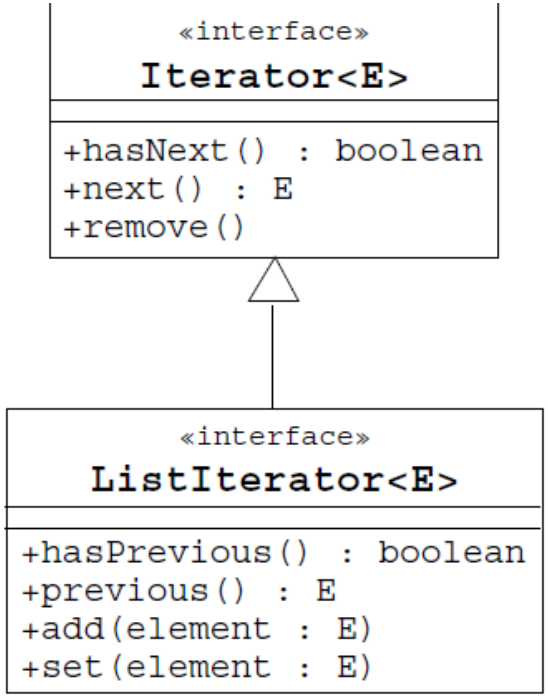
Output :
[second, one, third]
[2, 1st, III]
[second=2, one=1st, third=III]

Classic Collections

- The Vector class, which implements the List interface.
- The Stack class, which is a subclass of the Vector class and supports the push, pop, and peek methods.
- The Hashtable class, which implements the Map interface.
- The Properties class is an extension of Hashtable that only uses Strings for keys and values.
- Each of these collections has an elements method that returns an Enumeration object.

Iterators

- Iteration is the process of retrieving every element in a collection.
- The basic Iterator interface allows you to scan forward through any collection.



Thank You