COP 3330: Object-Oriented Programming Summer 2007

Arrays and Strings in Java – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2007

School of Electrical Engineering and Computer Science
University of Central Florida

The ArrayTools Class

In the next few slides we'll develop a class that will contain a number of class methods to allow us to manipulate one-dimensional arrays.

```
public class ArrayTools {
    //class constant
    private static final int MAX_LIST_SIZE = 100;

    //putList():  produces a string representation
    public static void putList(int[] data) {
           for (int i = 0; i < data.length; ++i){
                  System.out.println(data[i]);
           }
    }
```


```
//getList():  extract up to MAX_LIST_SIZE values
   public static int[] getList() throws IOException {
        BufferedReader stdin = new BufferedReader(
                new InputStreamReader(System.in));
        int[] buffer = new int[MAX_LIST_SIZE];
        int listSize = 0;
        for (int i = 0l i < MAX_LIST_SIZE; ++i){
                String currentInput = stdin.readLine();
                if (currentInput != null) {
                        int number = Integer.parseInt(currentInput);
                        buffer[i] = number;
                        ++listSize;
                }
                else {
                        break;
                }
        }
        int[] data = new int[listSize];
        for(int i =0; i < listSize; ++i) {
                data[i] = buffer[i];
        }
        return data;
   }
```


```
//reverse():  reverse the order of the element values
   public static void reverse(int[] list){
        int n = list.length;
        for (int i = 0; i < n/2; ++i){
           //swap the element from the front of the list with the
           //corresponding element from the end of the list
           int  remember = list[i];
           list[i] = list[n-1-i];
           list[n-1-i] = remember;
        }
   }

//doubleCapacity(): creates a duplicate list twice as big
   public static String[] doubleCapacity(String[] curList) {
        int n = curList.length;
        String[] biggerList = new String[2*n];
        for (int i = 0; i < n; ++i) {
            biggerList[i] = curList[i];
        }
        return biggerList;
   }
}//end class ArrayTools
```


Example Using the ArrayTools Class

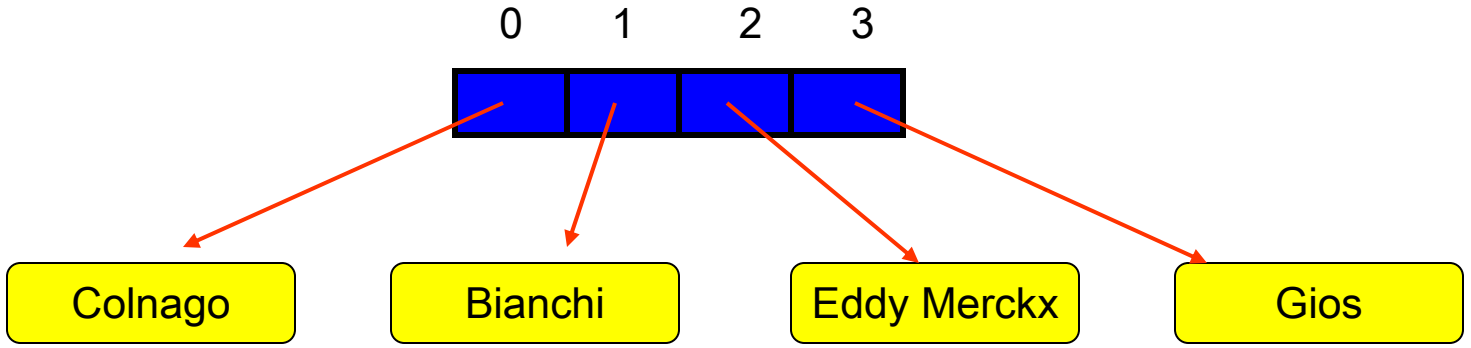
```
//extract and display a list in forward and reverse order
import java.io.*;
public class Demo{
   public static void main(String[] args) throws IOException {
      System.out.println();
      System.out.println("Enter a list of integers, one per line");
      int[] number = ArrayTools.getList();
      System.out.println();
      System.out.println("Your list");
      ArrayTools.putList(number);
      ArrayTools.reverse(number);
      System.out.println();
      System.out.println("Your list in reverse order:");
      ArrayTools.putList(number);
      System.out.println();
   }
}
```


Example – Doubling the Size of the Array

- Suppose that we declare the following array:

```
String[] bikes = {“Colnago”, “Bianchi”, “Eddy Merckx”,
“Gios”};
```

- There is no room to add another bike to this array. Each element of the array already references a value. The method doubleCapacity() will allow us to expand our array.

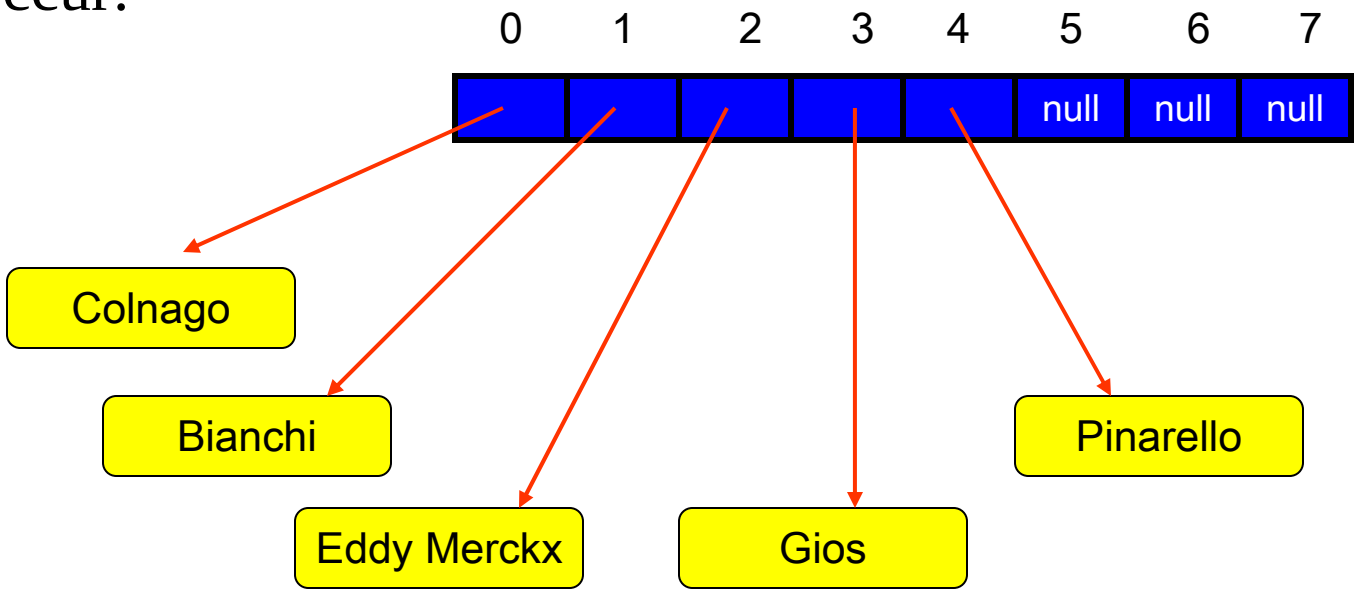


initial configuration

Example – Doubling the Size of the Array (cont.)

```
bikes = doubleCapacity(bikes);
bikes[4] = "Pinarello";
```

- The invocation and statements above causes the following to occur:



final configuration – after doubling capacity

Command-Line Parameters

- Many OS (e.g. Linux and Windows) provide command-line interpreters. These interpreters allow the user to type a command and then have the OS execute it.
- For example: `% cd \jdk\bin`
- The instruction above the string `\jdk\bin` is a command-line parameter to the command `cd`.
- The Java application on the next page simply echoes its command-line parameters to the standard output. Its operation is identical to that of the Linux/Windows commands `echo`.

Command-Line Parameters (cont.)

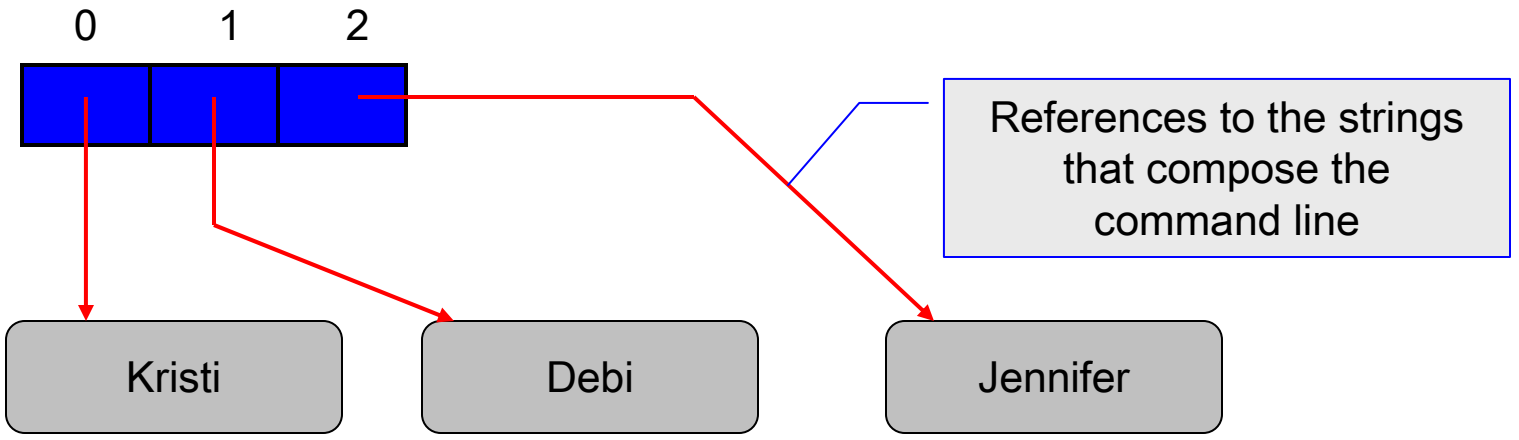
```
//mimics OS command echo
public class Echo {
   public static void main (String[] args){
        //display parameters one after the other
        for (int i = 0; i < args.length; ++i) {
               System.out.println(args[i] + " ");
        }
        System.out.println();
   } //end main
} //end Echo
```

```
Invocation:  java Echo Kristi Debi Jennifer

Output:  Kristi
         Debi
         Jennifer
```


Command-Line Parameters (cont.)

```
Invocation:  java Echo Kristi Debi Jennifer

Output: Kristi
        Debi
        Jennifer
```

Command line parameters are passed to method main()



Multidimensional Arrays

- Thus far, all of the arrays that we have examined have been one-dimensional arrays. It is also possible to define multidimensional arrays in Java (as well as other languages).
- There are many different application areas, such as matrices, graphical animation, economic forecast models, map representations, and microprocessor design, just to mention a few, where multidimensional arrays are extremely useful.
- Arrays of any dimension are possible in Java. Two and three dimensional arrays are quite common. Arrays with more than three dimensions are not commonly used, but are necessary for some types of problems.

Multidimensional Arrays (cont.)

- Let's look at 2-dimensional arrays for the time being before we look at more complex arrays.
- The following definition initializes `m` to reference a 2-dimensional array:

```
int[][] m = new int[3][4];
```

  - The 2-dimensional array `m` should be viewed as consisting of three component arrays: `m[0], m[1], and m[2].`
  - The component type of `m` is `int[]` and the element type of each of `m[0], m[1],` and `m[2]` is `int`.

Multidimensional Arrays (cont.)

- The definition for `m`

```
int[][] m = new int[3][4];
```

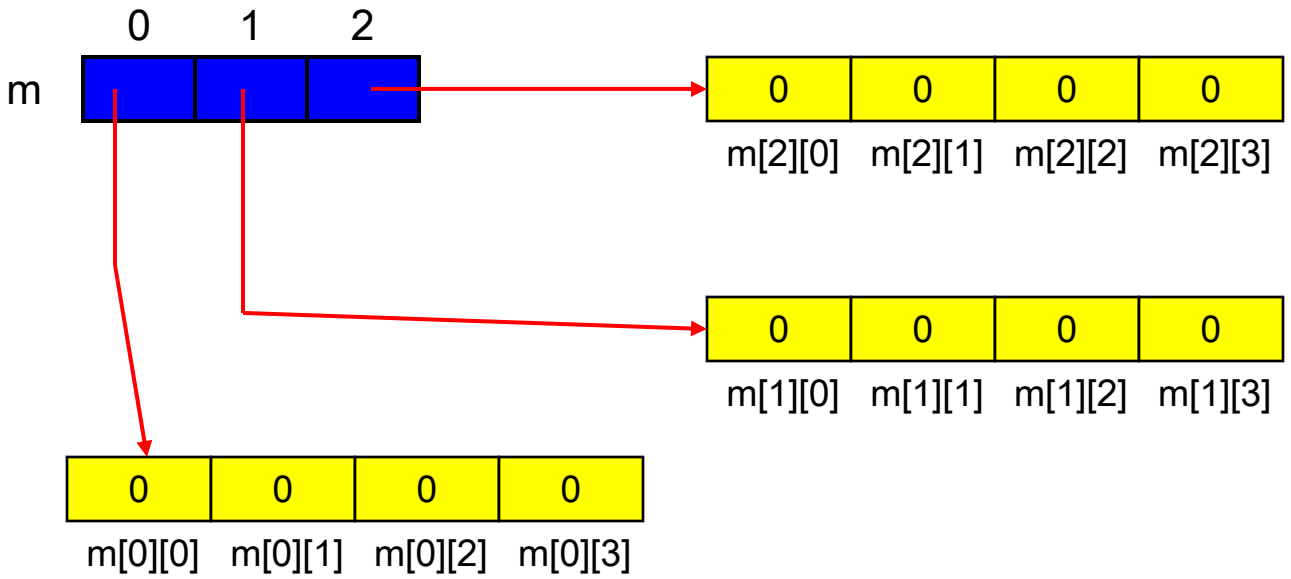
is shorthand for the following explicit definition:

```
int[][] m = new int[3][4];
m[0] = new int[4];
m[1] = new int[4];
m[2] = new int[4];
```

- The components of a 2-dimensional array are known as ***rows***. To refer to an individual element of a row, an additional subscript is required. For example, `m[i][j]` refers to the `jth` element of the `ith` row in `m`.

Multidimensional Arrays (cont.)

- The definition for `m` gives the array the following representation:



Multidimensional Arrays (cont.)

- Suppose that you want to set the values of a two-dimensional array `m` using the standard input. The easiest way to do this is to nest two for loops where the outer loop would iterate once per subarray. For each such iteration, the inner loop would iterate once for each element of the current subarray. This code is shown below:

```
for (int row = 0; r < m.length; ++row) {
   for(int column = 0; column < m[row].length; ++column) {
     System.out.println("Enter an int value: ");
     m[row][column] = Integer.parseInt(Stdin.readLine());
   }
}
```

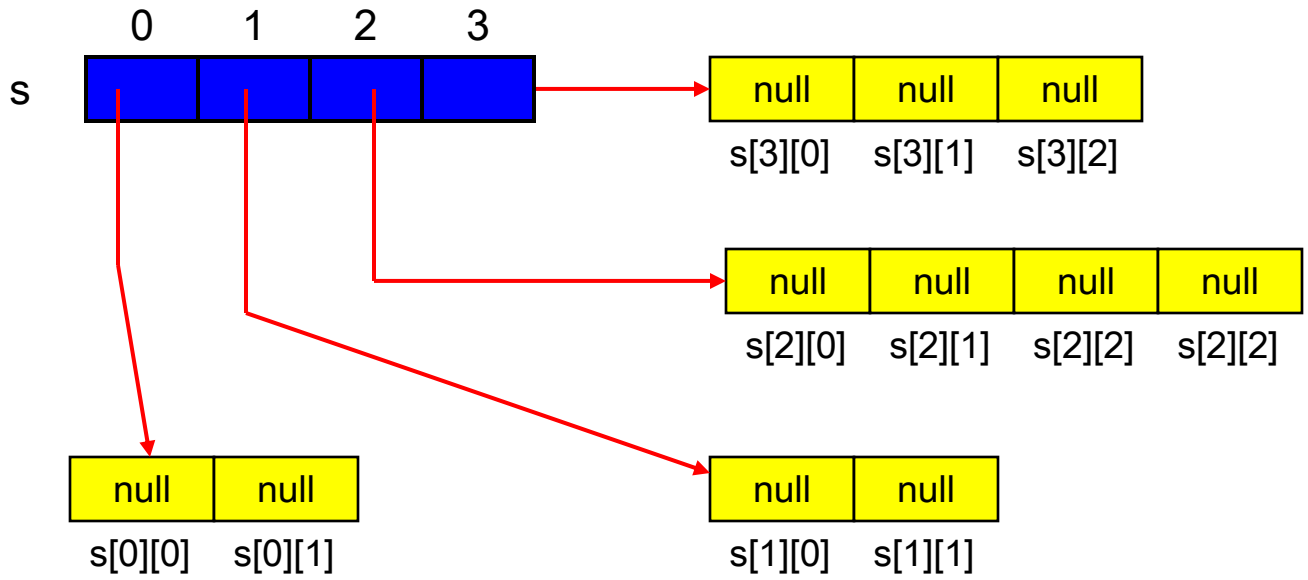

Multidimensional Arrays (cont.)

- Java does not require that the subarrays of a 2-dimensional array have the same length.
- Consider the following example:

```
String[][] s = new String[4][];
s[0] = new String[2];
s[1] = new String[2];
s[2] = new String[4];
s[3] = new String[3];
```

- The representation of this array is shown in the next slide.

Multidimensional Arrays (cont.)



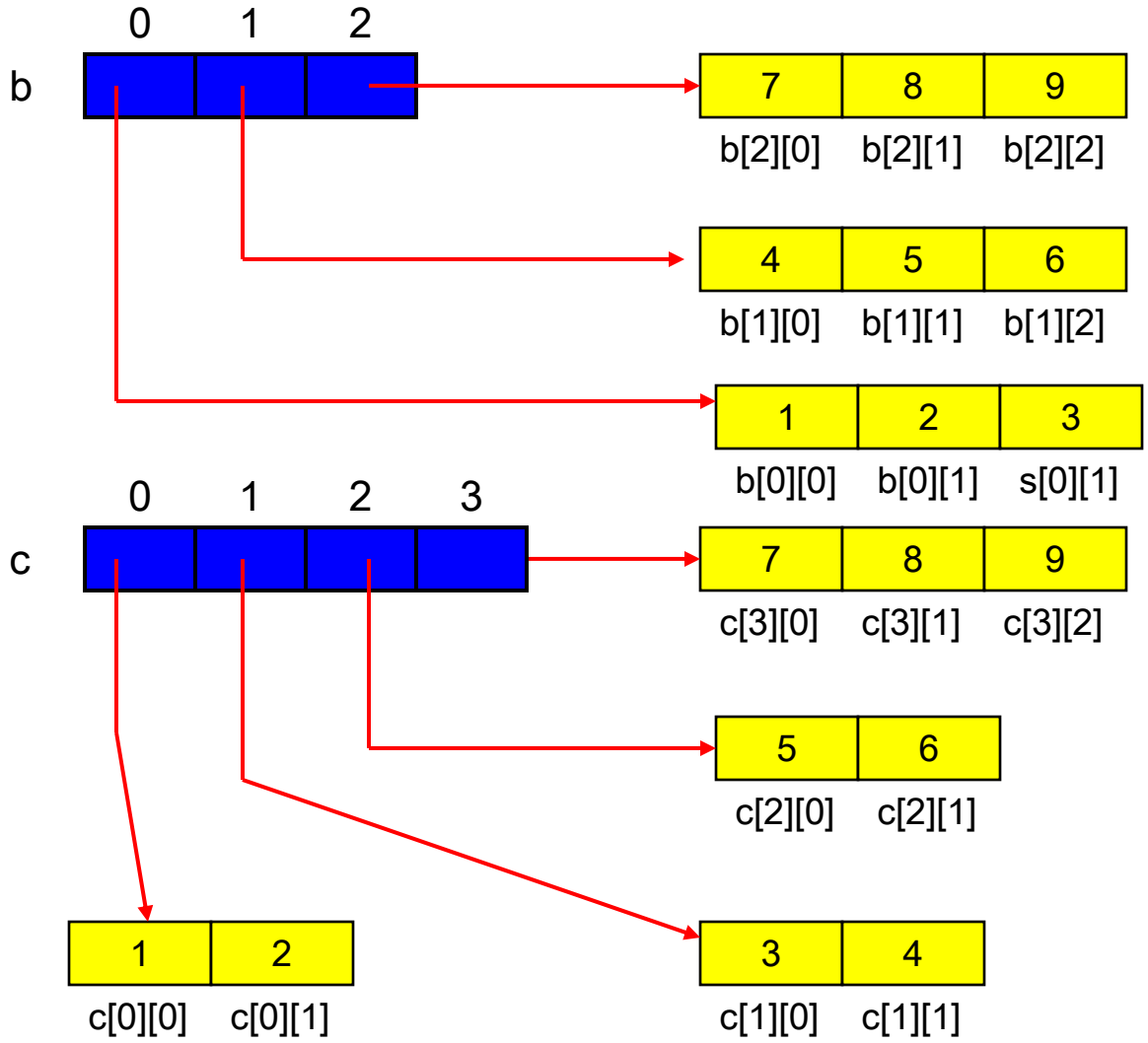
Multidimensional Arrays (cont.)

- The definition of multidimensional arrays can include initialization by specifying a block of values with each component of the array having its own initialization specification.
- For example, the following definitions initialize both `b` and `c` to be `int[][]` arrays.

```
int[][] b = {{1,2,3}, {4,5,6}, {7,8,9}};
int[][] c = {{1,2}, {3,4}, {5,6}, {7,8,9}};
```

- The representation of these two arrays is shown in the next slide.

Multidimensional Arrays (cont.)



Multidimensional Arrays (cont.)

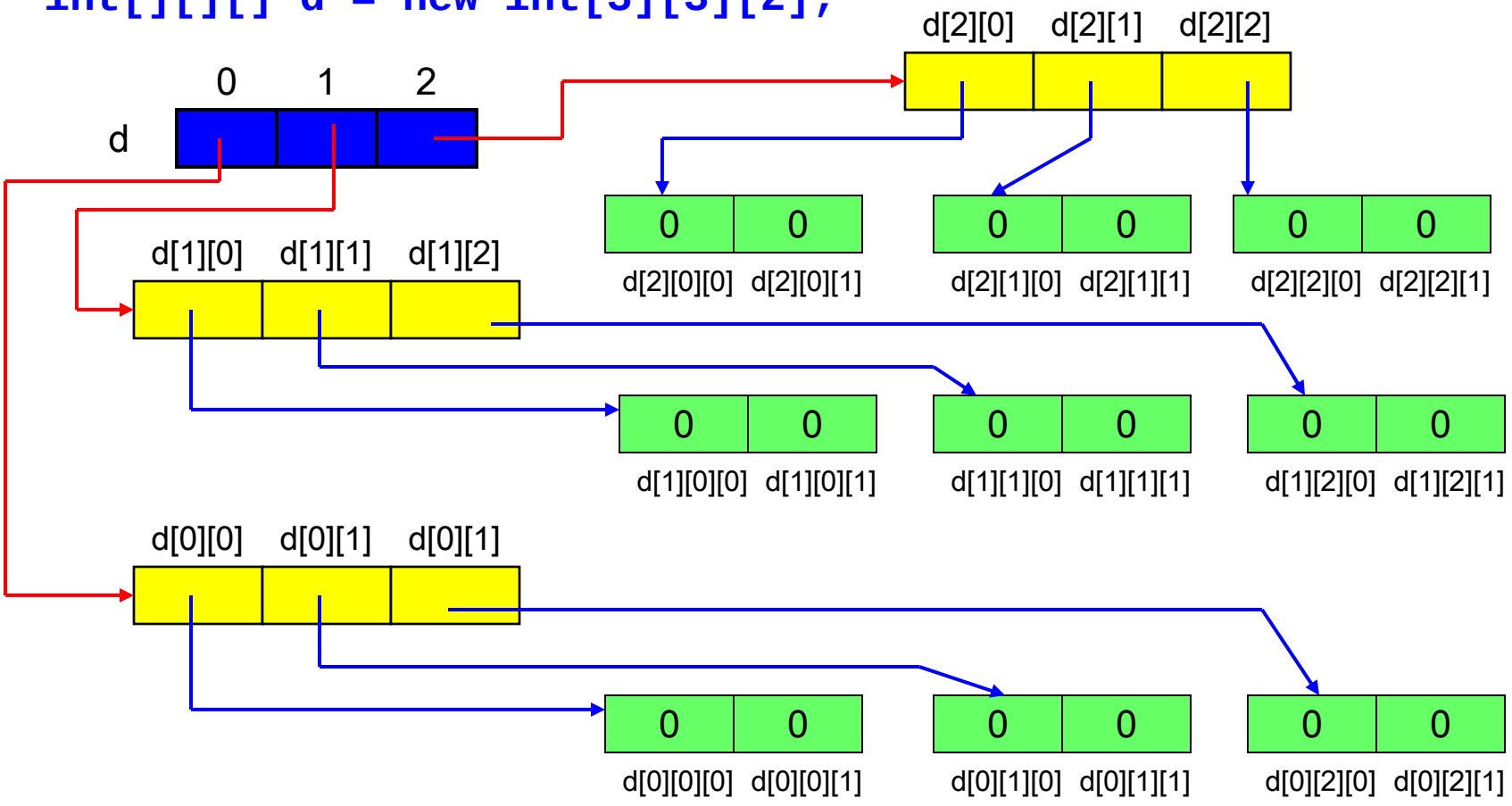
- Methods with parameters that are multidimensional arrays are permitted in Java.
- The following method zero() sets to 0 all the elements of the subarrays of its two-dimensional `int[][]` parameter array `inarray`.

```
public void zero (int[][] inarray){
   for(int row = 0; row < inarray.length; ++row){
     for(int column = 0; column < inarray[row].length; ++column)
         inarray[row][column] = 0;
     }
```


Multidimensional Arrays (cont.)

- The following example illustrates a 3-dimensional array.

```
int[][][] d = new int[3][3][2];
```



Sorting

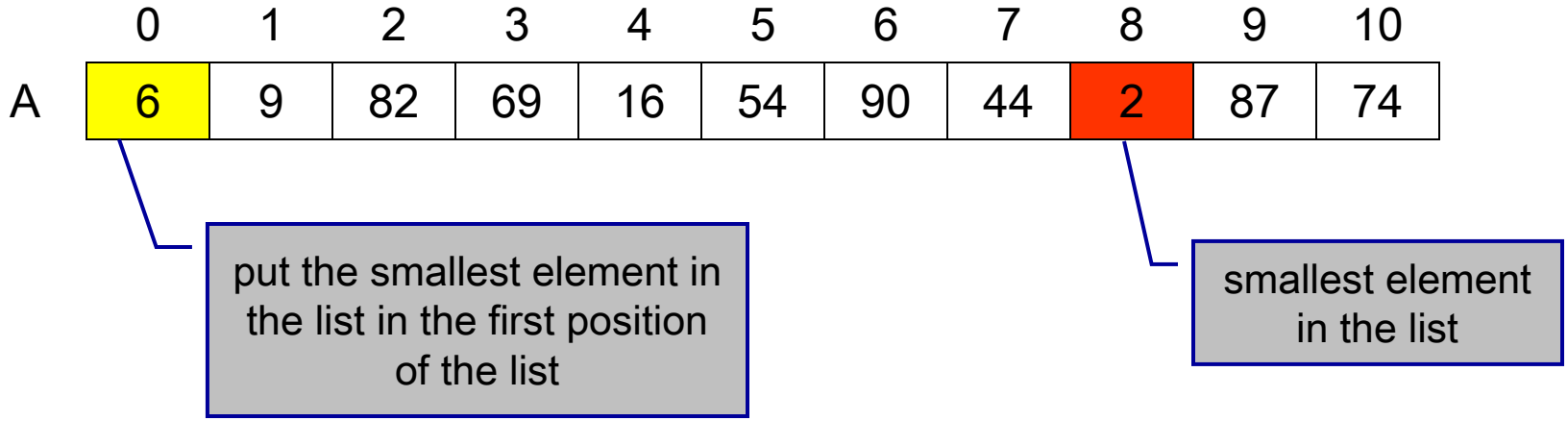
- Sorting is a common application which utilizes arrays to hold the list of elements which are to be sorted.
- A sort is often (not always) an iterative process such that during each iteration the elements in the list are rearranged in some manner. Each iterative step is designed to bring the list of elements closer to its final sorted order.
- Many different sort techniques are available and there are advantages and disadvantages associated with most sorting algorithms. We will consider only a couple of basic sorting algorithms which are suitable for sorting relatively small lists of elements.

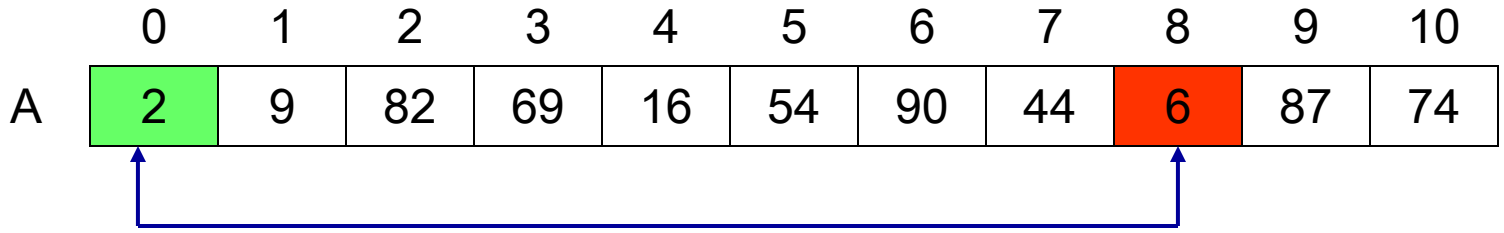
The Selection Sort

- The Selection sort is a simple, comparison-based sorting algorithm with complexity $O(n^2)$.
- The technique of the Selection sort is that on the first iteration through the elements of the list to be sorted the smallest (or largest) element is found and placed in the first (or last) position of the list. This is done by interchanging the smallest element with the element in the first position of the list.
  - Thus, the general technique of the Selection sort is such that on the `ith` iteration through the elements in the list to be sorted, the `ith` smallest (or largest) element is placed into the `ith` position in the list.

Example of Selection Sort

First iteration in the list

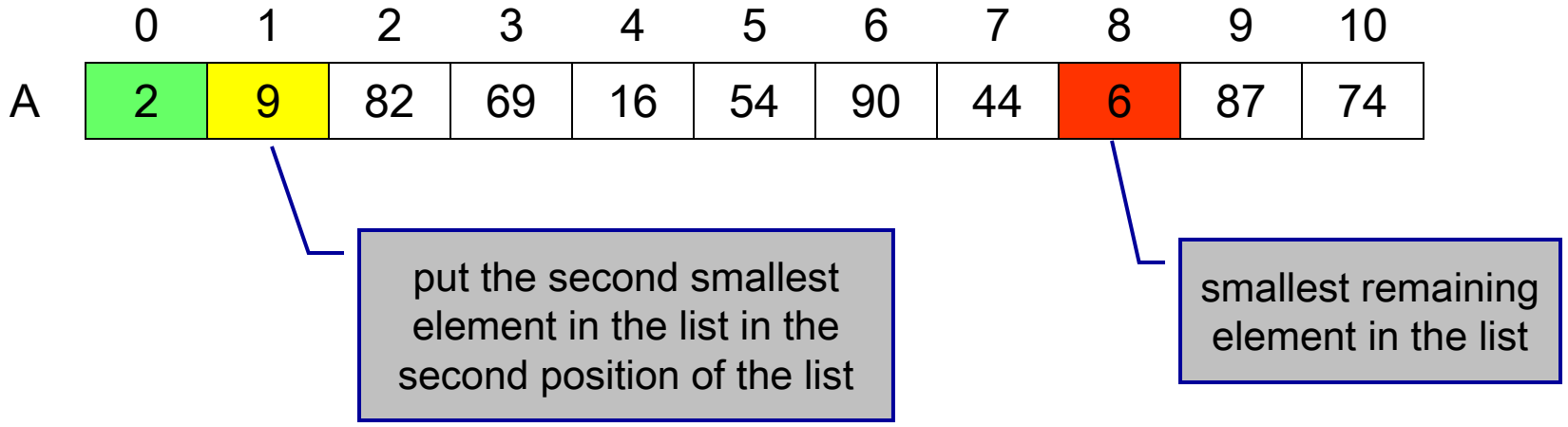


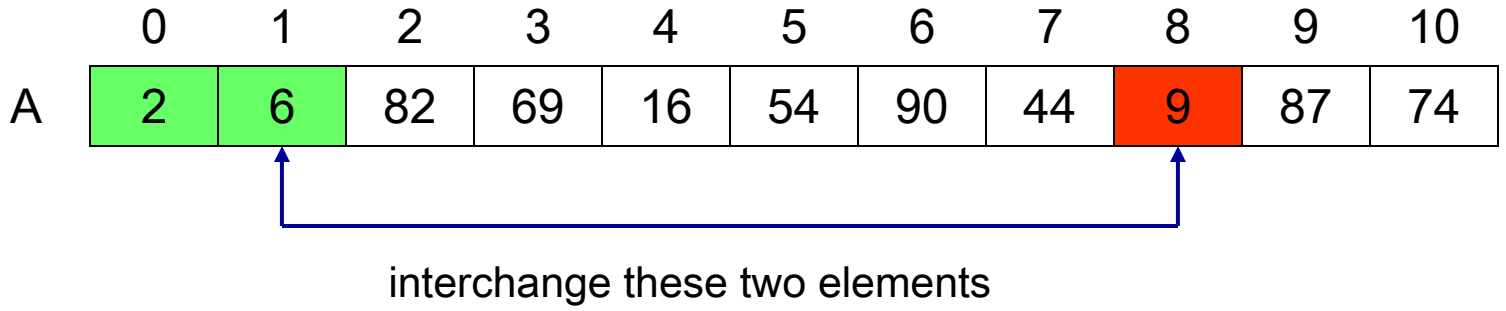


interchange these two elements

Selection Sort Example

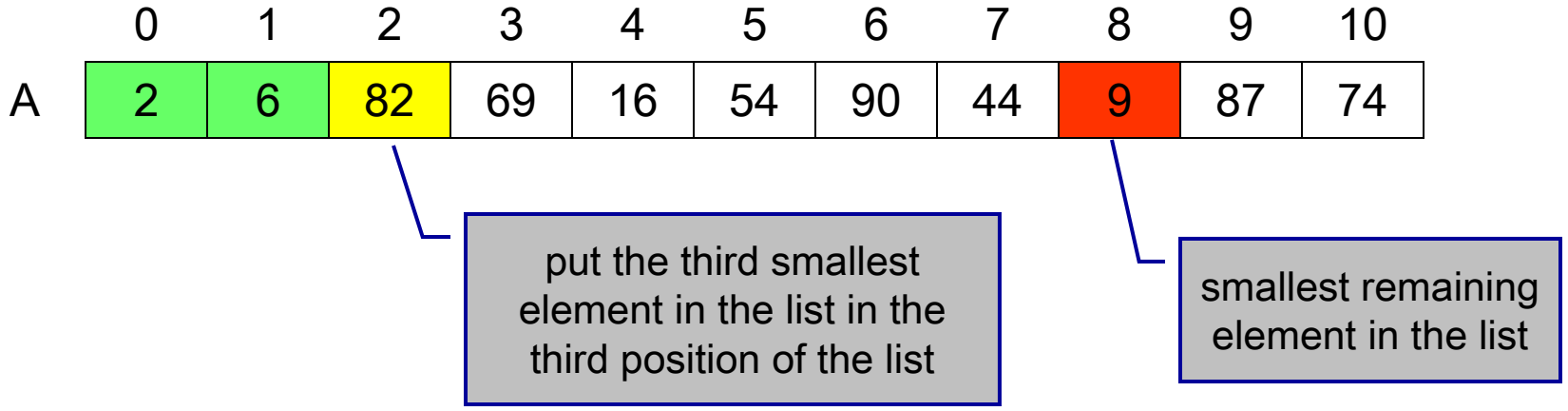
Second iteration in the list

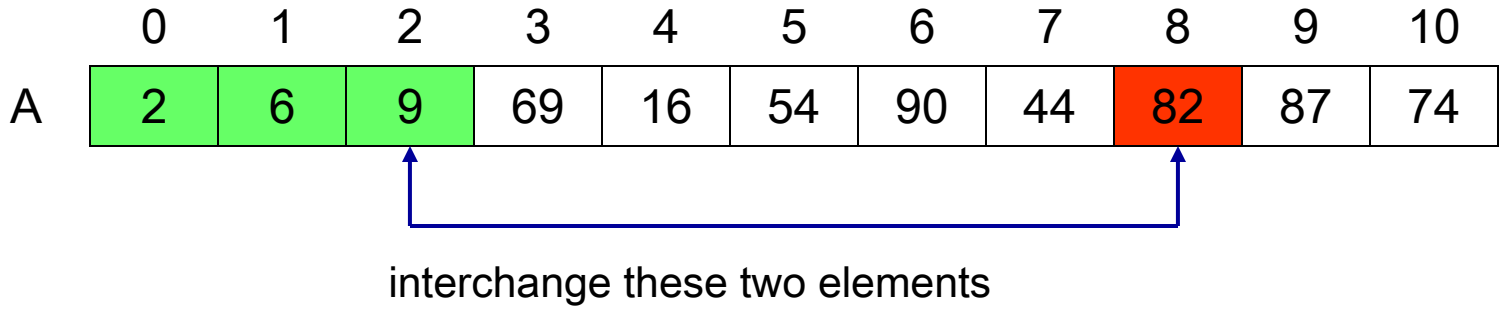




Selection Sort Example

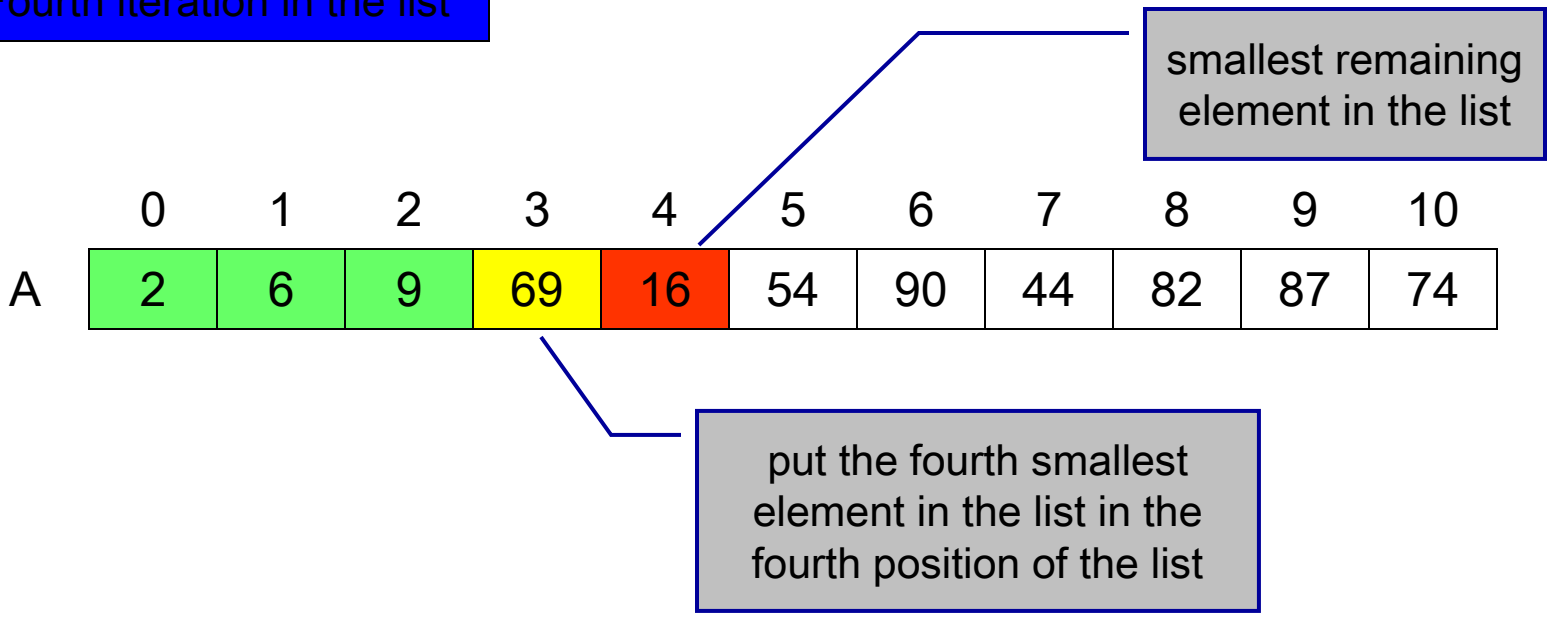
Third iteration in the list

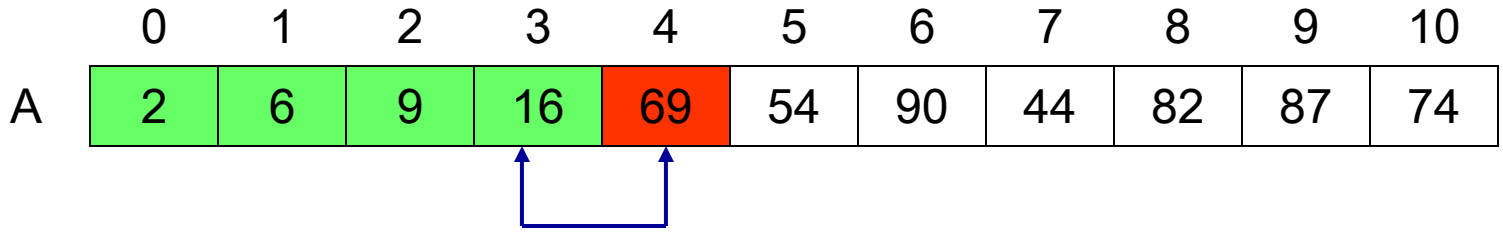




Selection Sort Example

Fourth iteration in the list

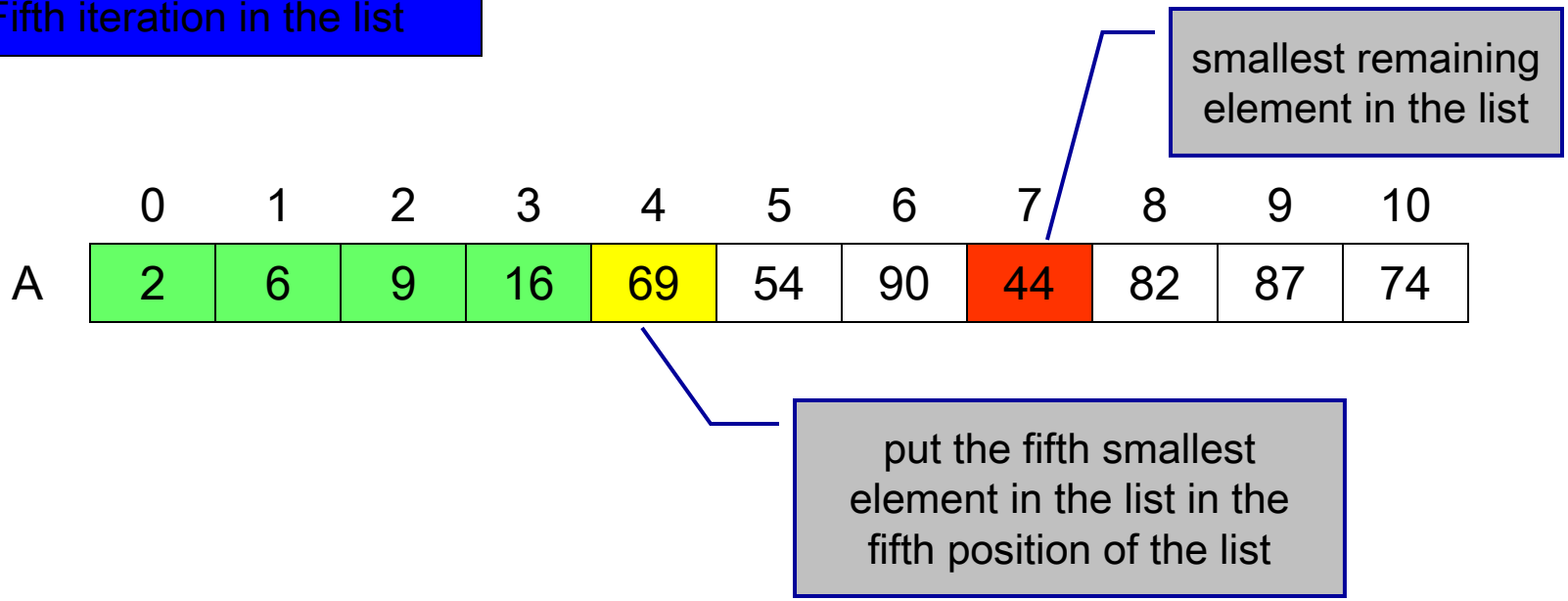


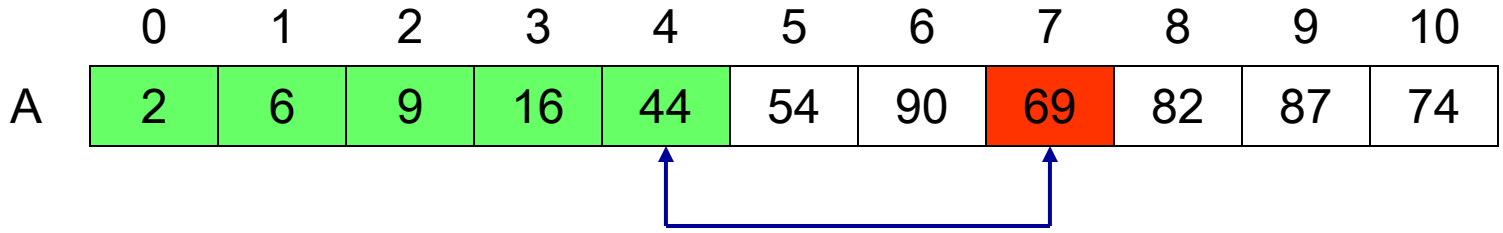


interchange these two elements

Selection Sort Example

Fifth iteration in the list

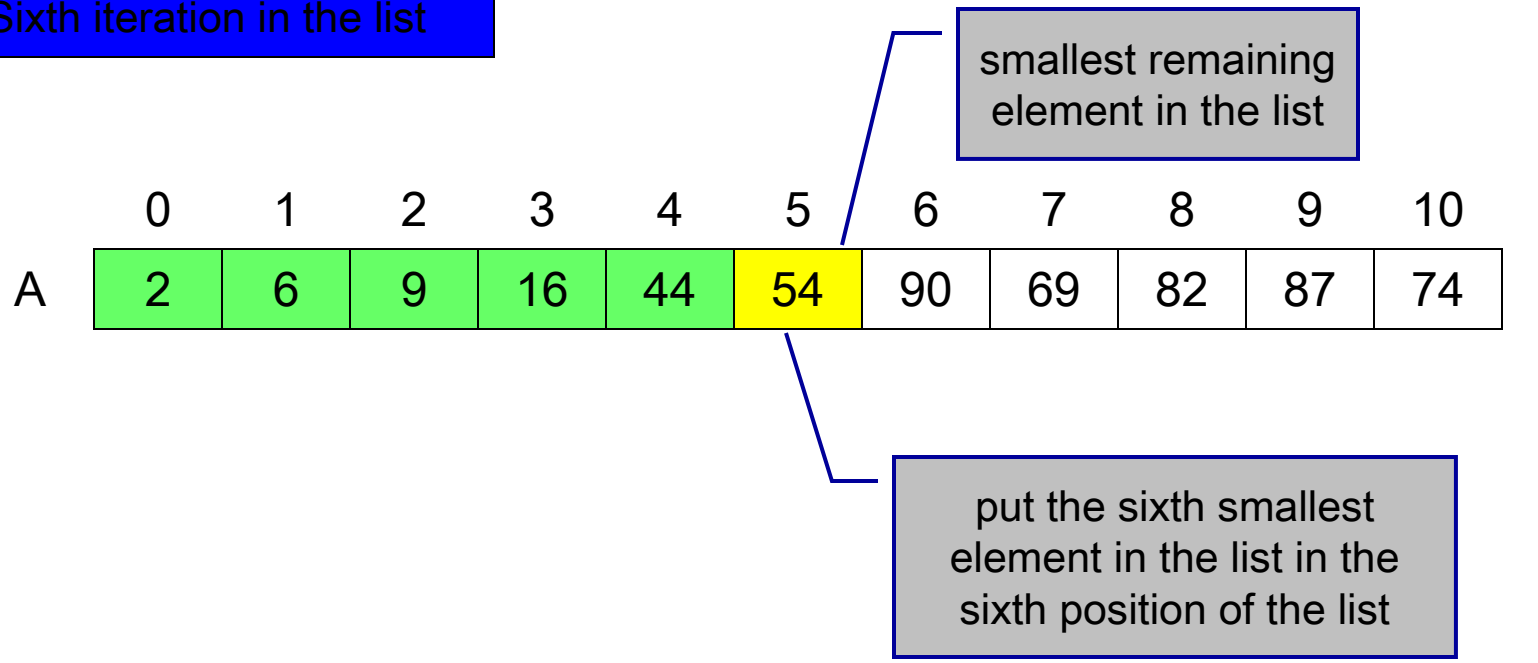


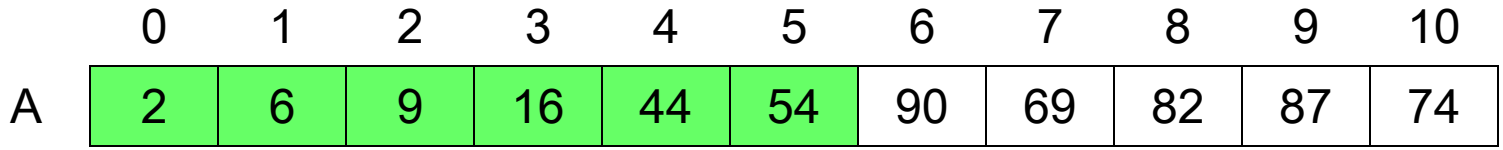


interchange these two elements

Selection Sort Example

Sixth iteration in the list

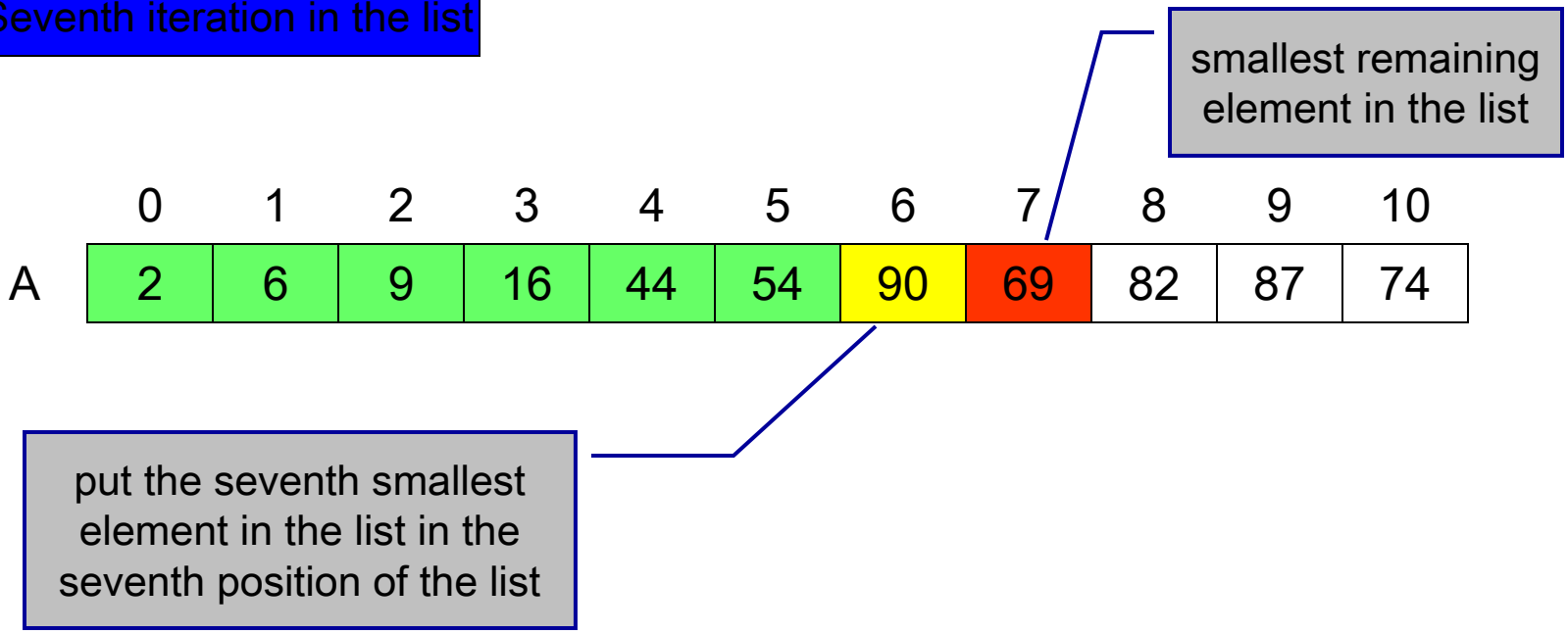


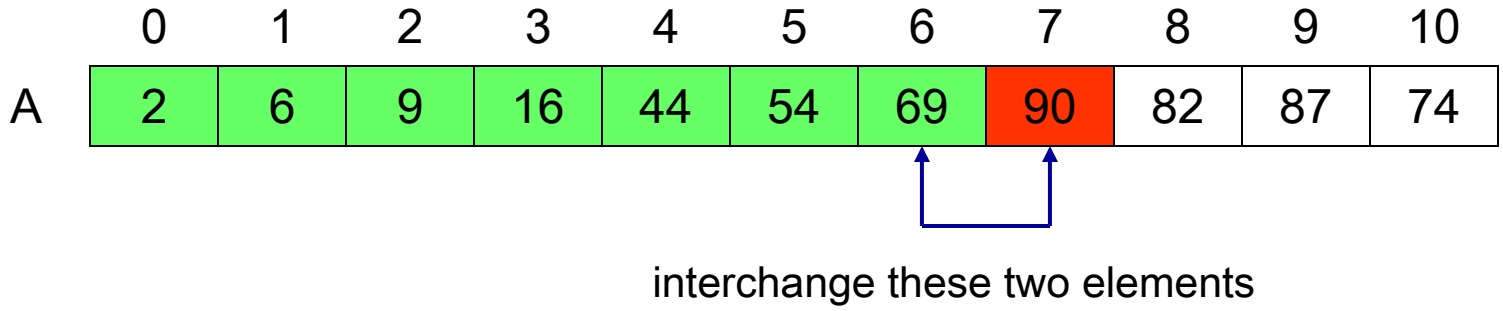


no interchange required

Selection Sort Example

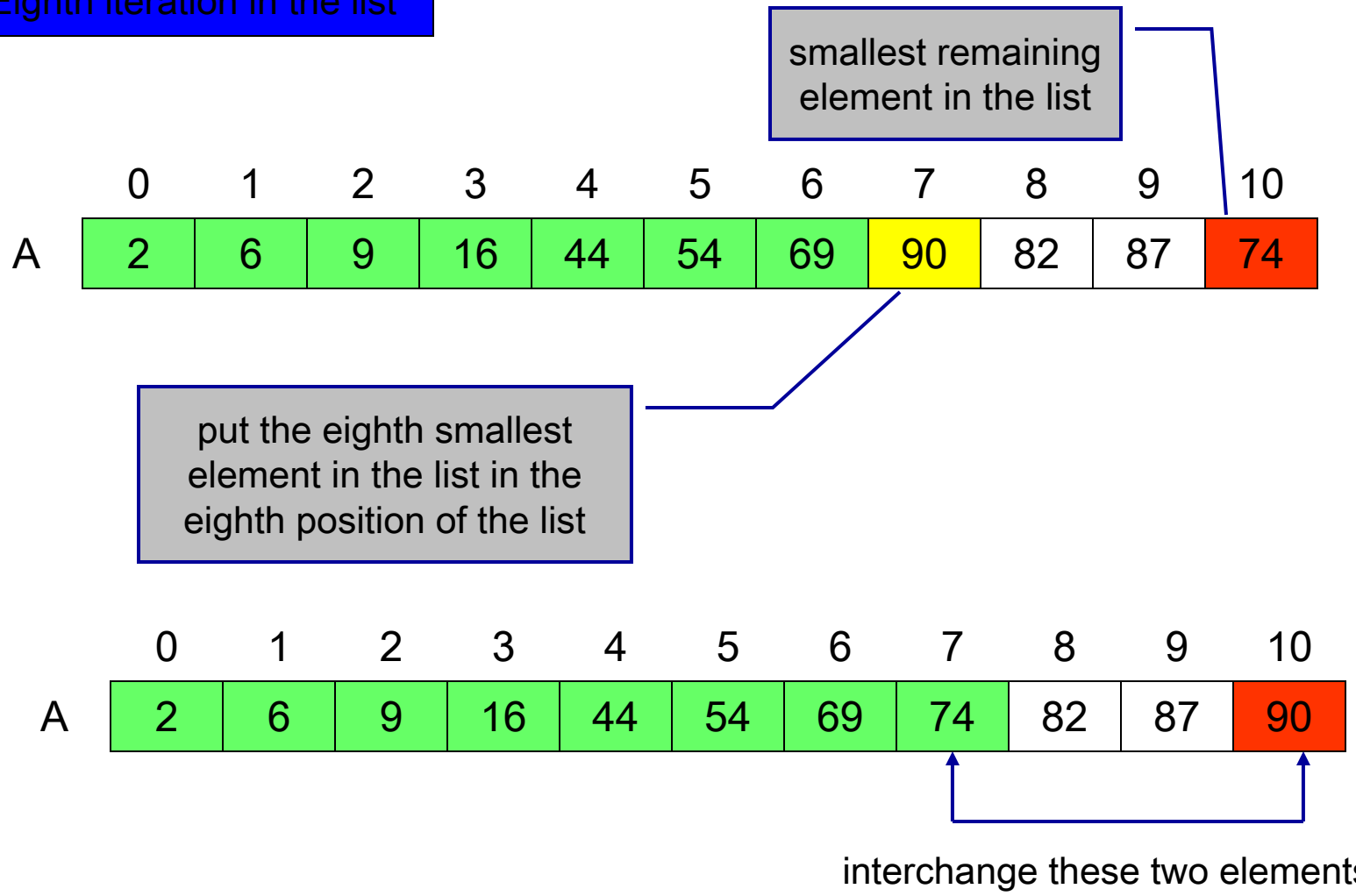
Seventh iteration in the list





Selection Sort Example

Eighth iteration in the list



Selection Sort Example

Ninth iteration in the list

smallest remaining element in the list

	0	1	2	3	4	5	6	7	8	9	10
A	2	6	9	16	44	54	69	74	82	87	90

put the ninth smallest element in the list in the ninth position of the list

	0	1	2	3	4	5	6	7	8	9	10
A	2	6	9	16	44	54	69	74	82	87	90

no interchange required

Selection Sort Example

Tenth and final iteration in the list

smallest remaining element in the list

	0	1	2	3	4	5	6	7	8	9	10
A	2	6	9	16	44	54	69	74	82	87	90

put the tenth smallest element in the list in the tenth position of the list

	0	1	2	3	4	5	6	7	8	9	10
A	2	6	9	16	44	54	69	74	82	87	90

Iterations end – last element is largest element in list and is in the last position

The Selection Sort

```
//selectionSort():  performs a selection sort on the
//elements of array a.
public static void selectionSort(int[] a){
  for(int i = 0; i < a.length-1; ++i){
     int smallest = i;
     for(int j = i+1; j< a.length; ++j) {
         //find the smallest remaining element
         if(a[j] < a[smallest]) {
            smallest = j;
         }
     }end inner loop on j
     //found the smallest remaining element
     int remember = a[i];
     a[i] = a[smallest];
     a[smallest] = remember;
  }//end outer loop on i
}
```


Example of Insertion Sort

First iteration in the list

	0	1	2	3	4	5	6	7	8	9	10
A	6	4	82	69	16	54	90	44	2	87	74

the first element in the list is sorted by default

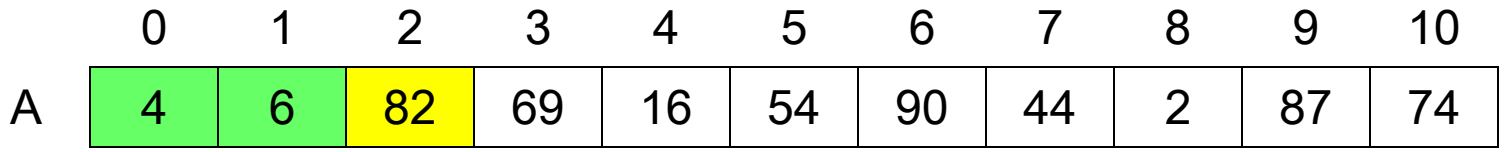
Second iteration in the list

	0	1	2	3	4	5	6	7	8	9	10
A	4	6	82	69	16	54	90	44	2	87	74

"Expand" window to consider the second element of the list. They are not in the correct order, so interchange them. Now the first two elements are sorted.

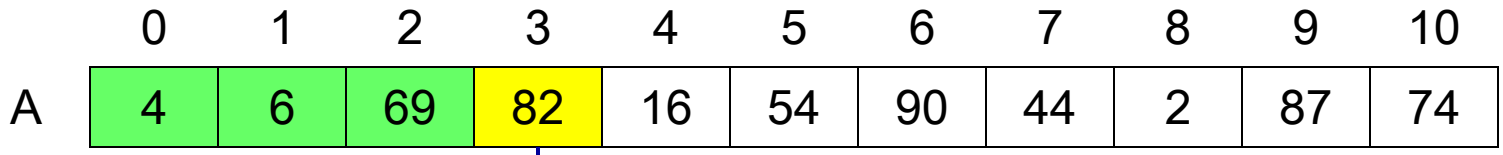
Insertion Sort Example

Third iteration in the list



"Expand" the window to include the third element. The new element is in its proper position with respect to the first three. Now the first three elements are sorted.

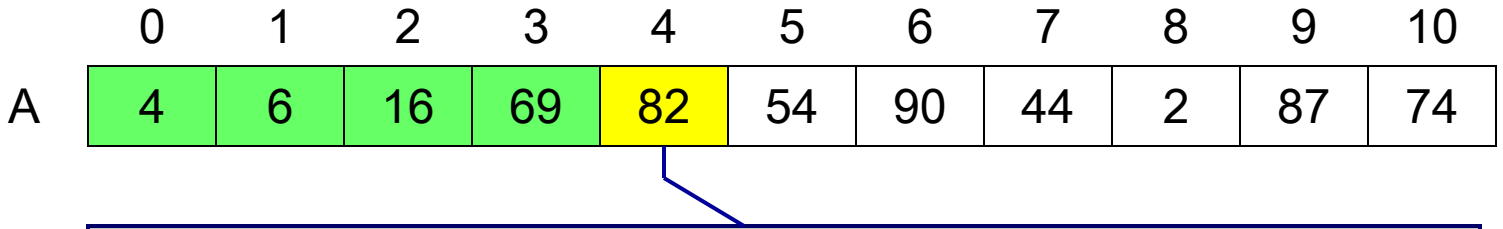
Fourth iteration in the list



"Expand" window to consider the fourth element of the list. It is not in its proper position relative to the first four elements, so put it in its proper position (interchange 69 and 82). Now the first four elements are sorted.

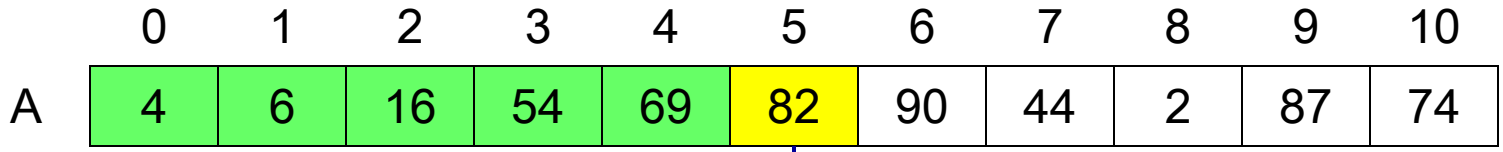
Insertion Sort Example

Fifth iteration in the list



"Expand" the window to include the fifth element. The new element is not in its proper position relative to the first five elements, so put it in its proper position (insert 16 at index 2 and shift 69 and 82 right). Now the first five elements are sorted.

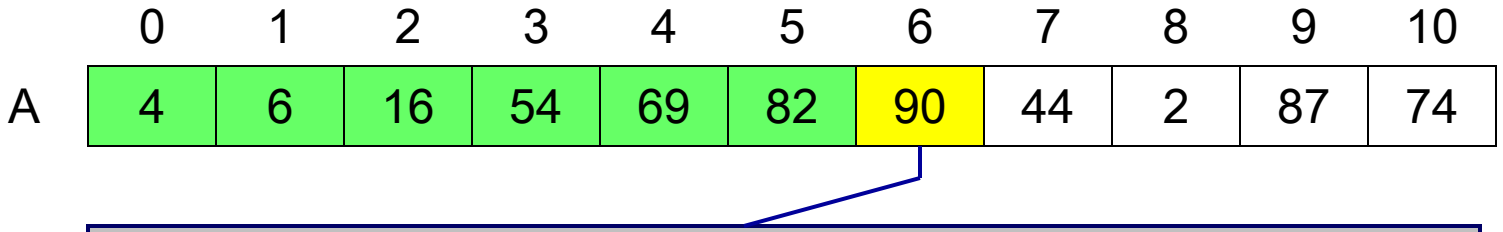
Sixth iteration in the list



"Expand" window to consider the sixth element of the list. It is not in its proper position relative to the first six elements, so put it in its proper position (insert 54 at index 3 and shift 69 and 82). Now the first six elements are sorted.

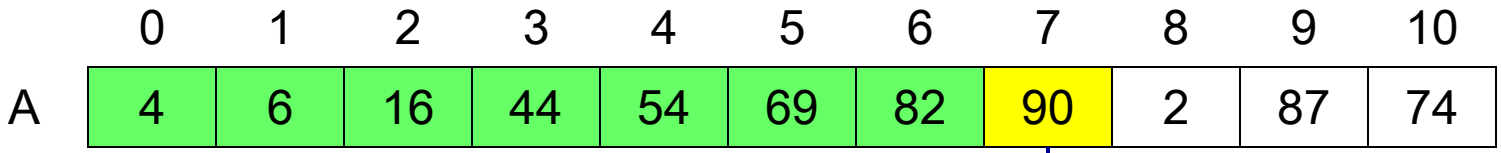
Insertion Sort Example

Seventh iteration in the list



"Expand" the window to include the seventh element. The new element is in its proper position relative to the first seven elements, so no repositioning is required. Now the first seven elements are sorted.

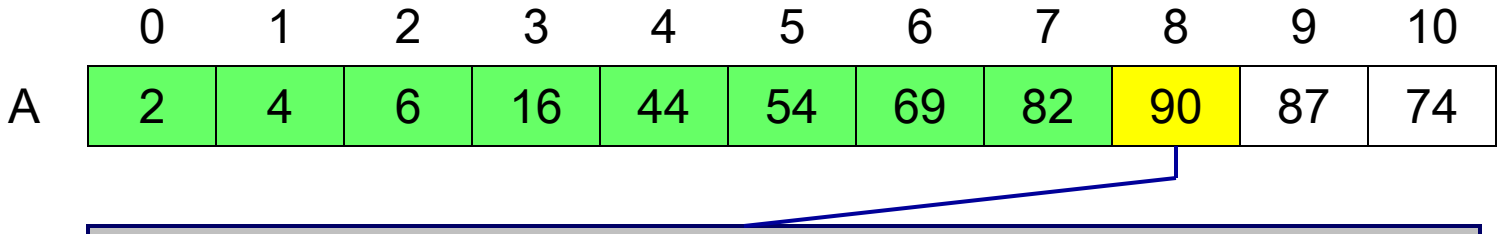
Eighth iteration in the list



"Expand" window to consider the eighth element of the list. It is not in its proper position relative to the first eight elements, so put it in its proper position (insert 44 at index 3 and shift 54, 69, 82, and 90). Now the first eight elements are sorted.

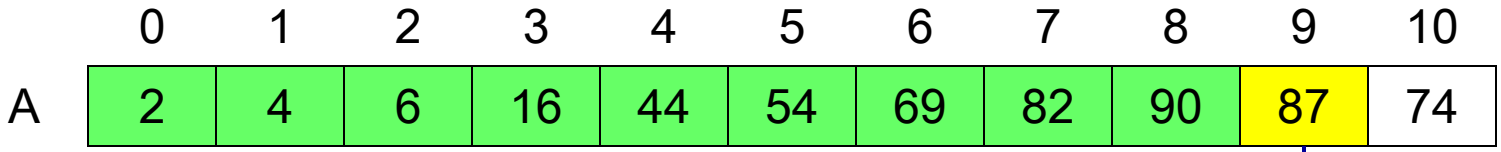
Insertion Sort Example

Ninth iteration in the list



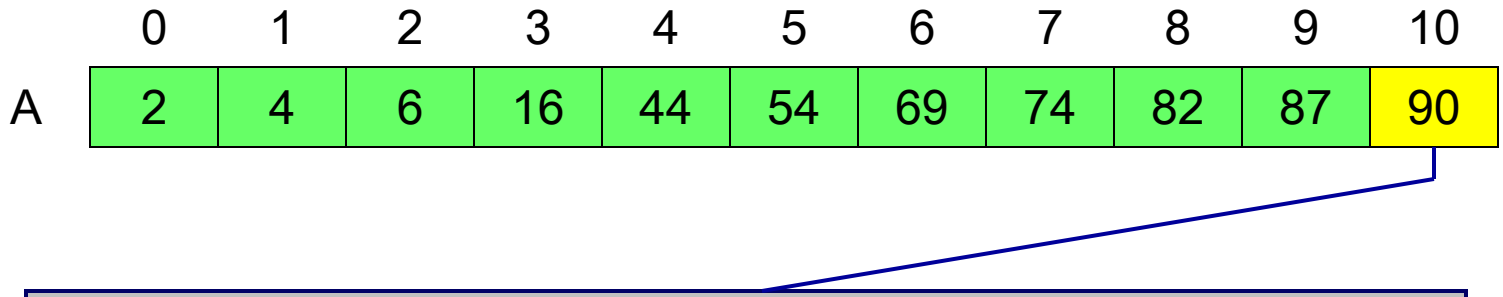
"Expand" the window to include the ninth element. The new element is not in its proper position relative to the first nine elements, so put it in its proper position (insert 2 at index 0 and shift 4, 6, 16, 44, 54, 69, 82, and 90 to the right). Now the first nine elements are sorted.

Tenth iteration in the list



"Expand" window to consider the tenth element of the list. It is not in its proper position relative to the first ten elements, so put it in its proper position (insert 87 at index 8 and shift 90 to the right). Now the first ten elements are sorted.

Eleventh and final iteration in the list



"Expand" window to consider the eleventh element of the list. It is not in its proper position relative to the first eleven elements, so put it in its proper position (insert 74 at index 7 and shift 82, 87, and 90 to the right). Now the entire list is sorted.

Insertion Sort

- The approach of Insertion Sort:
  - Pick any item and insert it into its proper place in a sorted sublist
  - repeat until all items have been inserted

- In more detail:
  - consider the first item to be a sorted sublist (of one item)
  - insert the second item into the sorted sublist, shifting items as necessary to make room to insert the new addition
  - insert the third item into the sorted sublist (of two items), shifting as necessary
  - repeat until all values are inserted into their proper position

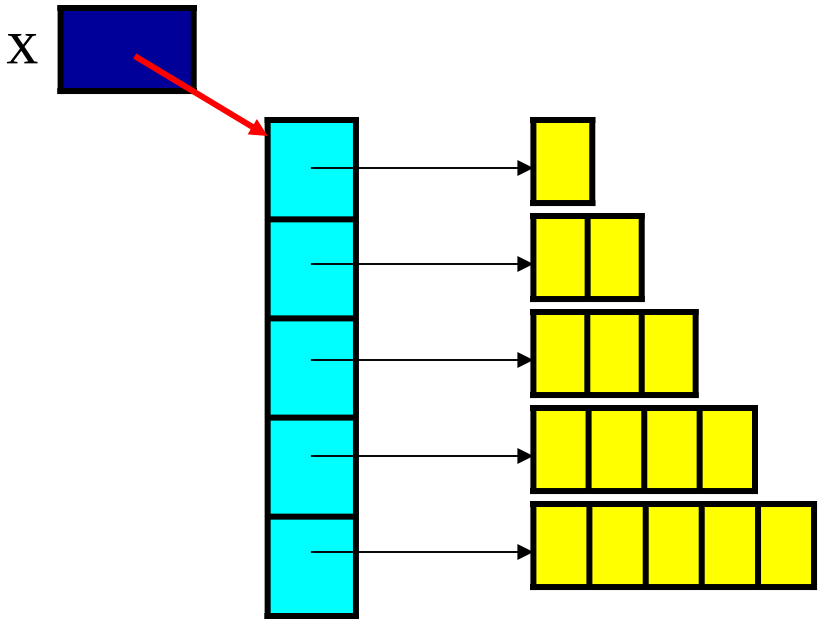
Sorting Objects

- Integers have an inherent order, but the order of a set of objects must be defined by the person defining the class
- Recall that a Java interface can be used as a type name and guarantees that a particular class has implemented particular methods
- We can use the `Comparable` interface to develop a generic sort for a set of objects

Arrays of -Varying Length

- The size of each row of a two-dimensional array can be different;

```
int[][] x;
x = new int[5][];
x[0] = new int[1];
x[1] = new int[2];
x[2] = new int[3];
x[3] = new int[4];
x[4] = new int[5];

x.length       ➔  5
x[0].length    ➔  1
x[1].length    ➔  2

? x[1] = new int[10];
```



```
public class ArrayOfArraysDemo2 {
   public static void main(String[] args) {
     int[][] aMatrix = new int[4][];
     //populate matrix
     for (int i = 0; i < aMatrix.length; i++) {
        aMatrix[i] = new int[5];    //create sub-array
          for (int j = 0; j < aMatrix[i].length; j++) {
              aMatrix[i][j] = i + j;
          }
     }
     //print matrix
     for (int i = 0; i < aMatrix.length; i++) {
        for (int j = 0; j < aMatrix[i].length; j++) {
           System.out.print(aMatrix[i][j] + " ");
        }
        System.out.println();
     }
  }
}
```


String and char Arrays in Java

- Strings in Java are not `char` arrays (in C and C++ strings are `char` arrays).
- But, we can convert a `char` array into a string or a string into a `char` array.
  - char array into string:

```
char[] name = {'m','a','r','k'};
String aname = new String(name};
```

  - string into char array:

```
String aname = "mary";
char[] name = aname.toCharArray();
```


toString() method

- Recall that the `toString()` method of a class allows the string representation of its instances.

```
public class ComplexNum {
  private double realpart, imaginarypart;
  public ComplexNum(double r, double i) {
    realpart=r; imaginarypart=i;
  }
  public String toString() {
 return(realpart+"+"+imaginarypart+"i");
  } }

ComplexNum cn1 = new ComplexNum(1.0,2.0);
System.out.println("The complex number: " + cn1);
```

"a"+cn1 is equivalent to **"a"+cn1.toString()**

Copy from standard input to standard output

```
import java.io.*;

public class CopyString {
  public static void main(String[] args) {
    try {
      BufferedReader stdin =
        new BufferedReader(new
                    InputStreamReader(System.in));
        String aline;
        while ((aline=stdin.readLine()) != null)
          System.out.println("your line: " + aline);
     } catch(IOException e) {}
   }
}
```

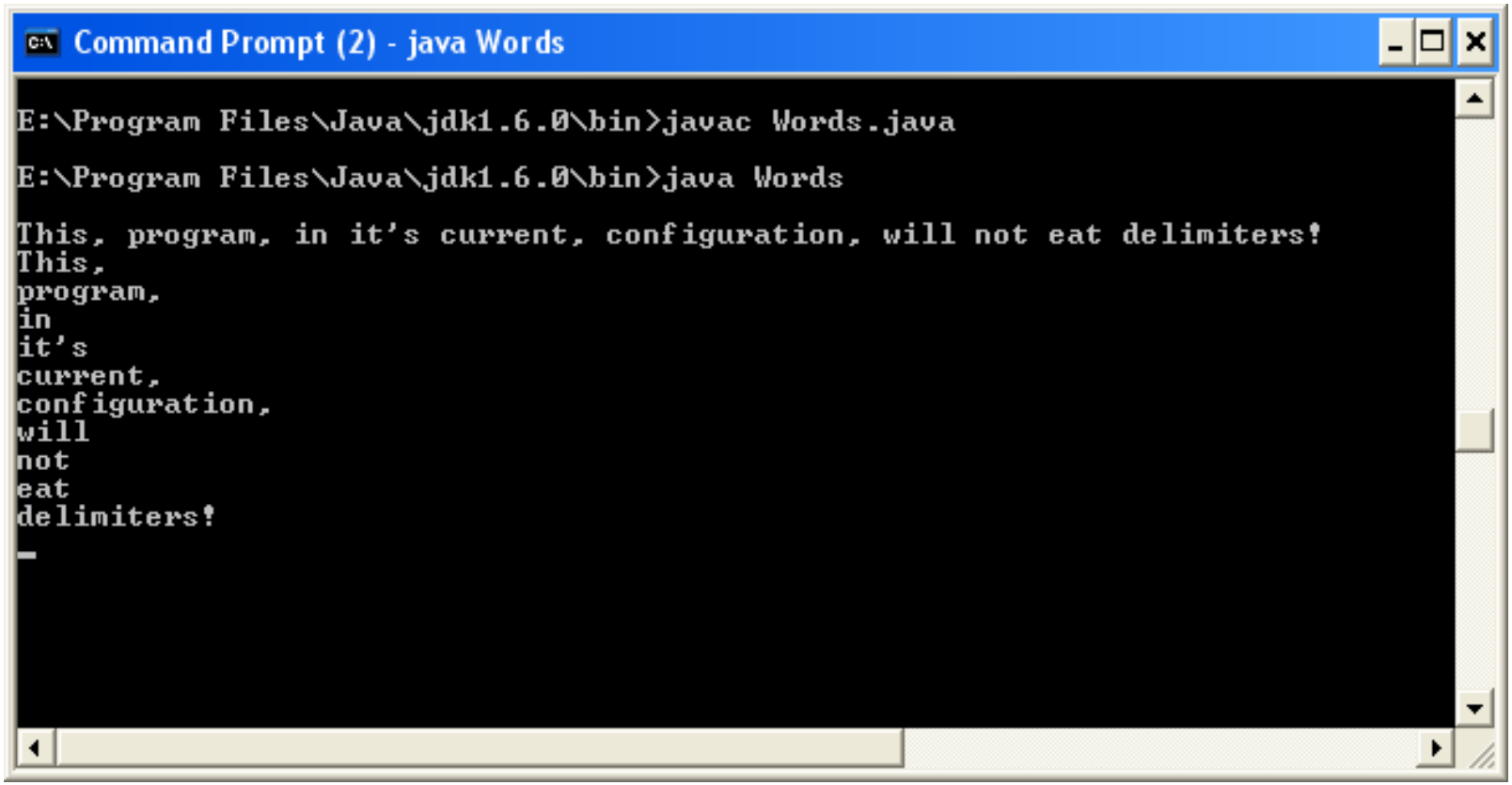

Copy from a file to another file

```
import java.io.*;
public class CopyFile {
  public static void main(String[] args) {
    try {
      BufferedReader stdin =
        new BufferedReader(new InputStreamReader(System.in));
      System.out.println("Input File Name: ");
      String infname = stdin.readLine();
      System.out.println("Output File Name: ");
      String outfname = stdin.readLine();
      BufferedReader infile = new BufferedReader(new
        FileReader(infname));
      PrintWriter outfile =
        new PrintWriter(new BufferedWriter(new
        FileWriter(outfname)));
      String aline;
      while ((aline=infile.readLine()) != null)
         outfile.println(aline);
      outfile.flush();  outfile.close();
    } catch(IOException e) {}
  }}
```


StringTokenizer Class

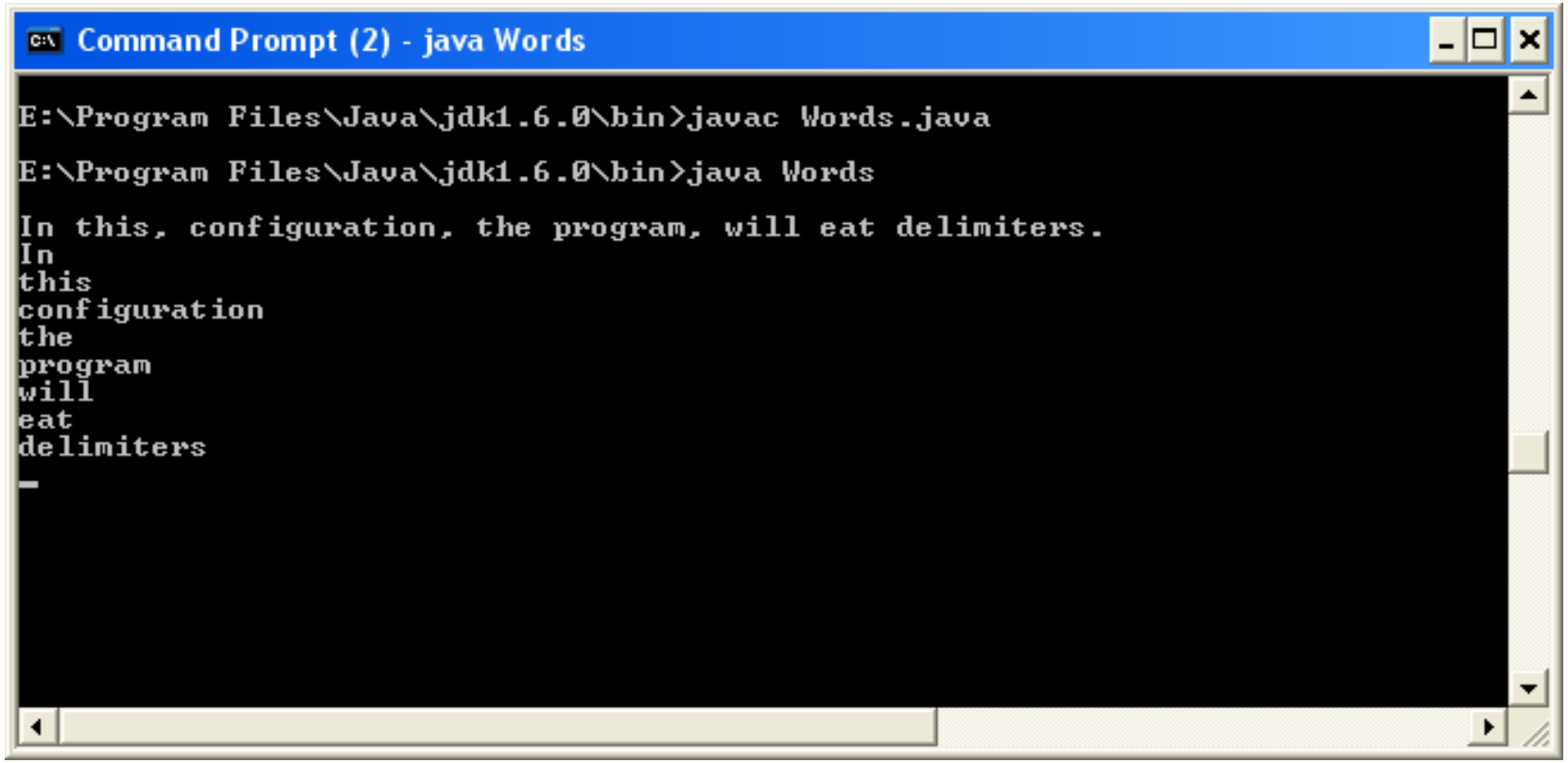
```
import java.io.*;
import java.util.*;
public class Words {
  public static void main(String[] args) {
    try {
      BufferedReader stdin =
        new BufferedReader(new InputStreamReader(System.in));
      String aline;
      //String delim=",. ";
      while ((aline=stdin.readLine()) != null) {
         StringTokenizer st = new StringTokenizer(aline);
         //StringTokenizer st = new StringTokenizer(aline,delim);
         while (st.hasMoreTokens())
           System.out.println(st.nextToken());
     }
    } catch(IOException e) {}
  } }
```


Output of Words Application



without delimeters

Output of Words Application



with delimeters , and .

Collections Framework

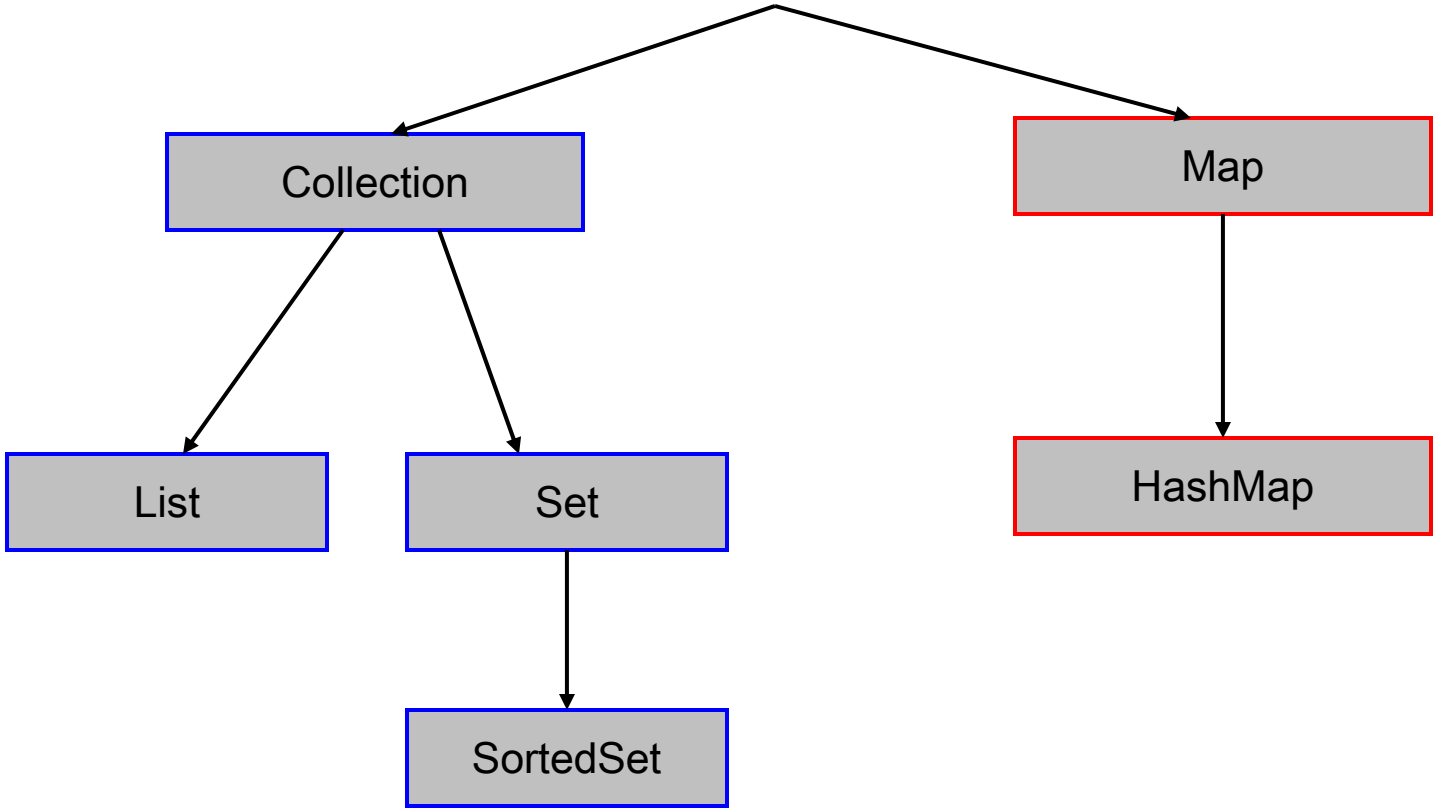
- Although arrays in Java are more robust than arrays in other programming languages they suffer from the traditional shortcoming - they cannot be resized; that is, there are no Java array operations that support the insertion of new elements or the deletion of existing elements.
- For software projects being developed in other languages, the restrictions on arrays often force developers to use nonportable alternative list representations.
  - The cost of using nonportable representations can be quite high because developers must create and support multiple versions of their software.

Collections Framework (cont.)

- Through its **collections framework**, Java has a large set of list representations.
  - With this framework, Java software developers typically can avoid the expense of developing new list representations.
  - In addition, because the framework provides an extensive collection of algorithms for examining and manipulating its list representations, software developers can rely on the correctness of these data structures and concentrate their resources on the problem-specific aspects of their projects.
- There are two types of list representations in the collections framework – those that implement the interface `java.util.Collection` and those that implement the interface `java.util.Map`.

Collections Framework (cont.)

The two interface hierarchies in the Collections network



Collections Framework (cont.)

- Classes that implement one of the derived `Collection` interfaces `List`, `Set`, and `SortedSet` represent lists as we normally imagine them. Such classes support a view of a list as a group of elements.
- Classes that implement the `Map` interface or its derived interface `HashMap` take a more associative view; that is, these classes provide the means to associate "keys" with values. The `Map`-based classes also provide the means to determine the value associated with a key and vice versa.

ArrayList Class

- We'll focus on the `ArrayList` class which provides a resizeable list representation that implements the `List` interface.
  - The name `ArrayList` is intended to be doubly suggestive. An `ArrayList` uses an array to represent the elements of its list. In addition, an `ArrayList` has been designed so that its element accessor and mutator methods are guaranteed to be very efficient, i.e., O(1).
  - Associated with each `ArrayList` is a *capacity*, which is the maximum number of elements that the list can store without growing. The capacity is the size of the array the `ArrayList` is currently using to store the elements of the list. If the capacity of that array becomes insufficient, then a new array is created with greater capacity for the `ArrayList` and the values from the old list are copied to it. The operation is very similar to the method `doubleCapacity()` from the notes of Day 15.

ArrayList Class (cont.)

- In addition to its accessor and mutator methods the `ArrayList` class provides the ability to add (append) an element to the end of the list.
  - The append operation is guaranteed to be O(1) on average.
- The class also provides a number of other methods for inserting and deleting elements to the list.
  - Most of these other methods require O(n) time (n is the number of elements in the list) to perform their tasks.

Selected ArrayList Constructors and Methods

```
public void add(int i, Object v)
```

- Inserts value v into the list such that v has index i. Any preexisting elements with indices i or greater are shifted backwards by one element (to a higher index value).

```
public boolean add(Object v)
```

- Appends the list with a new element with value v and returns true.

```
public void clear()
```

- Removes all elements from the list.

```
public Object clone()
```

- Returns a shallow copy of this list.

```
public Object get(int i)
```

- If i is a valid index, it returns the ith element; otherwise an exception is generated.

Selected ArrayList Constructors and Methods

`public boolean isEmpty()`

- Returns true if there are no more elements; otherwise, it returns false.

`public Object remove(int i)`

- If i is a valid index, it removes the ith element from the list by shifting forward (to a smaller index) elements i+1 and on. In addition, the removed value is returned. Otherwise, an exception is generated.

`public Object set(int i, Object v)`

- If i is a valid index, then the ith element is set to v and the previous value of the element is returned. Otherwise, an exception is thrown.

`public int size()`

- Returns the number of elements in the list.

Examples Using the ArrayList Class

```
ArrayList c = new ArrayList();
ArrayList d = new ArrayList();
c.add("Colnago");
c.add("Eddy Merckx");
```