ARRAYS

Computer Engineering Department

Java Course

Prof. Dr. Ahmet Sayar

Kocaeli University - Fall 2021

Introduction to Arrays

- An *array* is an object used to store a (possibly large) collection of data.
- All the data stored in the array must be of the same type.
- An array object has a small number of predefined methods.

Introduction to Arrays, cont.

- Sometimes you want to know several things about a collection of data:
  - the average
  - the number of items below the average
  - the number of items above the average
  - etc.
- This requires all the items to be stored as variables of one kind or another.

ARRAYS

- You can make an array of any type
  - int, double, String
- All elements of an array must have the same type

1. int[] values = new int[5];
2. int[] values = { 12, 24, -23, 47, 22 };

- values[1] = ?
- values[values.length] = ?
- int length = values.length;   // not lenght or length()!!!

Array Details

- Syntax for creating an array

```
Base_Type[] Array_Name = new Base_Type[Length];
```

- Arrays of primitive types or complex types
- Example

```
int[] pressure = new int[100];
```

or

```
int[] pressure;
pressure = new int[100];
```

Indices and `length`

- The indices of an array start with `0` and end with *`Array_Name`*`.length-1`.

- When a for loop is used to step through an array, the loop control variable should start at `0` and end at `length-1`.

- Example

```
 for (lcv = 0; lcv < temperature.length; index++)
```

Initializing Arrays

- An array can be initialized at the time it is declared.
- example

```
double[] reading = {3.3, 15.8, 9.7};
```

  – The size of the array is determined by the number of values in the *initializer list*.

Arguments for the Method `main`

- Recall the heading for method `main`:

```
public static void main(String[] args)
```

- Method `main` takes an array of `String` values as its argument.
- A default array of `String` values is when you run a program.

Arguments for the Method `main`, cont.

- Alternatively, an array of `String` values can be provided in the command line.
- Example

```
java TestProgram Mary Lou
```

  - `args[0]` is set to `“Mary”`
  - `args[1]` is set to `“Lou”`

```
System.out.println(“Hello “ + args[0] +
     “ “ + args[1]);
```

prints `Hello Mary Lou.`

Use of $=$ and $==$ with Arrays

- The assignment operator '='and the equality operator '==', when used with arrays, behave the same as when used with other objects.
  - The assignment operator creates an alias, **not** a copy of the array.
  - The equality operator determines if two references contain the same memory address, **not** if two arrays contain the same values.

Making a Copy of an Array

- 1.way

```
int[] a = new int[50];
int[] b = new int[50];
...
for (int j = 0; j  < a.length; j++)
  b[j] = a[j];
```

- 2.way

```
  Clone()
  b = a.clone();
```

Determining the “Equality” of Two Arrays, cont.

- A `while` loop can be used.
  - A boolean variable `match` is set to `true`.
  - Each element in the first array is compared to the corresponding element in the second array.
  - If two elements are different, `match` is set to `false` and the `while` loop is exited.
  - Otherwise, the loop terminates with `match` still set to `true`.

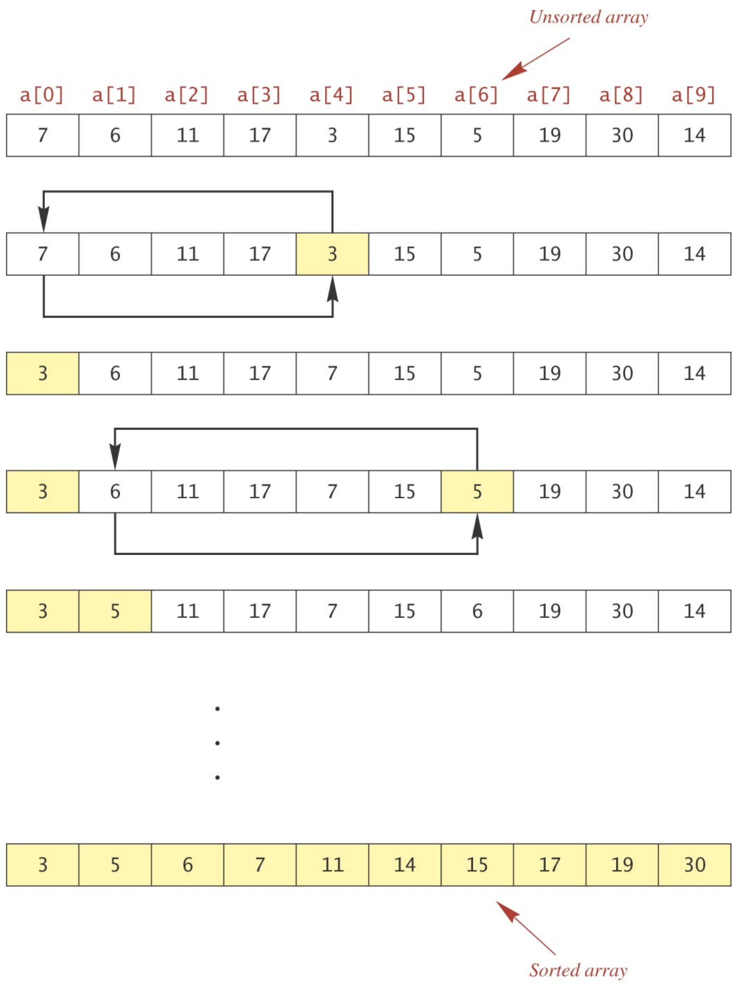
Sorting Arrays

- Sometime we want numbers in an array sorted from smallest to largest, or from largest to smallest.
- Sometimes we want the strings referenced by an array to be in alphabetical order.
- Sorting techniques typically are easy to adapt to sort any type that can be ordered.

Selection Sort, cont.

- Selection sort begins by finding a smallest item in the array and swapping it with the item in `a[0]`.
- Selection sort continue by finding a smallest item in the remainder of the array and swapping it with the next item in array `a`.
- Selection sort terminates when only one item remains.

Selection Sort, cont.



Display 6.12

Sorting by Swapping Values

Swapping Elements

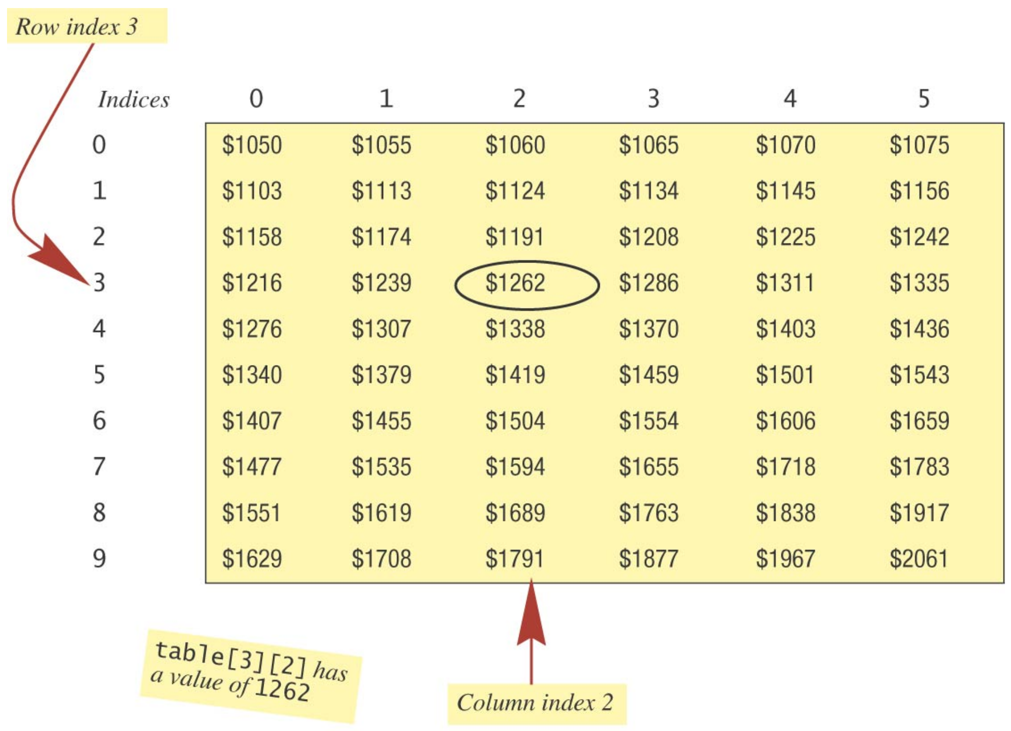
- To swap two elements `a[i]` and `a[j]`, one of them must be saved temporarily.

```
    /**
     Precondition: i and j are valid indices for the array a.
     Postcondition: Values of a[i] and a[j] have been interchanged.
    */
    private static void interchange(int i, int j, int[] a)
    {
        int temp;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp; //original value of a[i]
    }
}
```

Introduction to Multidimensional Arrays

- An array with more than one index sometimes is useful.
- Example: savings account balances at various interest rates for various numbers of years
  - columns for interest rates
  - rows for years
- This two-dimensional table calls for a *two-dimensional array*.

Introduction to Multidimensional Arrays, cont.



Row index 3
table[3][2] *has a value of* 1262
Column index 2

Indices	0	1	2	3	4	5
0	$1050	$1055	$1060	$1065	$1070	$1075
1	$1103	$1113	$1124	$1134	$1145	$1156
2	$1158	$1174	$1191	$1208	$1225	$1242
3	$1216	$1239	$1262	$1286	$1311	$1335
4	$1276	$1307	$1338	$1370	$1403	$1436
5	$1340	$1379	$1419	$1459	$1501	$1543
6	$1407	$1455	$1504	$1554	$1606	$1659
7	$1477	$1535	$1594	$1655	$1718	$1783
8	$1551	$1619	$1689	$1763	$1838	$1917
9	$1629	$1708	$1791	$1877	$1967	$2061

Display 6.16

Row and Column Indices for an Array Named table

Multidimensional-Array Basics

- example declaration

```
int[][] table = new int [10][6];
```

  or

```
int[][] table;
table = new int[10][6];
```

- The number of bracket pairs in the declaration is the same as the number of indices.

Multidimensional-Array Basics, cont.

- Syntax

```
Base_Type[]…[] Array_Name =
    new Base_Type[Length_1]…[Length_n];
```

- Examples

```
char[][] page = new char [100][80];
double[][][] threeDPicture =
                         new double[10][20][30];

SomeClass[][] entry = new SomeClass[100][80];
```

Multidimensional-Array Basics, cont.

- Nested `for` loops can be used to set the values of the elements of the array and to print the values of the elements of the array.

2-dim Arrays

- int [][] test = new int [][];
  - Error: Array dimension is missing
- int [][] test = new int [3][];
  - Raw number is 3, but column number is not defined yet – OK?
  - System.out.println(test.length);   // ??

  - test = new int [3][2];
  - System.out.println(test[1].length);   // ??

Decleration and Initialization in one line

- int[] test = new test[2][2];
- test[0][0]=0;
- test[0][1]=1;
- test[1][0]=2;
- test[1][1]=3;
- OR
- int[] test = {{0,1},{2,3}};

Ragged Arrays

- Since a two-dimensional array in Java is an array of arrays, each row can have a different number of elements (columns).
- Arrays in which rows have different numbers of elements are called *ragged arrays.*

Ragged Arrays, cont.

- Example

```
int[][] b = new int[3][];
b[0] = new int[5];
b[1] = new int[7];
b[2] = new int[4];
```

An Alternative to Arrays -ArrayList Class-

- Implemented by using Array Class
- ArrayList
  - No need to pre-define the size

```
ArrayList<String> list = new ArrayList<String>();
list.add("a");

for (int i = 0; i < list.size(); i++) {
      System.out.println("----- "+list.get(i));
}
```

ArrayList - I

- An ArrayList is a one-dimensional array.
- An ArrayList holds only objects, not primitive values like int, double, char or boolean.
- When you declare a standard array, you have to tell Java a fixed number of memory locations that you want the array to contain.
- When you declare an ArrayList, you don't have to tell Java how much storage space you need. It will resize itself when needed.

ArrayList - II

- ArrayList <Integer> nums = new ArrayList <Integer> ( );
- ArrayList <String> names = new ArrayList <String> ( );
- ArrayList <Student> students = new ArrayList <Student> ( );
- ArrayList <Integer> nums = new ArrayList <Integer> (1000);

- The ArrayList API says that all of the above ArrayLists will be constructed with an initial capacity of ten.
- an ArrayList holds only objects, not primitive values like int, double, char or boolean. But in particular Java has the Integer and Double classes so you can place numbers in an ArrayList

Generic or Raw ArrayList

- ArrayList <String> names = new ArrayList <String> ( );
- ArrayList names = new ArrayList ( );
- Difference:
  - In the second one, you could put all kinds of objects in the same ArrayList, but care had to be taken to make sure only one kind of object was placed in a list. You can still declare and instantiate a raw ArrayList, like the following, but most compilers will “complain” and at least give you a warning if you are using Java 1.5 or higher.

Advantages of ArrayList

- Operations are much much easier and less complex for:
  - traversing an ArrayList (accessing each element to print or perform an operation on it)
  - insertions anywhere in the ArrayList
  - removals anywhere in the ArrayList
  - searching an ArrayList

- An ArrayList tracks its own logical size and grows or shrinks automatically depending on the number of elements it has

ArrayList API – Some samples

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

ArrayList API – Some samples

clear()	Removes all of the elements from this list.
clone()	Returns a shallow copy of this ArrayList instance.
contains(Object o)	Returns true if this list contains the specified element.
ensureCapacity(int minCapacity)	Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
isEmpty()	Returns true if this list contains no elements.
lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).

Sample code

- // Creating an ArrayList of String
- List<String> animals = new ArrayList<>();

- // Adding new elements to the ArrayList
- animals.add("Lion");
- animals.add("Tiger");
- animals.add("Cat");
- animals.add("Dog");

- System.out.println(animals);

- // Adding an element at a particular index in an ArrayList
- animals.add(2, "Elephant");

- System.out.println(animals);

```
•     public class AccessElementsFromArrayListExample {
•        public static void main(String[] args) {
•           List<String> topCompanies = new ArrayList<>();

•           // Check if an ArrayList is empty
•           System.out.println("Is the topCompanies list empty? : " + topCompanies.isEmpty());

•           topCompanies.add("Google");
•           topCompanies.add("Apple");
•           topCompanies.add("Microsoft");
•           topCompanies.add("Amazon");
•           topCompanies.add("Facebook");

•           // Find the size of an ArrayList
•           System.out.println("Here are the top " + topCompanies.size() + " companies in the world");
•           System.out.println(topCompanies);

•           // Retrieve the element at a given index
•           String bestCompany = topCompanies.get(0);
•           String secondBestCompany = topCompanies.get(1);
•           String lastCompany = topCompanies.get(topCompanies.size() - 1);

•           System.out.println("Best Company: " + bestCompany);
•           System.out.println("Second Best Company: " + secondBestCompany);
•           System.out.println("Last Company in the list: " + lastCompany);

•           // Modify the element at a given index
•           topCompanies.set(4, "Walmart");
•           System.out.println("Modified top companies list: " + topCompanies);
•        }
•     }
```

An Alternative to Arrays - Vector Class-

- Implemented by using Array Class
- The `class Vector` can be used to implement a list, replacing a simple array
- The size of a Vector object can grow/shrink during program execution
- The Vector will automatically grow to accommodate the number of elements you put in it
- Vectors can hold only Objects and not primitive types (eg, int)

`class Vector` (continued)

- The `class Vector` is contained in the package `java.util`
- Programs must include either:
  - `import java.util.*;`
  - `import java.util.Vector;`

Vector Declaration

- Declare/initialize

```
Vector<Student> students = new Vector<Student>();
```

- The syntax <…> is used to declare the type of object that will be stored in the Vector
- If you add a different type of object, an exception is thrown
- Not strictly necessary, but highly recommended (compiler warning)
- Only stores Objects – does not have indexing as in Arrays

Vector Size/Capacity

- The *size* of a vector is the number of elements
- The *capacity* of a vector is the maximum number of elements before more memory is needed
- If size exceeds capacity when adding an element, the capacity is automatically increased
  - Declares a larger storage array
  - Copies existing elements, if necessary
  - Growing the capacity is expensive!
- By default, the capacity doubles each time

Setting Initial Capacity

- If you know you will need a large vector, it may be faster to set the initial capacity to something large

```
Vector<Student> students = new Vector<Student>(1000);
```

Size and capacity

- Get the current size and capacity:

```
Vector<Student> students = new Vector<Student>(1000);
students.size();      // returns 0
students.capacity();  // returns 1000
```

- Setting size and capacity:

```
// adds null elements or deletes elements if necessary
students.setSize(10);

// increases capacity if necessary
students.ensureCapacity(10000);
```

Adding Elements

```
Vector<String> stringList = new Vector<String>();
```

```
stringList.add("Spring");
stringList.add("Summer");
stringList.addElement("Fall");
stringList.addElement("Winter");
```

add and addElement have identical functionality

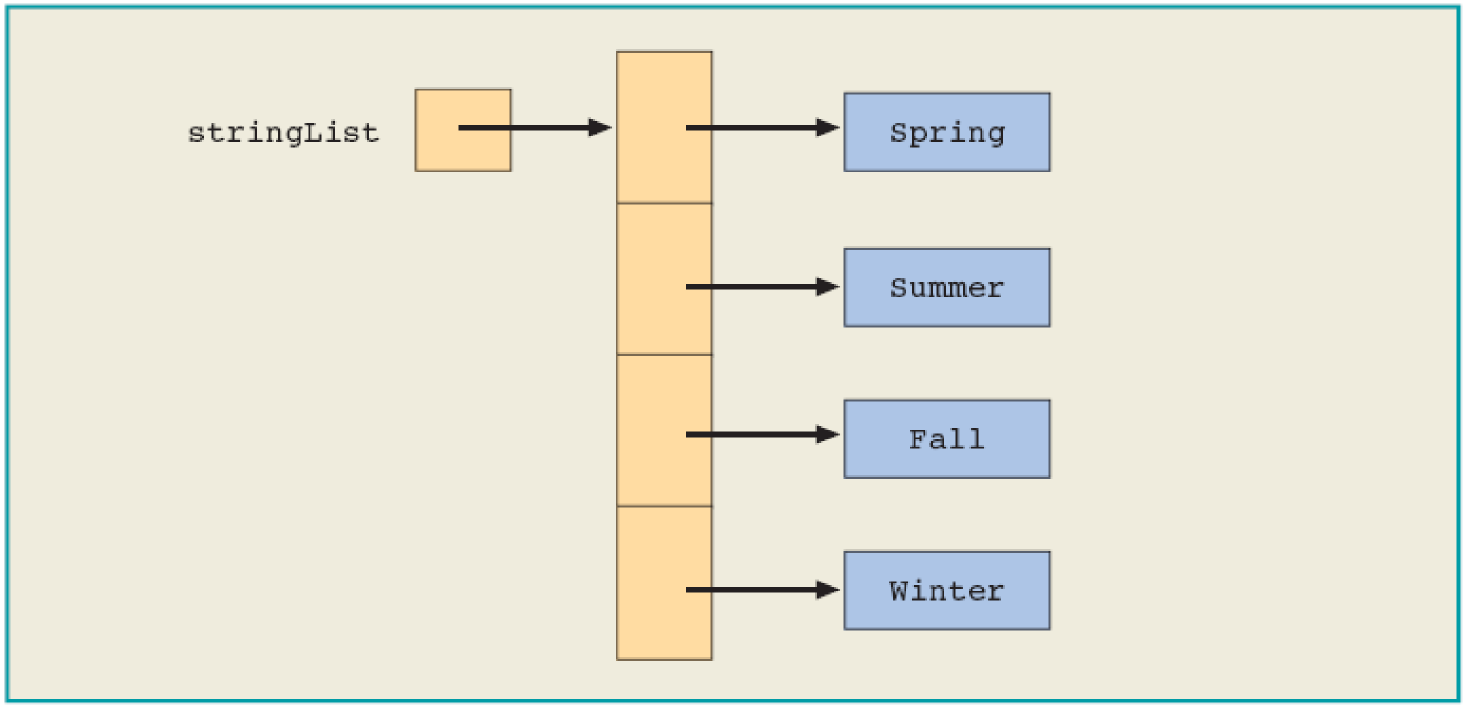


Figure 10-1 stringList after adding the four strings

Accessing Elements

```
stringList.get(0);   // "Spring"
stringList.get(3);   // "Winter"
stringList.get(4);   // ArrayIndexOutOfBounds Exception
```

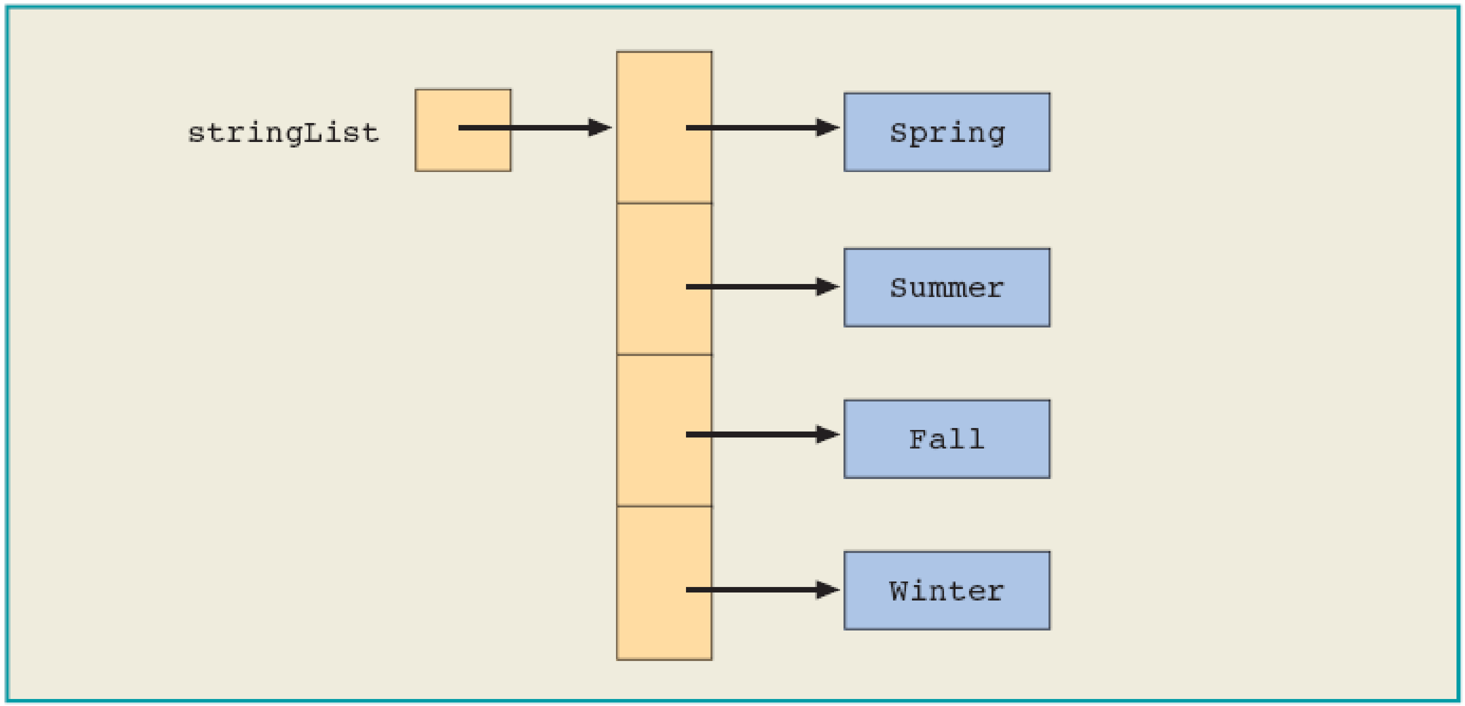


Figure 10-1 stringList after adding the four strings

Vector API – Some samples

`add(`**`value`**`)`	appends value at end of list
`add(int index, E element)`	Inserts the specified element at the specified position in this Vector.
`add(E e)`	Appends the specified element to the end of this Vector.
addAll(int index, Collection<? extends E> c)	Inserts all of the elements in the specified Collection into this Vector at the specified position.
addAll(Collection<? extends E> c)	Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator.
addElement(E obj)	Adds the specified component to the end of this vector, increasing its size by one.
capacity()	Returns the current capacity of this vector.

Vector API – Some samples

clear()	Removes all of the elements from this Vector.
clone()	Returns a clone of this vector.
contains(Object o)	Returns true if this vector contains the specified element.
containsAll(Collection<?> c)	Returns true if this Vector contains all of the elements in the specified Collection.
elementAt(int index)	Returns the component at the specified index.
ensureCapacity(int minCapacity)	Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.
equals(Object o)	Compares the specified Object with this Vector for equality.
get(int index)	Returns the element at the specified position in this Vector.

```
•   public static void main(String[] args) {

•           Vector<Integer> v = new Vector<>();
•           v.add(20);
•           v.add(30);
•           v.add(40);

•           // ... Get element 0.
•           System.out.println(v.get(0));

•           // ... Loop over all indexes, calling get.
•           for (int i = 0; i < v.size(); i++) {
•               int value = v.get(i);
•               System.out.println(value);
•           }
•       }
•   public static void main(String[] args) {

•           // Add ArrayList to Vector.
•           Vector<String> v = new Vector<>();
•           v.add("one");
•           v.add("two");
•           v.add("three");

•           // ... Convert Vector into String array.
•           String[] array = {};
•           String[] result = v.toArray(array);
•           for (String value : result) {
•               System.out.println(value);
•           }
•       }
```