



# Advanced Computer Programming

[Lecture 06]

Saeed Reza Kheradpisheh

kheradpisheh@ut.ac.ir

Department of Computer Science  
Shahid Beheshti University

# ARRAYS and ARRAY LISTS



In many programs, you need to collect large numbers of values. In Java, you use the **array** and **array list** constructs for this purpose. Arrays have a more concise syntax, whereas array lists can automatically grow to any desired size.

# Declaring Arrays

## Definition

An **array** is a sequence of values of the same type.

# Declaring Arrays

## Definition

An **array** is a sequence of values of the same type.

Declaring an array variable of a specific type:

```
type[] name;
```

e.g. `double[] values;`

# Declaring Arrays

## Definition

An **array** is a sequence of values of the same type.

Declaring an array variable of a specific type:

```
type[] name;
```

e.g. `double[] values;`

Declaring an array variable is not enough and you should initialize it:

```
type[] name = new type[length];
```

e.g. Initialize an array variable of type `double` with 10 `double` variables each has the value of zero:

```
double[] values = new double[10];
```

# Declaring Arrays

## Definition

An **array** is a sequence of values of the same type.

Declaring an array variable of a specific type:

```
type[] name;
```

e.g. `double[] values;`

Declaring an array variable is not enough and you should initialize it:

```
type[] name = new type[length];
```

e.g. Initialize an array variable of type `double` with 10 `double` variables each has the value of zero:

```
double[] values = new double[10];
```

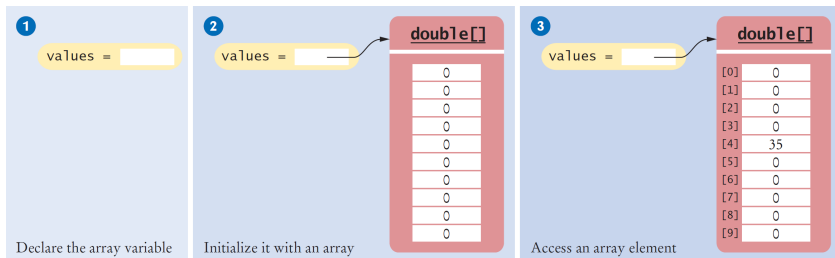
When you declare an array, you can specify the initial values:

```
double[] values = { 32, 54, 67.5, 29, 35, 80};
```

## Accessing Array Elements

- Individual elements in an array are accessed by an integer index  $i$ , using the notation `array[i]`.
- An array element can be used like any variable.


```
double[] values = new double[10];  
values[4] = 35;
```



Note that indices starts from 0 to `values.length - 1`

# Array Examples

Table 1 Declaring Arrays

|   |  |
|---|--|
| <pre>int[] numbers = new int[10];</pre>   | An array of ten integers. All elements are initialized with zero.                    |
| <pre>final int LENGTH = 10;<br/>int[] numbers = new int[LENGTH];</pre>  | It is a good idea to use a named constant instead of a “magic number”.               |
| <pre>int length = in.nextInt();<br/>double[] data = new double[length];</pre>   | The length need not be a constant.   |
| <pre>int[] squares = { 0, 1, 4, 9, 16 };</pre>  | An array of five integers, with initial values.                                      |
| <pre>String[] friends = { "Emily", "Bob", "Cindy" };</pre>  | An array of three strings.   |
|  <pre>double[] data = new int[10];</pre> | <b>Error:</b> You cannot initialize a double[] variable with an array of type int[]. |



# Array References

## Definition

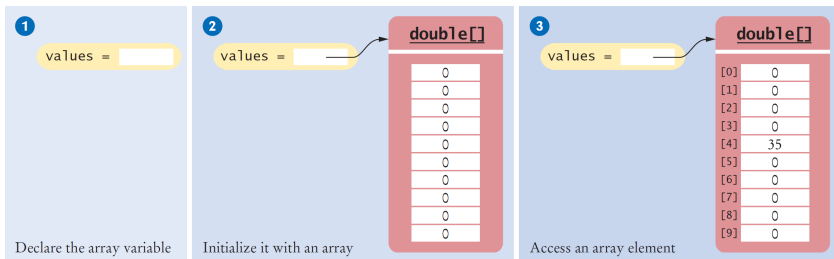
The **reference** denotes the location of the array in memory.

# Array References

## Definition

The **reference** denotes the location of the array in memory.

The array variable does not store any elements. Instead, the array elements are stored elsewhere in the memory and the array variable holds a reference to that location.



## Array References

Consider the following code:

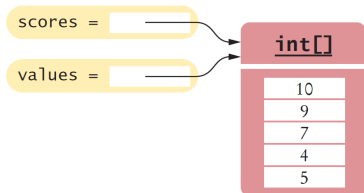
```
int[] scores = { 10, 9, 7, 4, 5 };  
int[] values = scores;
```

## Array References

Consider the following code:

```
int[] scores = { 10, 9, 7, 4, 5 };  
int[] values = scores;
```

When you copy an array variable into another, **both variables refer to the same array**, because you copy the reference not the elements!



You can modify the array through either of the variables:

```
scores[3] = 10;  
System.out.println(values[3]); // Prints 10
```

## Array Common Errors

- **Bounds Errors**

Perhaps the most common error in using arrays is accessing a nonexistent element.

```
double[] values = new double[10];  
values[10] = 5.4; // Error!
```

## Array Common Errors

- **Bounds Errors**

Perhaps the most common error in using arrays is accessing a nonexistent element.

```
double[] values = new double[10];  
values[10] = 5.4; // Error!
```

- **Uninitialized Arrays**

A common error is to allocate an array variable, but not an actual array.

```
double[] values;  
values[0] = 29.95; // Error-values not initialized
```

# The Enhanced for Loop

## Usage

You can use the **enhanced for loop** to visit (read) all elements of an array.

*Syntax*    `for (typeName variable : collection)`  
          `{`  
              `statements`  
          `}`

This variable is set in each loop iteration.  
It is only defined inside the loop.

An array

```
for (double element : values)
{
    sum = sum + element;
}
```

These statements  
are executed for each  
element.

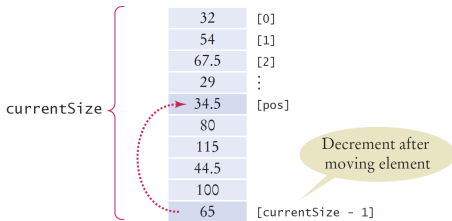
The variable  
contains an element,  
not an index.

# Common Array Algorithms

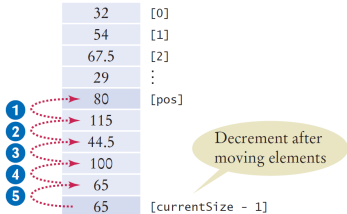
- Filling
- Sum and Average Value
- Maximum and Minimum
- Linear Search
- Removing an Element
- Inserting an Element
- Swapping Elements
- Copying Arrays
- Reading Input



# Removing an Element

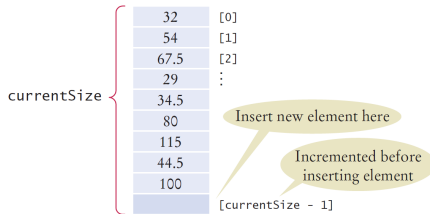


**Figure 4**  
Removing an Element in an Unordered Array

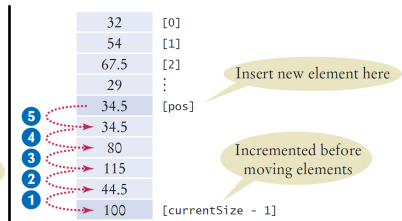


**Figure 5**  
Removing an Element in an Ordered Array

# Inserting an Element

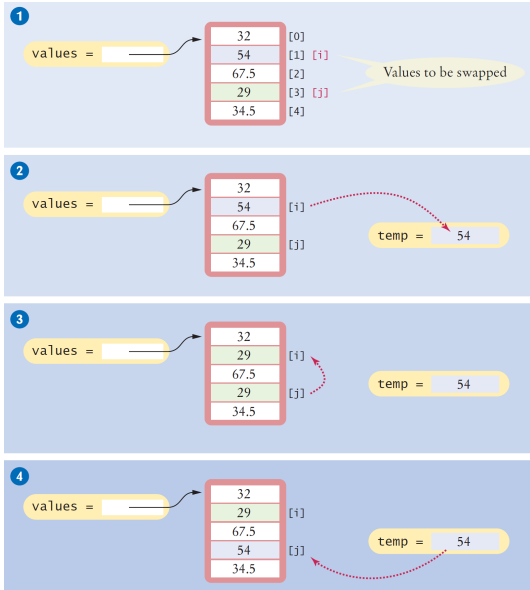


**Figure 6**  
Inserting an Element in an Unordered Array



**Figure 7**  
Inserting an Element in an Ordered Array

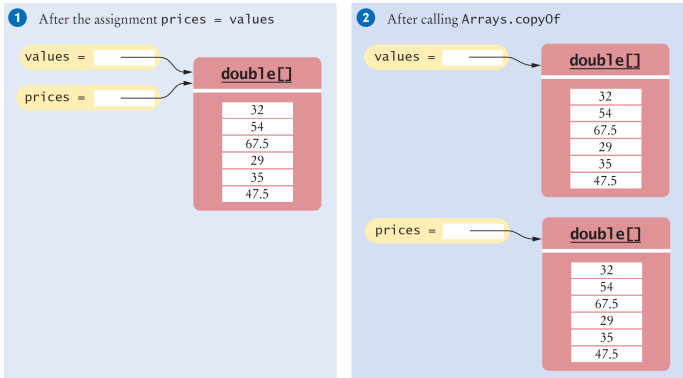
# Swapping Two Elements



## Copying Arrays

If you want to make a **true copy** of an array, call the `Arrays.copyOf` method. The call `Arrays.copyOf(values, n)` allocates an array of length `n`, copies the first `n` elements of `values` (or the entire `values` array if `n > values.length`) into it, and returns the new array.

```
double[] prices = Arrays.copyOf(values, values.length);
```



```
import java.util.Arrays;
```

# Using Arrays with Methods

## Using Arrays with Methods

What would be the output of the following code?

```
public static double sum(double[] values)
{
    for(int i = 1; i < values.length; i++)
        values[i] = values[i] + values[i - 1];
    return values[values.length - 1];
}
public static void main(String[] args)
{
    double[] data = {10, 20, 30};
    System.out.println(sum(data));
    System.out.println(data[2]);
}
```

## Using Arrays with Methods

What would be the output of the following code?

```
public static double sum(double[] values)
{
    for(int i = 1; i < values.length; i++)
        values[i] = values[i] + values[i - 1];
    return values[values.length - 1];
}
public static void main(String[] args)
{
    double[] data = {10, 20, 30};
    System.out.println(sum(data));
    System.out.println(data[2]);
}
```

Note that the copy of the array reference is passed instead of the copy of the array elements.

## Exercise (EvenSum.java)

Write a method that takes an integer array as input and returns the sum of elements at even positions.



### Exercise (`EvenSum.java`)

Write a method that takes an integer array as input and returns the sum of elements at even positions.

### Exercise (`RevArray.java`)

Write a method that takes an integer array as input and returns a new array which is filled with the input elements in reverse order.

### Exercise (`EvenSum.java`)

Write a method that takes an integer array as input and returns the sum of elements at even positions.

### Exercise (`RevArray.java`)

Write a method that takes an integer array as input and returns a new array which is filled with the input elements in reverse order.

### Exercise (`Login.java`)

Write a program that simulates a login system which takes a username and a password, checks for a match, and prompt the user with success or failure.

## Two-Dimensional Arrays

### Definition

An arrangement consisting of rows and columns of values is called a **two-dimensional array**, or a **matrix**.

```
double[][] tableEntries = new double[7][3];
```

Name      Element type      Number of rows  
Number of columns

All values are initialized with 0.

```
int[][] data = {  
    { 16, 3, 2, 13 },  
    { 5, 10, 11, 8 },  
    { 9, 6, 7, 12 },  
    { 4, 15, 14, 1 },  
};
```

Name

List of initial values

Individual elements in a two-dimensional array are accessed by using two index values, `array[i][j]`.

## Two-Dimensional Arrays

Two-dimensional arrays are in fact arrays of arrays!

## Two-Dimensional Arrays

Two-dimensional arrays are in fact arrays of arrays!

You can create rows of either the same size or different sizes

- `double[][] a = new double[5][3];`  
Creates 5 rows each has the length of 3;

## Two-Dimensional Arrays

Two-dimensional arrays are in fact arrays of arrays!

You can create rows of either the same size or different sizes

- `double[][] a = new double[5][3];`  
Creates 5 rows each has the length of 3;

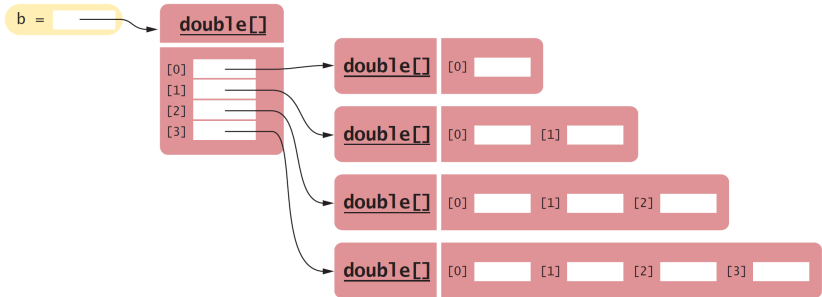
- `double[][] b = new double[3][];`  
Create 5 rows which are uninitialized!

So you can initialize each of them with an arbitrary size:

```
for(int i = 0; i < b.length; i++)  
    b[i] = new double[i + 1];
```

Note that `b[i]` is a reference to a one-dimensional array of type `double`.

# Two-Dimensional Arrays



# Multidimensional Arrays

Can you guess?



## Multidimensional Arrays

Can you guess?

```
type[] name = new type[size];  
type[][] name = new type[size][];  
type[][][] name = new type[size][][];  
type[][][][] name = new type[size][][][];  
type[][][][][] name = new type[size][][][][];  
.  
.  
.
```

### Exercise (`MatrixProduct.java`)

Write a method that takes two 2-D integer matrices  $a$  and  $b$  as inputs and returns  $a \times b$ .

### Exercise (`MatrixProduct.java`)

Write a method that takes two 2-D integer matrices  $a$  and  $b$  as inputs and returns  $a \times b$ .

### Exercise (`FallDown.java`)

Write a program that takes a map  $m$  and find the path that enters from the top and exits from the bottom.

Input maps have  $r$  rows ( $r$  is an odd integer) and  $c$  columns. Even rows of the map (starting from zero) are blocked with tiles. In each row of blocks, there is exactly one hole through which you can pass down.

```
# #####      #o#####
#           #      #oooo #
##### ##      #####o##
#           # → #  ooo #
## #####      ##o#####
#           #      # o  #
## #####      ##o#####
```

## Array Lists

### Definition

An **array list** is a container which stores a sequence of values and its size can change (arrays have a fixed size).

# Array Lists

## Definition

An **array list** is a container which stores a sequence of values and its size can change (arrays have a fixed size).

```
import java.util.ArrayList;
```

**Syntax** To construct an array list: `new ArrayList<typeName>()`  
To access an element: `arraylistReference.get(index)`  
`arraylistReference.set(index, value)`

Variable type \ Variable name \ An array list object of size 0  
`ArrayList<String> friends = new ArrayList<String>();`

Use the  
get and set methods  
to access an element.

```
friends.add("Cindy");  
String name = friends.get(i);  
friends.set(i, "Harry");
```

The add method  
appends an element to the array list,  
increasing its size.

The index must be  $\geq 0$  and  $< \text{friends.size}()$ .

## Array Lists

- The type `ArrayList<Type>` denotes an array list of `Type` elements (type parameter).
- You cannot use primitive types as type parameters.

## Array Lists

- The type `ArrayList<Type>` denotes an array list of `Type` elements (type parameter).
- You cannot use primitive types as type parameters.
- Array lists must be initialized:  
`ArrayList<Type> name = new ArrayList<Type> ();`
- An array list variable stores a reference to the actual location of elements.

## Array Lists

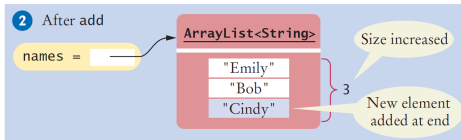
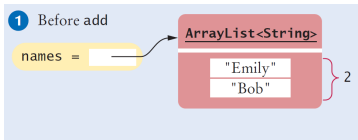
- The type `ArrayList<Type>` denotes an array list of `Type` elements (type parameter).
- You cannot use primitive types as type parameters.
- Array lists must be initialized:  

```
ArrayList<Type> name = new ArrayList<Type>() ;
```
- An array list variable stores a reference to the actual location of elements.
- When the `ArrayList<Type>` is first constructed, it has size 0.
- You can add elements to the end of an array list by use of the `add` method.



## Adding Elements to an Array List

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Emily");  
names.add("Bob");  
names.add("Cindy");
```



# Working with Array Lists

Table 2 Working with Array Lists

|   |  |
|---|--|
| <pre>ArrayList&lt;String&gt; names = new ArrayList&lt;String&gt;();</pre>   | Constructs an empty array list that can hold strings.                      |
| <pre>names.add("Ann");<br/>names.add("Cindy");</pre>  | Adds elements to the end.  |
| <pre>System.out.println(names);</pre>   | Prints [Ann, Cindy].   |
| <pre>names.add(1, "Bob");</pre>   | Inserts an element at index 1.<br>names is now [Ann, Bob, Cindy].          |
| <pre>names.remove(0);</pre>   | Removes the element at index 0.<br>names is now [Bob, Cindy].              |
| <pre>names.set(0, "Bill");</pre>  | Replaces an element with a different value.<br>names is now [Bill, Cindy]. |
| <pre>String name = names.get(i);</pre>  | Gets an element.   |
| <pre>String last = names.get(names.size() - 1);</pre>   | Gets the last element.   |
| <pre>ArrayList&lt;Integer&gt; squares = new ArrayList&lt;Integer&gt;();<br/>for (int i = 0; i &lt; 10; i++)<br/>{<br/>    squares.add(i * i);<br/>}</pre> | Constructs an array list holding the first ten squares.                    |

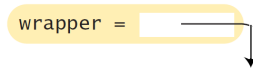
## Creating Array List of Primitive Types

Use the **wrapper classes** of each primitive type. Variables of wrapper types store reference.

```
Double wrapper = 29.95;
```

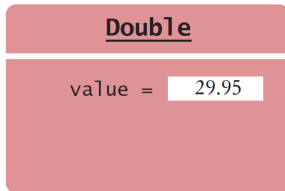
| Primitive Type | Wrapper Class |
|----------------|---------------|
| byte           | Byte          |
| boolean        | Boolean       |
| char           | Character     |
| double         | Double        |
| float          | Float         |
| int            | Integer       |
| long           | Long          |
| short          | Short         |

wrapper =

A yellow rounded rectangle contains the text 'wrapper = ' followed by a white rectangular box. An arrow points from the right side of this box down to the top of a larger red rounded rectangle below.

**Double**

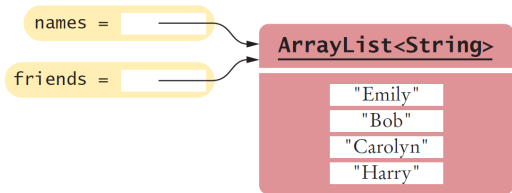
value =

A red rounded rectangle is divided into two horizontal sections. The top section is a lighter shade of red and contains the text 'Double' with a horizontal line underneath it. The bottom section is a darker shade of red and contains the text 'value = ' followed by a white rectangular box containing the number '29.95'.

## Copying Array List

### Reference Copy:

```
ArrayList<String> friends = names;  
names.add("Harry");
```



### True Copy:

```
ArrayList<String> newNames =  
new ArrayList<String> (names);
```

## Comparing Array and Array List Operations

| Operation                                  | Arrays   | Array Lists   |
|--|--|---|
| Get an element.                            | <code>x = values[4];</code>  | <code>x = values.get(4)</code>                                    |
| Replace an element.                        | <code>values[4] = 35;</code>   | <code>values.set(4, 35);</code>                                   |
| Number of elements.                        | <code>values.length</code>   | <code>values.size()</code>  |
| Number of filled elements.                 | <code>currentSize</code><br>(companion variable, see<br>Section 6.1.3) | <code>values.size()</code>  |
| Remove an element.                         | See Section 6.3.6  | <code>values.remove(4);</code>                                    |
| Add an element, growing<br>the collection. | See Section 6.3.7  | <code>values.add(35);</code>                                      |
| Initializing a collection.                 | <code>int[] values = { 1, 4, 9 };</code>                               | No initializer list syntax;<br>call <code>add</code> three times. |

length, length(), and size()

| Data Type  | Number of Elements |
|------------|--------------------|
| Array      | a.length           |
| Array list | a.size()           |
| String     | a.length()         |

### Exercise (`ArrayToList.java`)

Write a method that converts a 2-D array of strings into an array list of array lists of strings!

### Exercise (`ArrayToList.java`)

Write a method that converts a 2-D array of strings into an array list of array lists of strings!

### Exercise (`PrimeFactors.java`)

Write a method that takes an integer and returns all of its prime factors.



### Exercise (`ArrayToList.java`)

Write a method that converts a 2-D array of strings into an array list of array lists of strings!

### Exercise (`PrimeFactors.java`)

Write a method that takes an integer and returns all of its prime factors.

### Exercise (`ExtractWords.java`)

Write a method that takes a string and returns all of the words in that string.