ÖZYEĞİN UNIVERSITY

CS105
Introduction to Object-Oriented Programming

Prof. Dr. Nizamettin AYDIN

naydin@itu.edu.tr

nizamettin.aydin@ozyegin.edu.tr

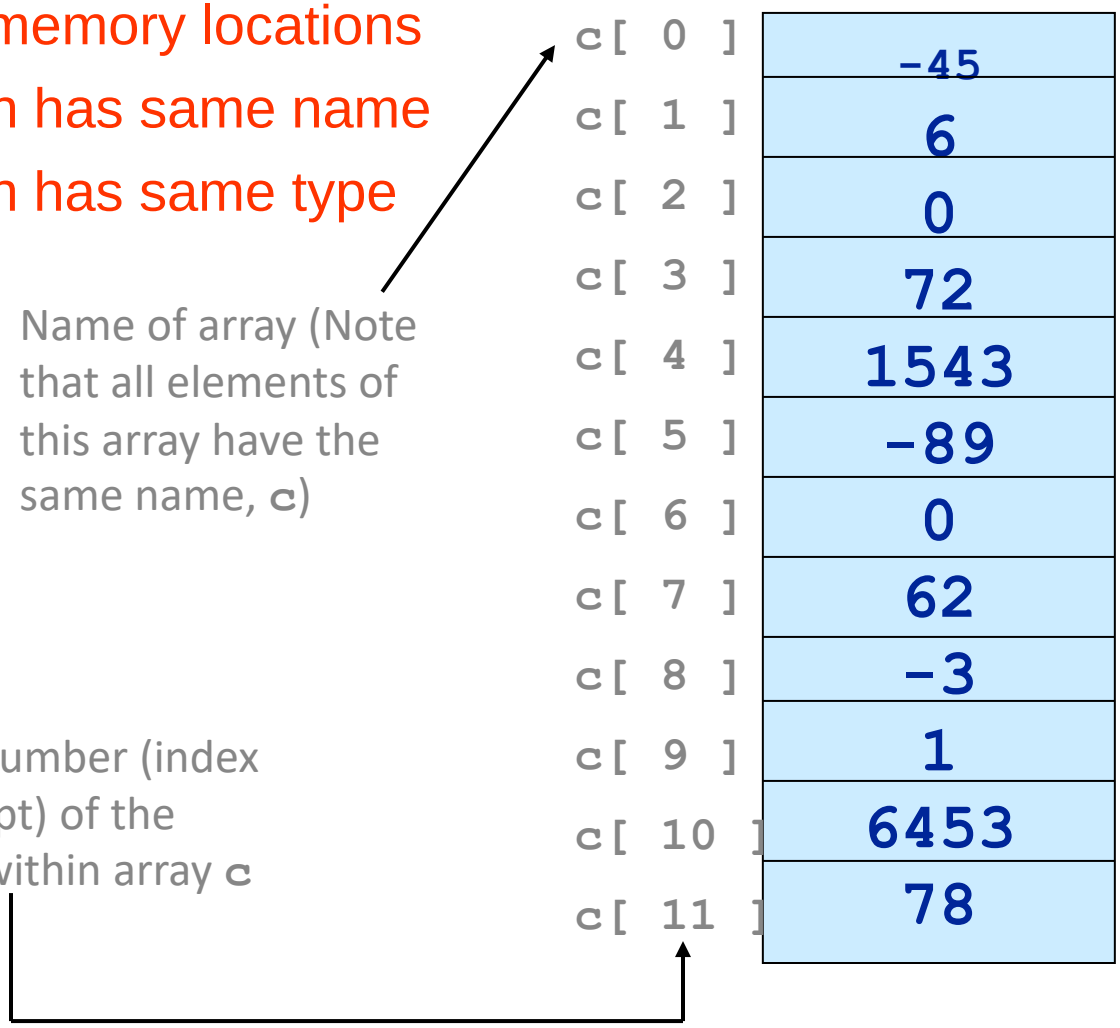
ArrayList

Outline

- Arrays
- Declaring and Allocating Arrays
- Allocating an Array and Initializing Its Elements
- Using an Initializer List to Initialize Elements of an Array
- Calculating the Value to Store in Each Array Element
- Summing the Elements of an Array
- Using Histograms to Display Array Data Graphically
- Using the Elements of an Array as Counters
- Using Arrays to Analyze Survey Results
- Passing Arrays to Methods
- The ArrayList Class
- The ArrayList Class

Arrays

- Array
  - –Group of contiguous memory locations
  - –Each memory location has same name
  - –Each memory location has same type



- **A 12-element array.**

Arrays

- Subscript
  - Also called an index
  - Position number in square brackets
  - Must be integer or integer expression

```
a = 5;
b = 6;
c[ a + b ] += 2;
```

    - Adds **2** to **c[11]**
    - Subscripted array name is an lvalue

- Examine array **c**
  - **c** is the array *name*
  - **c.length** accesses array **c**'s *length*
  - **c** has 12 *elements* (**c[0]**, **c[1]**, … **c[11]**)
    - The *value* of **c[0]** is **–45**
    - The brackets (**[]**) are in highest level of precedence in Java

Arrays

- Precedence and associavity of the operators

Operators	Associativity	Type
`() [] .`	left to right	highest
`++ --`	right to left	unary postfix
`++ -- + - ! (type)`	right to left	unary
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Declaring and Allocating Arrays

- Declaring and Allocating arrays
  - Arrays are objects that occupy memory
  - Allocated dynamically with operator **new**

```
int c[] = new int[ 12 ];
```

    - Equivalent to

```
int c[];          // declare array
c = new int[ 12 ]; // allocate array
```

  - We can allocate arrays of objects too

```
String b[] = new String[ 100 ];
```

Allocating an Array and Initializing Its Elements

```
1  // Fig. 7.3: InitArray.java
2  // Creating an array.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java
10    public static void main( String args[] )
11    {
12       int array[];              // declare r
13
14       array = new int[ 10 ]; // dynamical
15
16       String output = "Subscript\tValue\n";
17
18       // append each array element's value to String output
19       for ( int counter = 0; counter < array.length; counter++ )
20          output += counter + "\t" + array[ counter ] + "\n";
21
22       JTextArea outputArea = new JTextArea();
23       outputArea.setText( output );
24
25       JOptionPane.showMessageDialog( null, outputArea,
26          "Initializing an Array of int Values",
27          JOptionPane.INFORMATION_MESSAGE );
28
29       System.exit( 0 );
30    }
31 }
```

Declare **array** as an array of **int**s

Allocate **10 int**s for **array**; each **int** is initialized to **0** by default

array.length returns length of **array**

array[counter] returns **int** associated with index in **array**

InitArray.java

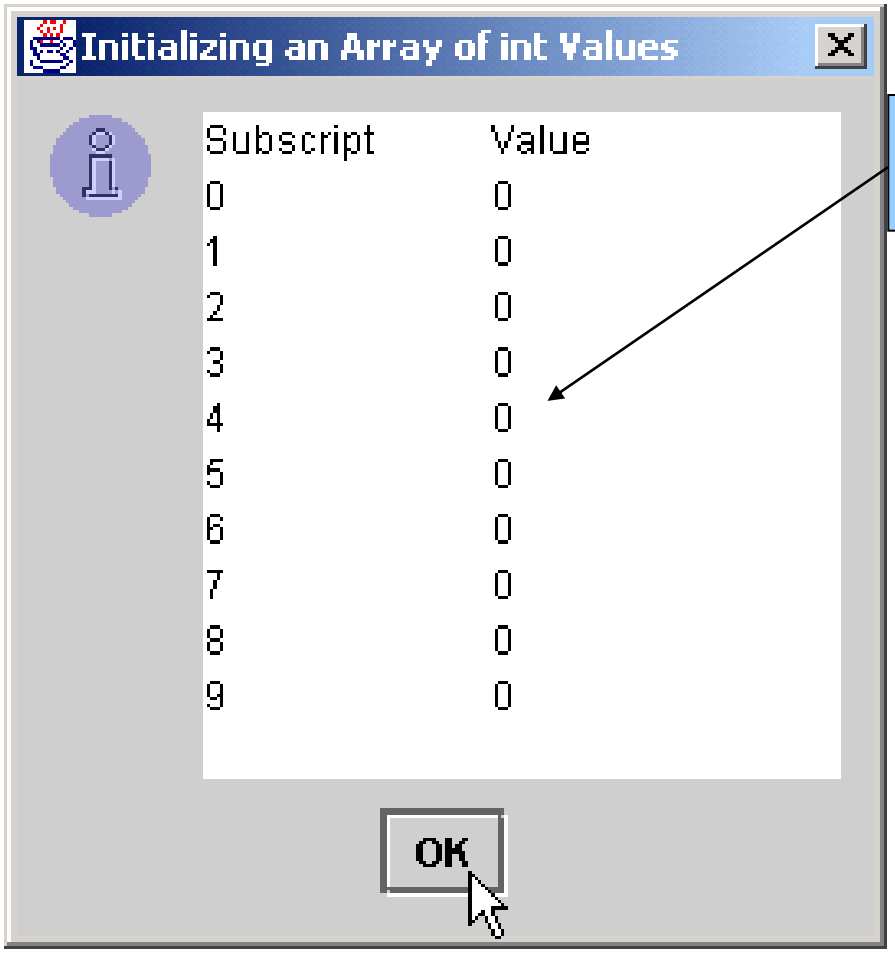
Line 12
Declare **array** s an array of ts

Line 14
Allocate **10 int**s for **array**; each **int** is initialized to **0** by default

Line 19
array.length returns length of **array**

Line 20
array[counter urns **int** ociated index in **array**

Allocating an Array and Initializing Its Elements



Each **int** is initialized to **0** by default

Using an Initializer List to Initialize Elements of an Array

- Initialize array elements
  - Use *initializer list*
    - Items enclosed in braces (**{}**)
    - Items in list separated by commas

```
int n[] = { 10, 20, 30, 40, 50 };
```

      - Creates a five-element array
      - Subscripts of **0**, **1**, **2**, **3**, **4**
  - Do not need operator **new**

Using an Initializer List to Initialize Elements of an Array

```
1  // Fig. 7.4: InitArray.java
2  // Initializing an array with a declaration.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       // initializer list specifies number
13       // value for each element
14       int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
15
16       String output = "Subscript\tValue\n";
17
18       // append each array element's value to String output
19       for ( int counter = 0; counter < array.length; counter++ )
20          output += counter + "\t" + array[ counter ] + "\n";
21
22       JTextArea outputArea = new JTextArea();
23       outputArea.setText( output );
24
25       JOptionPane.showMessageDialog( null, outputArea,
26          "Initializing an Array with a Declaration",
27          JOptionPane.INFORMATION_MESSAGE );
28
29       System.exit( 0 );
30    }
31 }
```

Declare **array** as an array of **int**s

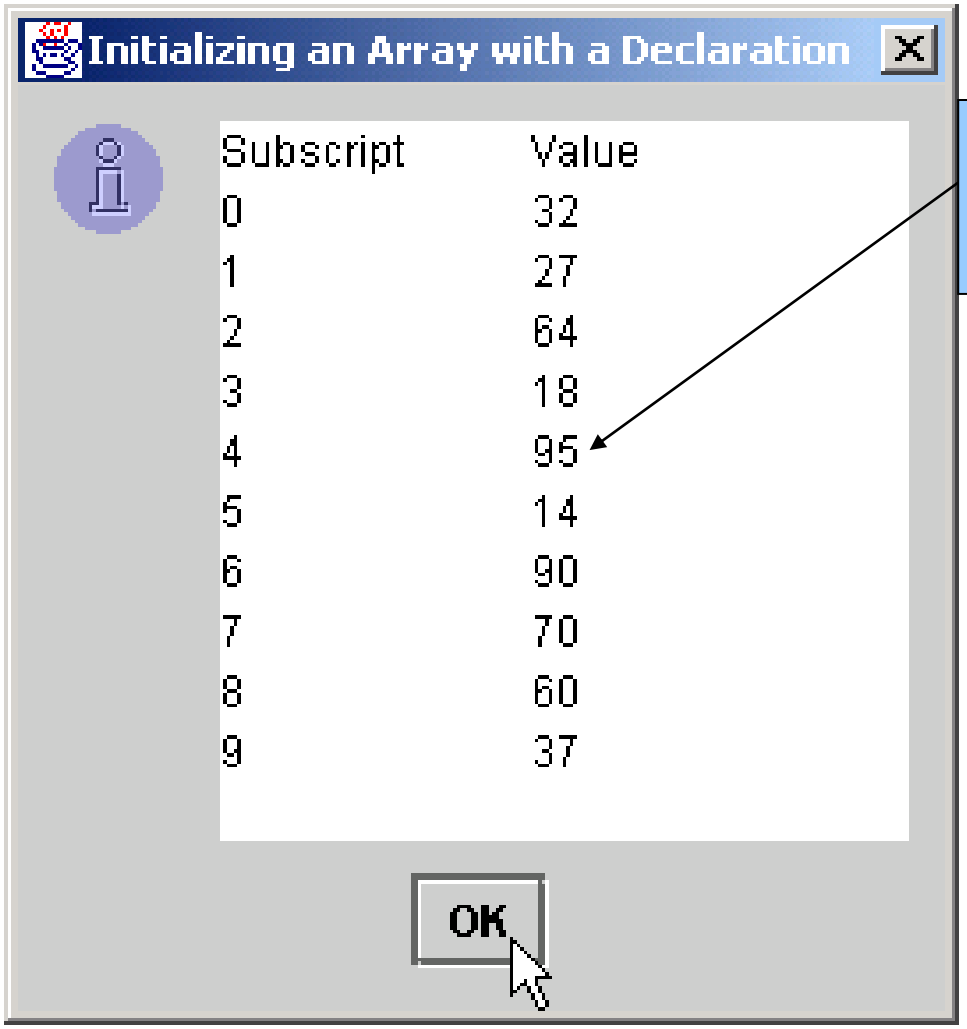
Compiler uses initializer list to allocate array

InitArray.java

Line 14
Declare **array** as an array of **int**s

Line 14
Compiler uses initializer list to allocate array

Using an Initializer List to Initialize Elements of an Array



Each **array** element corresponds to element in initializer list

Calculating the Value to Store in Each Array Element

- Calculate value stored in each array element
  - –Initialize elements of 10-element array to even integers

```
1  // Fig. 7.5: InitArray.java
2  // Initialize array with the even integers from 2 to 20.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       final int ARRAY_SIZE = 10;
13       int array[];                    // reference to int array
14
15       array = new int[ ARRAY_SIZE ];  // allocate array
16
17       // calculate value for each array element
18       for ( int counter = 0; counter < array.length; counter++ )
19          array[ counter ] = 2 + 2 * counter;
20
21       String output = "Subscript\tValue\n";
22
23       for ( int counter = 0; counter < array.length; counter++ )
24          output += counter + "\t" + array[ counter ] + "\n";
25
26       JTextArea outputArea = new JTextArea();
27       outputArea.setText( output );
28
29       JOptionPane.showMessageDialog( null, outputArea,
30          "Initializing to Even Numbers from 2 to 20",
31          JOptionPane.INFORMATION_MESSAGE );
32
33       System.exit( 0 );
34    }
35 }
```

Declare **array** as an array of **int**s

Allocate **10 int**s for **array**

Use **array** subscript to assign array value

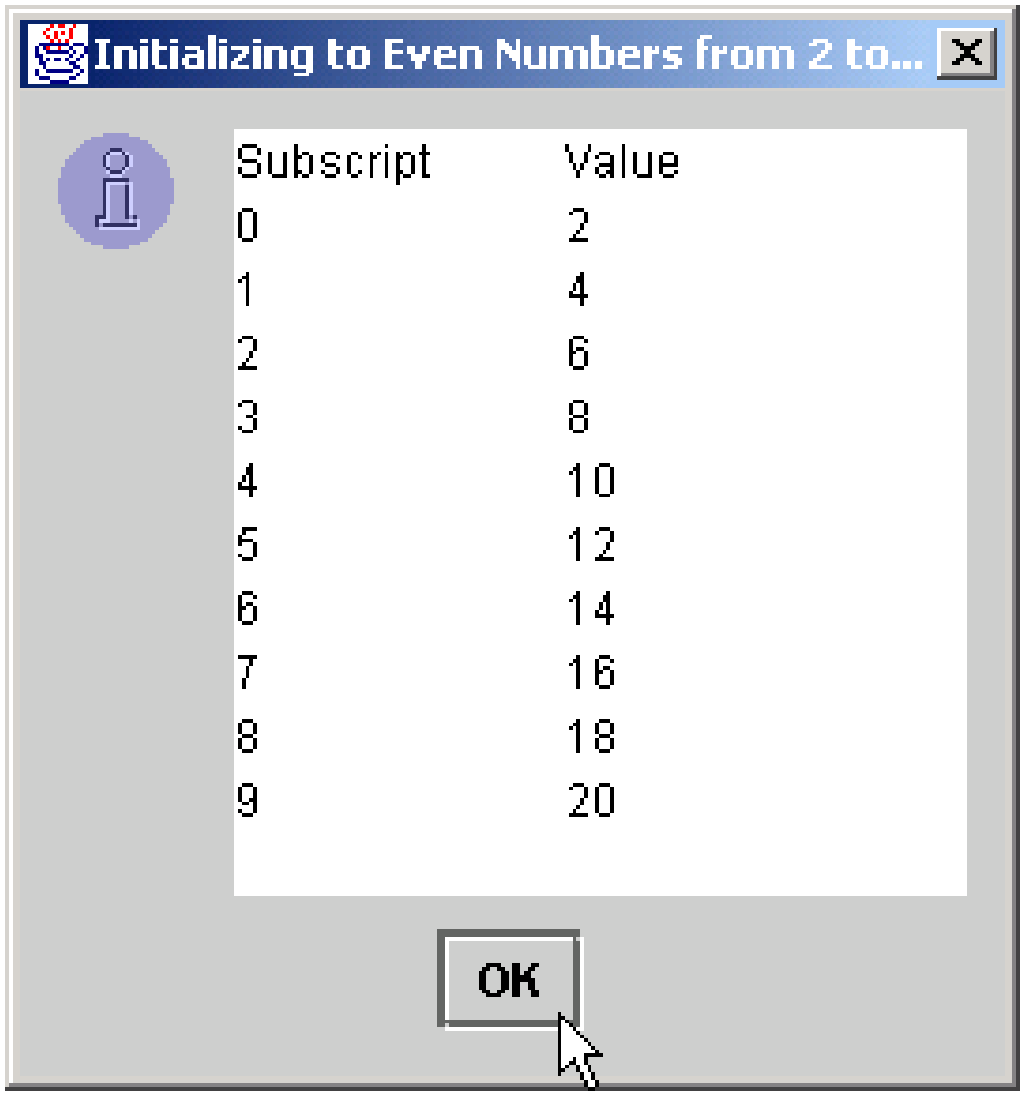
InitArray.java

Line 13
Declare **array** as an array of **int**s

Line 15
Allocate **10 int**s for **array**

Line 19
Use **array** subscript to assign array value

Calculating the Value to Store in Each Array Element



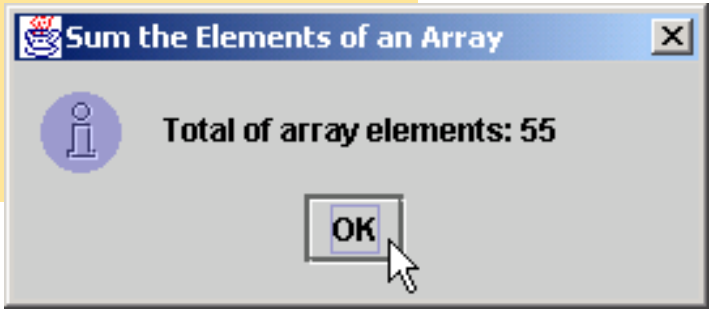
Summing the Elements of an Array

- Array elements
  - Can represent a series of values
    - We can sum these values

```
1  // Fig. 7.6: SumArray.java
2  // Total the values of the elements of an array.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class SumArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13       int total = 0;
14
15       // add each element's value to total
16       for ( int counter = 0; counter < array.length; counter++ )
17          total += array[ counter ];
18
19       JOptionPane.showMessageDialog( null,
20          "Total of array elements: " + total,
21          "Sum the Elements of an Array",
22          JOptionPane.INFORMATION_MESSAGE );
23
24       System.exit( 0 );
25    }
26 }
```

Declare **array** with initializer list

Sum all **array** values



SumArray.java

Line 12
Declare **array** with initializer list

Line 17
Sum all **array** values

Using Histograms to Display Array Data Graphically

- Present array values graphically
  - –Histogram
    - Plot each numeric value as bar of asterisks (*)

```
1  // Fig. 7.7: Histogram.java
2  // Histogram printing program.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class Histogram {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int array[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
13
14       String output = "Element\tValue\tHistogram";
15
16       // for each array element, output a bar in histogram
17       for ( int counter = 0; counter < array.length; counter++ ) {
18          output +=
19             "\n" + counter + "\t" + array[ counter ] + "\t";
20
21          // print bar of asterisks
22          for ( int stars = 0; stars < array[ counter ]; stars++ )
23             output += "*";
24       }
25
26       JTextArea outputArea = new JTextArea();
27       outputArea.setText( output );
28
29       JOptionPane.showMessageDialog( null, outputArea,
30          "Histogram Printing Program",
31          JOptionPane.INFORMATION_MESSAGE );
32
33       System.exit( 0 );
34    }
35 }
```

Declare **array** with initializer list

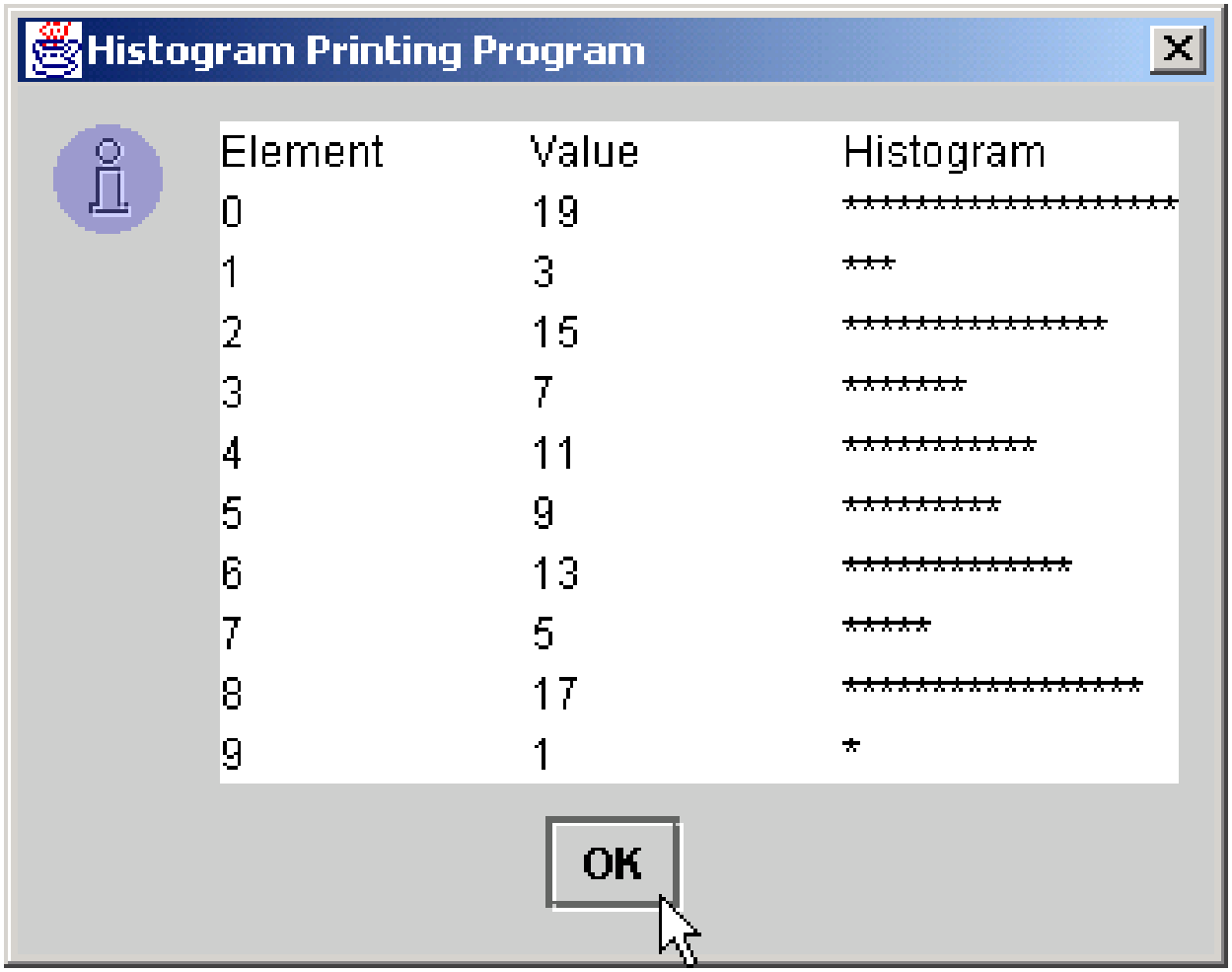
For each **array** element, print associated number of asterisks

Histogram.java

Line 12
Declare **array** with initializer list

Line 23
For each **array** element, print associated number of asterisks

Using Histograms to Display Array Data Graphically



Using the Elements of an Array as Counters

- Use series of counters to summarize data
  - –Array can store these counters

```
1  // Fig. 7.8: RollDie.java
2  // Roll a six-sided die 6000 times
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class RollDie {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int face, frequency[] = new int[ 7 ];
13
14       // roll die 6000 times
15       for ( int roll = 1; roll <= 6000; roll++ ) {
16          face = 1 + ( int ) ( Math.random() * 6 );
17
18          // use face value as subscript for frequency array
19          ++frequency[ face ];
20       }
21
22       String output = "Face\tFrequency";
23
24       // append frequencies to String output
25       for ( face = 1; face < frequency.length; face++ )
26          output += "\n" + face + "\t" + frequency[ face ];
27
28       JTextArea outputArea = new JTextArea();
29       outputArea.setText( output );
30
31       JOptionPane.showMessageDialog( null, outputArea,
32          "Rolling a Die 6000 Times",
33          JOptionPane.INFORMATION_MESSAGE );
34
```

Declare **frequency** as array of **7 int**s

Generate **6000** random integers in range **1–6**

Increment **frequency** values at index associated with random number

RollDie.java

Line 12
Declare **frequency** as array of **7 int**s

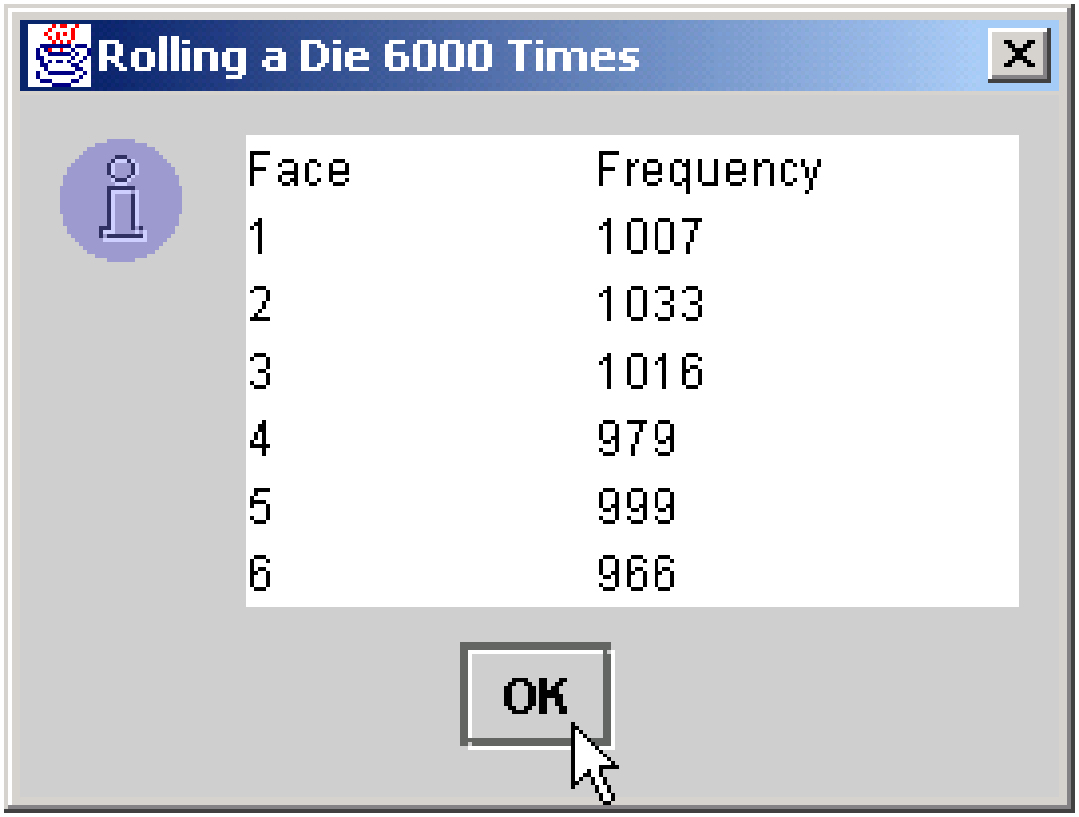
Lines 15-16
Generate **6000** random integers in range **1-6**

ent
ncy values at index associated with random number

Using the Elements of an Array as Counters

```
35         System.exit( 0 );
36      }
37   }
```

RollDie.java



Using Arrays to Analyze Survey Results

- Problem statement
  - –40 students rate the quality of food
    - 1-10 Rating scale: 1 mean awful, 10 means excellent
  - –Place 40 responses in array of integers
  - –Summarize results

```
1  // Fig. 7.9: StudentPoll.java
2  // Student poll program
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class StudentPoll {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
13                           1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
14                           6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
15                           5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
16       int frequency[] = new int[ 11 ];
17
18       // for each answer, select value of an element of
19       // responses array and use that value as subscript in
20       // frequency array to determine element to increment
21       for ( int answer = 0; answer < responses.length; answer++ )
22          ++frequency[ responses[ answer ] ];
23
24       String output = "Rating\tFrequency\n";
25
26       // append frequencies to String output
27       for ( int rating = 1; rating < frequency.length; rating++ )
28          output += rating + "\t" + frequency[ rating ] + "\n";
29
30       JTextArea outputArea = new JTextArea();
31       outputArea.setText( output );
32
33       JOptionPane.showMessageDialog( null, outputArea,
34          "Student Poll Program",
35          JOptionPane.INFORMATION_MESSAGE );
```

Declare **responses** as array to store **40** responses

Declare **frequency** as array of **11** **int** and ignore the first element

For each response, increment **frequency** values at index associated with that response

StudentPoll.java

Lines 12-15
Declare **responses** as array to store **40** responses

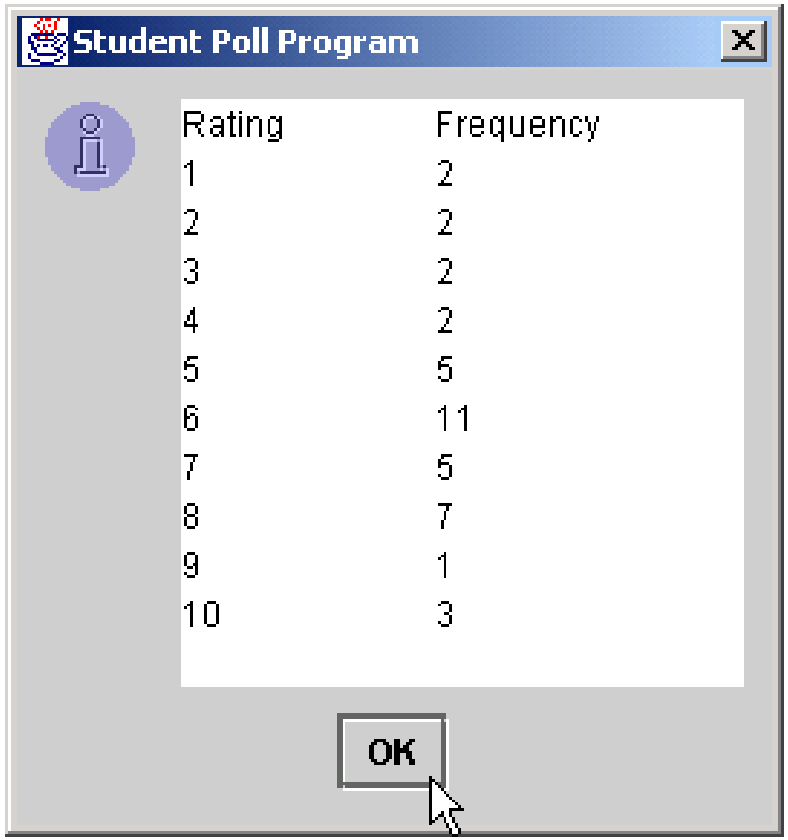
Line 16
Declare **frequency** as array of **11 int** and ignore the first element

Lines 21-22
For each response, increment **frequency** values at index associated with that response

Using Arrays to Analyze Survey Results

```
36
37          System.exit( 0 );
38       }
39    }
```

StudentPoll.java



Using Arrays to Analyze Survey Results

- Some additional points
  - When looping through an array
    - Subscript should never go below 0
    - Subscript should be less than total number of array elements
  - When invalid array reference occurs
    - Java generates ArrayIndexOutOfBoundsException
      - This is relevant to exception handling

Passing Arrays to Methods

- To pass array argument to a method
  - –Specify array name without brackets
    - Array **hourlyTemperatures** is declared as

```
int hourlyTemperatures = new int[ 24 ];
```

  - –The method call

```
modifyArray( hourlyTemperatures );
```

  - –Passes array **hourlyTemperatures** to method **modifyArray**

```
1  // Fig. 7.10: PassArray.java
2  // Passing arrays and individual array elements to methods
3
4  // Java core packages
5  import java.awt.Container;
6
7  // Java extension packages
8  import javax.swing.*;
9
10 public class PassArray extends JApplet {
11
12    // initialize applet
13    public void init()
14    {
15       JTextArea outputArea = new JTextArea();
16       Container container = getContentPane();
17       container.add( outputArea );
18
19       int array[] = { 1, 2, 3, 4, 5 };
20
21       String output =
22          "Effects of passing entire array by reference:\n" +
23          "The values of the original array are:\n";
24
25       // append original array elements to String output
26       for ( int counter = 0; counter < array.length; counter++ )
27          output += "   " + array[ counter ];
28
29       modifyArray( array );  // array passed by reference
30
31       output += "\n\nThe values of the modified array are:\n";
32
33       // append modified array elements to String output
34       for ( int counter = 0; counter < array.length; counter++ )
35          output += "   " + array[ counter ];
```

Declare **5-int array** with initializer list

Pass **array** by reference to method **modifyArray**

PassArray.java

Line 19
Declare **5-int array** with initializer list

Line 29
Pass **array** by reference to method **modifyArray**

Passing Arrays to Methods

```
36
37       output += "\n\nEffects of passing array " +
38                 "element by value:\n" +
39                 "a[3] before modifyElement: " + array[ 3 ];
40
41       // attempt to modify array[ 3 ]
42       modifyElement( array[ 3 ] );
43
44       output += "\na[3] after modifyElement: " + array[ 3 ];
45       outputArea.setText( output );
46
47    }  // end method init
48
49    // multiply each element of an array by 2
50    public void modifyArray( int array2[] )
51    {
52       for ( int counter = 0; counter < array2.length; counter++ )
53          array2[ counter ] *= 2;
54    }
55
56    // multiply argument by 2
57    public void modifyElement( int element )
58    {
59       element *= 2;
60    }
61
62 }  // end class PassArray
```

Pass **array[3]** by value to method **modifyElement**

Method **modifyArray** manipulates the array directly

Method **modifyElement** manipulates a primitive's copy

The original primitive is left unmodified

PassArray.java

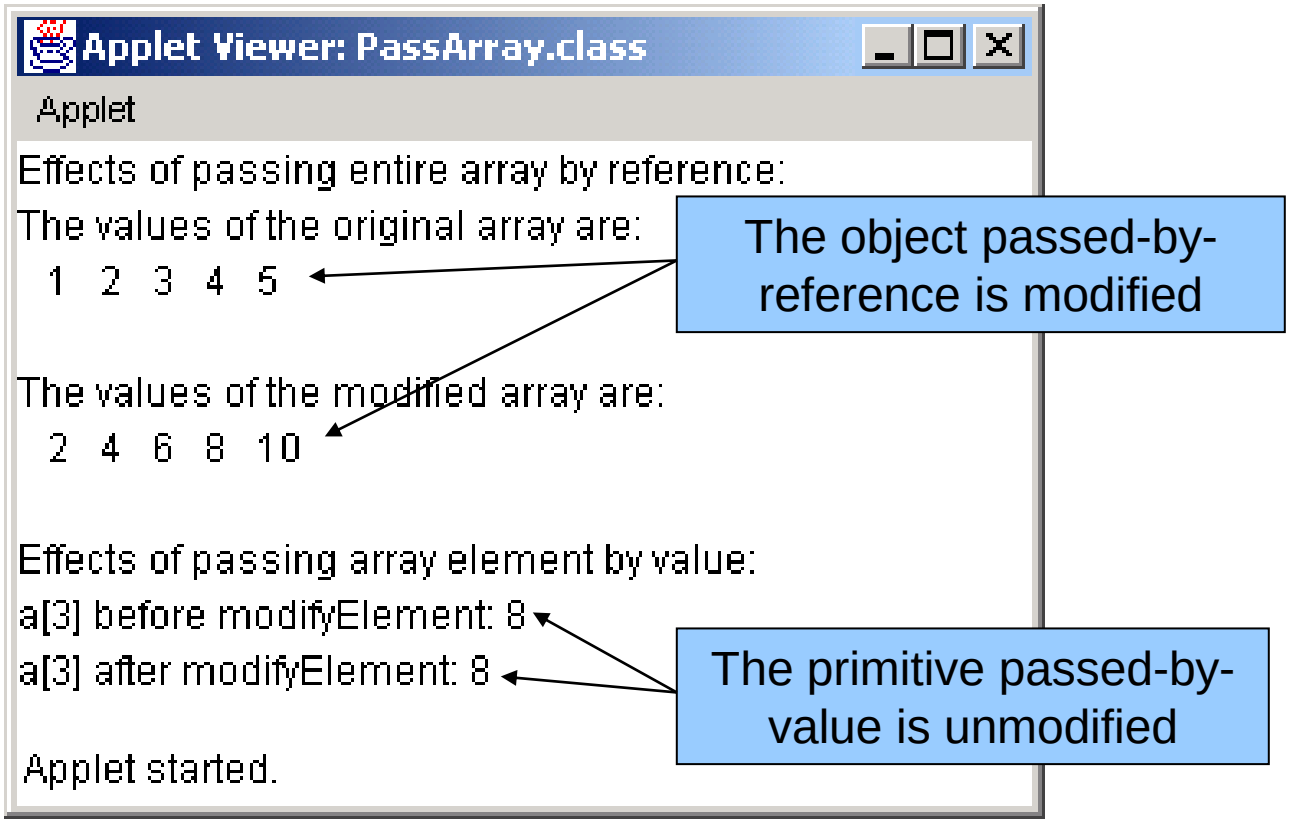
Line 42
Pass **array[3]** by value to method **modifyElement**

Lines 50-54
Method **modifyArray** manipulates the array directly

Lines 57-60
Method **modifyElement** manipulates a primitive's copy

Lines 59
The original primitive is left unmodified

Passing Arrays to Methods



PassArray.java

The object passed-by-reference is modified

The primitive passed-by-value is unmodified

The ArrayList Class

- Arrays are conceptually important as a data structure,
  - they are not used as much in Java as they are in most other languages.
- The reason is that the java.util package includes a class called ArrayList that provides the standard array behavior along with other useful operations.
- ArrayList is a Java class rather than a special form in the language.
  - So, all operations on ArrayLists are indicated using method calls.
  - For example,
    - the most obvious differences include:
      - You create a new ArrayList by calling the ArrayList constructor.
      - You get the number of elements by calling the size method rather than by selecting a length field.
      - You use the get and set methods to select individual elements.
      - the most important methods in the
- The next slides summarize ArrayList class.
  - The notation <T> indicates the base type.

Methods in the ArrayList Class

- boolean add(<T> element)
  - Adds a new element to the end of the ArrayList; the return value is always true.
- void add(int index, <T> element)
  - Inserts a new element into the ArrayList before the position specified by index.
- <T> remove(int index)
  - Removes the element at the specified position and returns that value.
- boolean remove(<T> element)
  - Removes the first instance of element, if it appears; returns true if a match is found.
- void clear()
  - Removes all elements from the ArrayList.
- int size()
  - Returns the number of elements in the ArrayList.
- <T> get(int index)
  - Returns the object at the specified index.
- <T> set(int index, <T> value)
  - Sets the element at the specified index to the new value and returns the old value.
- int indexOf(<T> value)
  - Returns the index of the first occurrence of the specified value, or -1 if it does not appear.
- boolean contains(<T> value)
  - Returns true if the ArrayList contains the specified value.
- boolean isEmpty()
  - Returns true if the ArrayList contains no elements.

The ArrayList Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new
ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
    array1[i] = i;
}

for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
}
```

The ArrayList Class

```
int[] array1 = new int[5];

for(int i = 0; i < 5; i++) {
    array1[i] = i;
}
```

- i = 0 ➔

0	0	0	0	0

- i = 1 ➔

0	1	0	0	0

- i = 2 ➔

0	1	2	0	0

- i = 3 ➔

0	1	2	3	0

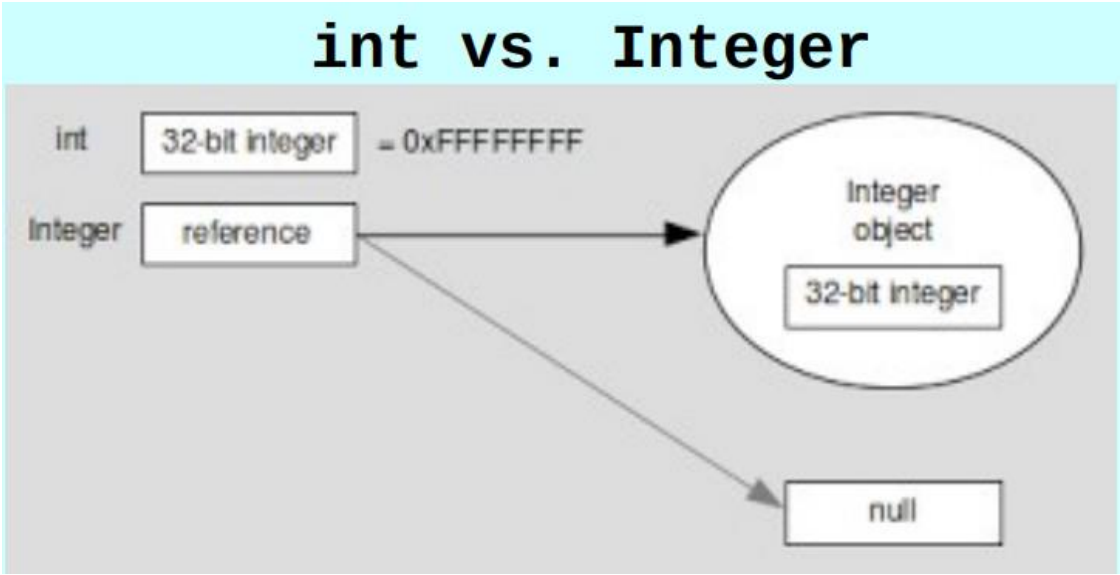
- i = 4 ➔

0	1	2	3	4

The ArrayList Class

```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
     // add a new element to the end of the list
     arrayList1.add(i);
 }
```

- Java Collection classes do not accept primitive types.
  - Instead they use wrapper classes.



The ArrayList Class

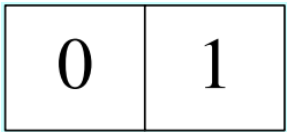
```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
     // add a new element to the end of the list
     arrayList1.add(i);
 }
```

The ArrayList Class

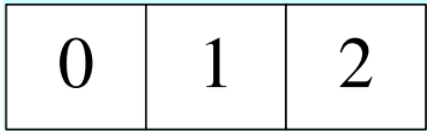
```
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();

for(int i = 0; i < 5; i++) {
     // add a new element to the end of the list
     arrayList1.add(i);
 }
```

- i = 0 ➔

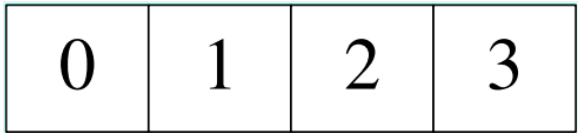
- i = 1 ➔



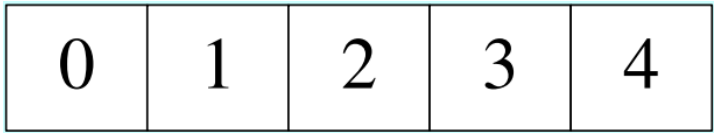
- i = 2 ➔



- i = 3 ➔



- i = 4 ➔



The ArrayList Class

```
int[] array1 = new int[5];
ArrayList<Integer> arrayList1 = new ArrayList<Integer>();
for(int i = 0; i < 5; i++) {
    array1[i] = i;
    println(array1.length);
}
for(int i = 0; i < 5; i++) {
    // add a new element to the end of the list
    arrayList1.add(i);
    println(arrayList1.size());
}
```

```
5
5
5
5
5
---------
1
2
3
4
5
```

add *method extends the size by one by adding a new element to the end of the list.*

The ArrayList Class

```
array[] = 42;
println(array[2]);

arrayList(2, 42);
println(arrayList.get(2));
```

The ArrayList Class

```
array[] = 42;
println(array[2]);

arrayList(2, 42);
println(arrayList.get(2));
```

The ArrayList Class

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.add(44);
arrayList.set(2, 45);
println(arrayList.get(2));
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
println(arrayList.get(2));
arrayList.add(44);
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.set(2, 45);
arrayList.add(44);
println(arrayList.get(2));
```

?

The ArrayList Class

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.add(44);
arrayList.set(2, 45);
println(arrayList.get(2)); // prints 45
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
println(arrayList.get(2)); // ERROR!!!
arrayList.add(44);
```

?

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(42);
arrayList.add(43);
arrayList.set(2, 45); // ERROR!!!
arrayList.add(44);
println(arrayList.get(2));
```

?

The ArrayList Class

```
ArrayList<Integer> intList = new ArrayList<Integer>();
ArrayList<Double> doubleList = new ArrayList<Double>();
ArrayList<String> stringList = new ArrayList<String>();

intList.add(42);
doubleList.add(42.0);
stringList.add("42");
```

- **Generic Types in Java 5.0:**
  - The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language.
  - In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array.
  - Class definitions that include a type parameter are called **generic types**.

Generic Types in Java 5.0

- The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language.
- In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array.
- Class definitions that include a type parameter are called **generic types**.
- When you declare or create an **ArrayList**, it is a good idea to specify the element type in angle brackets.
  - For example, to declare and initialize an **ArrayList** called **names** that contains elements of type **String**, you would write

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

Generic Types in Java 5.0

- The advantage of specifying the element type is that Java now knows what type of value the **ArrayList** contains.
- When you call **set**, Java can ensure that the value matches the element type.
- When you call **get**, Java knows what type of value to expect, eliminating the need for a type cast.

Any Questions?