Chapter 7 Single-Dimensional Arrays

Opening Problem

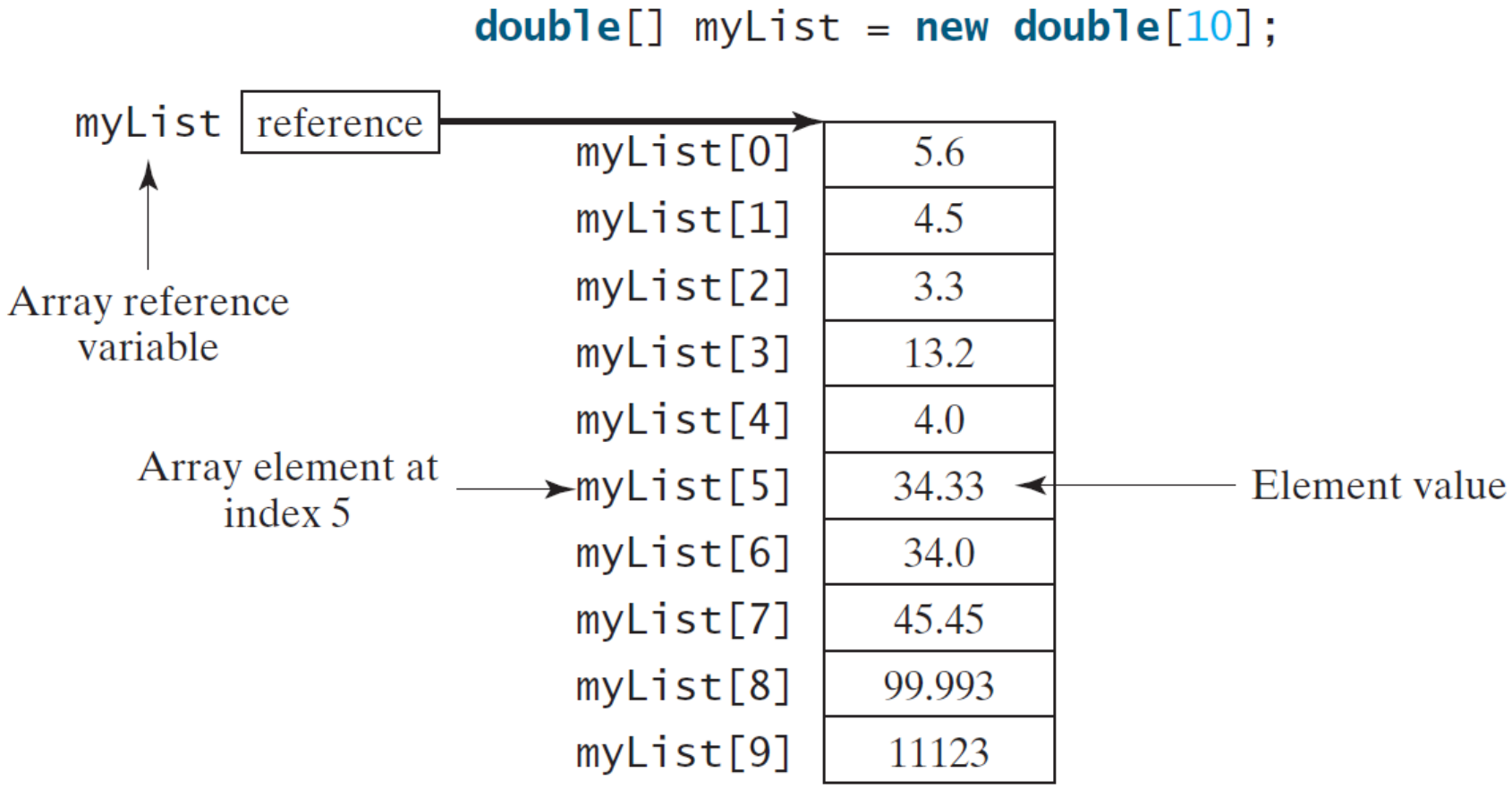
Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Objectives

- To describe why arrays are necessary in programming (§7.1).
- To declare array reference variables and create arrays (§§7.2.1–7.2.2).
- To obtain array size using **arrayRefVar.length** and know default values in an array (§7.2.3).
- To access array elements using indexes (§7.2.4).
- To declare, create, and initialize an array using an array initializer (§7.2.5).
- To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements, random shuffling, and shifting elements) (§7.2.6).
- To simplify programming using the foreach loops (§7.2.7).
- To apply arrays in application development (**AnalyzeNumbers**, **DeckOfCards**) (§§7.3–7.4).
- To copy contents from one array to another (§7.5).
- To develop and invoke methods with array arguments and return values (§§7.6–7.8).
- To define a method with a variable-length argument list (§7.9).
- To search elements using the linear (§7.10.1) or binary (§7.10.2) search algorithm.
- To sort an array using the selection sort approach (§7.11).
- To use the methods in the **java.util.Arrays** class (§7.12).
- To pass arguments to the main method from the command line (§7.13).

Introducing Arrays

Array is a data structure that represents a collection of the same types of data.



Declaring Array Variables

- `datatype[] arrayRefVar;`

  Example:

  ```
  double[] myList;
  ```

- `datatype arrayRefVar[];` // This style is allowed, but not preferred

  Example:

  ```
  double myList[];
  ```

Creating Arrays

```
arrayRefVar = new datatype[arraySize];
```

Example:

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

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.

Declaring and Creating in One Step

- `datatype[] arrayRefVar = new datatype[arraySize];`

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

- `datatype arrayRefVar[] = new datatype[arraySize];`

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

The Length of an Array

Once an array is created, its size is fixed. It cannot be changed. You can find its size using

```
arrayRefVar.length
```

For example,

```
myList.length returns 10
```

Default Values

When an array is created, its elements are assigned the default value of

0 for the numeric primitive data types,

'\u0000' for char types, and

false for boolean types.

Indexed Variables

The array elements are accessed through the index. The array indices are *0-based*, i.e., it starts from 0 to arrayRefVar.length-1. In the example in Figure 6.1, myList holds ten double values and the indices are from 0 to 9.

Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayRefVar[index];
```

Using Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in myList[0] and myList[1] to myList[2].

```
myList[2] = myList[0] + myList[1];
```

Array Initializers

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

Declaring, creating, initializing Using the Shorthand Notation

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

CAUTION

Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

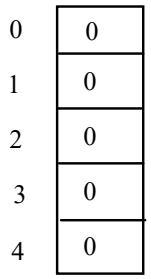
animation

Trace Program with Arrays

Declare array variable values, create an array, and assign its reference to values

```
public class Test {
  public static void main(Str[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



animation

Trace Program with Arrays

i becomes 1

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

animation

Trace Program with Arrays

i (=1) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

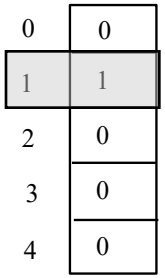
animation

Trace Program with Arrays

After this line is executed, value[1] is 1

```
public class Test {
  public static void main(Strin      gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration



animation

Trace Program with Arrays

After i++, i becomes 2

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

animation

Trace Program with Arrays

```
public class Test {
  public static void main(String[]
      args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] +
      values[4];
  }
}
```

i (= 2) is less than 5

After the first iteration

0	0
1	1
2	0
3	0
4	0

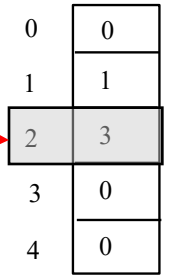
animation

Trace Program with Arrays

After this line is executed, values[2] is 3 (2 + 1)

```
public class Test {
  public static void main(Strin      gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration



animation

Trace Program with Arrays

After this, i becomes 3.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

animation

Trace Program with Arrays

i (=3) is still less than 5.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

animation

Trace Program with Arrays

After this line, values[3] becomes 6 (3 + 3)

```
public class Test {
  public static void main(String[] arg
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

animation

Trace Program with Arrays

After this, i becomes 4

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

animation

Trace Program with Arrays

i (=4) is still less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

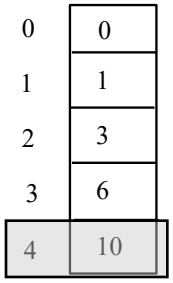
animation

Trace Program with Arrays

After this, values[4] becomes 10 (4 + 6)

```
public class Test {
  public static void main(Stri      args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration



animation

Trace Program with Arrays

After i++, i becomes 5

```
public class Test {
  public static void main(String[] ar
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

i (=5) < 5 is false. Exit the loop

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

After this line, values[0] is 11 (1 + 10)

```
public class Test {
  public static void m       tring[] args) {
    int[] values = ne       [5];
    for (int i = 1; i       ; i++) {
      values[i] = i       values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

0	11
1	1
2	3
3	6
4	10

Processing Arrays

See the examples in the text.

1. (Initializing arrays with input values)
2. (Printing arrays)
3. (Initializing arrays with random values)
4. (Summing all elements)
5. (Finding the largest element)
6. (Finding the smallest index of the largest element)
7. (*Random shuffling*)
8. (*Shifting elements*)

Initializing arrays with input values

```
java.util.Scanner input = new java.util.Scanner(System.in);
double[] myList = new double[10];
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

Printing arrays values/elements

```
for (int i = 0; i < myList.length; i++) {
  System.out.print(myList[i] + " ");
}
```

Initializing arrays with random values

```
double[] myList = new double[10];
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
for (int i = 0; i < myList.length; i++) {
  System.out.print(myList[i] + " ");
}
```

Summing all elements

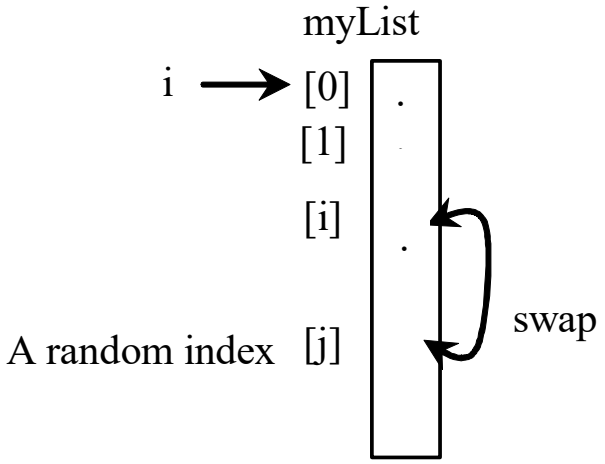
```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
System.out.print("Total = "+total);
```

Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max)
     max = myList[i];
}
System.out.print("Max = "+max);
```

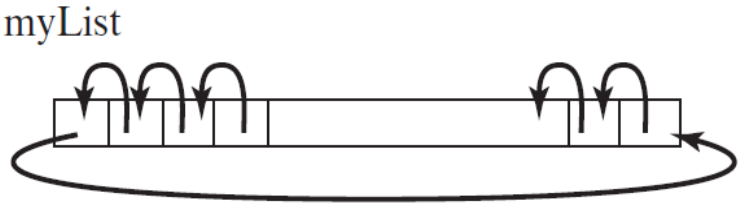
Random shuffling

```
for (int i = 0; i < myList.length - 1; i++) {
  // Generate an index j randomly
  int j = (int)(Math.random()
    * myList.length);

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



Shifting Elements

```
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
  myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```



Enhanced for Loop (for-each loop)

JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```
for (double value: myList)
  System.out.println(value);
```

In general, the syntax is

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Analyze Numbers

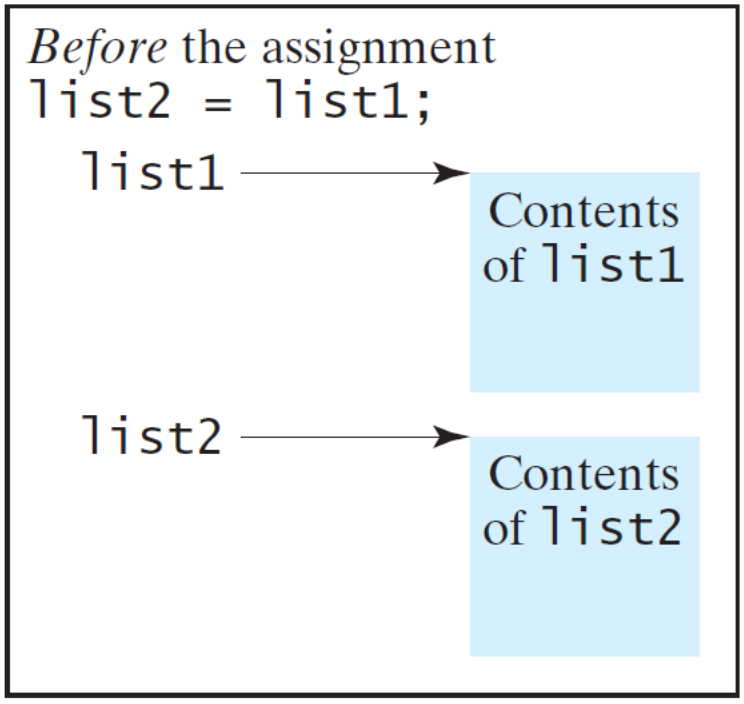
Read one hundred numbers, compute their average, and find out how many numbers are above the average.

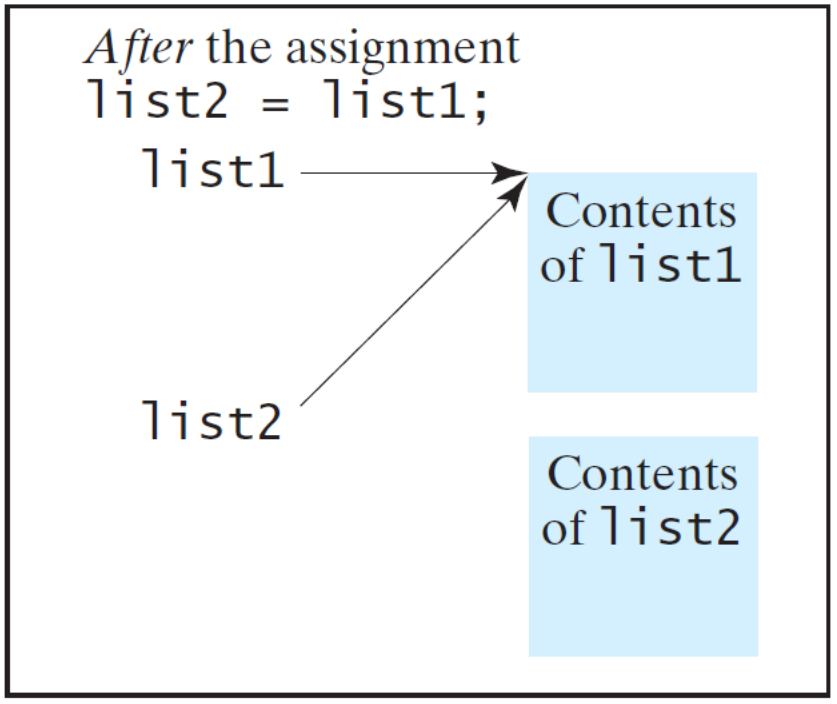
AnalyzeNumbers Run

Copying Arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;





Copying Arrays

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

The `arraycopy` Utility

```
arraycopy(sourceArray, src_pos,
  targetArray, tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0,
  targetArray, 0, sourceArray.length);
```

Passing Arrays to Methods

```
public class PassingArrayToMethod {
   public static void main(String[] args) {
      int[] list = {3, 1, 2, 6, 4, 2};
      printArray(list);
      System.out.println();
      //Invoke the method using anonymous array
      printArray(new int[]{13, 11, 12, 16, 14, 12});
      System.out.println();
   }
  public static void printArray(int[] myArray) {
      for (int i = 0; i < myAarray.length; i++)
         System.out.print(myAarray[i] + " ");
   }
}
```

Anonymous Array

The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

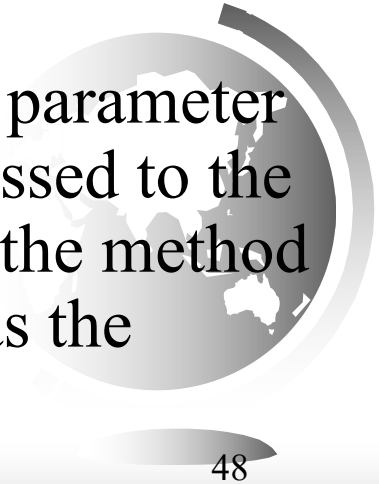
There is no explicit reference variable for the array. Such array is called an *anonymous array*.

Pass By Value

Java uses *pass by value* to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

✦ For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.

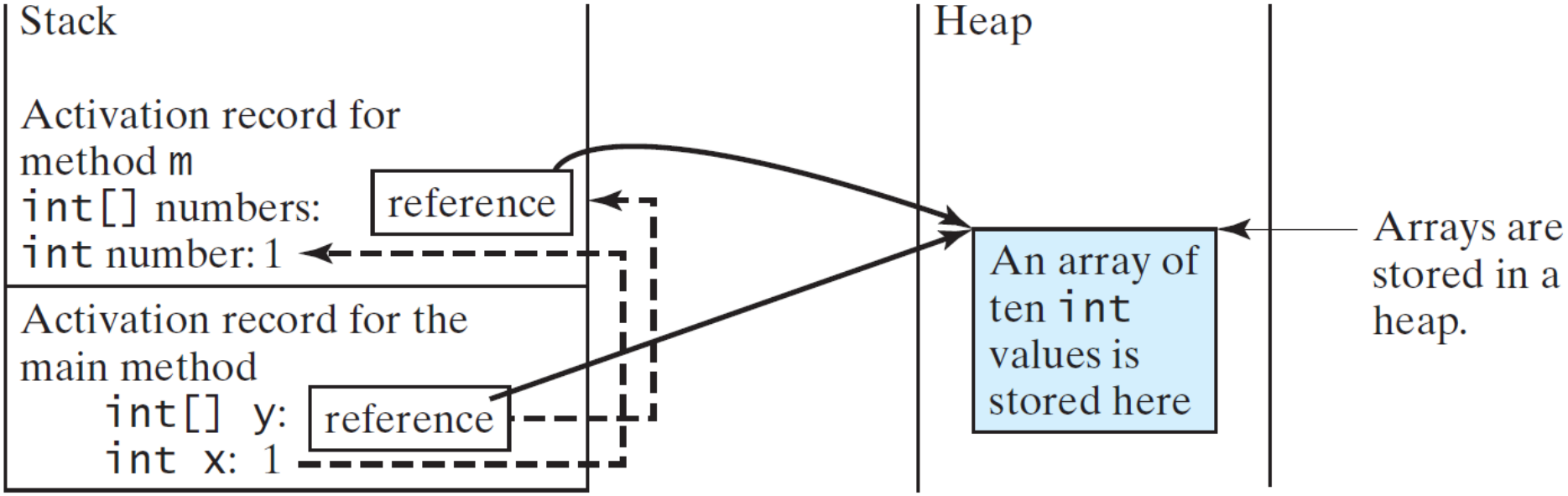
✦ For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.



Simple Example to pass by value and by reference

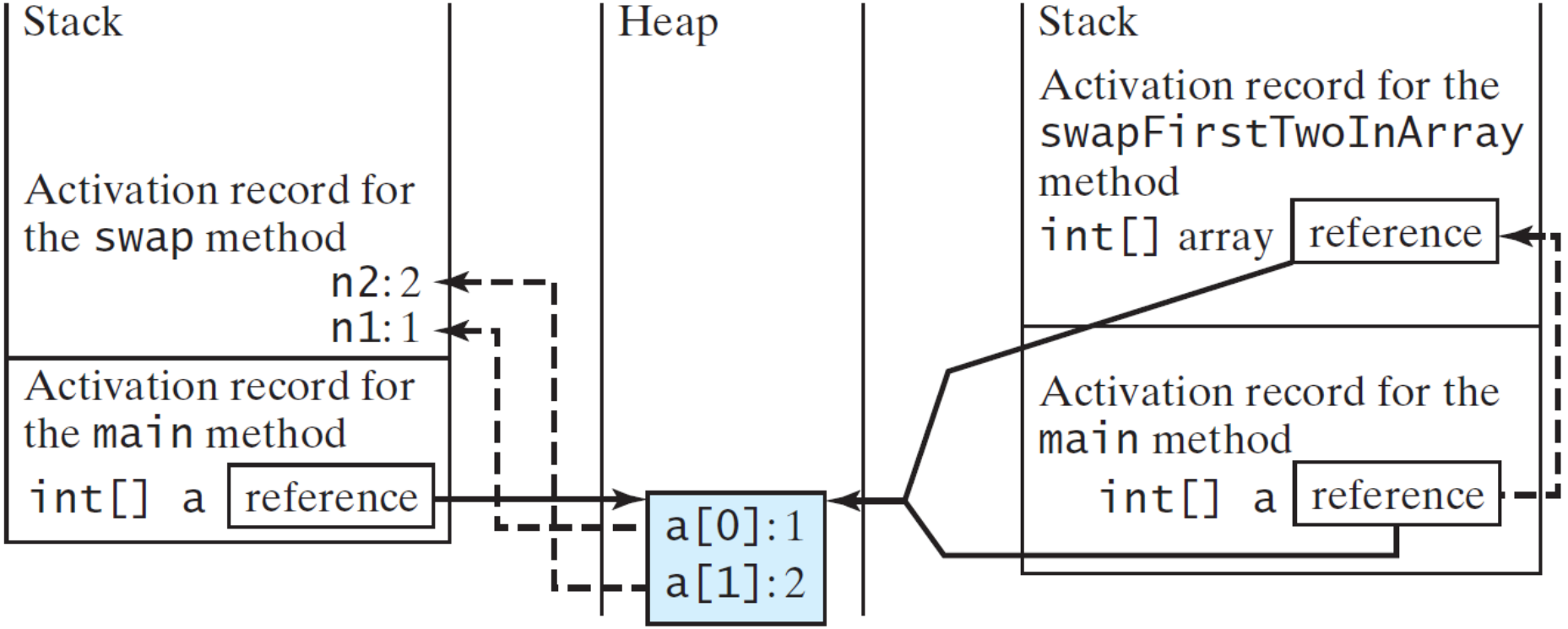
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

Call Stack



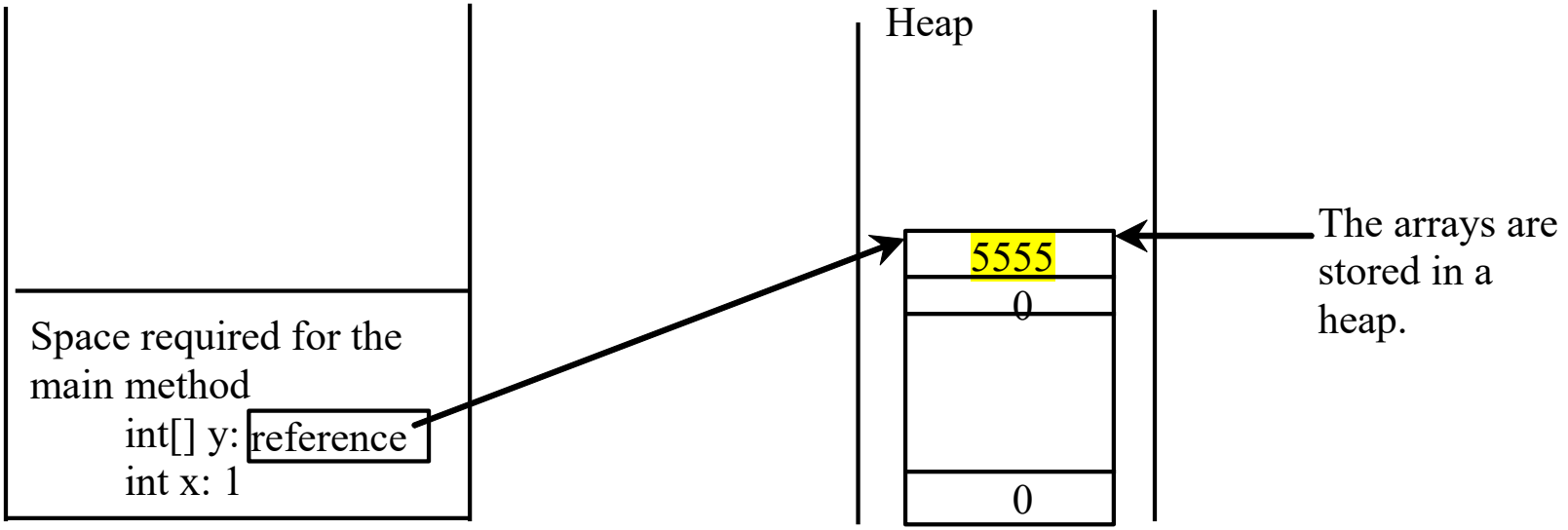
When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Call Stack



When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Heap



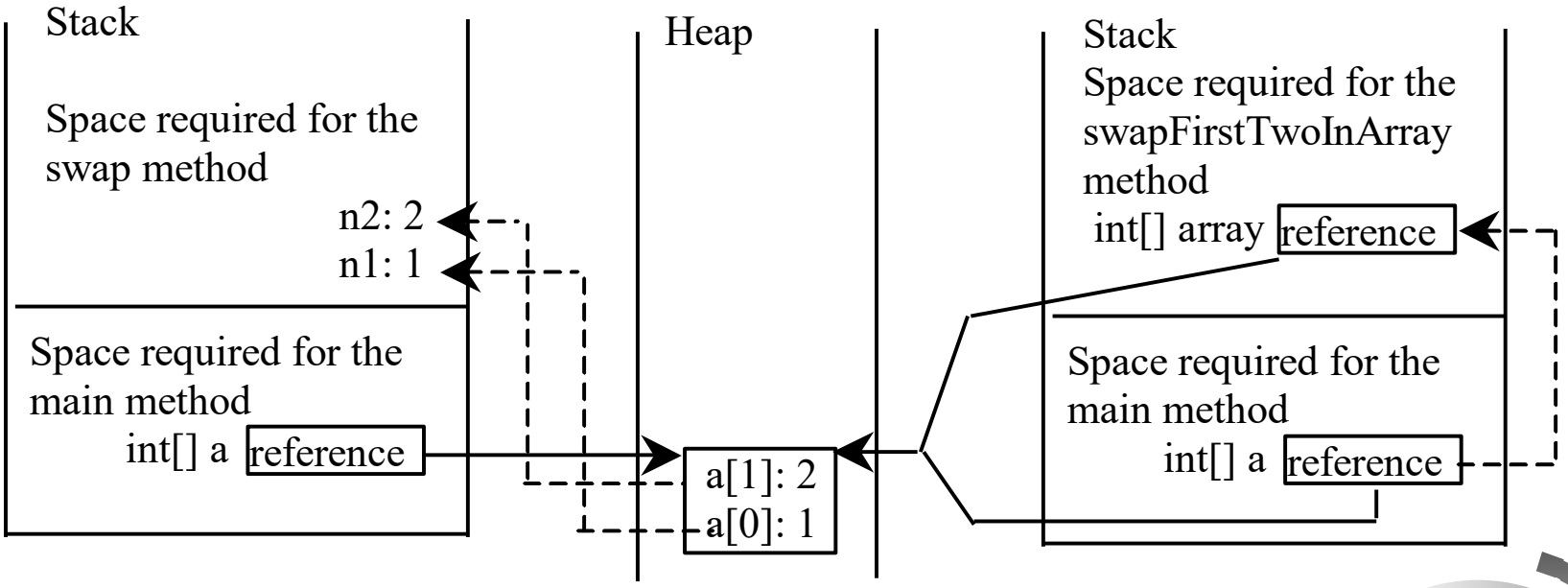
The JVM stores the array in an area of memory, called *heap*, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

Passing Arrays as Arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables.

TestPassArray Run

Example, cont.



Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

The arrays are stored in a heap.

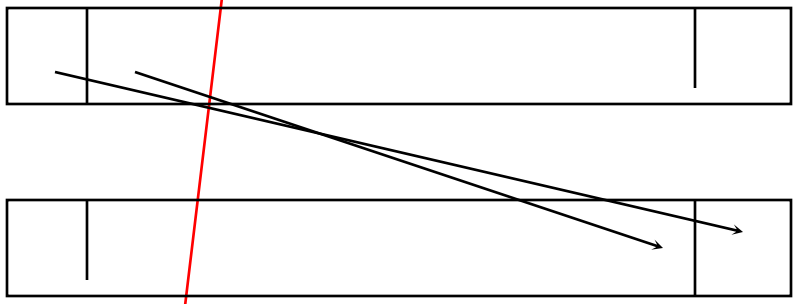
Invoke swapFirstTwoInArray(int[] array). The reference value in a is passed to the swapFirstTwoInArray method.

Returning an Array from a Method

```
public static void reverse(int[] list) {

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

//Write it inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
for (int i = 0, i < list1.length; i++) {
    System.out.print(list[i]+" ");

reverse(list1);

for (int i = 0, i < list1.length; i++) {
    System.out.print(list[i]+" ");
```

Returning an Array from a Method

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

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

  return result;
}
```



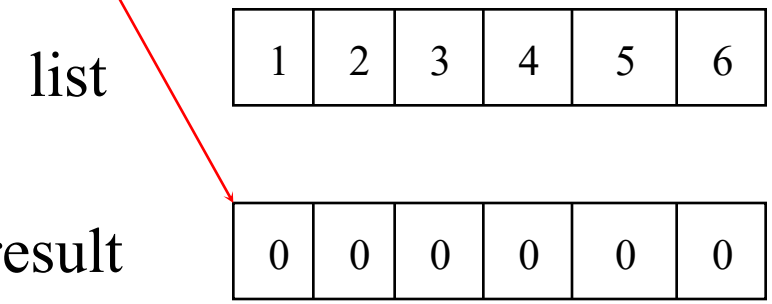

```
//Write it inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

animation

Trace the reverse Method

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

Declare result and create array

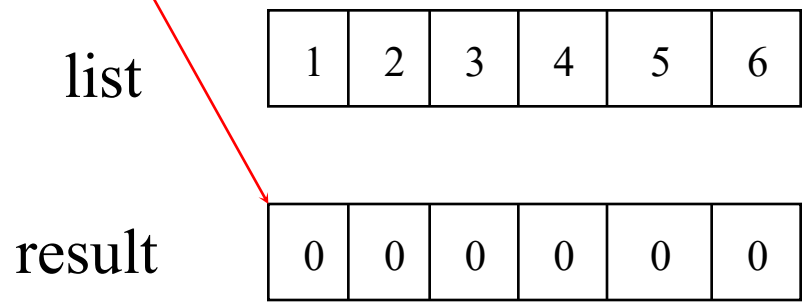


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 0 and j = 5

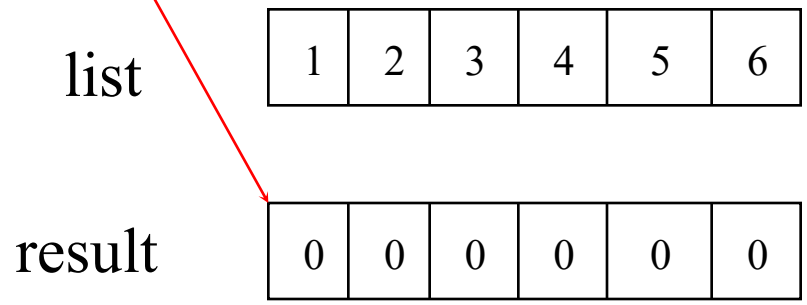


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (= 0) is less than 6

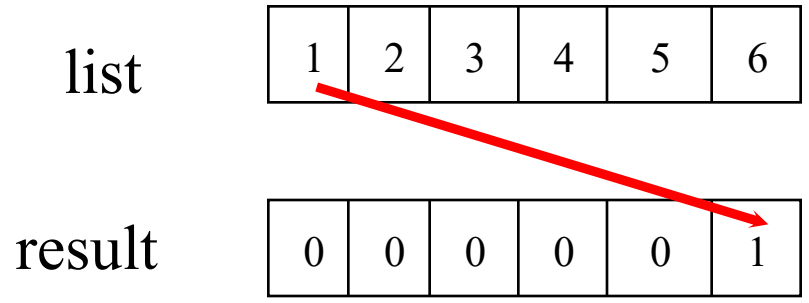


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 0 and j = 5
Assign list[0] to result[5]

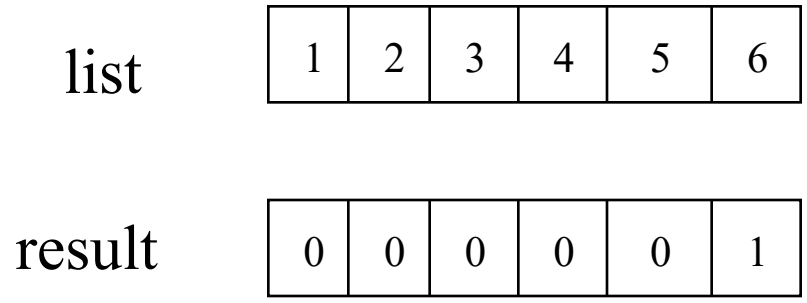


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

After this, i becomes 1 and j becomes 4

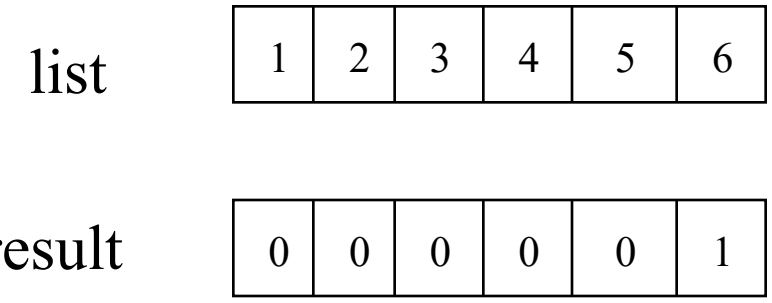


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=1) is less than 6

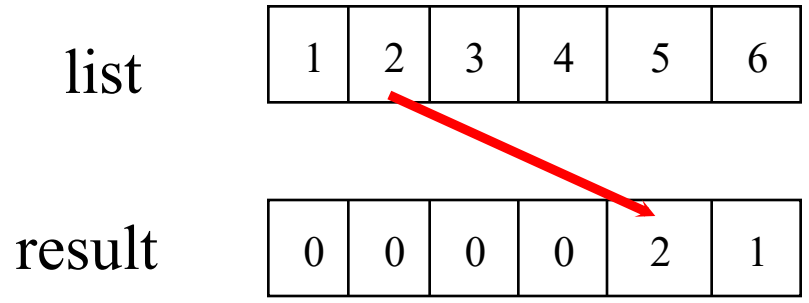


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 1 and j = 4
Assign list[1] to result[4]



animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

After this, i becomes 2 and j becomes 3

list	1	2	3	4	5	6

result	0	0	0	0	2	1

animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=2) is still less than 6

list	1	2	3	4	5	6

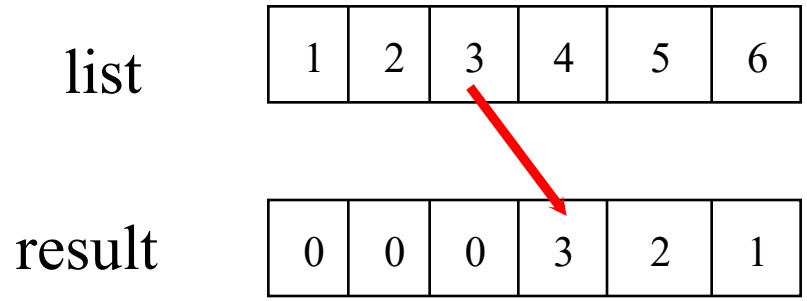
result	0	0	0	0	2	1

animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 2 and j = 3
Assign list[i] to result[j]

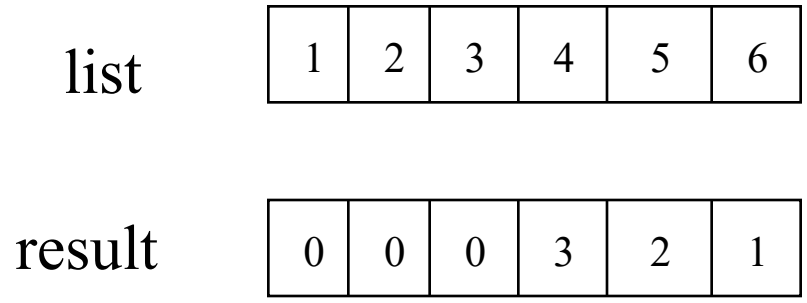


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

After this, i becomes 3 and j becomes 2



animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=3) is still less than 6

list	1	2	3	4	5	6

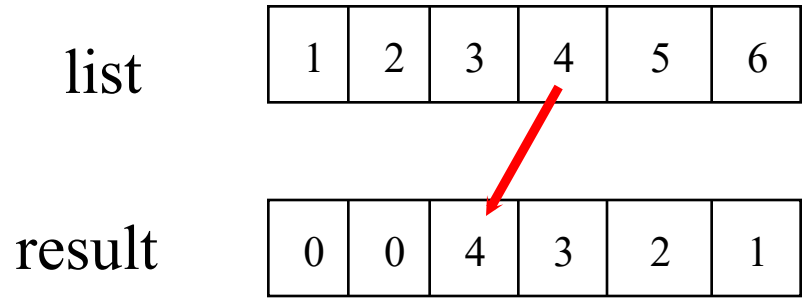
result	0	0	0	3	2	1

animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 3 and j = 2
Assign list[i] to result[j]

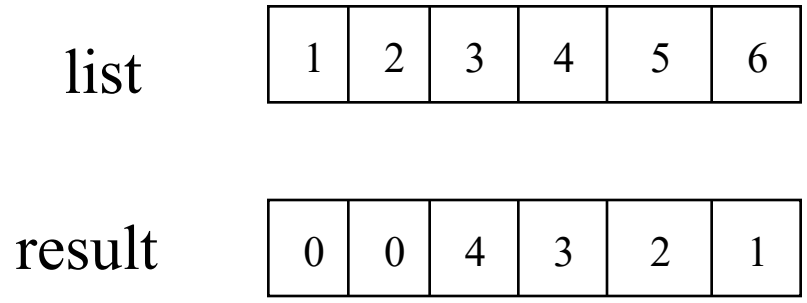


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

After this, i becomes 4 and j becomes 1

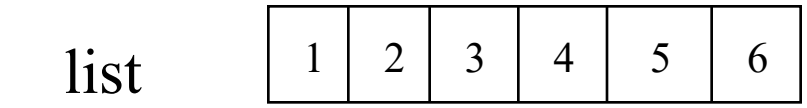


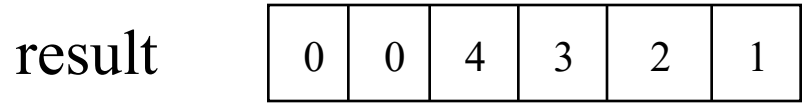
animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=4) is still less than 6



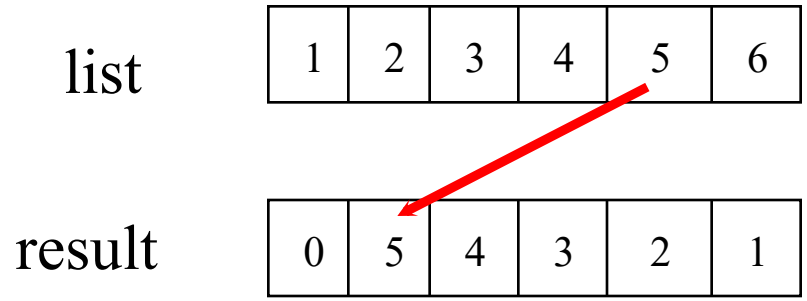


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 4 and j = 1
Assign list[i] to result[j]



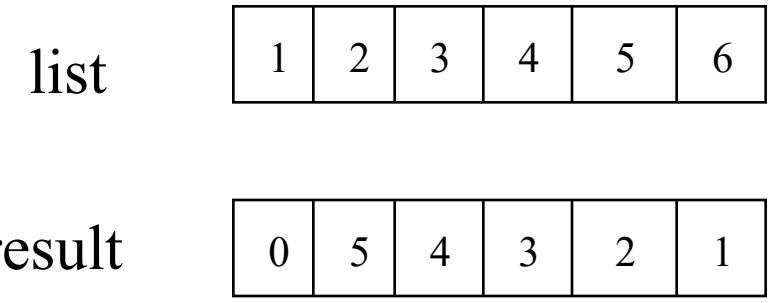
animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
```

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

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

  return result;
}
```

After this, i becomes 5 and j becomes 0



animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=5) is still less than 6

list	1	2	3	4	5	6

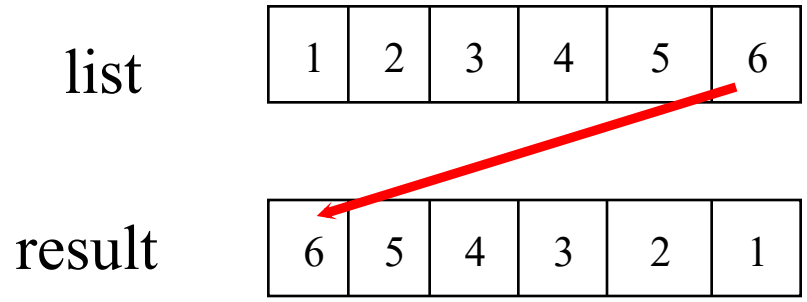
result	0	5	4	3	2	1

animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i = 5 and j = 0
Assign list[i] to result[j]

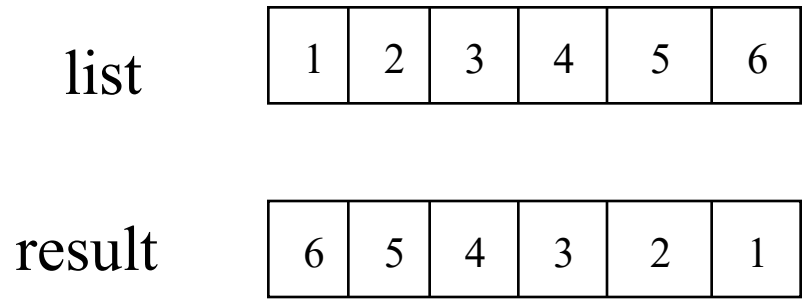


animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

After this, i becomes 6 and j becomes -1



animation

Trace the reverse Method, cont.

```
// inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```

i (=6) < 6 is false. So exit the loop.

list	1	2	3	4	5	6

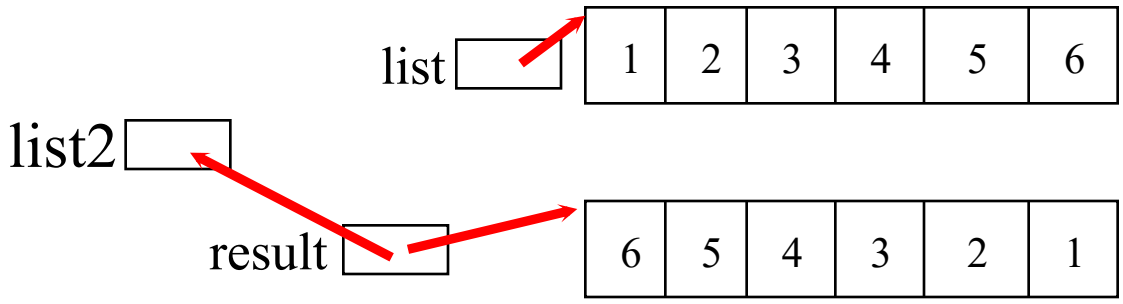
result	6	5	4	3	2	1

animation

Trace the reverse Method, cont.

```
// This code goes inside main method
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
// another method inside my class

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

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

  return result;
}
```

Return result



Variable-Length Arguments

You can pass a variable number of arguments of the same type to a method.

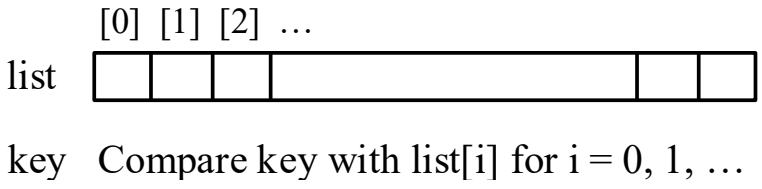
VarArgsDemo

Run

Searching Arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, *linear search* and *binary search*.

```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
}
```



Linear Searching Complete Program

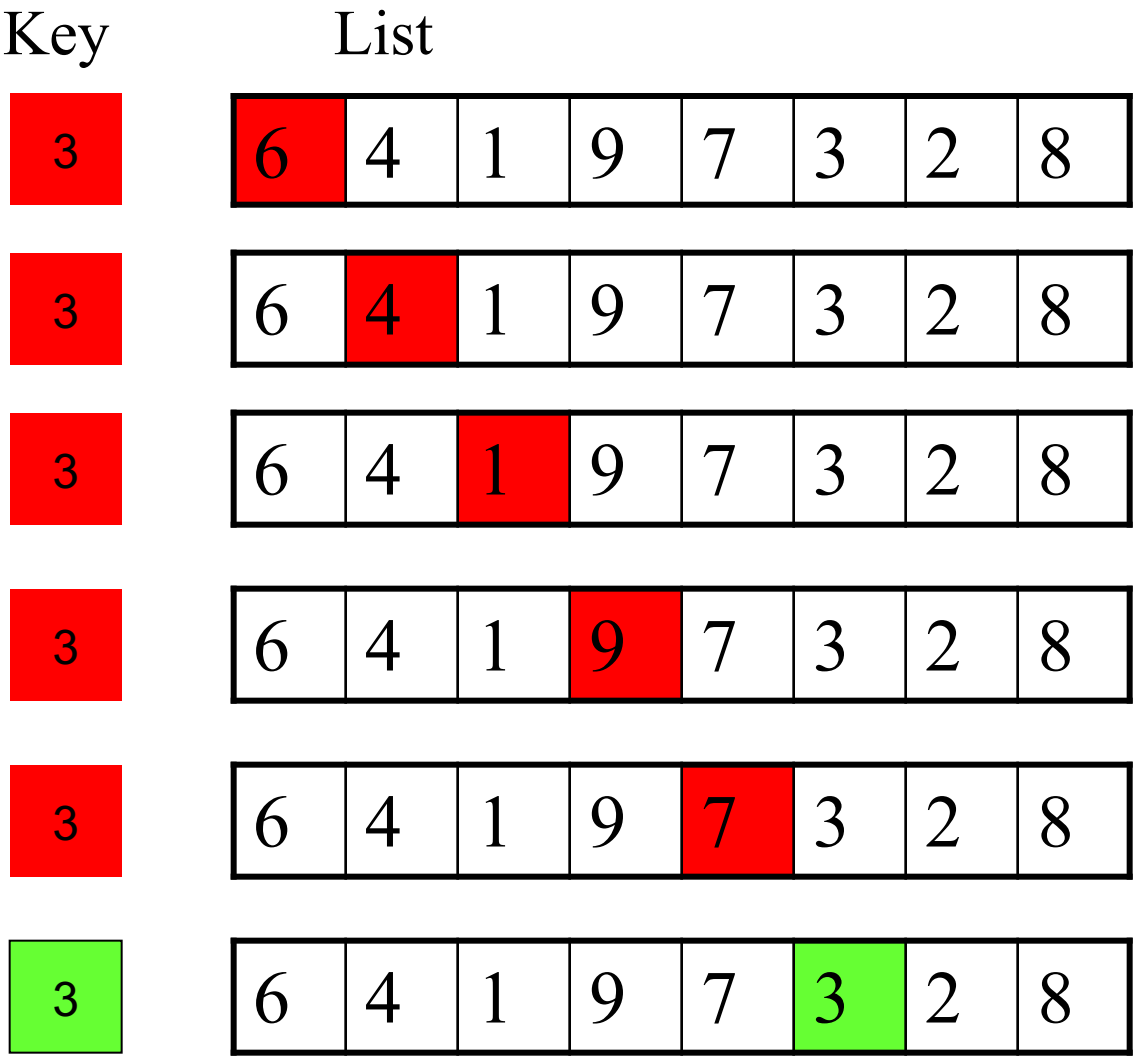
```
public class LinearSearch {
   public static void main(String[] args) {
   int[] y = {6,4,1,9,7,3,2,8}; // y represents an array of int values
   Scanner input=new Scanner(System.in);
   System.out.println("Enter the Element you want to search for =");
   int n=input.nextInt();
   int elementFound=linearSearch(y,n);
   if(elementFound>0)
      System.out.println("Element found on index = "+elementFound);
   else
      System.out.println("Element was not found in the list");
  }
   public static int linearSearch(int[] list, int key) {
      for (int i = 0; i < list.length; i++)
         if (key == list[i])
            return i;
      return -1;
  }
}
```

Linear Search

The linear search approach compares the key element, key, *sequentially* with each element in the array list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns -1.

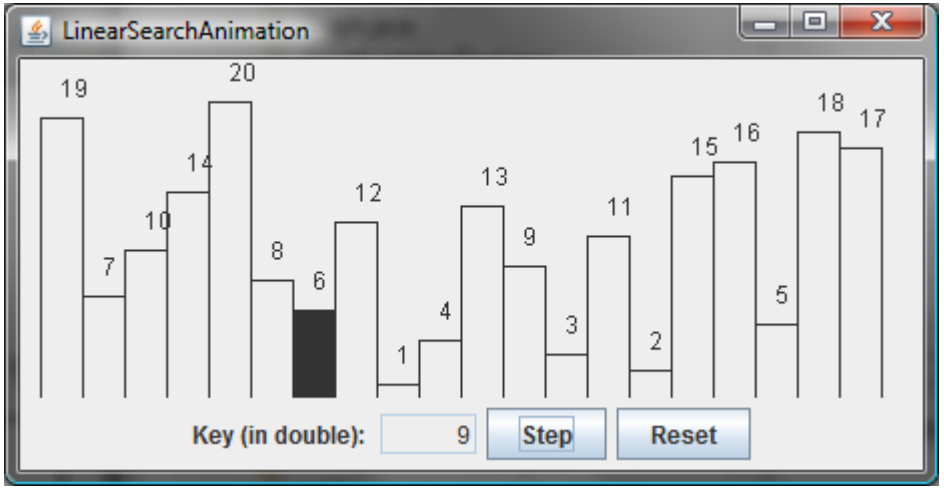
animation

Linear Search Animation



Linear Search Animation

http://www.cs.armstrong.edu/liang/animation/web/LinearSearch.html



Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithms: *selection sort*.

The Arrays.sort Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code sorts an array of numbers and an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

Java 8 now provides Arrays.parallelSort(list) that utilizes the multicore for fast sorting.

The Arrays.toString(list) Method

The Arrays.toString(list) method can be used to return a string representation for the list.