COMP20220
Programming II (Conversion)

Michael O'Mahony

Chapter 7 Single-Dimensional Arrays

Opening Problem

Suppose you are asked to write a program to read 100 numbers from the console, compute their average, and find out how many numbers are above the average.

Up to now, we would approach this problem by declaring 100 double variables, read in 100 numbers, and assign each number to one of the declared variables…

Using an array greatly simplify this process…

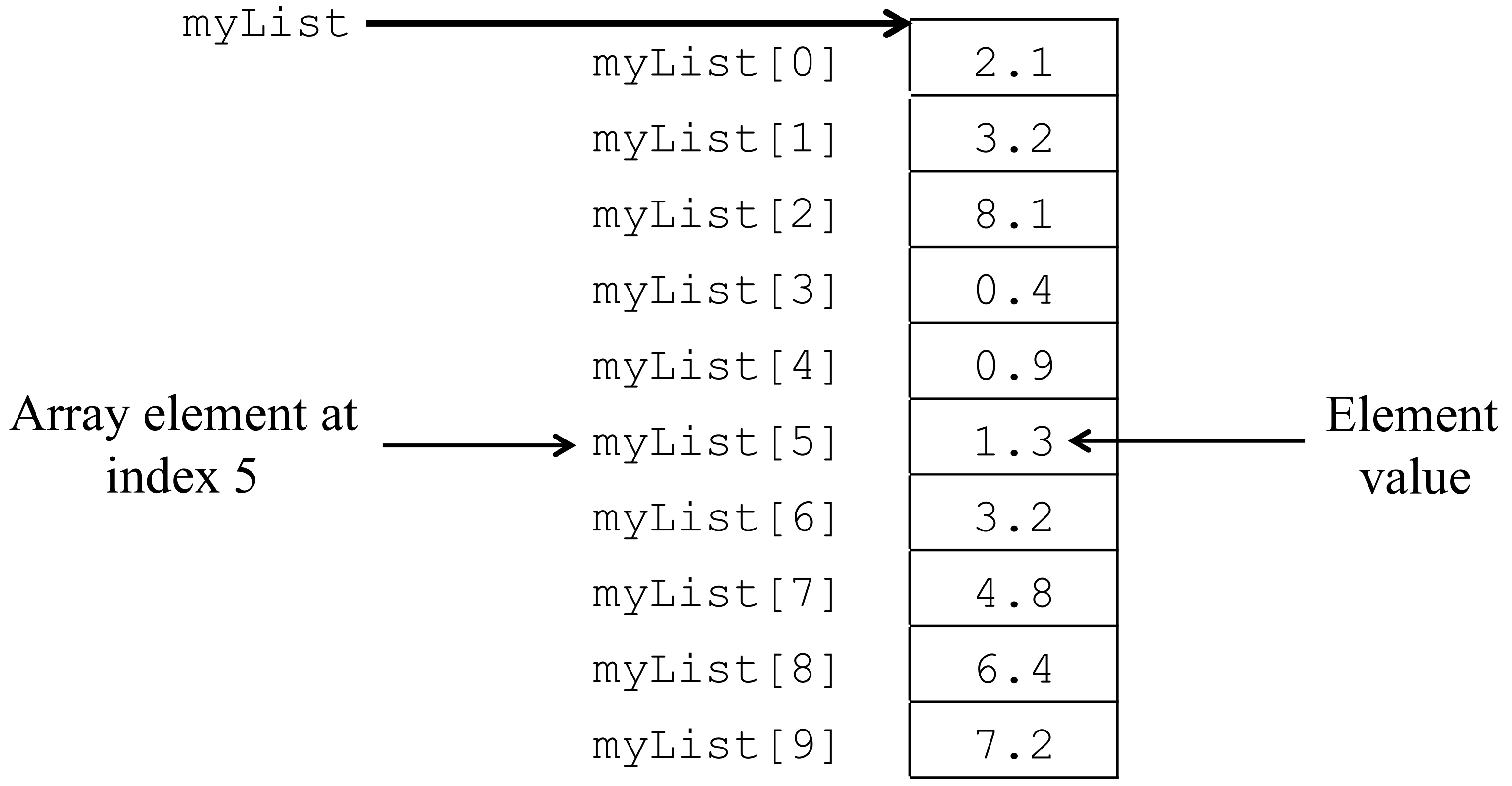
Objectives

- 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 and know the default values in an array (§7.2.3).
- To access array elements using indices (§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, and shifting elements) (§7.2.6).
- To simplify programming using **foreach** loops (§7.2.7).
- 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 search elements using the linear (§7.10.1) or binary (§7.10.2) search algorithm.
- To use the methods in the **java.util.Arrays** class (§7.12).

Introducing Arrays

An array is a data structure that stores a fixed-size, sequential collection of elements of the same type.

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



Declaring Array Variables

- Syntax:

  ```
  datatype[] arrayRefVar;
  ```

  Example:

  ```
  double[] myList;
  ```

- Alternative syntax (this style is allowed but not preferred):

  ```
  datatype arrayRefVar[];
  ```

  Example:

  ```
  double myList[];
  ```

Creating Arrays

- Syntax:

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

Example:

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

Declaring and Creating in One Step

- Syntax:

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

Example:

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

The Length of an Array

When an array is created, the array size must be given, specifying the number of elements that can be stored in it.

Once an array is created, its size is fixed and cannot be changed.

You can find the *size of* (aka *length of*, aka *number of elements in*) an array using:

```
arrayRefVar.length
```

Example:

```
double[] myList = new double[10];
int len = myList.length // len is 10
```

Default Values

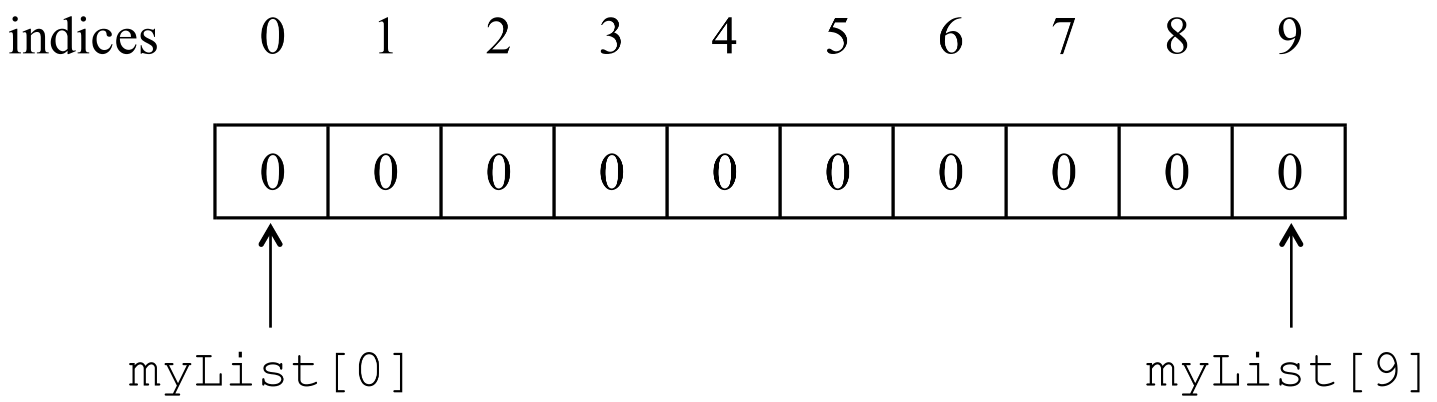
When an array is created, its elements are assigned the default value of:

- 0 for the numeric primitive data types
- `'\u0000'` for `char` types
- `false` for `boolean` types

Indexed Variables

Array elements are accessed through *indices*. For example:

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



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

```
arrayRefVar[index];
```

For example: `myList[0], myList[1], …`

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 assigns values to `myList[0]` and `myList[1]`:

```
myList[0] = 2.1;
myList[1] = 3.2;
```

And the following adds the values in `myList[0]` and `myList[1]` and assigns the result to `myList[2]`:

```
myList[2] = myList[0] + myList[1]; // myList[2] is 5.3
```

Declaring, Creating, Initializing

Declaring, creating, initializing in one step:

```
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;
```

Note

Using the shorthand notation, you have to declare, create, and initialize the array all in one statement.

Splitting it causes a syntax error. For example, the following is incorrect:

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

animation

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```

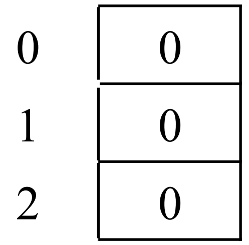
animation

Trace Program with Arrays

Declare and create an array `values`

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



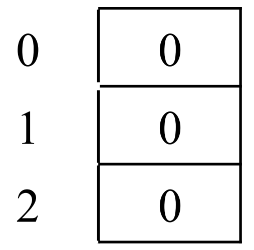
animation

Trace Program with Arrays

Initialize i to 0

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



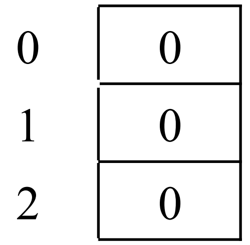
animation

Trace Program with Arrays

0 is less than 3

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



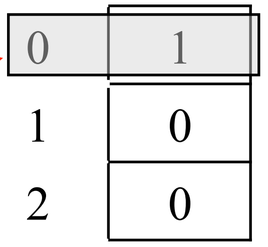
animation

Trace Program with Arrays

After this statement, `values[0]` is 1 (0 + 1)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



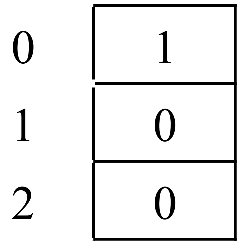
animation

Trace Program with Arrays

Increment `i` (`i` is now 1)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



animation

Trace Program with Arrays

1 is less than 3

```
public class Test {
  public static void main(  ring[] args) {
    int[] values = new int 3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```

0	1
1	0
2	0

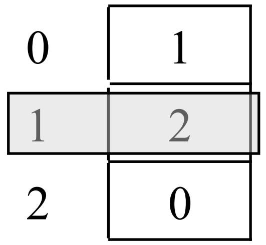
animation

Trace Program with Arrays

After this statement, `values[1]` is 2 (1 + 1)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



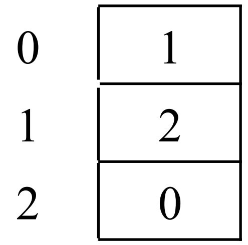
animation

Trace Program with Arrays

Increment `i` (`i` is now 2)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++) {
      values[i] = i + 1;
  }
}
```



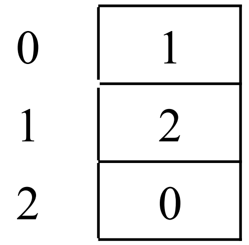
animation

Trace Program with Arrays

2 is less than 3

```
public class Test {
  public static void main(  ring[] args) {
    int[] values = new int 3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



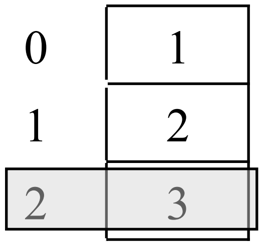
animation

Trace Program with Arrays

After this statement, `values[2]` is 3 (2 + 1)

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



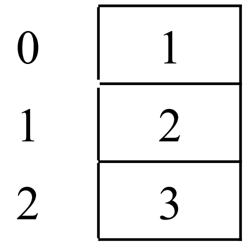
animation

Trace Program with Arrays

Increment `i` (`i` is now 3)

```
public class Test {
  public static void main(String[] args)
    int[] values = new int[3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```



animation

Trace Program with Arrays

3 is not less than 3, exit loop

```
public class Test {
  public static void main(  ring[] args) {
    int[] values = new int 3];

    for (int i = 0; i < values.length; i++)
      values[i] = i + 1;
  }
}
```

0	1
1	2
2	3

Processing Arrays

Some code snippets to show examples of using arrays:

1. Initializing arrays with input values
2. Initializing arrays with random values
3. Printing arrays
4. Summing all elements
5. Finding the largest element
6. Finding the index of the largest element
7. Shifting elements

Initializing arrays with input values

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

Scanner input = new Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");

for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

Initializing arrays with random values

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

for (int i = 0; i < myList.length; i++)
  myList[i] = Math.random() * 100;
```

Printing arrays

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

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

Summing all elements

```
double[] myList = new double[10];
double sum = 0;

for (int i = 0; i < myList.length; i++)
  sum += myList[i];
```

Finding the largest element

```
double[] myList = new double[10];
double max = myList[0];

for (int i = 1; i < myList.length; i++)
  if (myList[i] > max)
    max = myList[i];
```

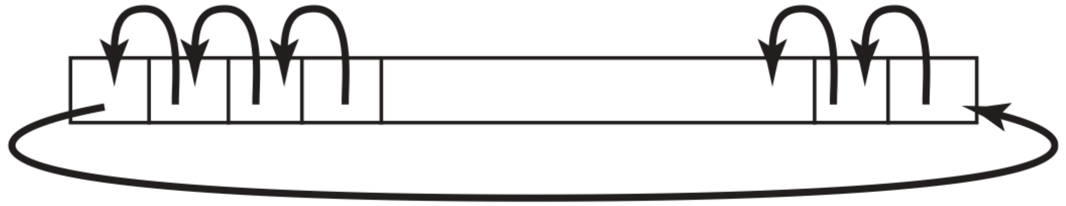
Finding the index of the largest element

```
double[] myList = new double[10];
int maxIndex = 0;

for (int i = 1; i < myList.length; i++)
  if (myList[i] > myList[maxIndex])
    maxIndex = i;
```

Shifting elements

Sometimes you need to shift the elements left or right. Here is an example of shifting the elements one position to the left and filling the last element with the first element:



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

// Shift elements one position to the left
for (int i = 1; i < myList.length; i++)
  myList[i - 1] = myList[i];

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

`foreach` Loop

`foreach` loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

`foreach` Loop

`foreach` loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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

`foreach` Loop

`foreach` loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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

```
for (double x: myList)
   System.out.print(x + " ");
```

`foreach` Loop

`foreach` loop – enables you to traverse the complete array sequentially without using an index variable.

The following are equivalent and display all elements in the array:

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

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

```
for (double x: myList)
   System.out.print(x + " ");
```

General syntax:

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

Note: you need to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Opening Problem

Read *n* numbers from the console, compute their average, and find out how many numbers are above the average.

AnalyzeNumbers

Problem: Lotto Numbers

Suppose you play *Pick-10* lotto. Each ticket has 10 numbers ranging from 1 to 99, inclusive.

Assume that the numbers in a ticket are picked randomly. Further, assume that the same number may appear in a ticket more than once.

Over all your tickets, you wish to have all the numbers from 1 to 99 included at least once.

Write a program that generates lotto tickets. Print the tickets to the standard output. The program should exit when all numbers from 1 to 99 are included at least once in the tickets, and the number of tickets generated should be displayed.

LottoTickets

Reference Types

Declaring array variables:

- Consider: `int[] list1;`
- An array variable (e.g. `list1`) is a *reference variable* – when declared, it does not allocate space in memory for an array, it just contains a reference to an array.

Creating arrays:

- An array is created by using the `new` operator and its reference is assigned to the array variable `list1` using the following syntax:

```
list1 = new int[10];
```

- The above statement does two things:
  1) It creates an array using `new int[10];`
  2) It assigns the reference of the newly created array to the variable `list1`.
- You cannot assign elements to an array unless it has already been created.

Reference Types

Suppose you created two arrays as follows:

```
int[] list1 = {1, 2, 3, 4, 5};
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;
list1[0] = 100;
list2[1] = 200;
```

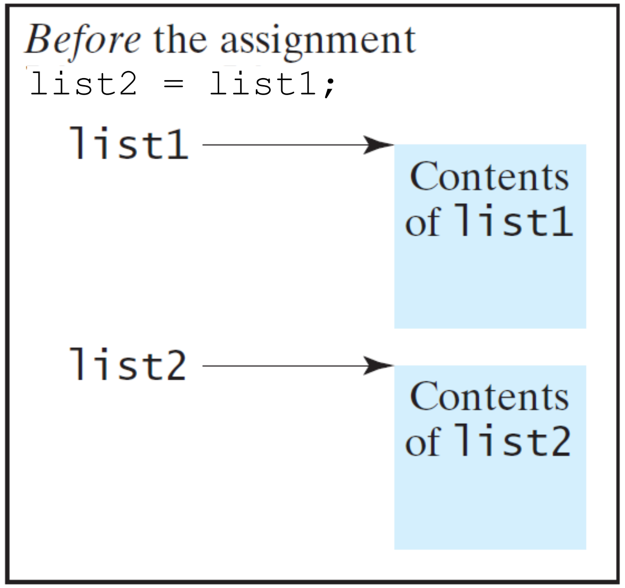
Reference Types

Suppose you created two arrays as follows:

```
int[] list1 = {1, 2, 3, 4, 5};
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;
list1[0] = 100;
list2[1] = 200;
```



Reference Types

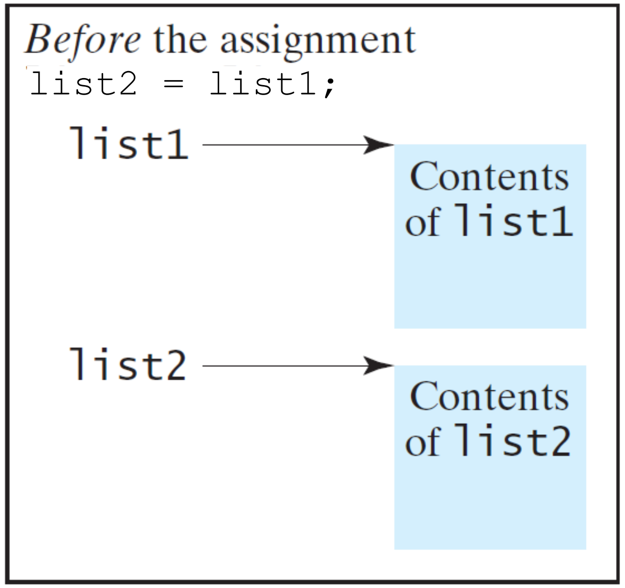
ArrayReferenceTest

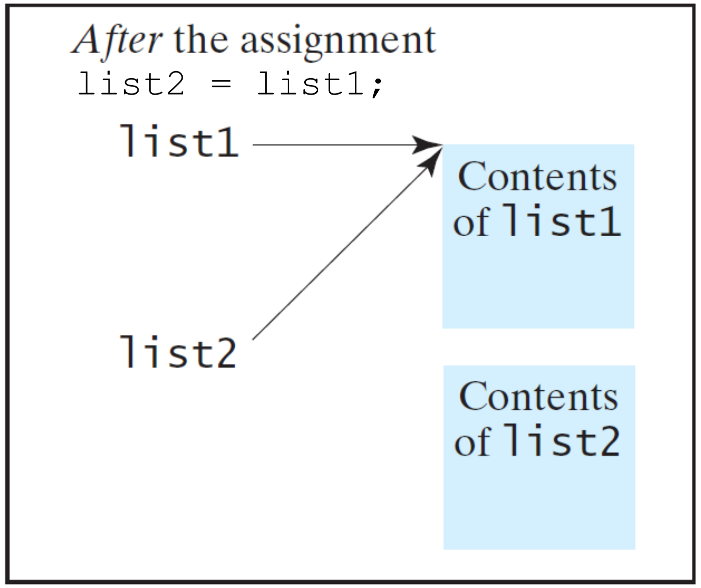
Suppose you created two arrays as follows:

```
int[] list1 = {1, 2, 3, 4, 5};
int[] list2 = {10, 20, 30};
```

What happens after the following statements?

```
list2 = list1;
list1[0] = 100;
list2[1] = 200;
```





Copying Arrays

To copy the contents of one array into another, you have to copy the array’s individual elements into the other array.

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

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

The `arraycopy` Utility

Java provides a method to copy arrays. The syntax is:

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length);
```

The parameters `srcPos` and `tarPos` indicate the starting positions in `sourceArray` and `targetArray`, respectively. The number of elements copied from `sourceArray` to `targetArray` is indicated by length.

Example:

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

Note: the `arraycopy` method does not allocate memory space for the target array. The target array must have already been created with its memory space allocated.

Passing Arrays to Methods

When passing an array to a method, the *reference* of the array is passed to the method.

Example – a method to print an array:

```
public static void main(String[] args) {
  int[] list = {3, 1, 2, 6, 4, 2};
  printArray(list); // Invoke the method
}

public static void printArray(int[] arr) {
  for (int i = 0; i < arr.length; i++)
    System.out.print(arr[i] + " ");
}
```

Pass By Value

Java uses *pass by value* to pass arguments to a method.

There are important differences between passing primitive data type variables and passing array variables to methods…

For a parameter of a primitive type, changing the value of the parameter inside the method does not affect the value of the variable passed to the method (we have seen this before…).

For a parameter of an array type, any changes made to the array elements inside the method will affect the original array that was passed to the method.

Passing Arrays as Arguments

Examples to demonstrate the differences between passing primitive data type variables and array variables to methods.

TestPassArray1

TestPassArray2

Returning an Array from a Method

When a method returns an array, the *reference* of the array is returned. For example:

```
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;
}
```

// Invoke this method in another part of the program (e.g. in method `main`)

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);  // what happens?
```

animation

Trace the reverse Method

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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;
}
```

animation

Trace the reverse Method

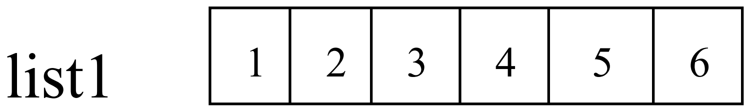
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Declare `list1` and create array

```
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;
}
```



animation

Trace the reverse Method

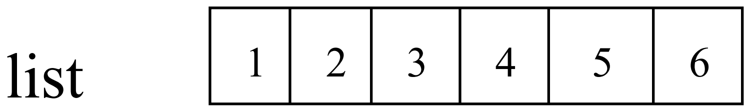
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Invoke method `reverse`

```
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;
}
```



animation

Trace the reverse Method

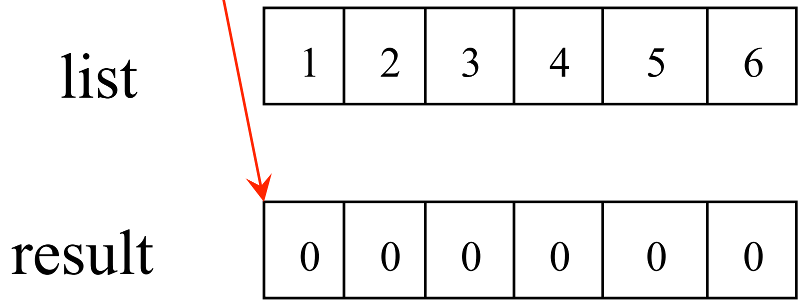
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Declare `result` and create array

```
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;
}
```



animation

Trace the reverse Method, cont.

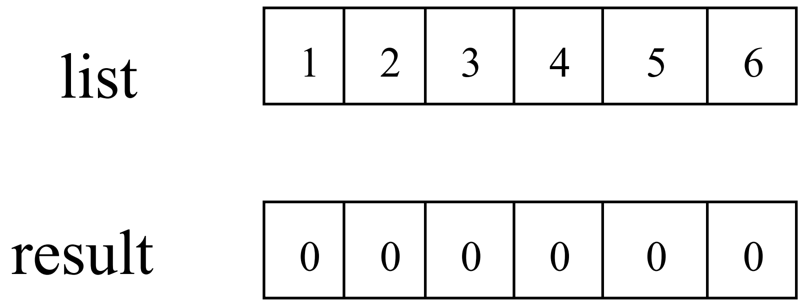
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i = 0 and j = 5

```
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;
}
```



animation

Trace the reverse Method, cont.

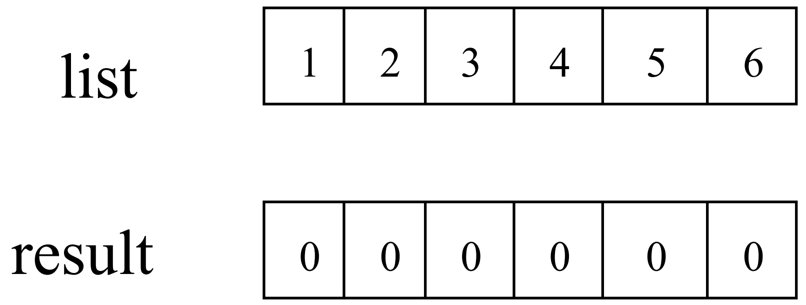
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (= 0) is less than 6

```
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;
}
```



animation

Trace the reverse Method, cont.

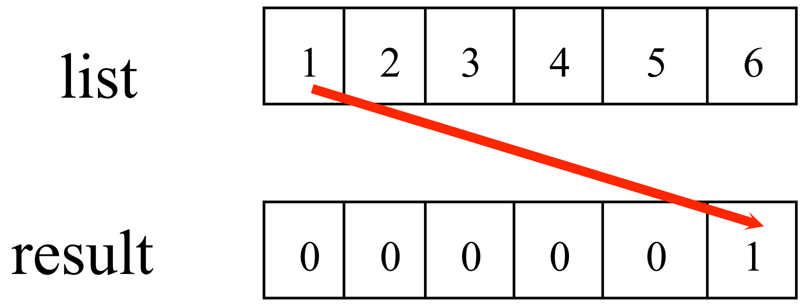
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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.

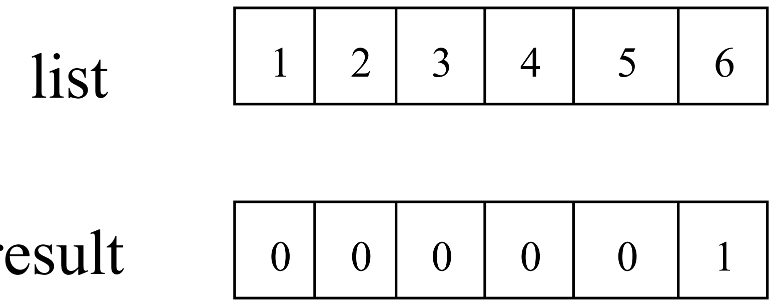
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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.

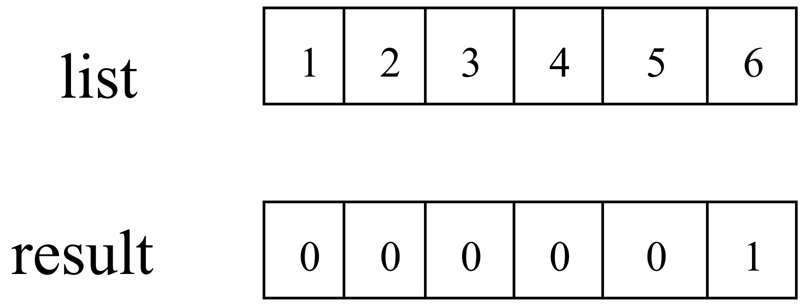
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=1) is less than 6

```
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;
}
```



animation

Trace the reverse Method, cont.

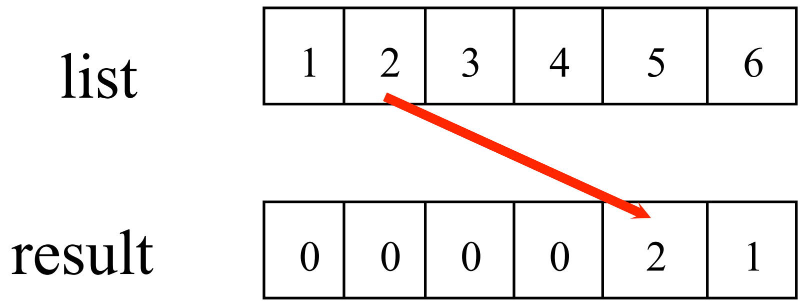
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 2 and j becomes 3

```
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;
}
```

list	1	2	3	4	5	6
result	0	0	0	0	2	1

animation

Trace the reverse Method, cont.

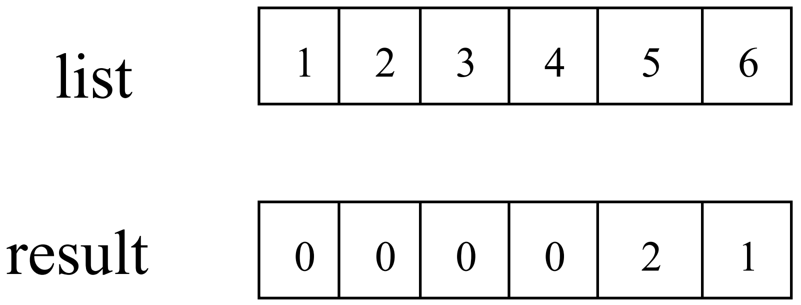
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=2) is still less than 6

```
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;
}
```



animation

Trace the reverse Method, cont.

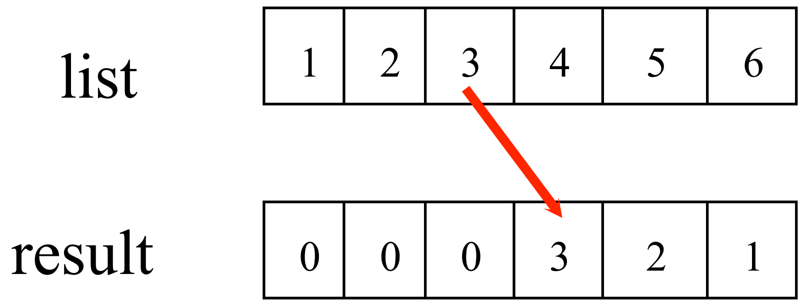
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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[2] to result[3]



animation

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 3 and j becomes 2

```
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;
}
```

list	1	2	3	4	5	6
result	0	0	0	3	2	1

animation

Trace the reverse Method, cont.

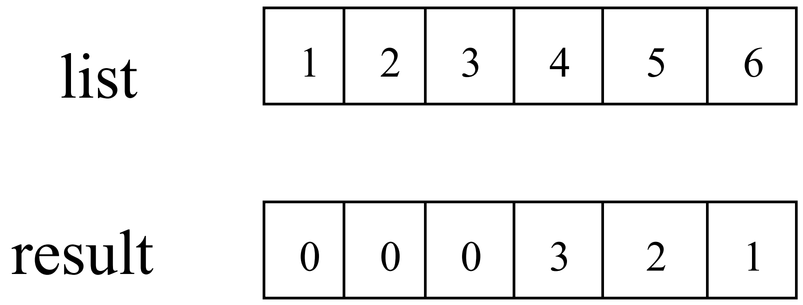
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=3) is still less than 6

```
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;
}
```



animation

Trace the reverse Method, cont.

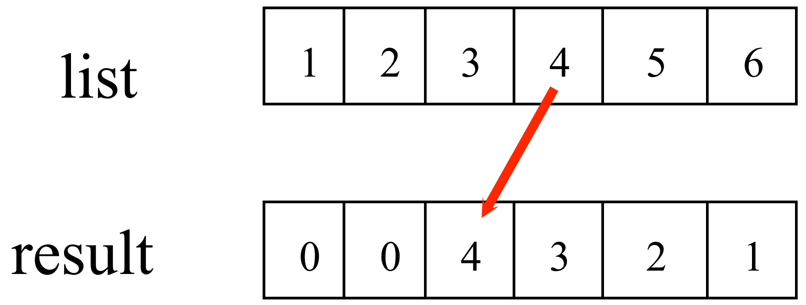
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

```
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;
}
```



animation

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 4 and j becomes 1

```
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;
}
```

list	1	2	3	4	5	6
result	0	0	4	3	2	1

animation

Trace the reverse Method, cont.

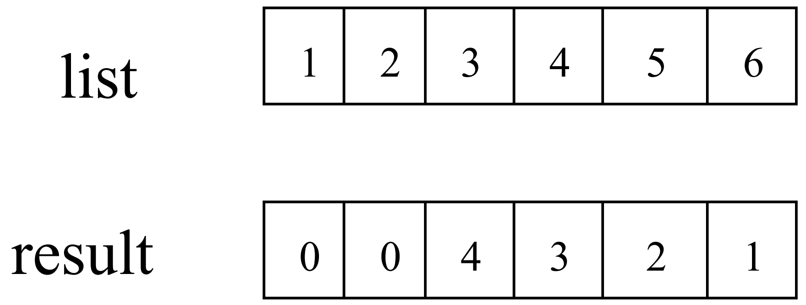
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=4) is still less than 6

```
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;
}
```



animation

Trace the reverse Method, cont.

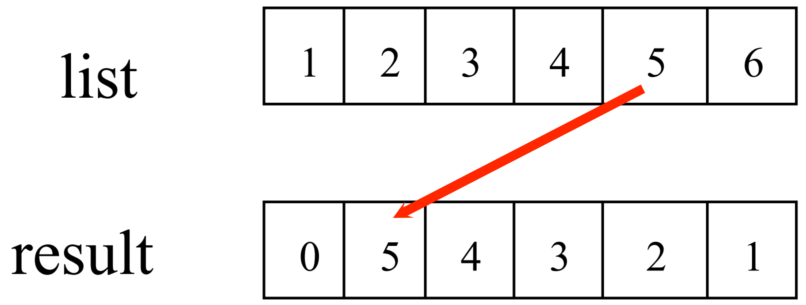
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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[4] to result[1]



animation

Trace the reverse Method, cont.

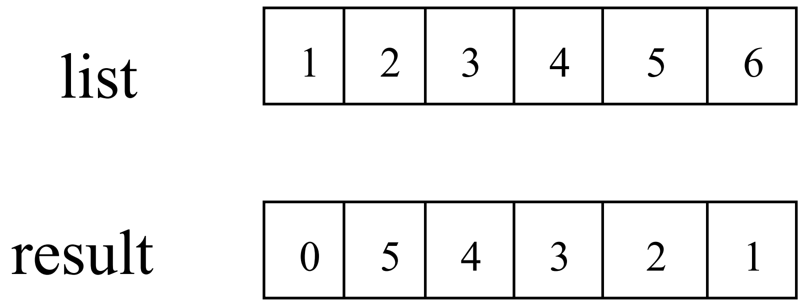
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 5 and j becomes 0

```
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;
}
```



animation

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=5) is still less than 6

```
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;
}
```

list	1	2	3	4	5	6
result	0	5	4	3	2	1

animation

Trace the reverse Method, cont.

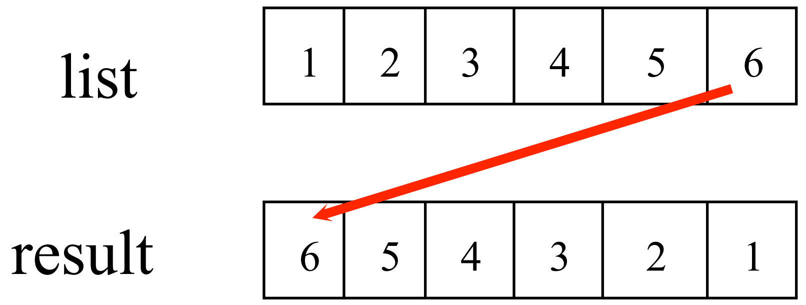
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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[5] to result[0]



animation

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 6 and j becomes -1

```
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;
}
```

list	1	2	3	4	5	6
result	6	5	4	3	2	1

animation

Trace the reverse Method, cont.

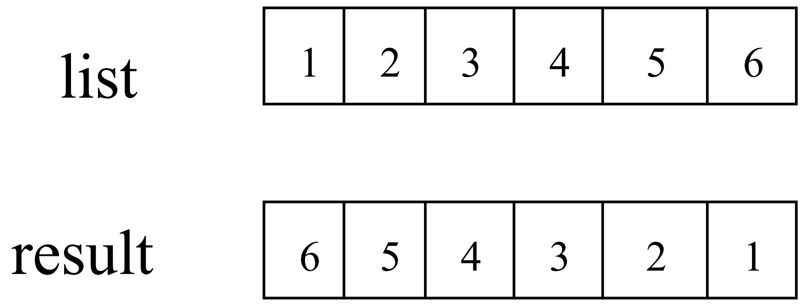
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

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

```
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;
}
```



animation

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Return result

```
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;
}
```

list	1	2	3	4	5	6
result	6	5	4	3	2	1

animation

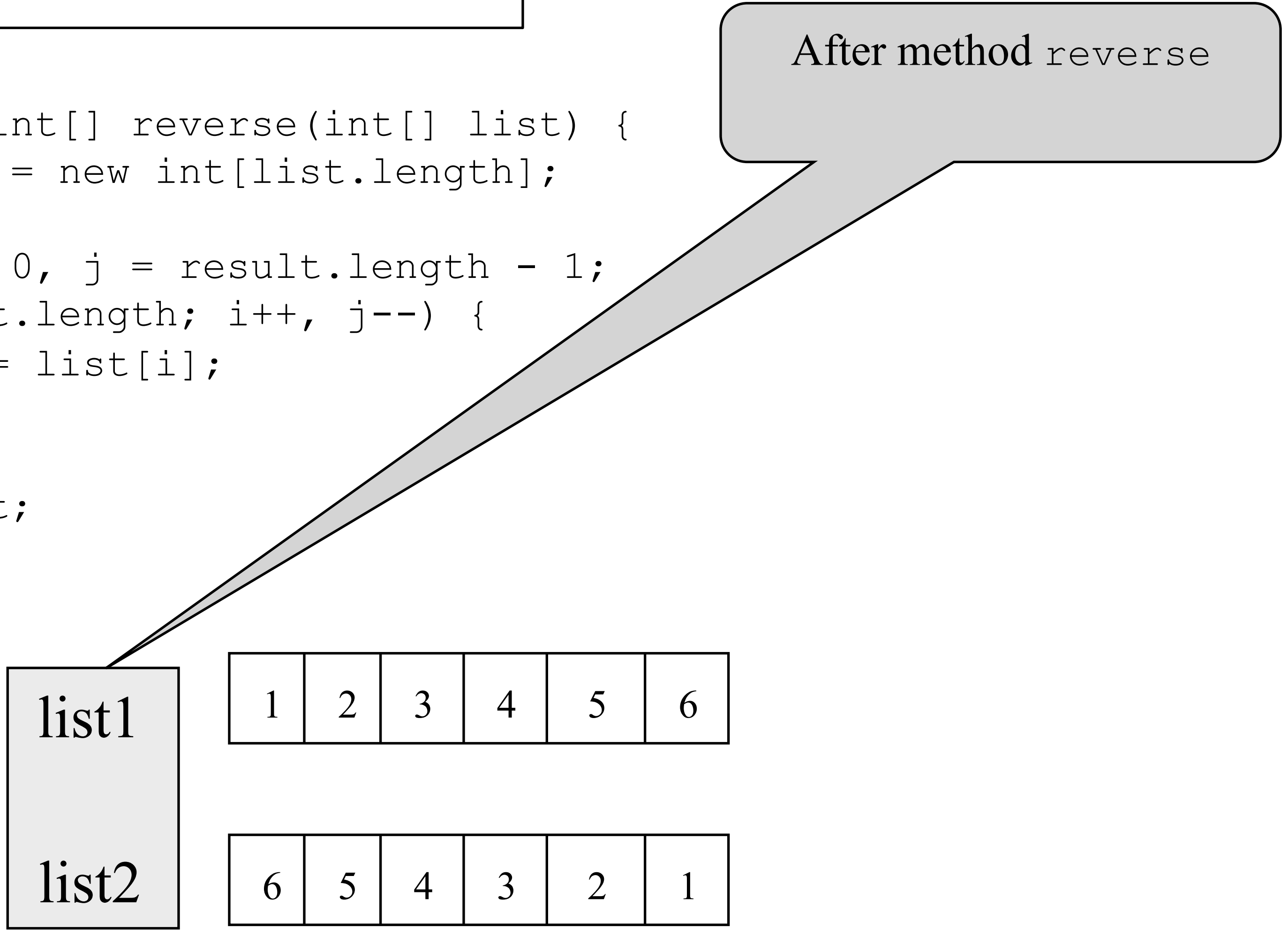
Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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;
}
```



Problem: Counting the Occurrence of Each Letter

Randomly generate 50 lowercase letters and assign to an array of characters.

Count the occurrence of each letter in the array and display the result.

CountLettersInArray

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.

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 (1) an array of numbers and (2) an array of characters.

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 (1) an array of numbers and (2) an array of characters.

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

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 (1) an array of numbers and (2) an array of characters.

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

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 (1) an array of numbers and (2) an array of characters.

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

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

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 (1) an array of numbers and (2) an array of characters.

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

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

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
`'0'` to `'9'`	48 to 57	\u0030 to \u0039
`'A'` to `'Z'`	65 to 90	\u0041 to \u005A
`'a'` to `'z'`	97 to 122	\u0061 to \u007A

The `Arrays.toString` Method

The `java.util.Arrays.toString` method can be used to return a string representation for the array.

For example:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
System.out.println(Arrays.toString(numbers));
```

// Prints: [6.0, 4.4, 1.9, 2.9, 3.4, 3.5]

Passing Arguments to the `main` Method

The `main` method can be invoked just like a regular method

For example, the `main` method in class `B` is invoked by a method in `A`, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Next Topic

Chapter 8:

- Multidimensional arrays…