Chapter 6 Arrays

Basic computer skills such as using Windows, Internet Explorer, and Microsoft Word

Chapter 1 Introduction to Computers, Programs, and Java

Chapter 2 Primitive Data Types and Operations

Chapter 3 Selection Statements

Chapter 4 Loops

Chapter 5 Methods → §§19.1-19.3 in Chapter 19 Recursion
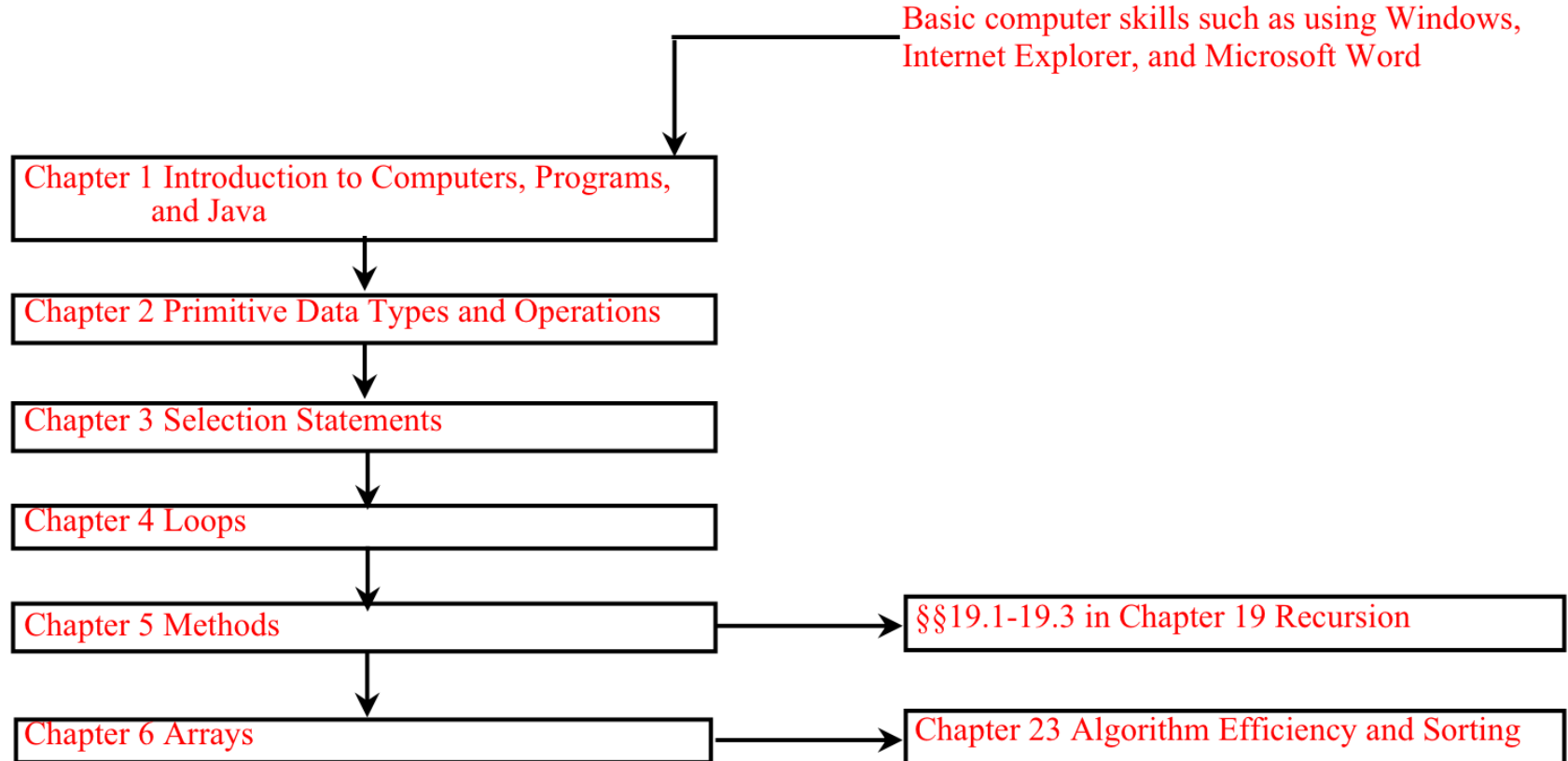
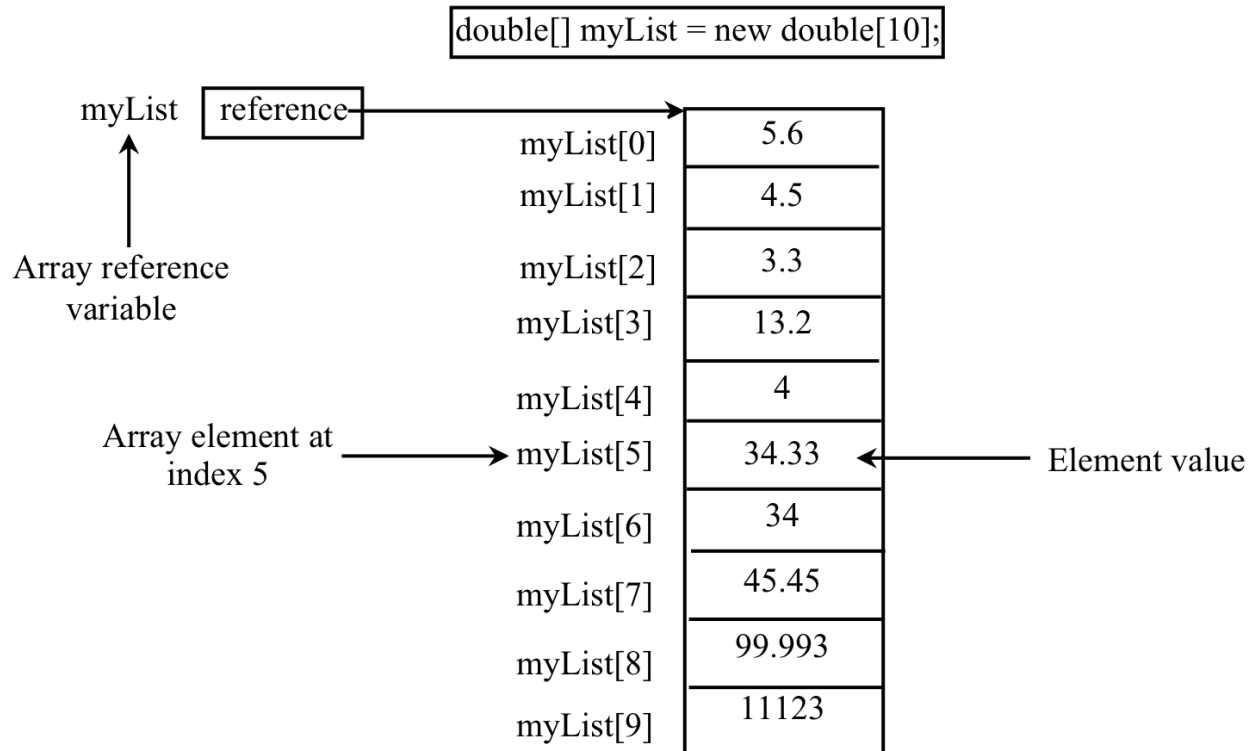Chapter 6 Arrays → Chapter 23 Algorithm Efficiency and Sorting

Objectives

- To describe why an array is necessary in programming (§6.1).
- To learn the steps involved in using arrays: declaring array reference variables and creating arrays (§6.2).
- To initialize the values in an array (§6.2).
- To simplify programming using JDK 1.5 enhanced for loop (§6.2).
- To copy contents from one array to another (§6.3).
- To develop and invoke methods with array arguments and ruturn type (§6.4-6.5).
- To sort an array using the selection sort algorithm (§6.6).
- To search elements using the linear or binary search algorithm (§6.7).
- To declare and create multidimensional arrays (§6.8).
- To declare and create multidimensional arrays (§6.9 Optional).

Introducing Arrays

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

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

myList[0]	5.6
myList[1]	4.5
myList[2]	3.3
myList[3]	13.2
myList[4]	4
myList[5]	34.33
myList[6]	34
myList[7]	45.45
myList[8]	99.993
myList[9]	11123

myList | reference

Array reference variable

Array element at index 5

Element value

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

    <u>0</u> for the numeric primitive data types,

    <u>'\u0000'</u> for <u>char</u> types, and

    <u>false</u> for <u>boolean</u> 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];

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};
```

Enhanced <u>for</u> 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);
```
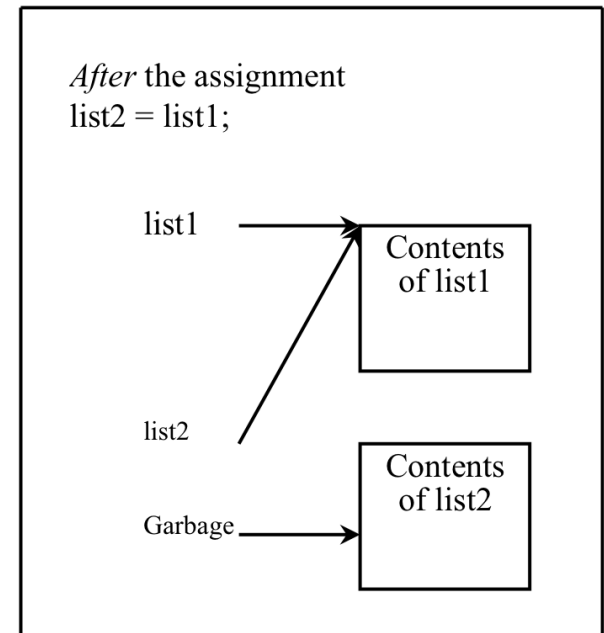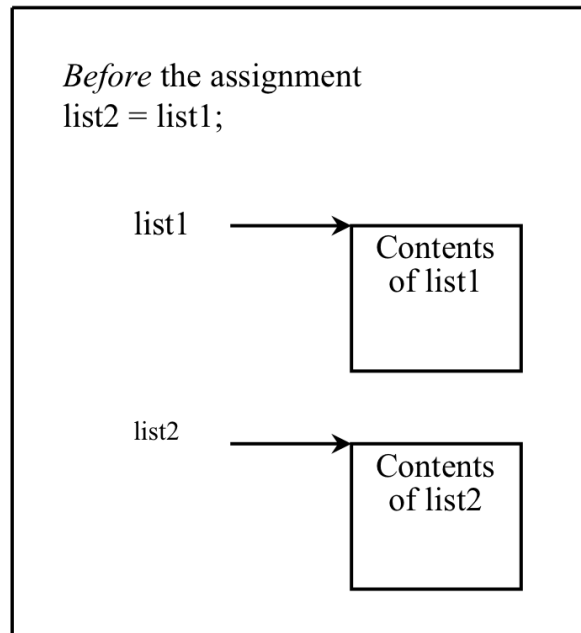
In general, the syntax is

```
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.

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;

Before the assignment
list2 = list1;

list1 → Contents of list1

list2 → Contents of list2

After the assignment
list2 = list1;

list1 → Contents of list1

list2 ↗

Garbage → Contents of list2

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

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

Invoke the method

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

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

Anonymous array

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*.
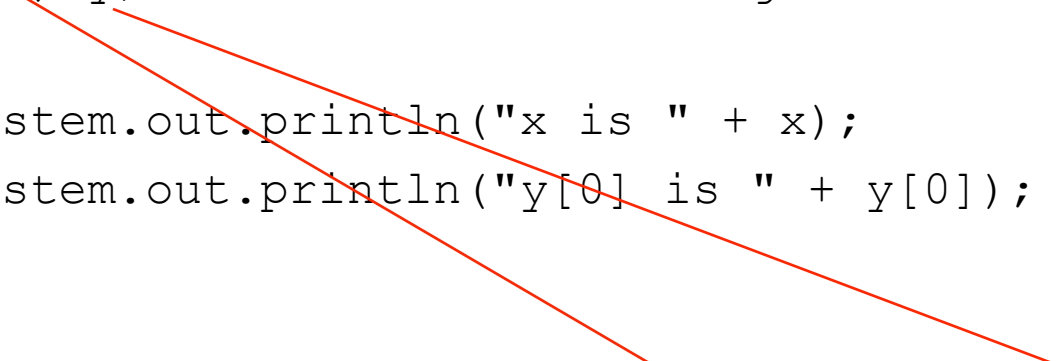
Pass By Value

Java uses *pass by value* to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays (or objects).

• 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 (or object), the value of the parameter contains a reference to an array (or object); this reference is passed to the method. Any changes to the array (or object) that occur inside the method body will affect the original array (or object) that was passed as the argument.

Simple Example

```
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]
  }
}
```
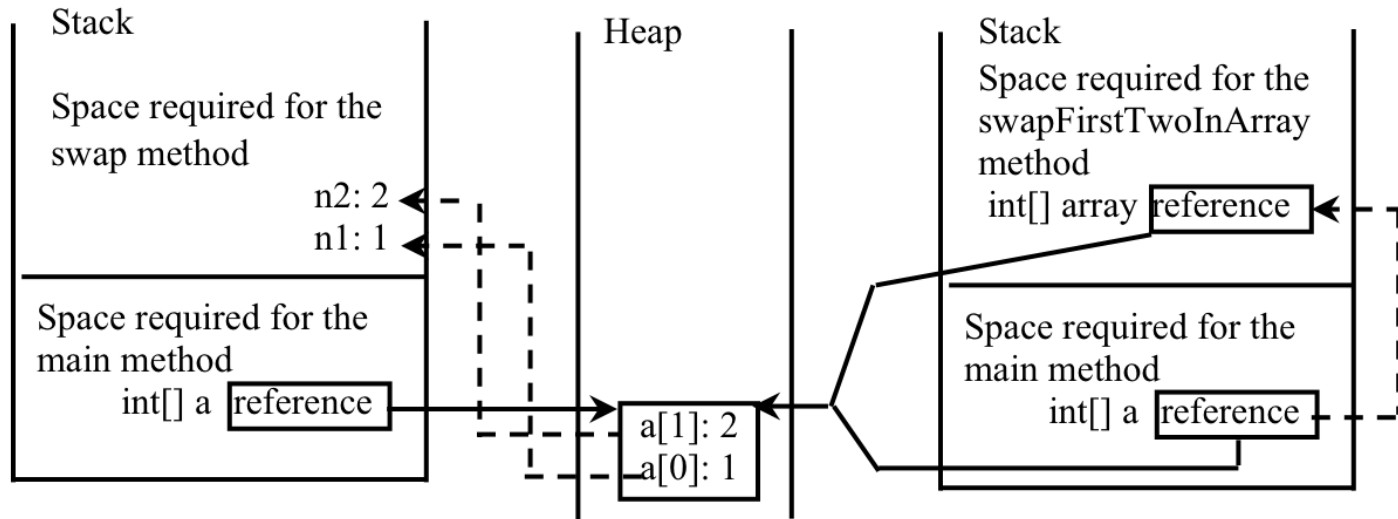
Example:
Passing Arrays as Arguments

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

TestPassArray

Example, cont.

Stack

Space required for the
swap method
        n2: 2
        n1: 1

Space required for the
main method
    int[] a reference

Heap

a[1]: 2
a[0]: 1

Stack
Space required for the
swapFirstTwoInArray
method
 int[] array reference

Space required for the
main method
    int[] a reference

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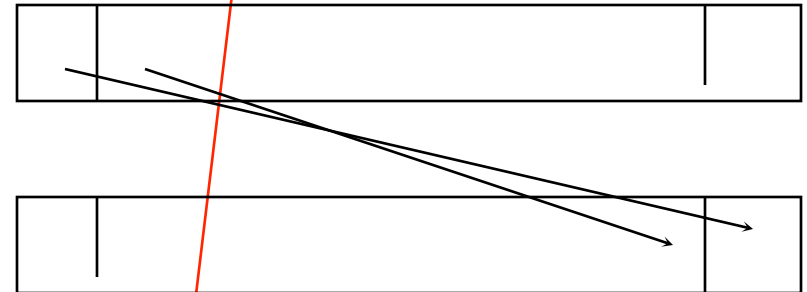swap method.

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 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

result

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

Trace the reverse Method

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	0	0	0	0	0

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	0	0	0	0	0

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	0	0	0	0	0

Trace the reverse Method, cont.

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

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

list

1	2	3	4	5	6

result

0	0	0	0	0	1

Trace the reverse Method, cont.

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

After this, i becomes 1 and j becomes 4

```java
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

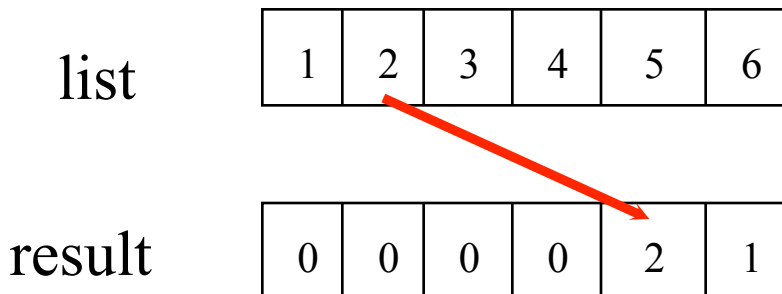| 1	2	3	4	5	6

result

0	0	0	0	0	1

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	0	0	0	0	1

Trace the reverse Method, cont.

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

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

```
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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

Trace the reverse Method, cont.

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

i (=2) is still less than 6

```java
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

Trace the reverse Method, cont.

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

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

```java
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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

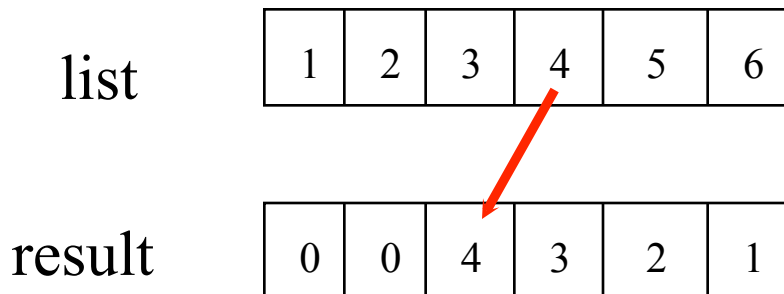| 1	2	3	4	5	6

result

0	0	0	3	2	1

Trace the reverse Method, cont.

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

i (=3) is still less than 6

```java
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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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 = 3 and j = 2
Assign list[i] to result[j]

list

1	2	3	4	5	6

result

0	0	4	3	2	1

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	0	4	3	2	1

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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[i] to result[j]

list

1	2	3	4	5	6

result

0	5	4	3	2	1

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

list

1	2	3	4	5	6

result

0	5	4	3	2	1

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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[i] to result[j]

list

1	2	3	4	5	6

result

6	5	4	3	2	1

42

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

Trace the reverse Method, cont.

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

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

```java
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

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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 |

list2

result | 6 | 5 | 4 | 3 | 2 | 1 |
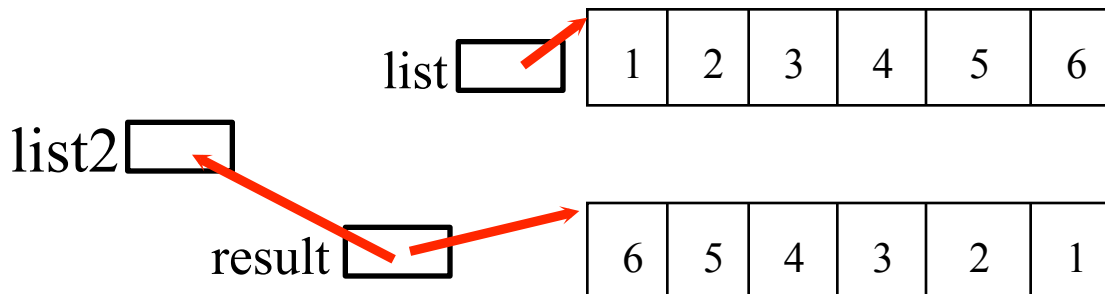
Two-dimensional Arrays

```
// Declare array ref var
dataType[][] refVar;


// Create array and assign its reference to variable
refVar = new dataType[10][10];


// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];


// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

```
int[][] matrix = new int[10][10];
  or
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);

double[][] x;
```

Two-dimensional Array Illustration


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

```
        0  1  2  3  4
    0 
    1 
    2        7
    3 
    4 
matrix[2][1] = 7;
```

```
     0    1    2
  0  1    2    3
  1  4    5    6
  2  7    8    9
  3  10   11   12

int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

matrix.length? 5

matrix[0].length? 5

array.length? 4

array[0].length? 3

Declaring, Creating, and Initializing Using Shorthand Notations

You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,
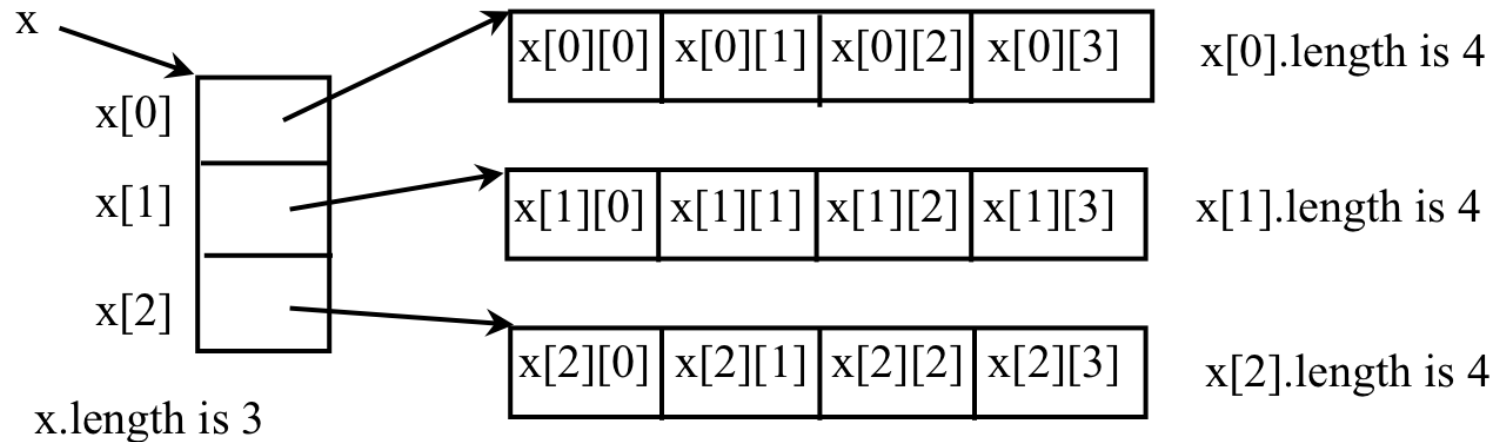
```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

Lengths of Two-dimensional Arrays

int[][] x = new int[3][4];

Lengths of Two-dimensional Arrays, cont.

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

array.length
array[0].length
array[1].length
array[2].length
array[3].length

array[4].length   ArrayIndexOutOfBoundsException
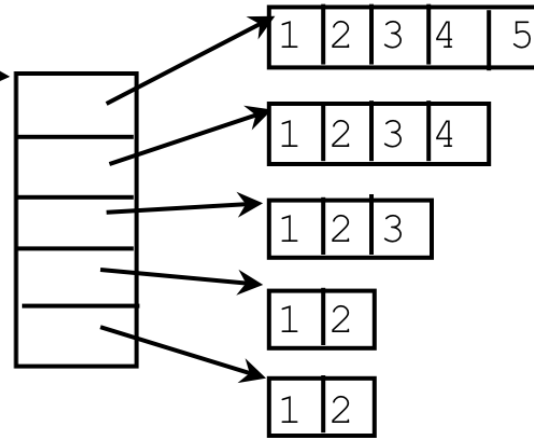
Ragged Arrays

Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1

Ragged Arrays, cont.

```
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
```

Multidimensional Arrays

Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.

The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3. For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

```
double[][][] scores = new double[10][5][2];
```

Example: Calculating Total Scores

- Objective: write a program that calculates the total score for students in a class. Suppose the scores are stored in a three-dimensional array named <u>scores</u>. The first index in <u>scores</u> refers to a student, the second refers to an exam, and the third refers to the part of the exam. Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part. So, <u>scores[i][j][0]</u> represents the score on the multiple-choice part for the i's student on the j's exam. Your program displays the total score for each student.

[TotalScore]