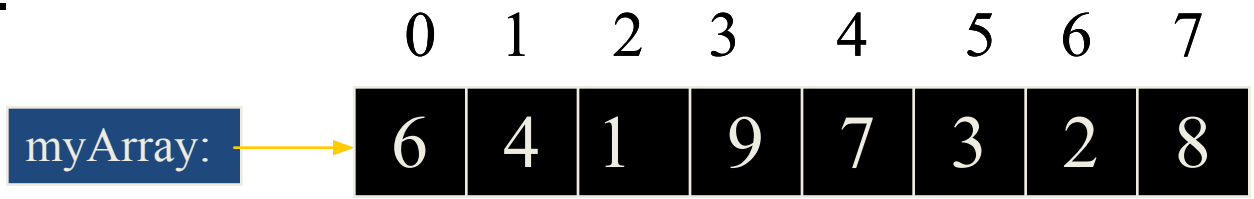
Single-Dimensional Arrays

Arrays

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

A single-dimensional array is one that stores data elements in one row.



Strings are arrays.

```
String Name = "John Smith";

String courseNumber;
courseNumber = new String ("CS 1301");
```

Declaring Array Variables

```
datatype[] arrayRefVar;
```

Example:

```
    double[] myList;     //vs. double myList;
    int[] yourList;
    boolean[] herList;
    char[] hisList;
```

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

Example:

```
  double myList[];
```

Creating Arrays

This is to allocate memory space for the array.

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

Example:

```
myList = new double[10];
yourList = new int[100];
herList = new boolean[20];
hisList = new char[500];
```

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

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

In one step you can declare and create:

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

The Length of an Array

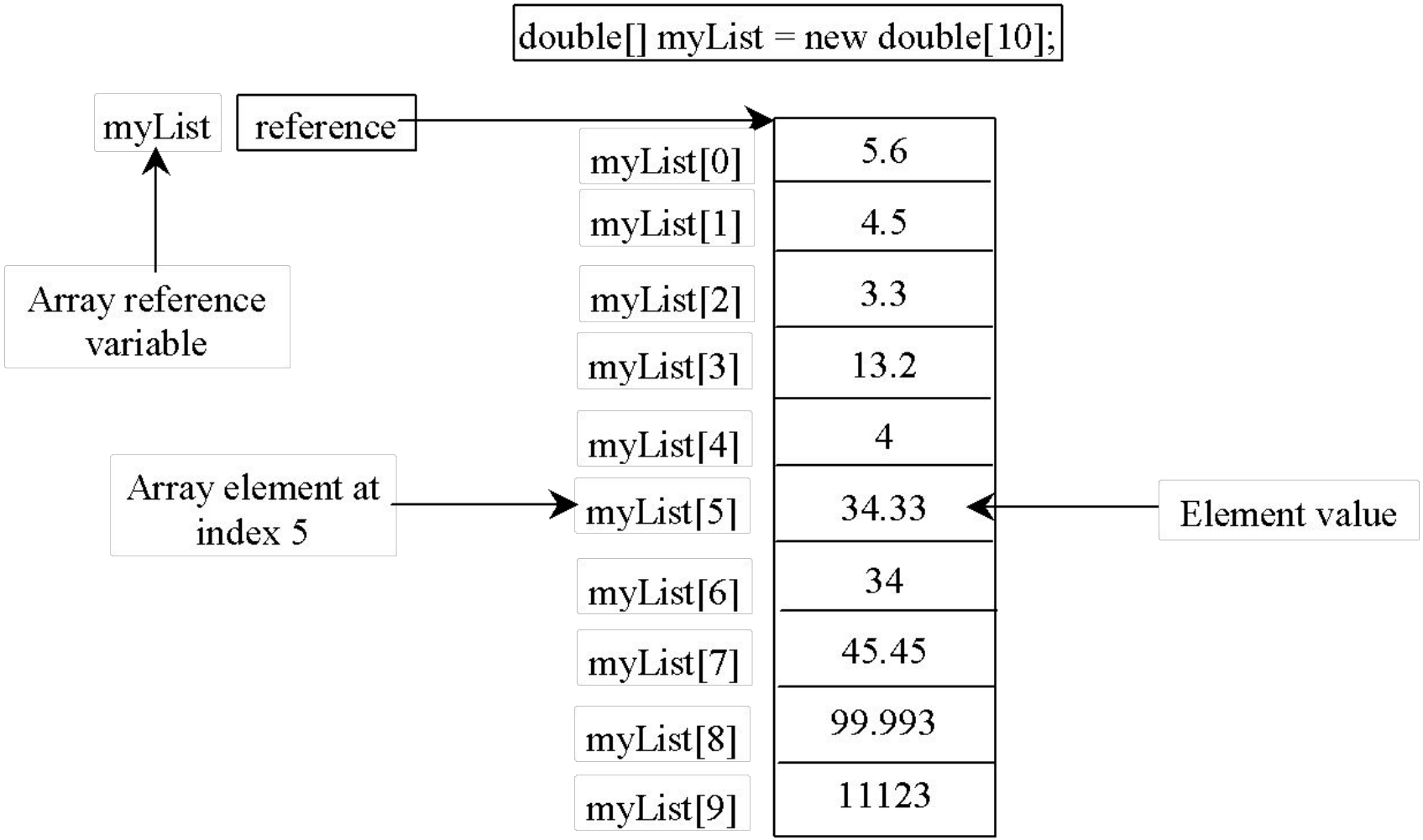
Once an array is created, **its size is fixed**. It cannot be changed. You can find its size using

```
arrayRefVar.length; (not length()as with strings)
```

For example,

```
int length = myList.length;     // returns 10
int length = yourList.length;   // returns 100;
int length = herList.length;    // returns 20;
int length = hisList.length;    // returns 500;
```

Array Representation



Default Values

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

- 0 for the numeric primitive types (byte, short, int, long)
- 0.0 for the numeric primitive data types (float, double)
- '\u0000' for char types (Null value)
- false for boolean type.

Indexed Variables

The array elements are accessed through the index.

The array indices start from **0** to **arrayRefVar.length-1**

Example:

```
myList = new double[5];
```

Array `myList` holds five double values
The indices are **0** to **4**.

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

```
arrayRefVar[index];
```

```
myList[0]= 25.7; //value in first element
double price = myList[0];
```

Using Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable is used.

Example:

The following code adds the values in the first and second positions in array `myList` and stores the result in the third position.

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

Example:

The following code prints out the content of array `myList`.

```
for (int i = 0; i <= myList.length-1; i++)
    System.out.println("Value in position " + i +
                       " = "  + myList[i]);
```

Array Initialization

Declaring, creating, initializing in one step:

```
 double[] myList      = {1.9, 2.9, 3.4, 3.5};
int[] numberGrades  = {70, 65, 87, 93, 90};
char[] letterGrades = {'C', 'D', 'B', 'A', 'A'};
boolean myFlags     = {true, false, true, false};
```

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;
```

Be Careful! This code gives an error. It must be in one statement.

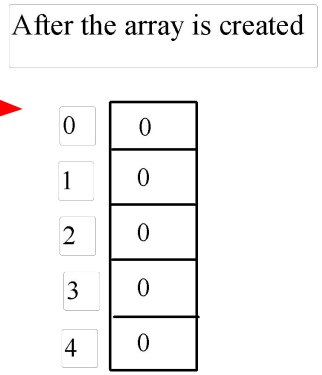
```
int[] numberGrades;
numberGrades = {70, 65, 87, 93, 90};
```

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(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];
  }
}
```



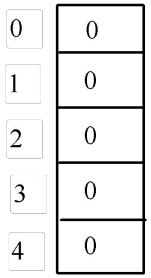
animation

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



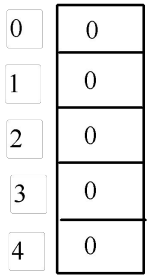
animation

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

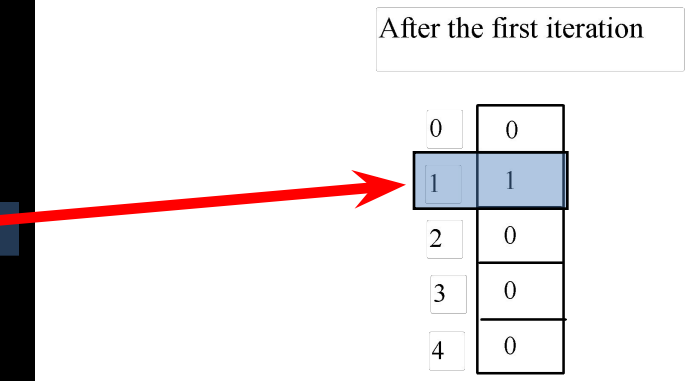


animation

Trace Program with Arrays

After this line is executed, value[1] is 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];
  }
}
```



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

i (= 2) 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 first iteration

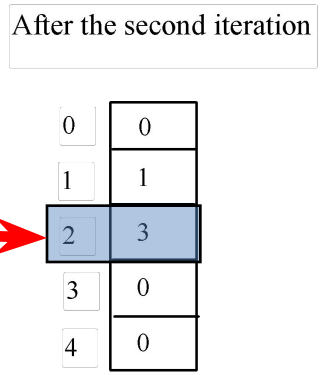
0	0
1	1
2	0
3	0
4	0

animation

Trace Program with Arrays

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

```
public class Test {
  public static void main(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];
  }
}
```



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

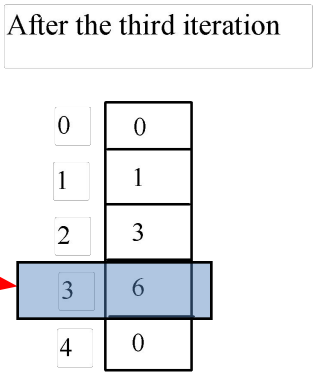
0	0
1	1
2	3
3	0
4	0

animation

Trace Program with Arrays

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

```
public class Test {
  public static void main(String[] 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];
  }
}
```



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

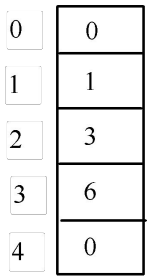
animation

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

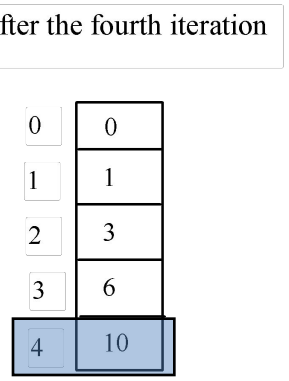


animation

Trace Program with Arrays

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

```
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];
  }
}
```



animation

Trace Program with Arrays

After i++, i becomes 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 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

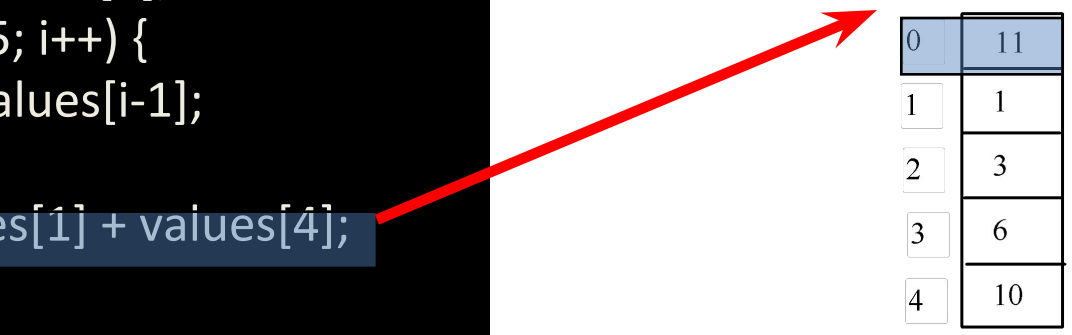
0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

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

```
public class Test {
  public static void 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];
  }
}
```



0	11
1	1
2	3
3	6
4	10

Processing Arrays

Some common examples:

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 smallest index of the largest element)
7. (*Random shuffling*)
8. (*Shifting elements*)

Initializing arrays with input values

```
...  //Some code

double[] myList = new double[5];

java.util.Scanner input = new java.util.Scanner(System.in);


System.out.print("Enter " + myList.length + " double values: ");

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

...  //other code
```

Initializing arrays with random values

```
...  //Some code

double[] myList = new double[5];

...  //initialize the array

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

...  //other code
```

Printing arrays

```
...  //Some code

double[] myList = new double[5];

...  //initialize the array

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

...  //other code
```

Summing all elements

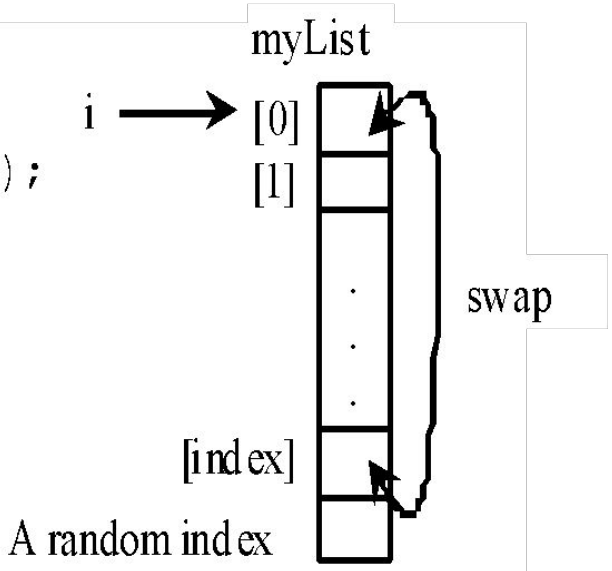
```
...  //Some code

double total = 0;

double[] myList = new double[5];

...  //initialize the array

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

...  //other code
```

Finding the largest element

```
...  //Some code

double[] myList = new double[5];

double max = myList[0];

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

...  //other code
```

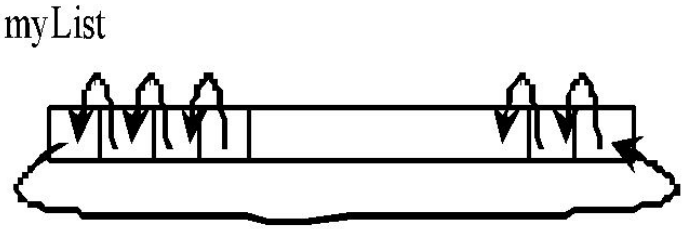
Random shuffling

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

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

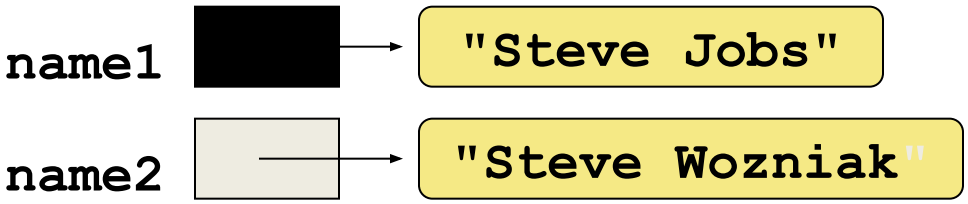
```
For (elementType value: arrayRefVar) {
   // Process the value
}
```

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.

Copying Arrays

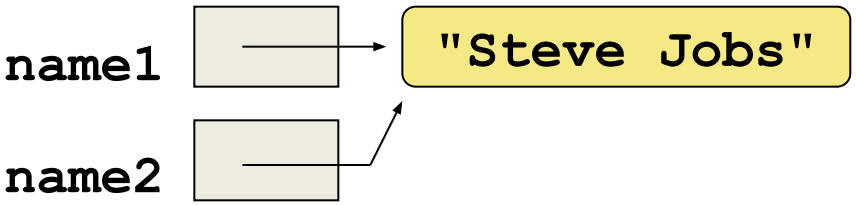
For object references, an assignment statement copies the memory address NOT the array content:

Before:




```
name2 = name1;
```

After:



Copying Arrays

Use 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);
```

This will copy every element in `sourceArray` to `targetArray`.

Must create `targetArray` first.

```
System.arraycopy(sourceArray, 10, targetArray, 10, length);
```

This will copy every element in `sourceArray` to `targetArray` starting at position 10.

Must create `targetArray` first.

Passing Arrays to Methods

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

Invoke the method:

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

Invoke the method again:

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

Anonymous (nameless) array

Anonymous Array

The statement

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

creates un-named array and pass it to method printArray().

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

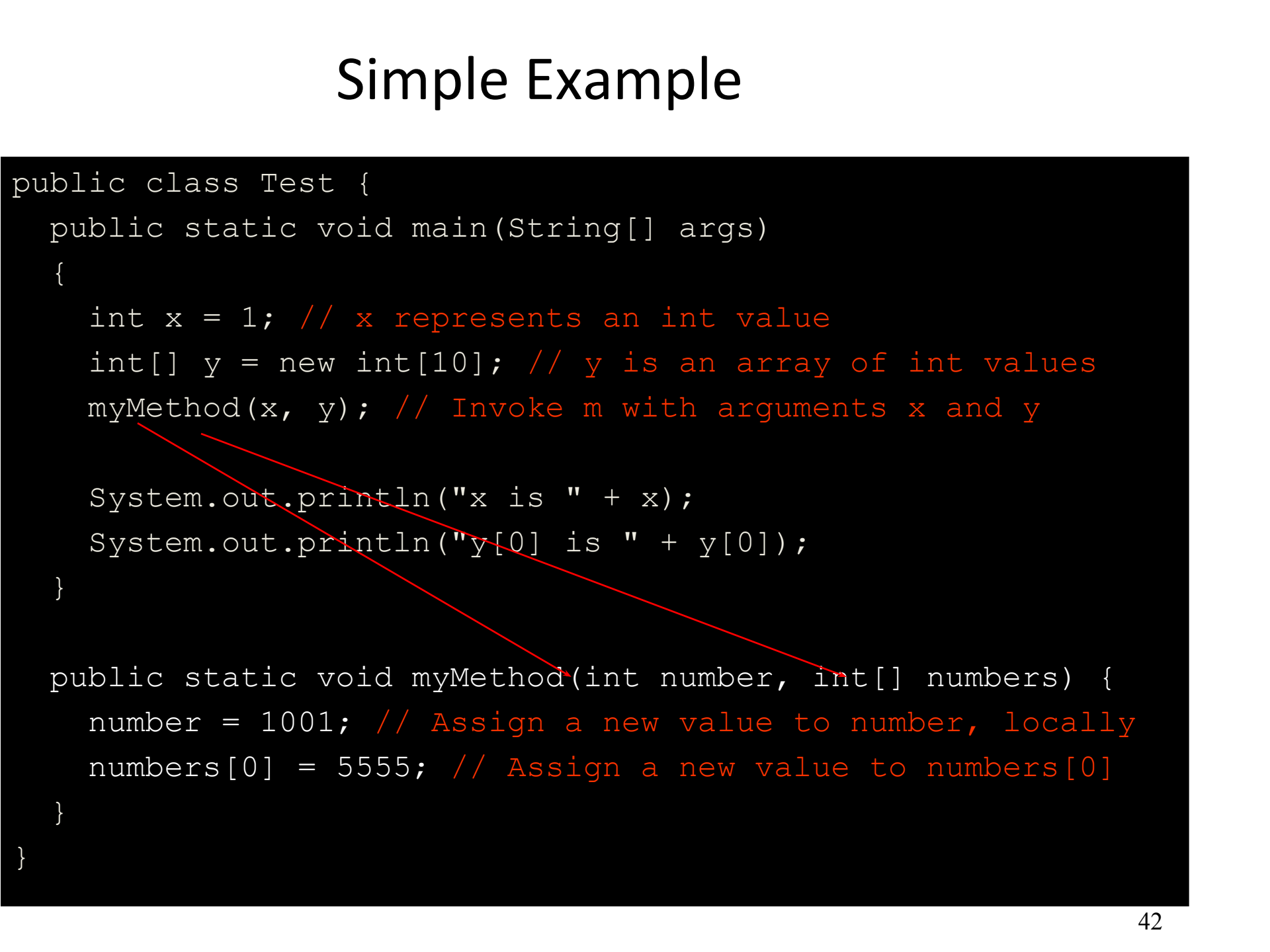
Pass By Value - Revisited

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.

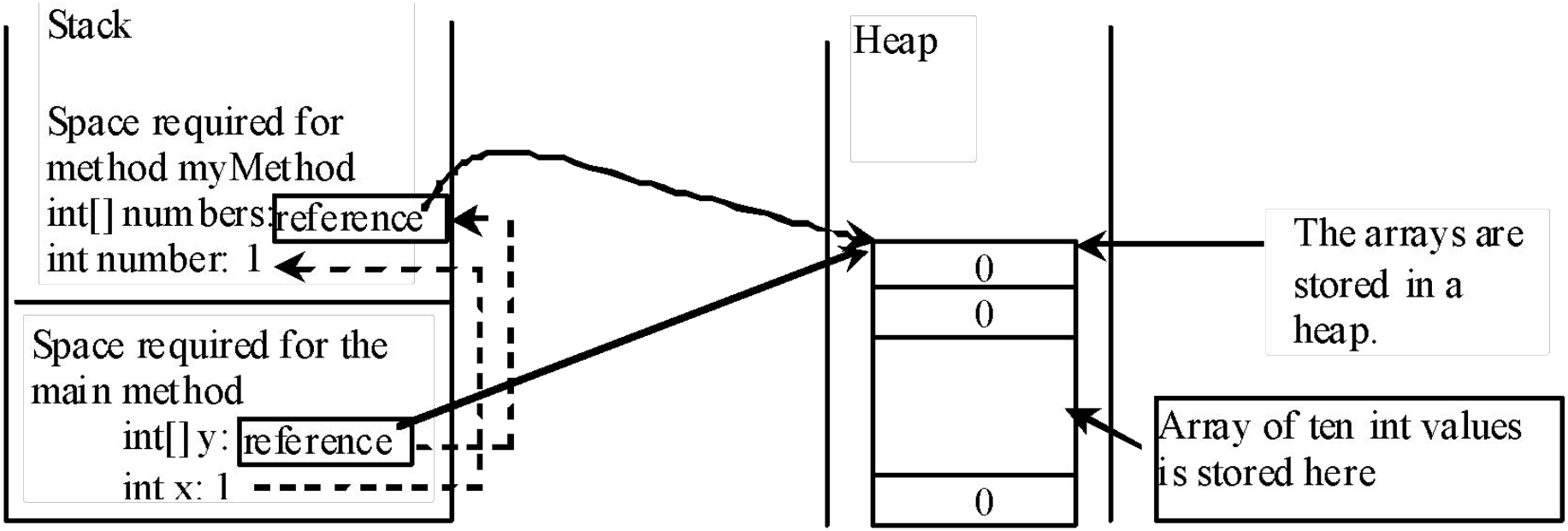
- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method **does not affect** the value of the variable outside the method.
- For a parameter of an array type, the value of the parameter *contains a reference to an array; this 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

```
public class Test {
  public static void main(String[] args)
  {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y is an array of int values
    myMethod(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 myMethod(int number, int[] numbers) {
    number = 1001; // Assign a new value to number, locally
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

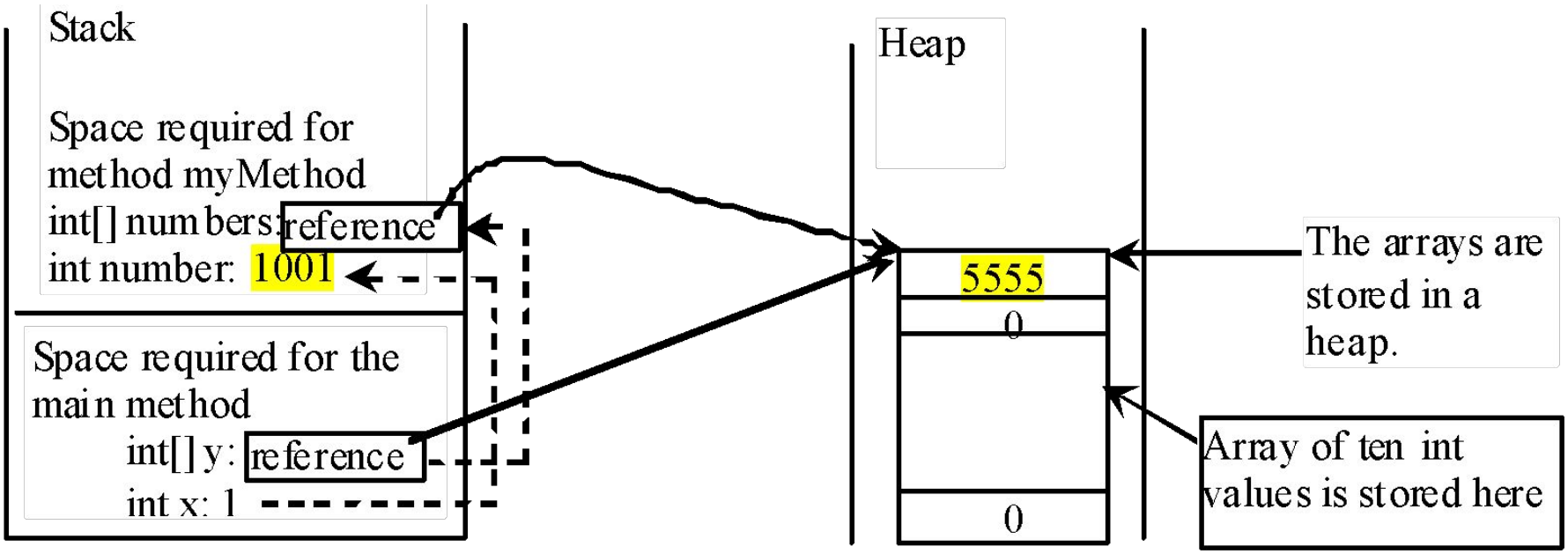


Call (Run-Time) Stack

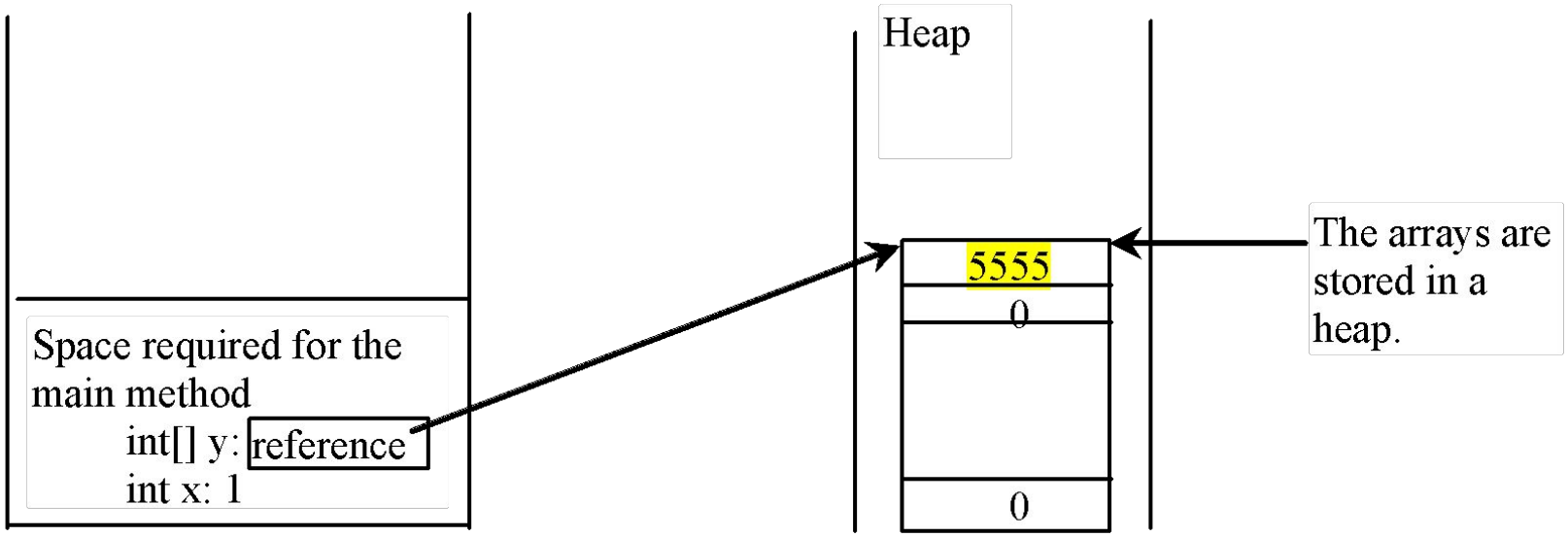


When invoking myMethod(x, y), the values of x and y are passed to number and numbers. Since y contains the **reference value** to the array (memory address) , numbers now contains the same reference value to the same array.

Call (Run-Time) Stack



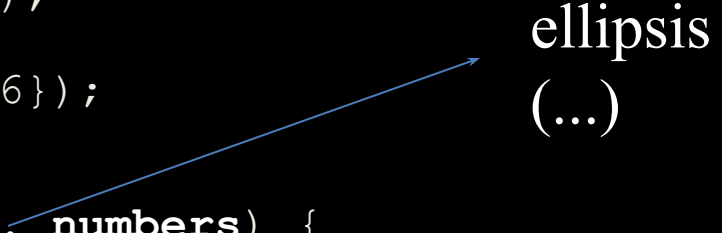
Call (Run-Time) Stack



Variable-Length Parameter List

Java allows a method to take variables number of parameters of the same type. The parameter list is treated as an array.

```
public class VarArgsDemo {
   public static void main (String[] args) {
      printMax(50, 29, 19, 2, 98, 16);
      printMax(150, 300, 275);
      printMax(new int[] {1,2,3,4,5,6});
   }

   public static void printMax(int... numbers) {
      if (numbers.length == 0) {
         System.out.println("No arguments passed! ");
         return; // to exit the method, not to return a value
      }
      int result = numbers[0];
      for (int i=1; i<numbers.length; i++)
         if (numbers[i] > result)
            result = numbers[i];
      System.out.println("The max value is " + result);
   }
}
```



Searching Arrays

Searching is the process of looking for a specific element in an array. The element may be found or may not.

How? Two commonly used methods are:

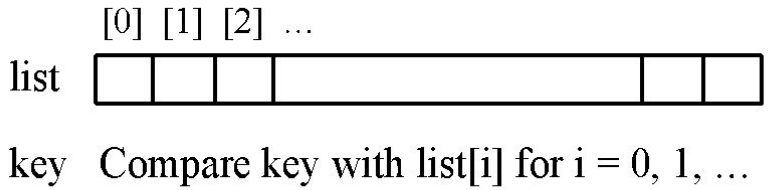
- *linear search:* search all elements in sequence from first to last.

- *binary search:* search an ordered array taking into consideration one-half of the array in each stop.

Linear Search

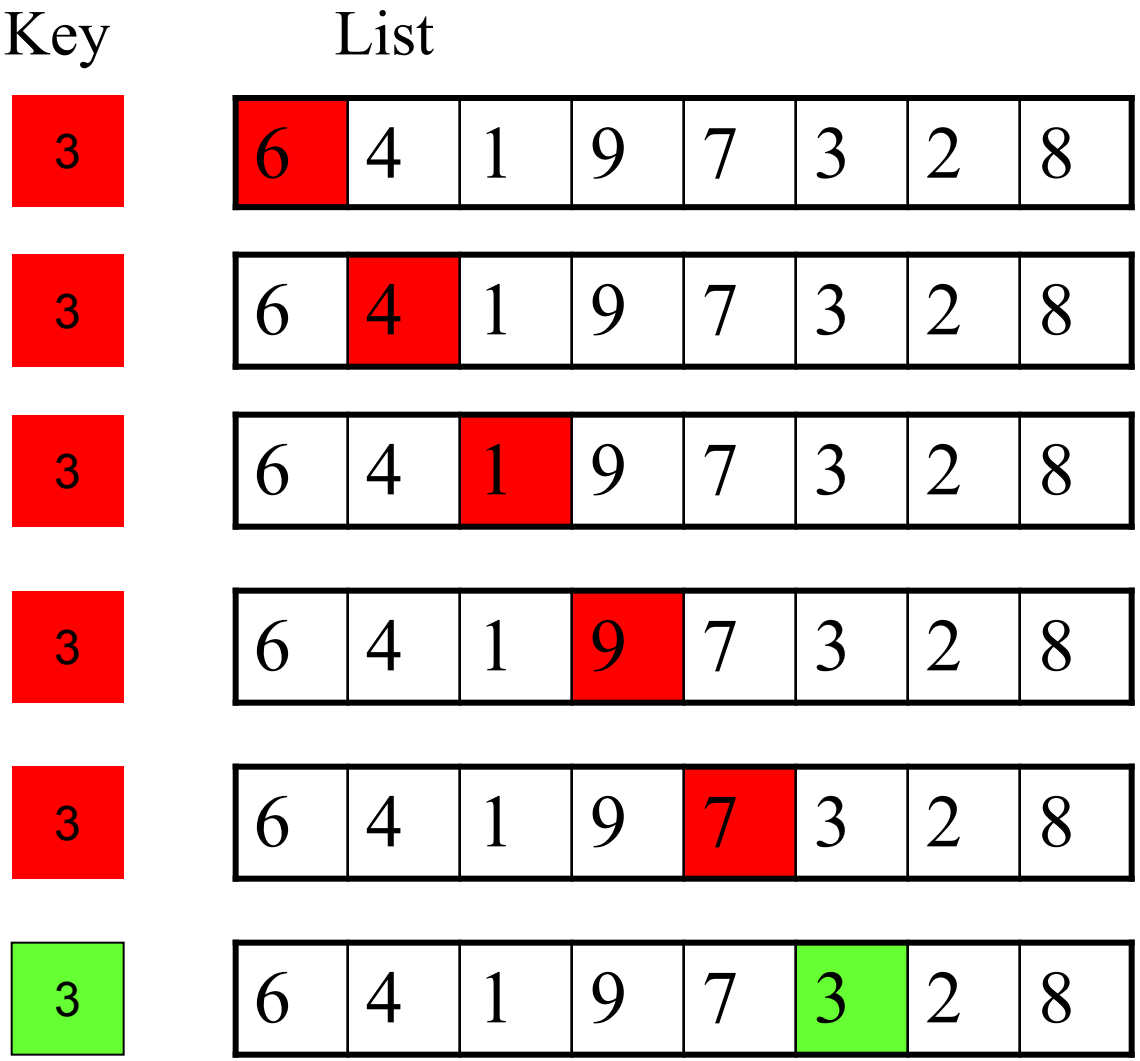
The linear search approach compares the key element, key or target, *sequentially* with each element in the array list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index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

```
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;
  }
}
```



animation

Linear Search Animation



Binary Search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

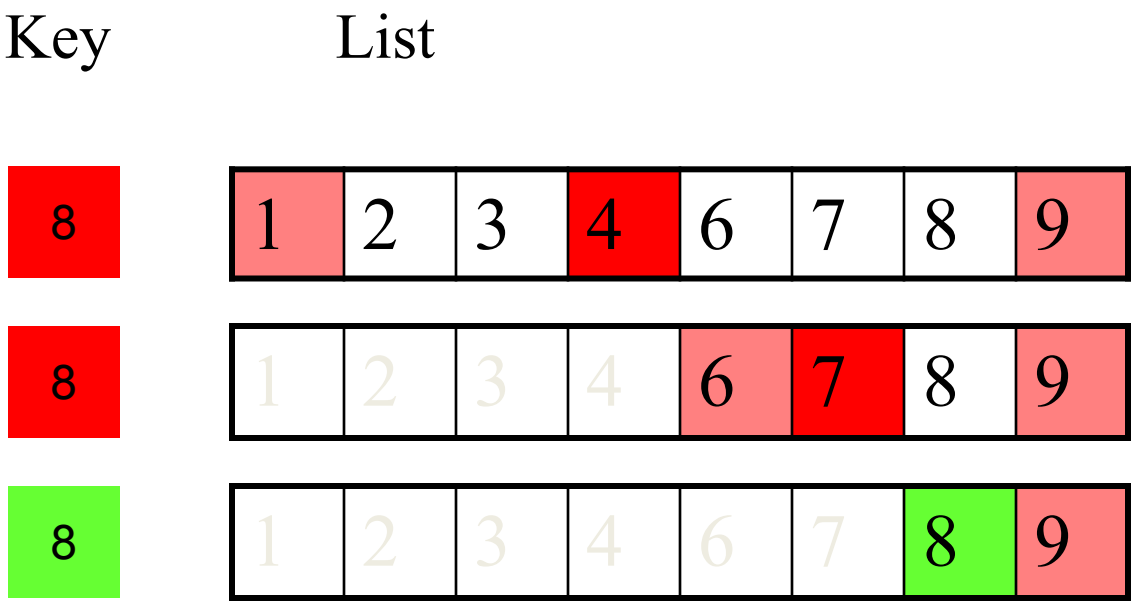
e.g.: 2 4 7 10 11 45 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the array.

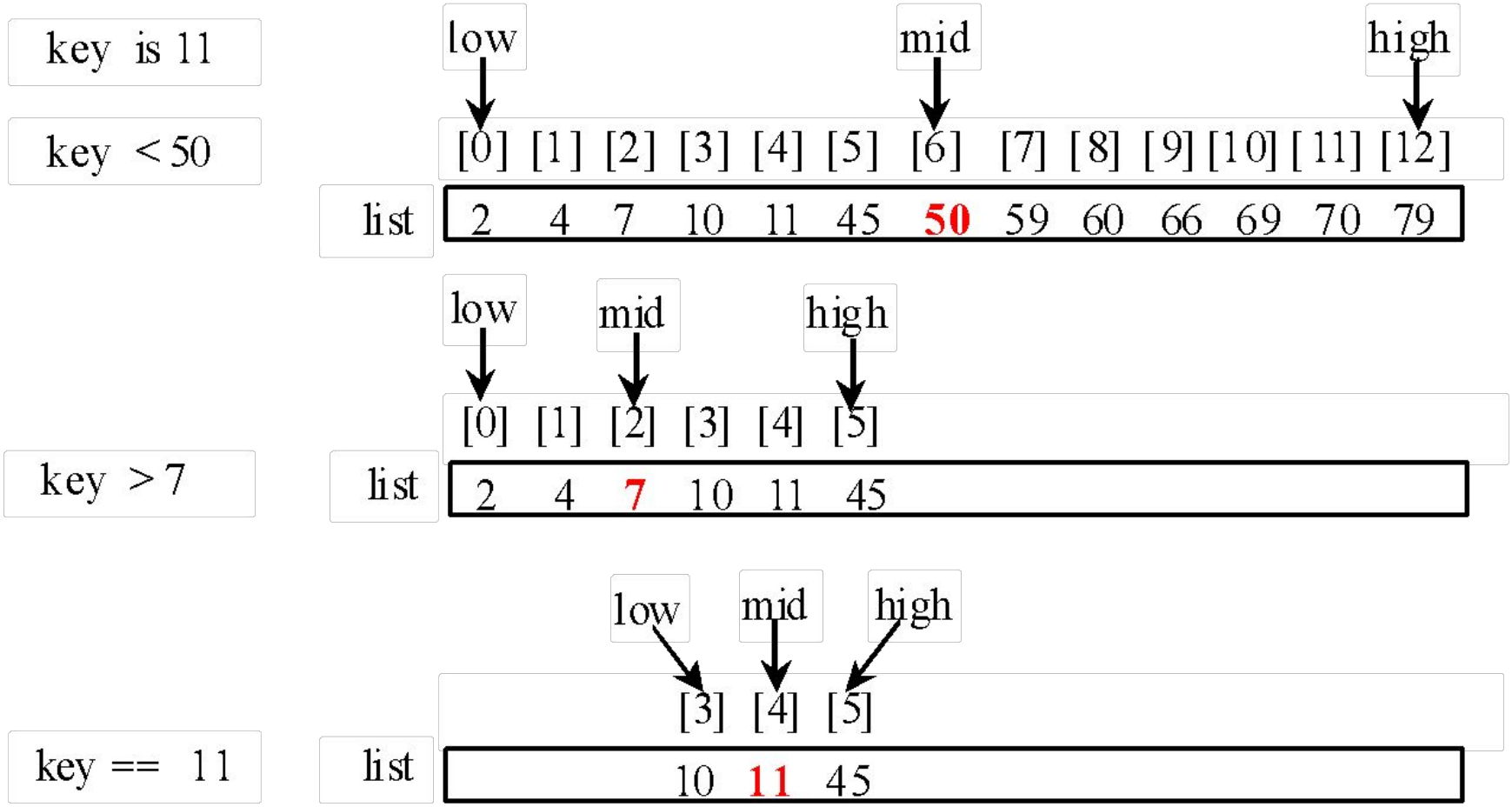
- If the key is less than the middle element, you only need to search the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the array.

animation

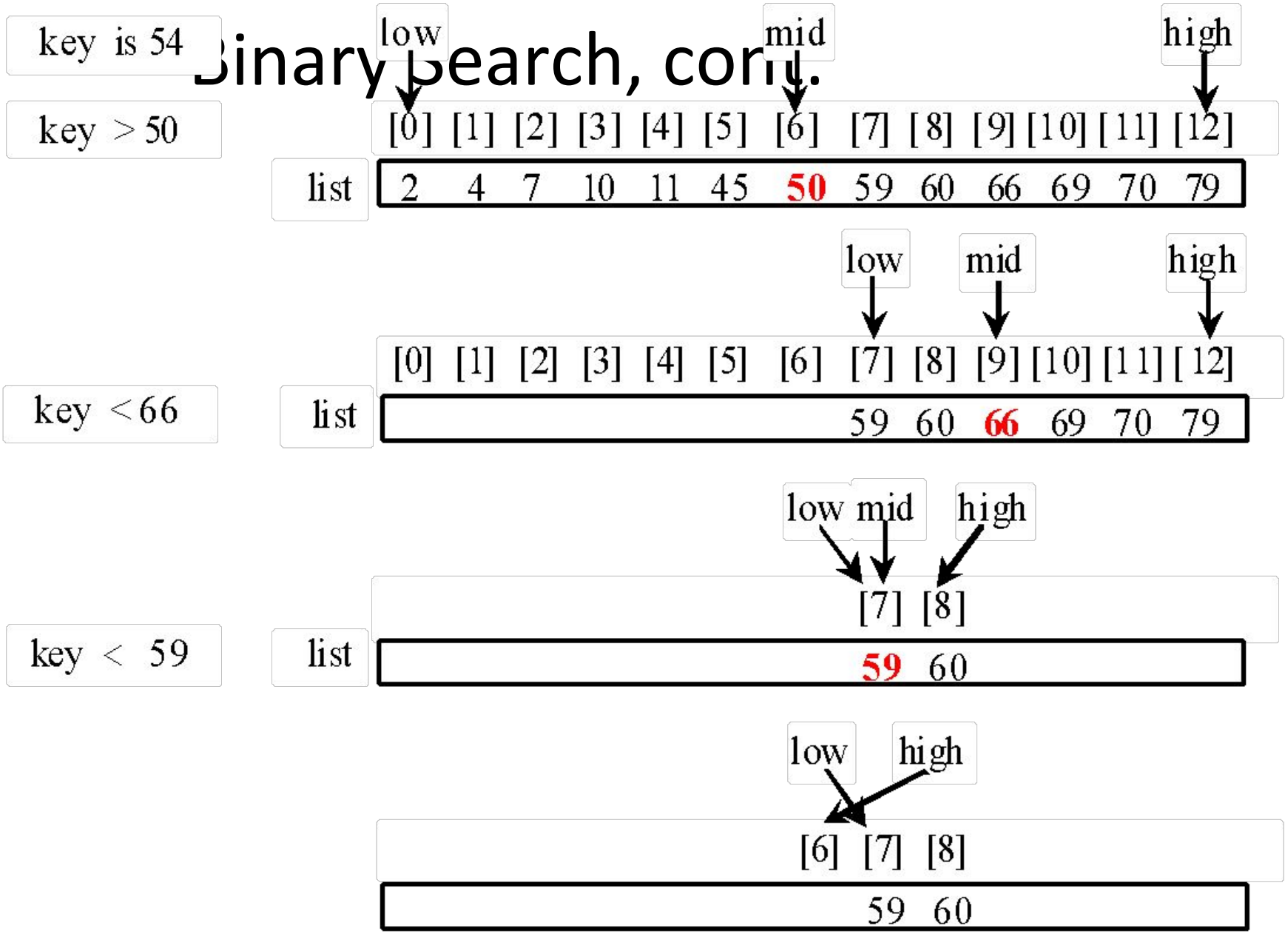
Binary Search



Binary Search, cont.



Binary Search, cont.



The Solution

```
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;  // middle element
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;  // target is found
    else
      low = mid + 1;
  }

  return -1 - low;  // target not found!
}
```

Class **Arrays**

Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the **java.util.Arrays** class. For example:

```
import java.util.*;
int[] list = {2,4,7,10,11,45,50,59,60,66,69,70,79};
System.out.println("Index of value 11 is " +
                   Arrays.binarySearch(list, 11));


char[] chars = {'a','c','g','x','y','z'};
System.out.println("Index of letter t is " +
                   Arrays.binarySearch(chars, 't'));
```

Note: This call returns –4 (insertion point is 3, so return is -3-1)

For the binarySearch method to work, the array must be pre-sorted in increasing order. See section 7.12, page 272, for other methods in class Arrays (**`sort`**, **`equals`**, **`fill`**, **`toString`**)

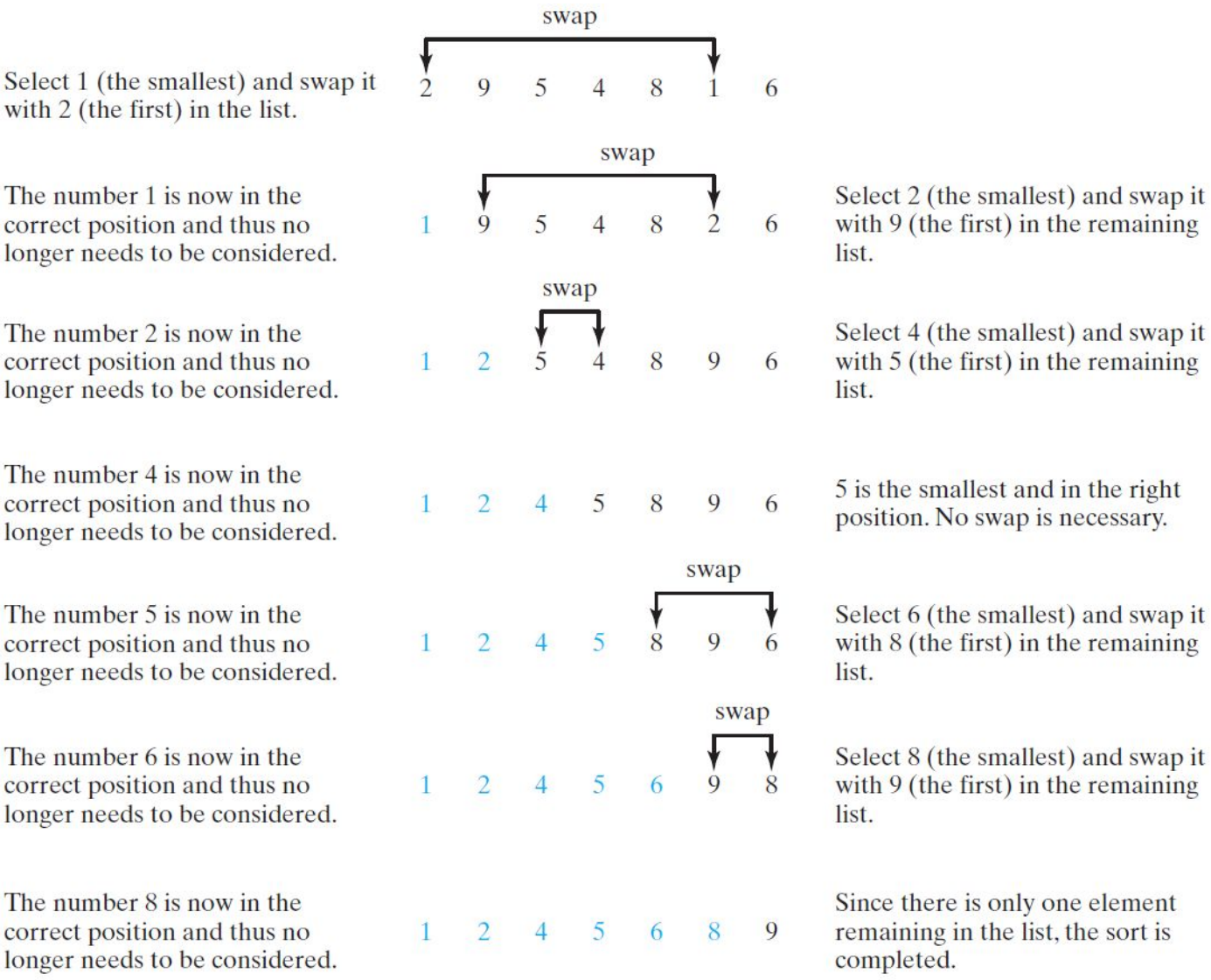
Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting.

This section introduces algorithm *selection sort*.

Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



The Algorithm

The Code

```
// Find the minimum in the list[i..list.length-1]




// Swap list[i] with list[currentMinIndex] if necessary;
```

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.

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

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

Arrays.toString(list) Method

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

```
import java.util.*;
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
Arrays.sort(chars);  //sort the list
//print the list
for (int i = 0; i < chars.length; i++)
     System.out.print(chars[i]+ " ");
System.out.println();
//convert to string
String myString = Arrays.toString(chars);
System.out.println (myString);
```

Output:

```
4 A D F P a
[4, A, D, F, P, a]
```

Passing Arguments to Main Method

You can call a regular method by passing actual parameters. Can you pass arguments to main? Of course, yes. 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]);
  }
}
```

End of Chapter 7