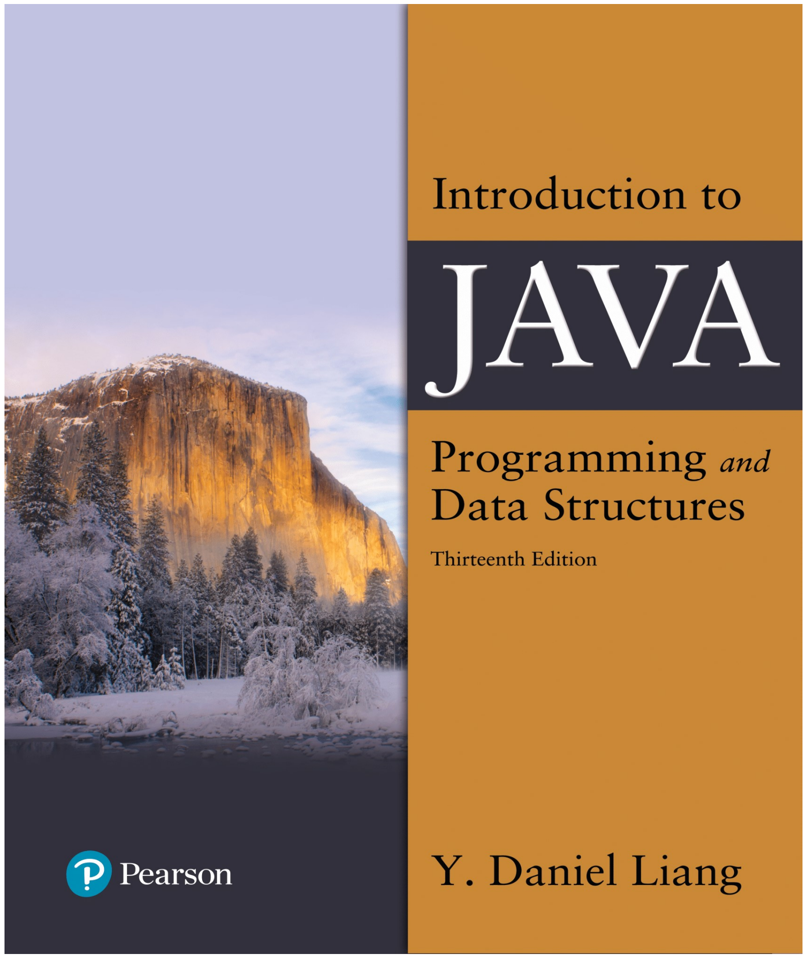
Introduction to Java Programming and Data Structures

Thirteenth Edition



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 (1 of 2)

7.1 To describe why arrays are necessary in programming (§7.1).

7.2 To declare array reference variables and create arrays (§§7.2.1–7.2.2).

7.3 To obtain array size using **arrayRefVar.length** and know default values in an array (§7.2.3).

7.4 To access array elements using indexes (§7.2.4).

7.5 To declare, create, and initialize an array using an array initializer (§7.2.5).

7.6 To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements, random shuffling, and shifting elements) (§7.2.6).

7.7 To simplify programming using the foreach loops (§7.2.7).

Objectives (2 of 2)

7.8 To apply arrays in application development (**AnalyzeNumbers**, **DeckOfCards**) (§§7.3–7.4).

7.9 To copy contents from one array to another (§7.5).

7.10 To develop and invoke methods with array arguments and return values (§§7.6–7.8).

7.11 To define a method with a variable-length argument list (§7.9).

7.12 To search elements using the linear (§7.10.1) or binary (§7.10.2) search algorithm.

7.13 To sort an array using the selection sort approach (§7.11).

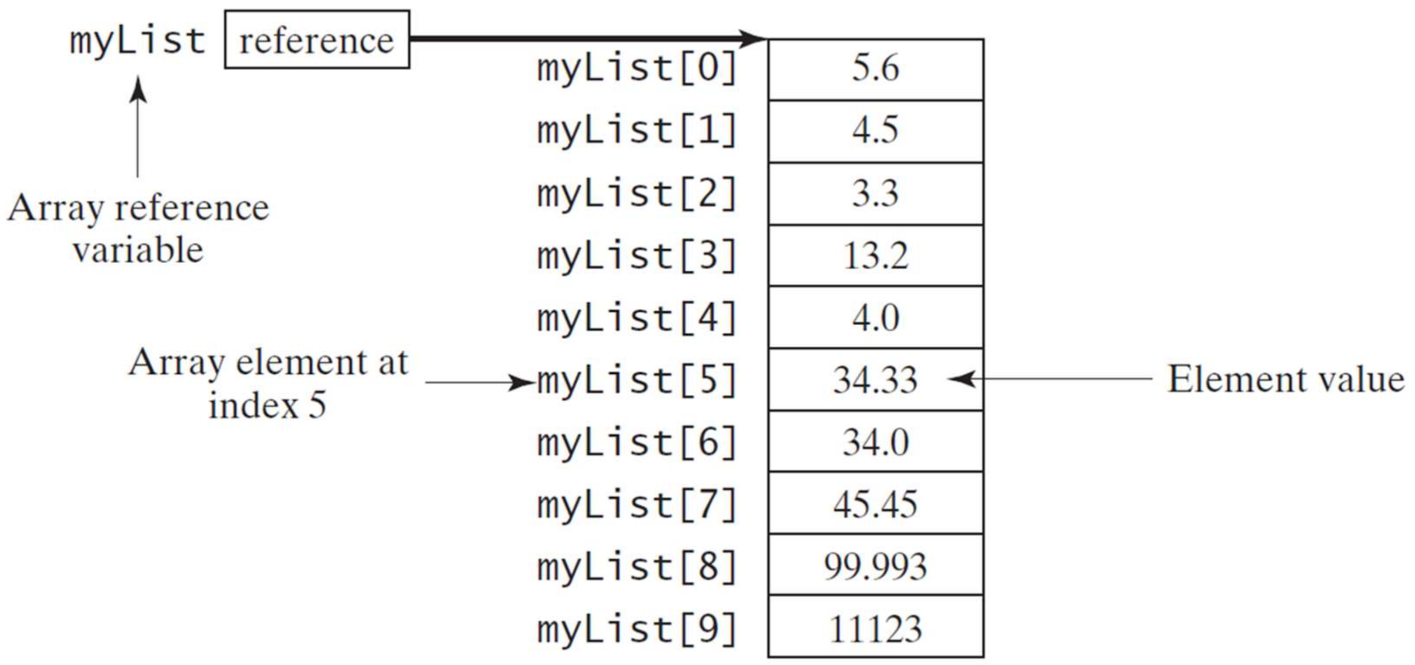
7.14 To use the methods in the **java.util.Arrays** class (§7.12).

7.15 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.

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



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];
```

Array Initializers

- Declaring, creating, initializing in one step:

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

This shorthand syntax must be contained within a single statement.

The following is wrong:

```
double[] myList;

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

Declaring, Creating, Initializing Using the Shorthand Notation

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

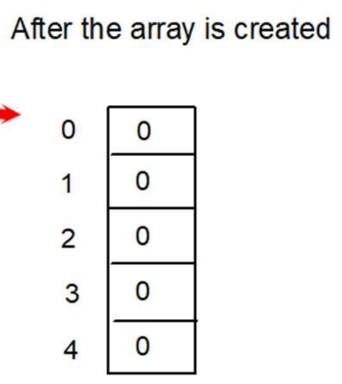
This shorthand notation is equivalent to the following statements:

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

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

Trace Program with Arrays (1 of 16)

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

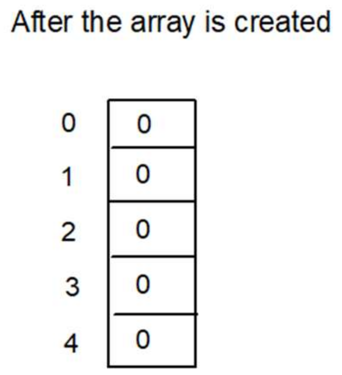
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



Trace Program with Arrays (2 of 16)

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];
  }
}
```

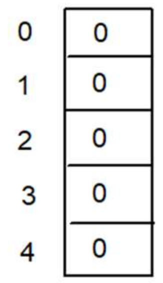


Trace Program with Arrays (3 of 16)

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];
  }
}
```

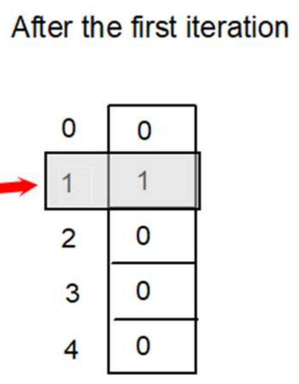
After the array is created



Trace Program with Arrays (4 of 16)

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

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

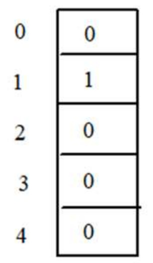


Trace Program with Arrays (5 of 16)

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];
  }
}
```

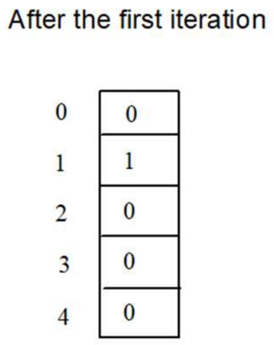
After the first iteration



Trace Program with Arrays (6 of 16)

```
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

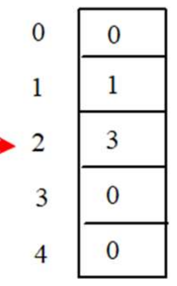


Trace Program with Arrays (7 of 16)

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

```
public class Test {
  public static void main(St          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

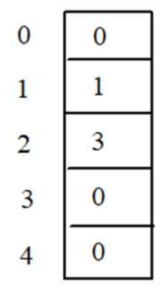


Trace Program with Arrays (8 of 16)

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



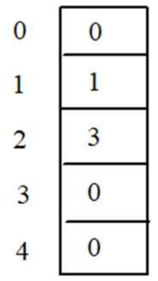
0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays (9 of 16)

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];
  }
}
```

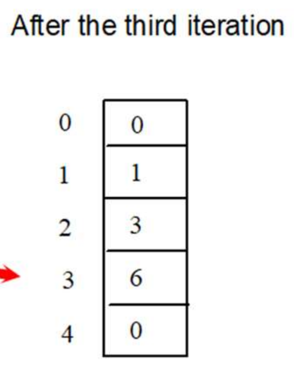
After the second iteration



Trace Program with Arrays (10 of 16)

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];
  }
}
```

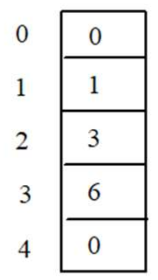


Trace Program with Arrays (11 of 16)

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];
  }
}
```

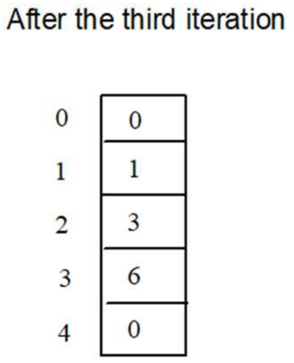
After the third iteration



Trace Program with Arrays (12 of 16)

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];
  }
}
```

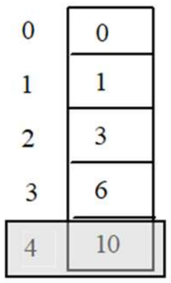


Trace Program with Arrays (13 of 16)

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

```
public class Test {
  public static void main(St [] 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

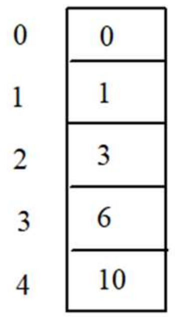


Trace Program with Arrays (14 of 16)

After i++, i becomes 5

```
public class Test {
  public static void main(String[] a          ) {
    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



Trace Program with Arrays (15 of 16)

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];
  }
}
```

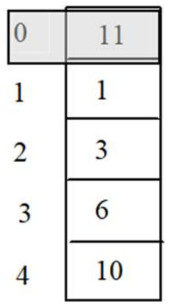
After the fourth iteration

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays (16 of 16)

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];
  }
}
```



Processing Arrays

See the examples in the text.

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**) – shift-left & shift-right

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

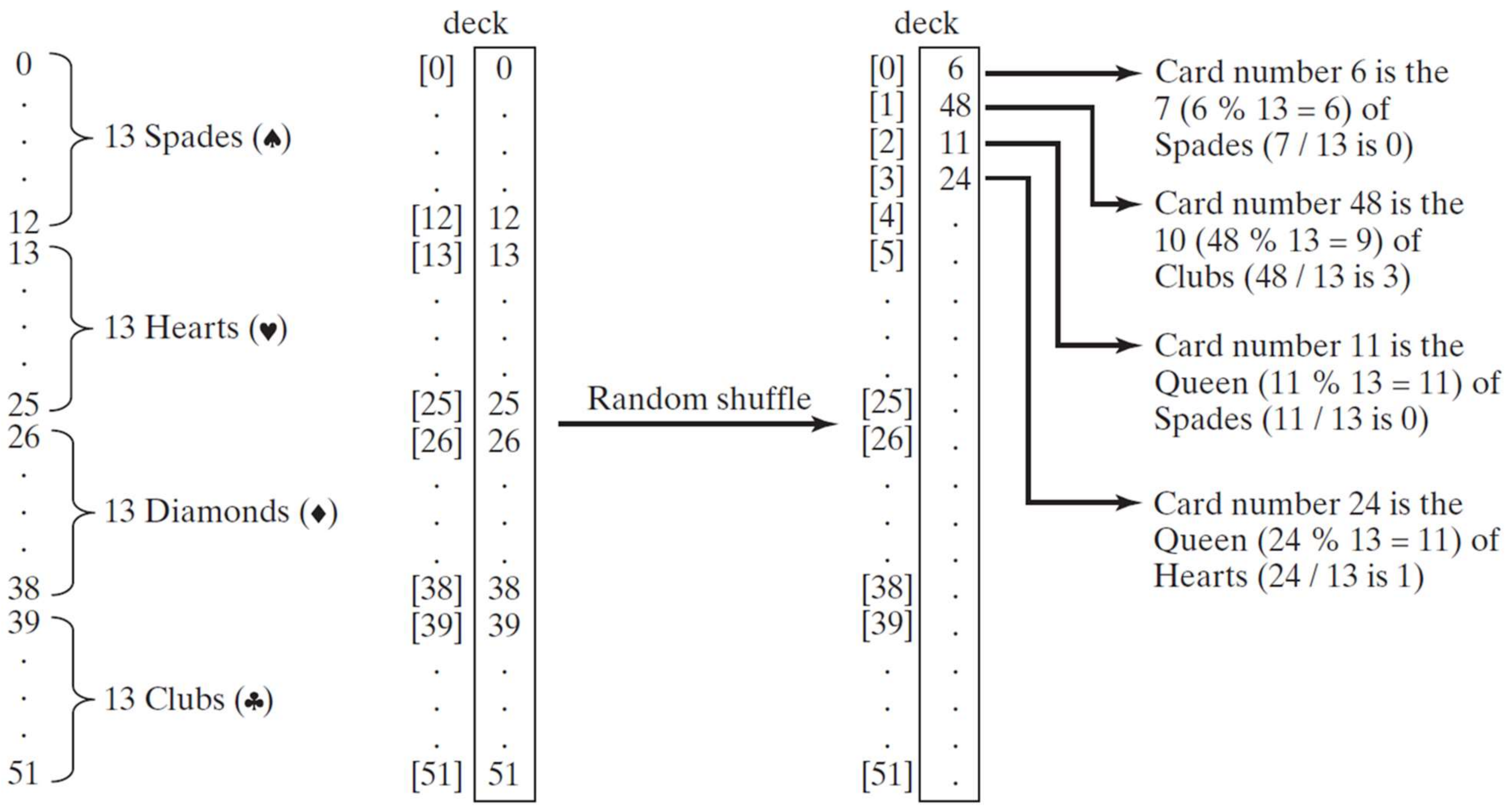
Problem: Deck of Cards (1 of 4)

The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows:

```
int[] deck = new int[52];
// Initialize cards
for (int i = 0; i < deck.length; i++)
  deck[i] = i;
```

DeckOfCards

Problem: Deck of Cards (2 of 4)


0
12
13 Spades (♠)
13
25
13 Hearts (♥)
26
38
13 Diamonds (♦)
39
51
13 Clubs (♣)
deck
[0] 0
[12] 12
[13] 13
[25] 25
[26] 26
[38] 38
[39] 39
[51] 51
Random shuffle
deck
[0] 6
[1] 48
[2] 11
[3] 24
[4]
[5]
[25]
[26]
[38]
[39]
[51]
Card number 6 is the 7 (6 % 13 = 6) of Spades (7 / 13 is 0)
Card number 48 is the 10 (48 % 13 = 9) of Clubs (48 / 13 is 3)
Card number 11 is the Queen (11 % 13 = 11) of Spades (11 / 13 is 0)
Card number 24 is the Queen (24 % 13 = 11) of Hearts (24 / 13 is 1)

Problem: Deck of Cards (3 of 4)

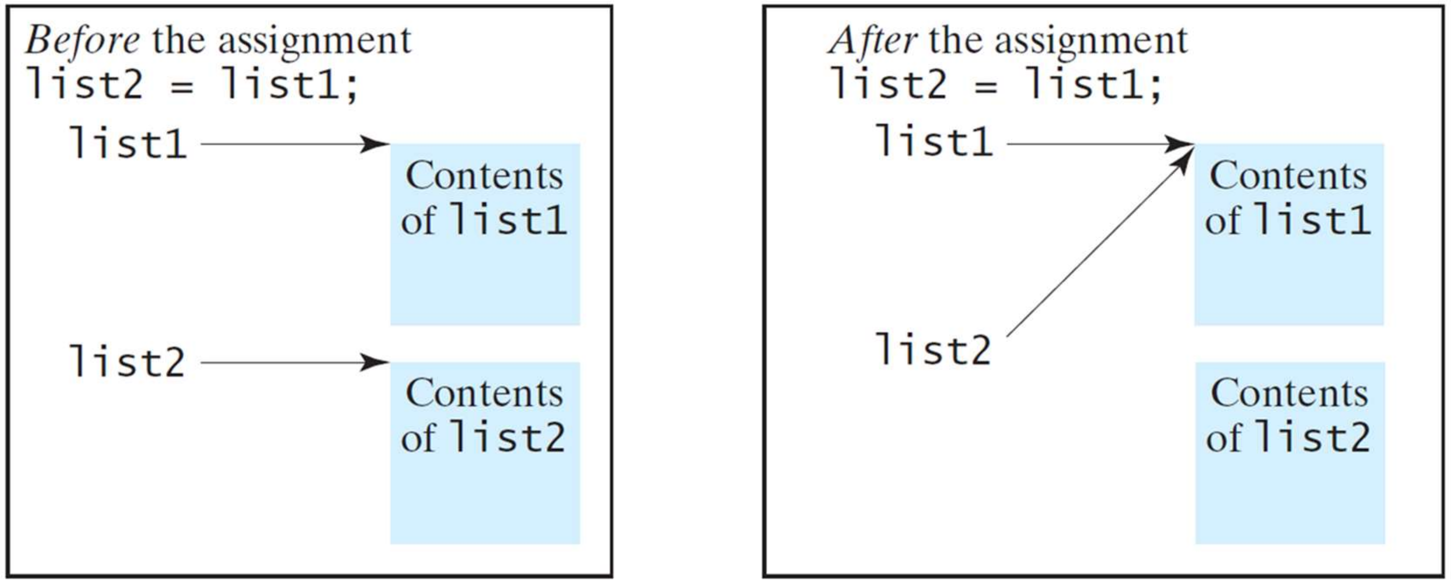
$$\text{cardNumber / 13} = \left\{\begin{array}{lcl} 0 & \longrightarrow & \text{Spades} \\ 1 & \longrightarrow & \text{Hearts} \\ 2 & \longrightarrow & \text{Diamonds} \\ 3 & \longrightarrow & \text{Clubs} \end{array}\right. \qquad \text{cardNumber \% 13} = \left\{\begin{array}{lcl} 0 & \longrightarrow & \text{Ace} \\ 1 & \longrightarrow & 2 \\ . & & \\ . & & \\ 10 & \longrightarrow & \text{Jack} \\ 11 & \longrightarrow & \text{Queen} \\ 12 & \longrightarrow & \text{King} \end{array}\right.$$

DeckOfCards

Copying Arrays (1 of 2)

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 (2 of 2)

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 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
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

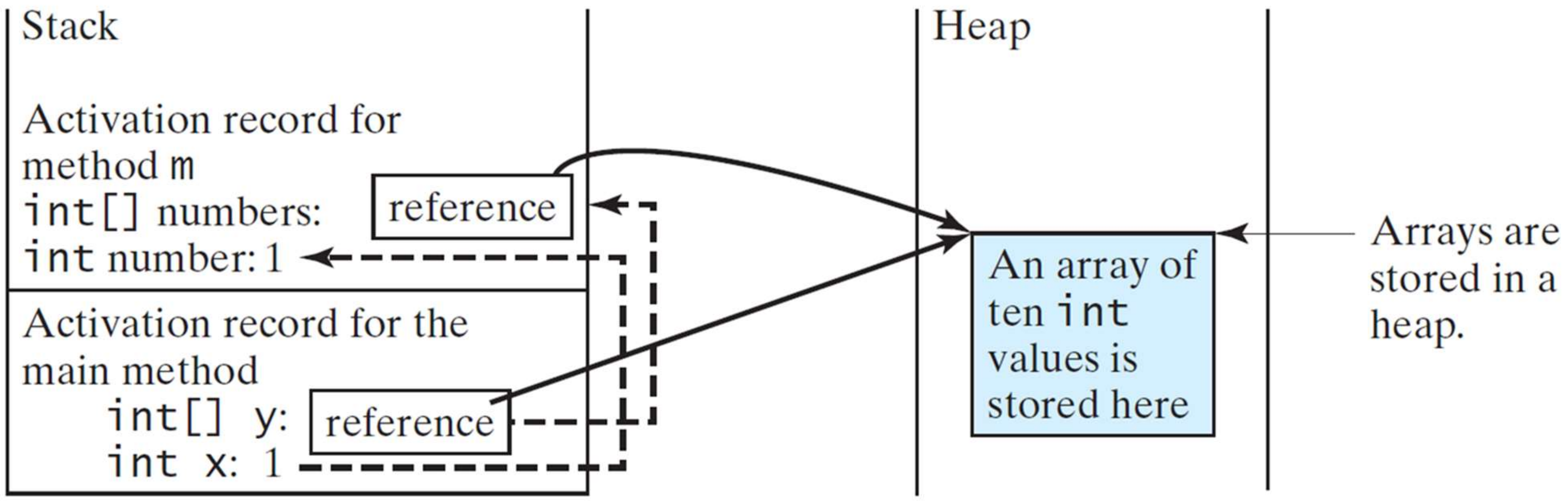
Java uses **pass by value** to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

- 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

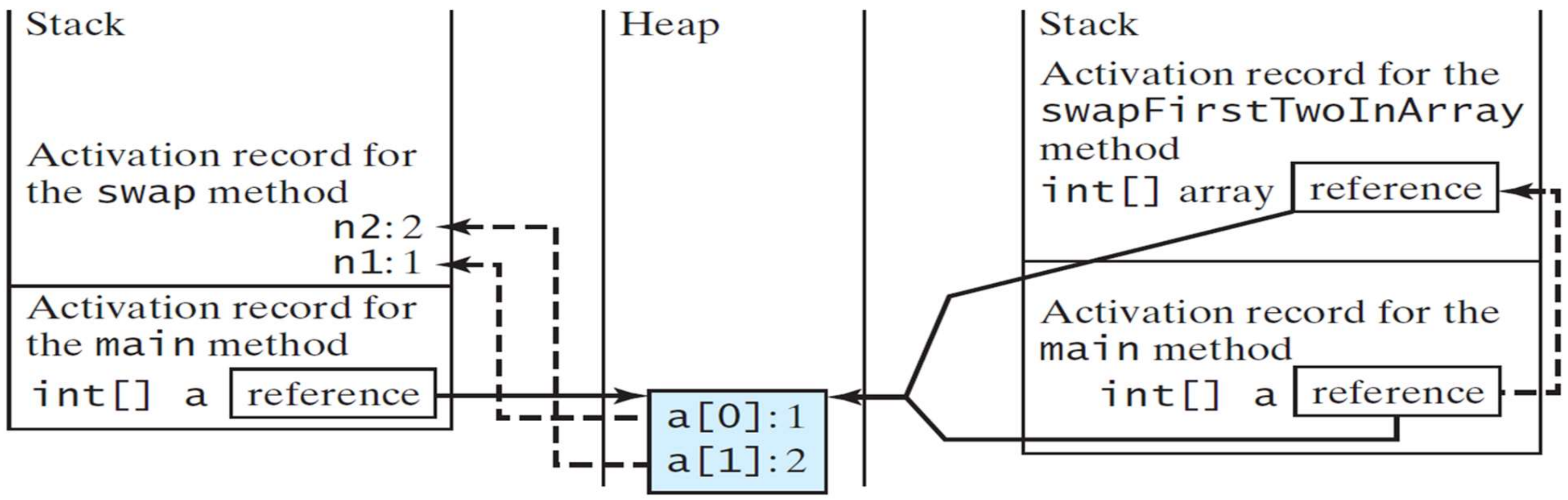
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    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 (1 of 2)


Stack
Activation record for method m
int[] numbers:
reference
int number: 1
Activation record for the main method
int[] y:
reference
int x: 1
Heap
An array of ten int values is stored here
Arrays are stored in a heap.

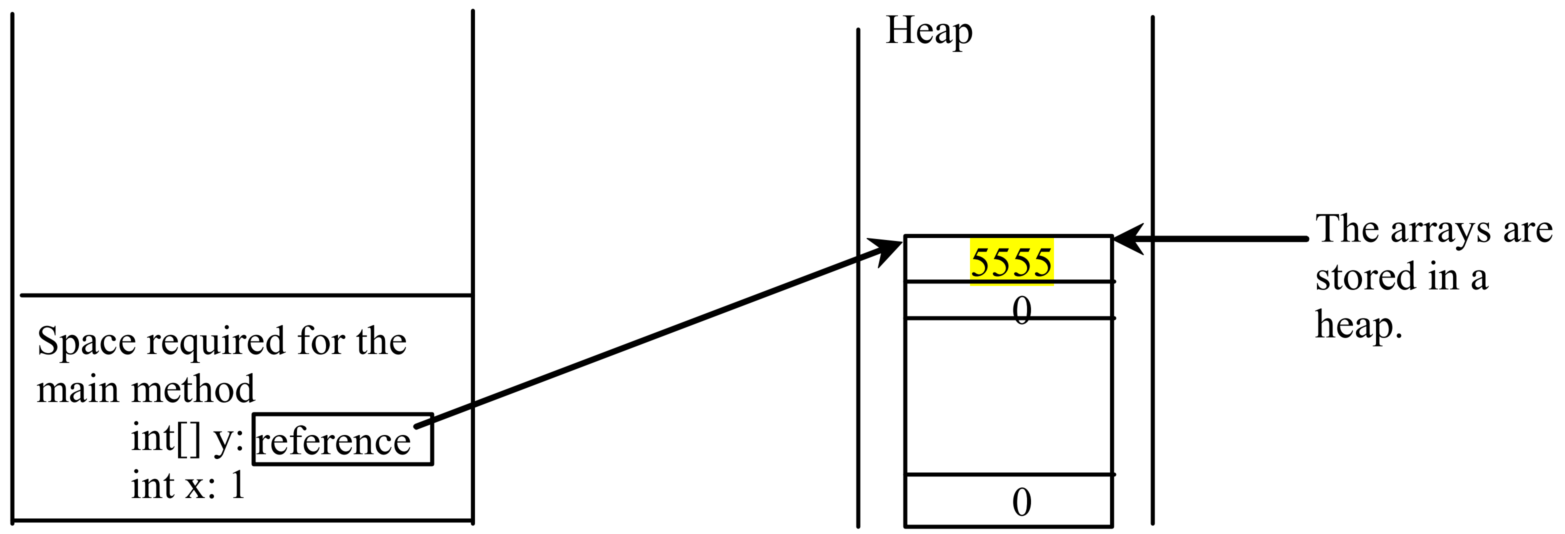
When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Call Stack (2 of 2)


Stack
Activation record for the swap method
n2: 2
n1: 1
Activation record for the main method
int[] a reference
Heap
a[0]: 1
a[1]: 2
Stack
Activation record for the swapFirstTwoInArray method
int[] array reference
Activation record for the main method
int[] a reference

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


Space required for the main method
int[] y: reference
int x: 1
Heap
5555
0
0
The arrays are stored in a heap.

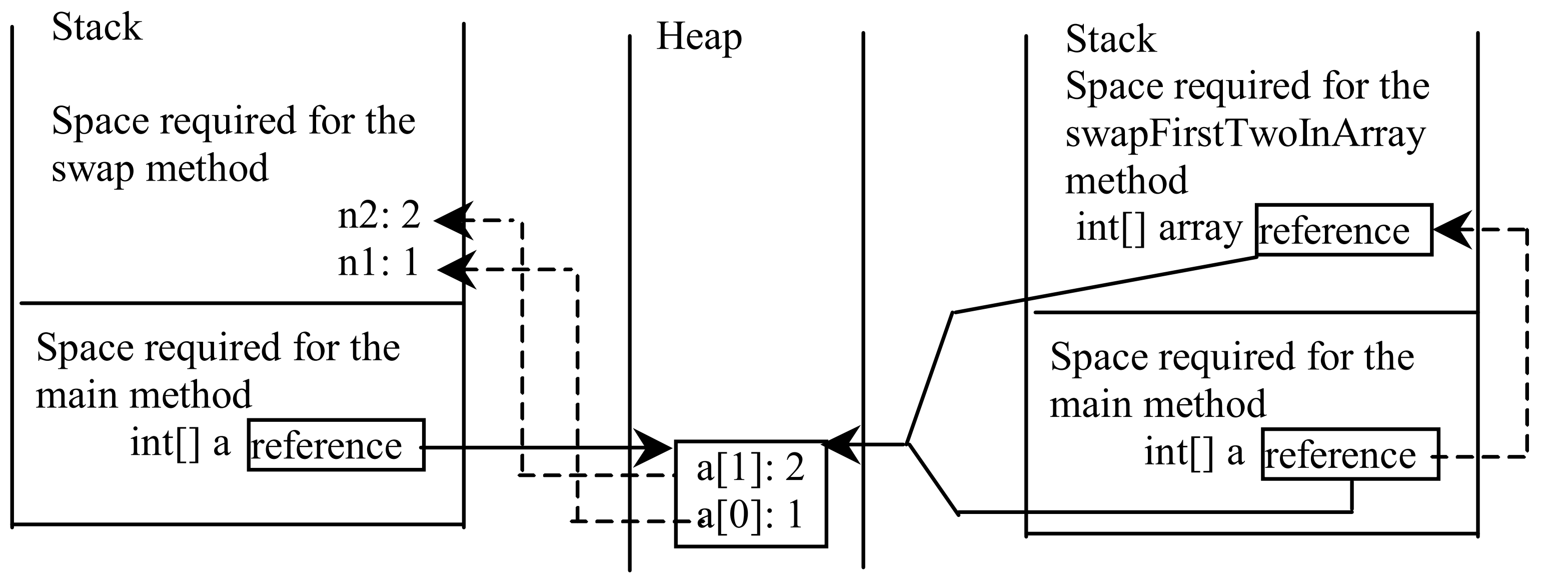
The JVM stores the array in an area of memory, called **heap**, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

Passing Arrays as Arguments

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

TestPassArray

Example


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

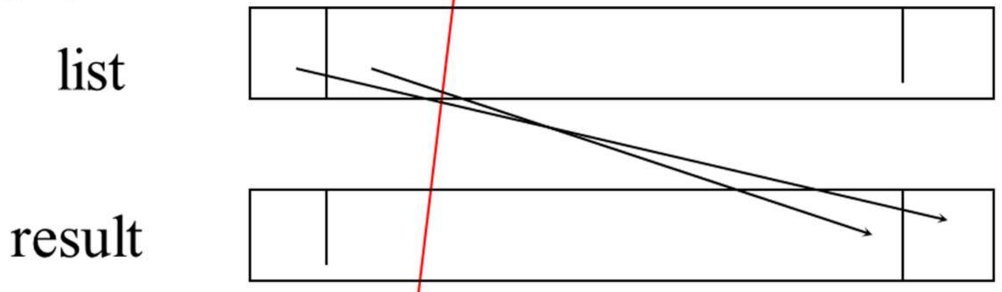
Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

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

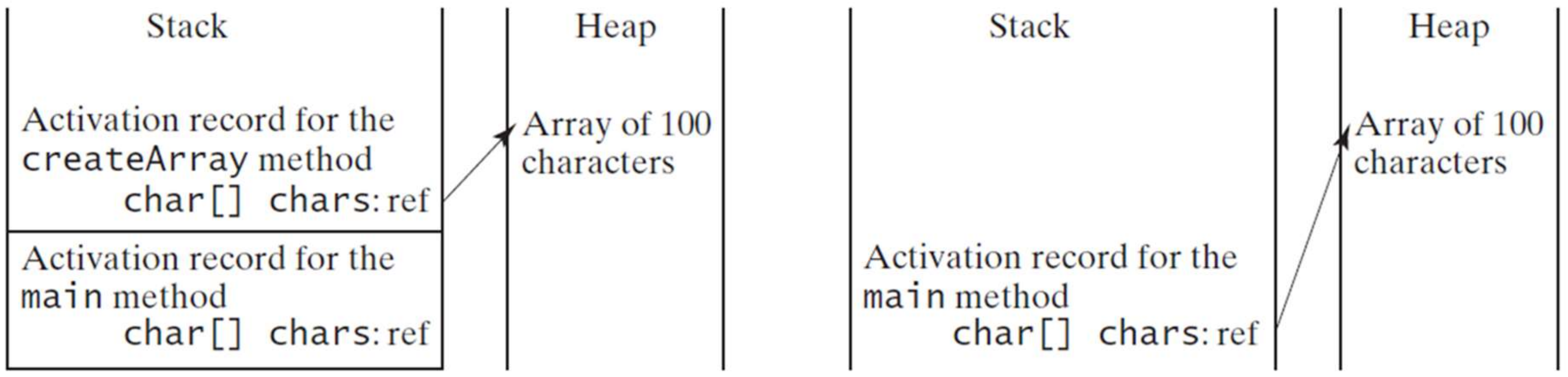
```java
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

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

  return result;
}
```



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

Problem: Counting Occurrence of Each Letter

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.



Stack
Heap
Activation record for the createArray method
char[] chars: ref
Activation record for the main method
char[] chars: ref
Array of 100 characters
Stack
Heap
Array of 100 characters
Activation record for the main method
char[] chars: ref

(a) Executing createArray in line 5

(b) After exiting createArray in line 5

CountLettersInArray

Variable-Length Arguments

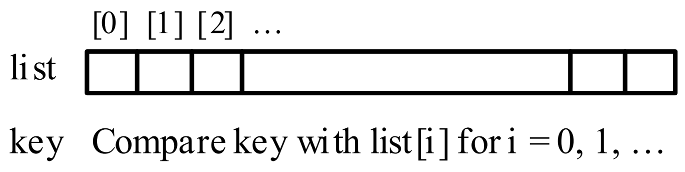
You can pass a variable number of arguments of the same type to a method.

VarArgsDemo

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;
  }
}
```

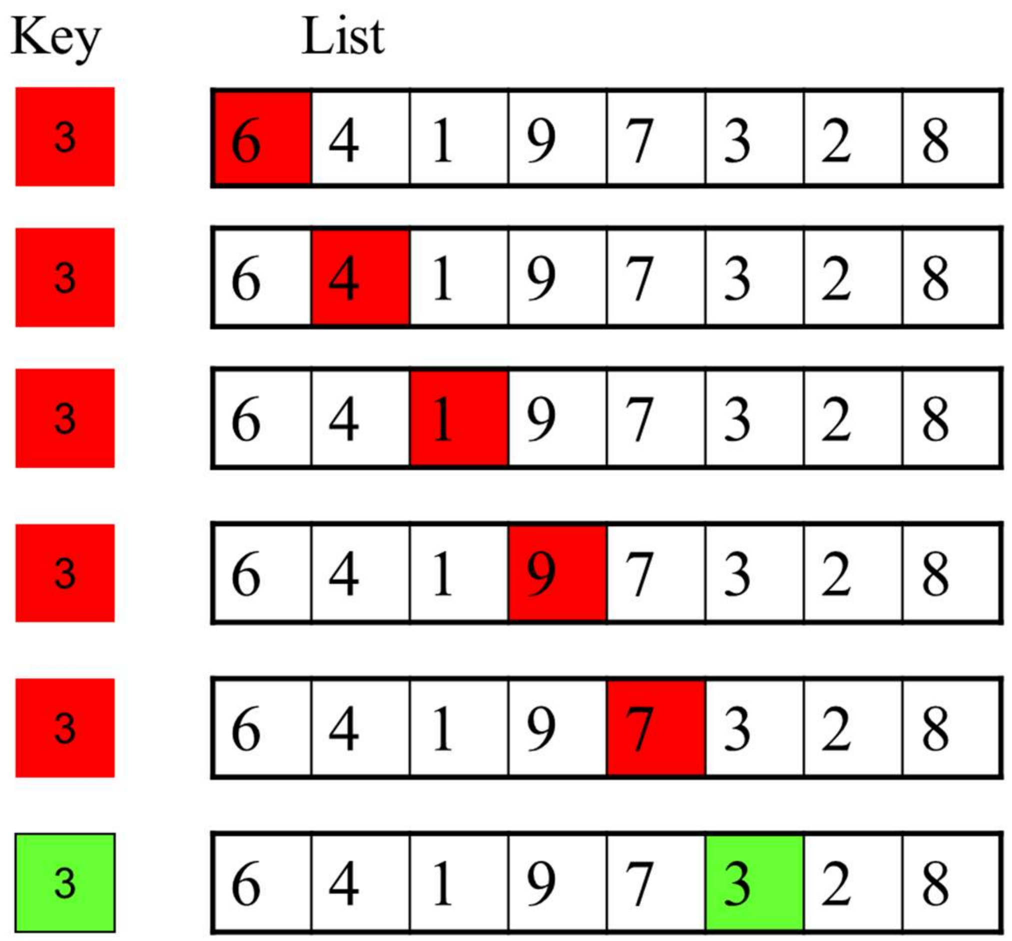


Linear Search

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.

Linear Search Animation (1 of 2)



Binary Search (1 of 6)

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

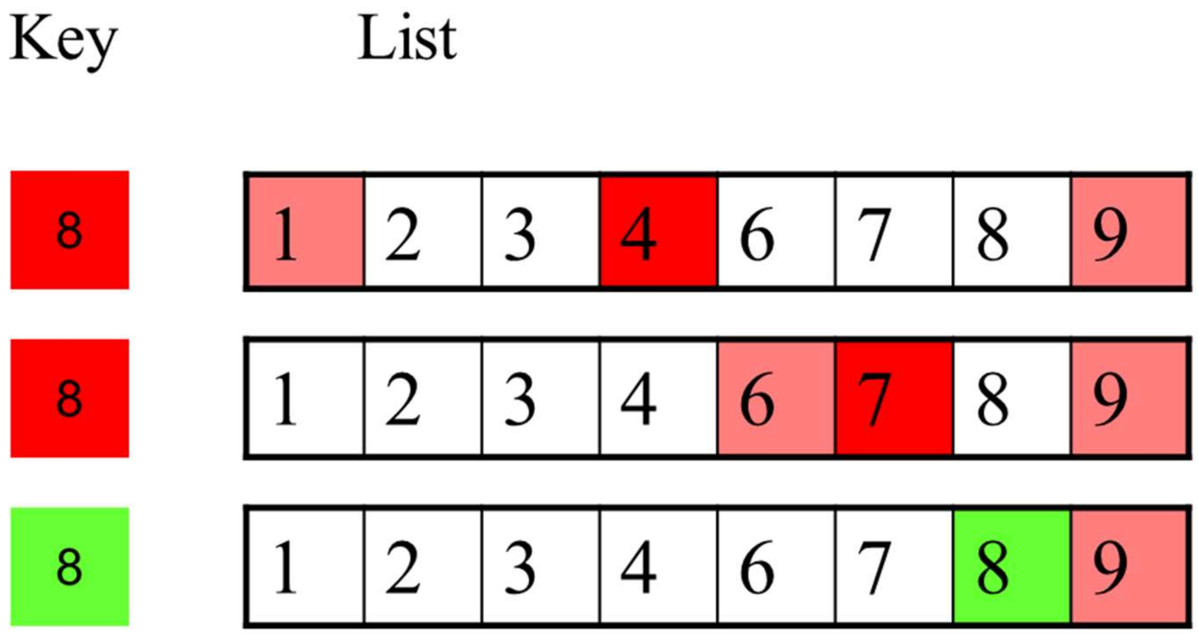
The binary search first compares the key with the element in the middle of the array.

Binary Search (2 of 6)

Consider the following three cases:

- 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.

Binary Search (3 of 6)



Binary Search Animation

https://liveexample.pearsoncmg.com/dsanimation/BinarySearcheBook.html

Binary Search (6 of 6)

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns

-insertion point - 1.

The insertion point is the point at which the key would be inserted into the list.

The Arrays.binarySearch Method

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, the following code searches the keys in an array of numbers and an array of characters.

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

Return is 4

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

Return is –4 (insertion point is 3, so return is -3-1)

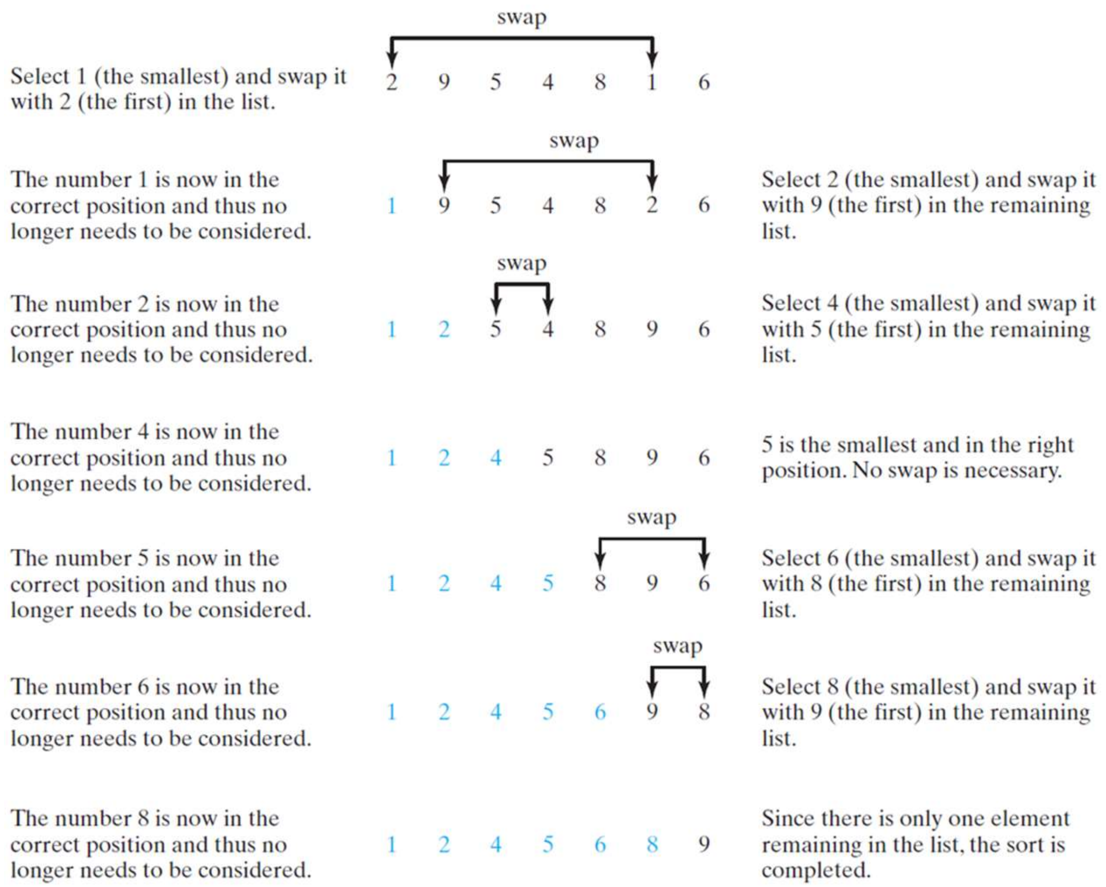
For the binarySearch method to work, the array must be pre-sorted in increasing order.

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**.

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.



swap
Select 1 (the smallest) and swap it with 2 (the first) in the list.
2 9 5 4 8 1 6
swap
The number 1 is now in the correct position and thus no longer needs to be considered.
1 9 5 4 8 2 6
Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.
swap
The number 2 is now in the correct position and thus no longer needs to be considered.
1 2 5 4 8 9 6
Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.
The number 4 is now in the correct position and thus no longer needs to be considered.
1 2 4 5 8 9 6
5 is the smallest and in the right position. No swap is necessary.
swap
The number 5 is now in the correct position and thus no longer needs to be considered.
1 2 4 5 8 9 6
Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.
swap
The number 6 is now in the correct position and thus no longer needs to be considered.
1 2 4 5 6 9 8
Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.
The number 8 is now in the correct position and thus no longer needs to be considered.
1 2 4 5 6 8 9
Since there is only one element remaining in the list, the sort is completed.

Selection Sort Animation

https://liveexample.pearsoncmg.com/dsanimation/SelectionSortNew.html

Wrap it in a Method

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length; i++) {
    // Find the minimum in the list[i..list.length-1]
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Invoke it

selectionSort(yourList)

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.

Main Method Is Just a Regular 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]);
  }
}
```

Command-Line Parameters

```
class TestMain {
 public static void main(String[] args) {
  ...
  }
}
java TestMain arg0 arg1 arg2 ... argn
```

Processing Command-Line Parameters

In the main method, get the arguments from `args[0]`, `args[1], ..., args[n]`, which corresponds to `arg0`, `arg1, ..., argn` in the command line.

Problem: Calculator

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

```
java Calculator 2 + 3

java Calculator   2 – 3

java Calculator   2 / 3

java Calculator 2.3
```

Calculator