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

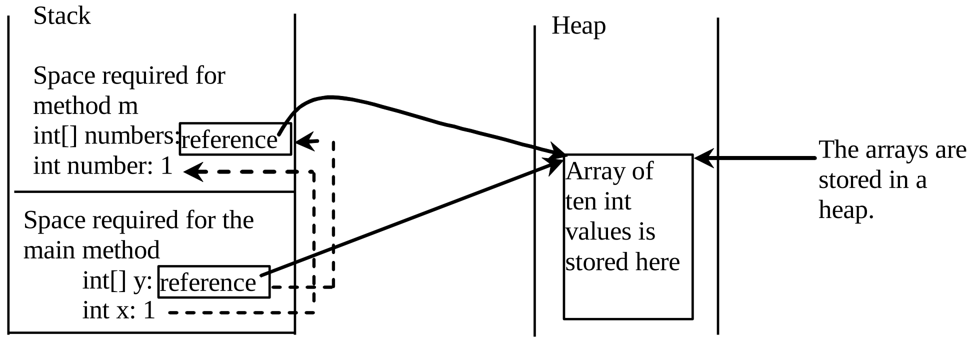

Stack
Space required for method m
int[] numbers:
reference
int number: 1
Space required for the main method
int[] y:
reference
int x: 1
Heap
Array of ten int values is stored here
The arrays are stored in a heap.

FIGURE 7.5 The primitive type value in x is passed to number, and the reference value in y is passed to numbers

LISTING 7.3 TestPassArray: Passing Arrays as Arguments

- 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.
- **Example**: write two methods for swapping elements in an array. The first method, named *swap*, fails to swap two int arguments. The second method, named *swapFirstTwoInArray*, successfully swaps the first two elements in the array argument.

```
public class TestPassArray {
  /** Main method */
  public static void main(String[] args) {
    int[] a = {1, 2};

    // Swap elements using the swap method
    System.out.println("Before invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swap(a[0], a[1]);
    System.out.println("After invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");

    // Swap elements using the swapFirstTwoInArray method
    System.out.println("Before invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swapFirstTwoInArray(a);
    System.out.println("After invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
  }

  /** Swap two variables */
  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }

  /** Swap the first two elements in the array */
  public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
}
```

```
Before invoking swap
array is {1, 2}
After invoking swap
array is {1, 2}
Before invoking swapFirstTwoInArray
array is {1, 2}
After invoking swapFirstTwoInArray
array is {2, 1}
```

- The first method doesn't work. The two elements are not swapped using the *swap* method.
- The second method works. The two elements are actually swapped using the *swapFirstTwoInArray* method.
- Since the arguments in the first method are primitive type, the values of a[0] and a[1] are passed to n1 and n2 inside the method when invoking *swap(a[0], a[1])*.
- The memory locations for n1 and n2 are independent of the ones for a[0] and a[1].
- The contents of the array are not affected by this call.

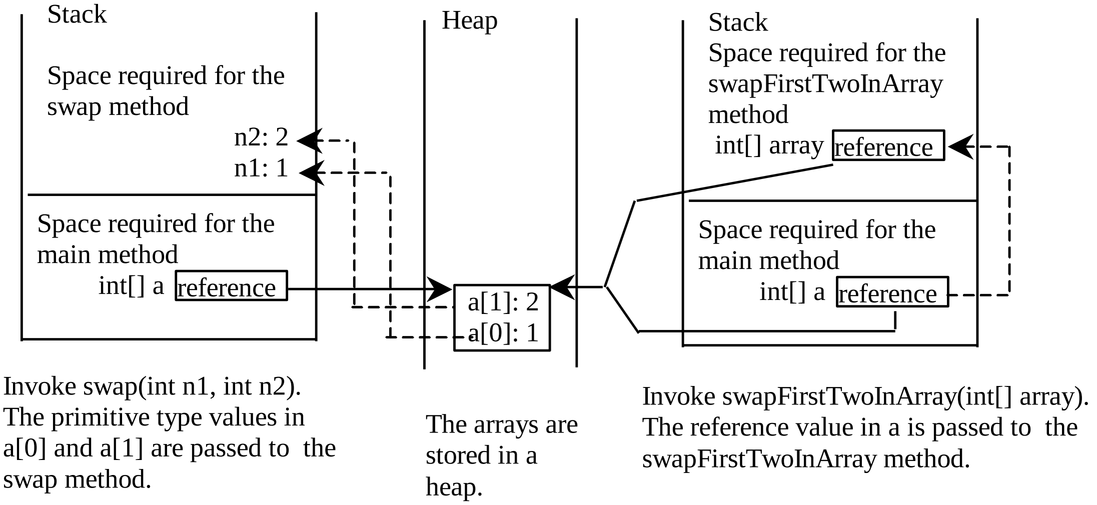

Stack
Space required for the swap method
n2: 2
n1: 1
Space required for the main method
int[] a reference
Heap
a[1]: 2
a[0]: 1
Stack
Space required for the swapFirstTwoInArray method
int[] array reference
Space required for the main method
int[] a reference
Invoke swap(int n1, int n2). The primitive 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.

FIGURE 7.6 When passing an array to a method, the reference of the array is passed to the method.

- The parameter in the `swapFirstTwoInArray` method is an array.
- As shown above, the reference of the array is passed to the method.
- Thus the variables *a* (outside the method) and *array* (inside the method) both refer to the same array in the same memory location.
- Therefore, swapping array[0] with array[1] inside the method `swapFirstTwoInArray` is the same as swapping a[0] with a[1] outside of the method.

7.7 Returning an Array from a Method

- You can pass arrays to invoke a method. A method may also return an array.
- For example, the method below returns an array that is the reversal of another array:

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

  for (int i = 0, j = result.length - 1; //  copies elements from array
       i < list.length; i++, j--) {      //  list to array result
    result[j] = list[i];
  }
  return result;
}
```

- The following statement returns a new array list2 with elements 6, 5, 4, 3, 2, 1:

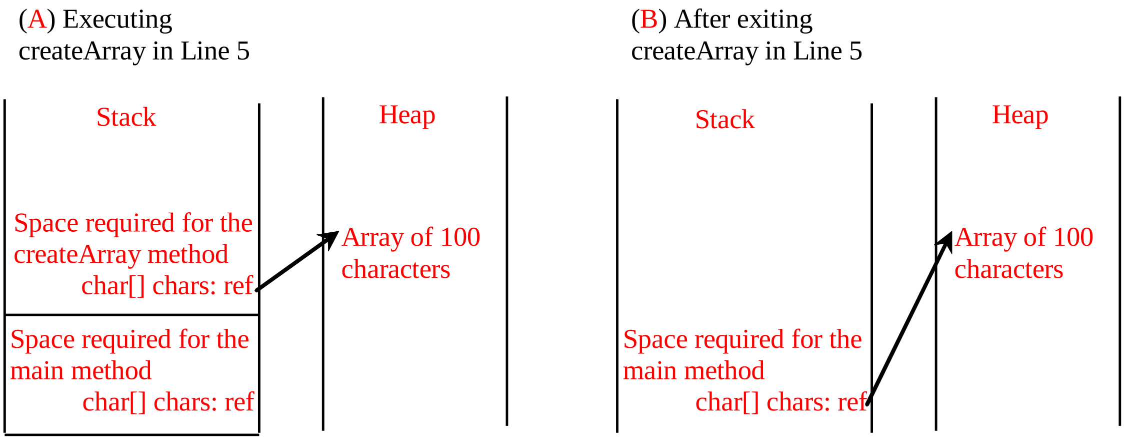
```
int[] list1 = new int[]{1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

7.8 Case Study: Counting the Occurrences of Each Letters

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

LISTING 7.4 CountLettersInArray.java

```
/* Output
      The lowercase letters are:
      e n v e v n s f w x i u b x w v w m y v
      h o c j d d y t b e c p w w q h e w d u
      v t q p c d k q m v j o k n u x w f c b
      p p n z t x f e m o g g n o y y l b s b
      h f a h t e i f a h f x l e y u i w v g

      The occurrences of each letter are:
      2 a 5 b 4 c 4 d 7 e 6 f 3 g 5 h 3 i 2 j
      2 k 2 l 3 m 5 n 4 o 4 p 3 q 0 r 2 s 4 t
      4 u 7 v 8 w 5 x 5 y 1 z
*/

public class CountLettersInArray {
  /** Main method */
  public static void main(String args[]) {
    // Declare and create an array
    char[] chars = createArray();

    // Display the array
    System.out.println("The lowercase letters are:");
    displayArray(chars);

    // Count the occurrences of each letter
    int[] counts = countLetters(chars);

    // Display counts
    System.out.println();
    System.out.println("The occurrences of each letter are:");
    displayCounts(counts);
  }

  /** Create an array of characters */
  public static char[] createArray() {
    // Declare an array of characters and create it
    char[] chars = new char[100];

    // Create lowercase letters randomly and assign
    // them to the array
    for (int i = 0; i < chars.length; i++)
      chars[i] = RandomCharacter.getRandomLowerCaseLetter();

    // Return the array
    return chars;
  }

  /** Display the array of characters */
  public static void displayArray(char[] chars) {
    // Display the characters in the array 20 on each line
    for (int i = 0; i < chars.length; i++) {
      if ((i + 1) % 20 == 0)
        System.out.println(chars[i] + " ");
      else
        System.out.print(chars[i] + " ");
    }
  }

  /** Count the occurrences of each letter */
  public static int[] countLetters(char[] chars) {
    // Declare and create an array of 26 int
    int[] counts = new int[26];

    // For each lowercase letter in the array, count it
    for (int i = 0; i < chars.length; i++)
      counts[chars[i] - 'a']++;

    return counts;
  }

  /** Display counts */
  public static void displayCounts(int[] counts) {
    for (int i = 0; i < counts.length; i++) {
      if ((i + 1) % 10 == 0)
        System.out.println(counts[i] + " " + (char)(i + 'a'));
      else
        System.out.print(counts[i] + " " + (char)(i + 'a') + " ");
    }
  }
}
```


(A) Executing createArray in Line 5

(B) After exiting createArray in Line 5

FIGURE 7.8 (a) An array of 100 characters is created when executing createArray. (b) This array is returned and assigned to the variable chars in the main method

7.9 Variable-Length Argument Lists

- A variable number of arguments of **the same type** can be passed to a method and treated as an **array**.

```
TypeName… parameterName
```

LISTING 7.5 VarArgsDemo.java

```
public class VarArgsDemo {
  public static void main(String args[]) {
    printMax(34, 3, 3, 2, 56.5);
    printMax(new double[]{1, 2, 3});
  }

  public static void printMax(double... numbers) {
    if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
    }

    double 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);
  }
}
```

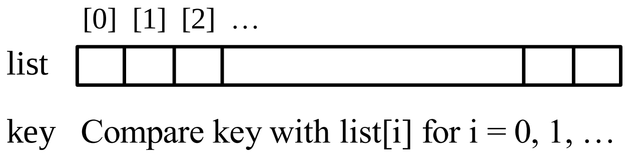
```
The max value is 56.5
The max value is 3.0
```

7.10 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.

7.10.1 The Linear Search Approach

- 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.



```
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;
  }
}
```

- The linear search method compares the key with each element in the array.

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = LinearSearch.linearSearch(list, 4); // Returns 1
int j = LinearSearch.linearSearch(list, -4); // Returns -1
int k = LinearSearch.linearSearch(list, -3); // Returns 5
```

7.10.2 The Binary Search Approach

- 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.

  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.
- 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.

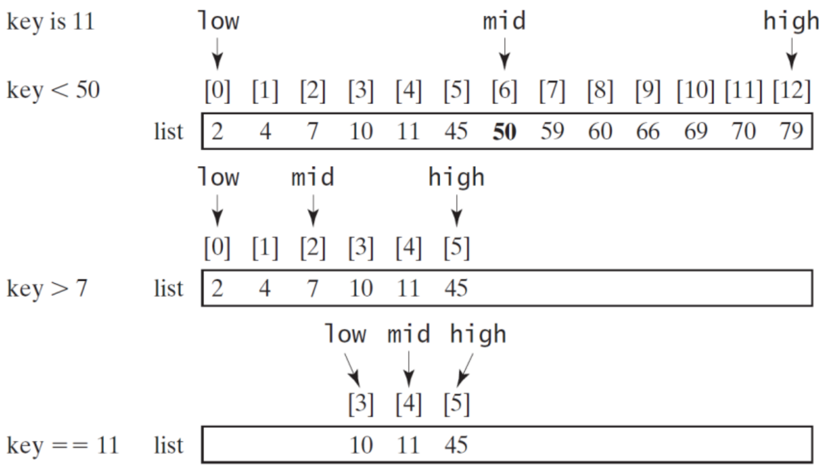


FIGURE 7.9 Binary search eliminates half of the list from further consideration after each comparison.

LISTING 7.7 BinarySearch.java

```
public class BinarySearch {
  /** 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;
      if (key < list[mid])
        high = mid - 1;
      else if (key == list[mid])
        return mid;
      else
        low = mid + 1;
    }
    return -low - 1; // Now high < low
  }
}
```

- To better understand this method, trace it with the following statements and identify low and high when the method returns.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
int i = BinarySearch.binarySearch(list, 2); // Returns 0
int j = BinarySearch.binarySearch(list, 11); // Returns 4
int k = BinarySearch.binarySearch(list, 12); // Returns -6
int l = BinarySearch.binarySearch(list, 1); // Returns -1
int m = BinarySearch.binarySearch(list, 3); // Returns -2
```

Method	Low	High	Value Returned
binarySearch(list, 2)	0	1	0
binarySearch(list, 11)	3	5	4
binarySearch(list, 12)	5	4	-6
binarySearch(list, 1)	0	-1	-1
binarySearch(list, 3)	1	0	-2

7.11 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** 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.

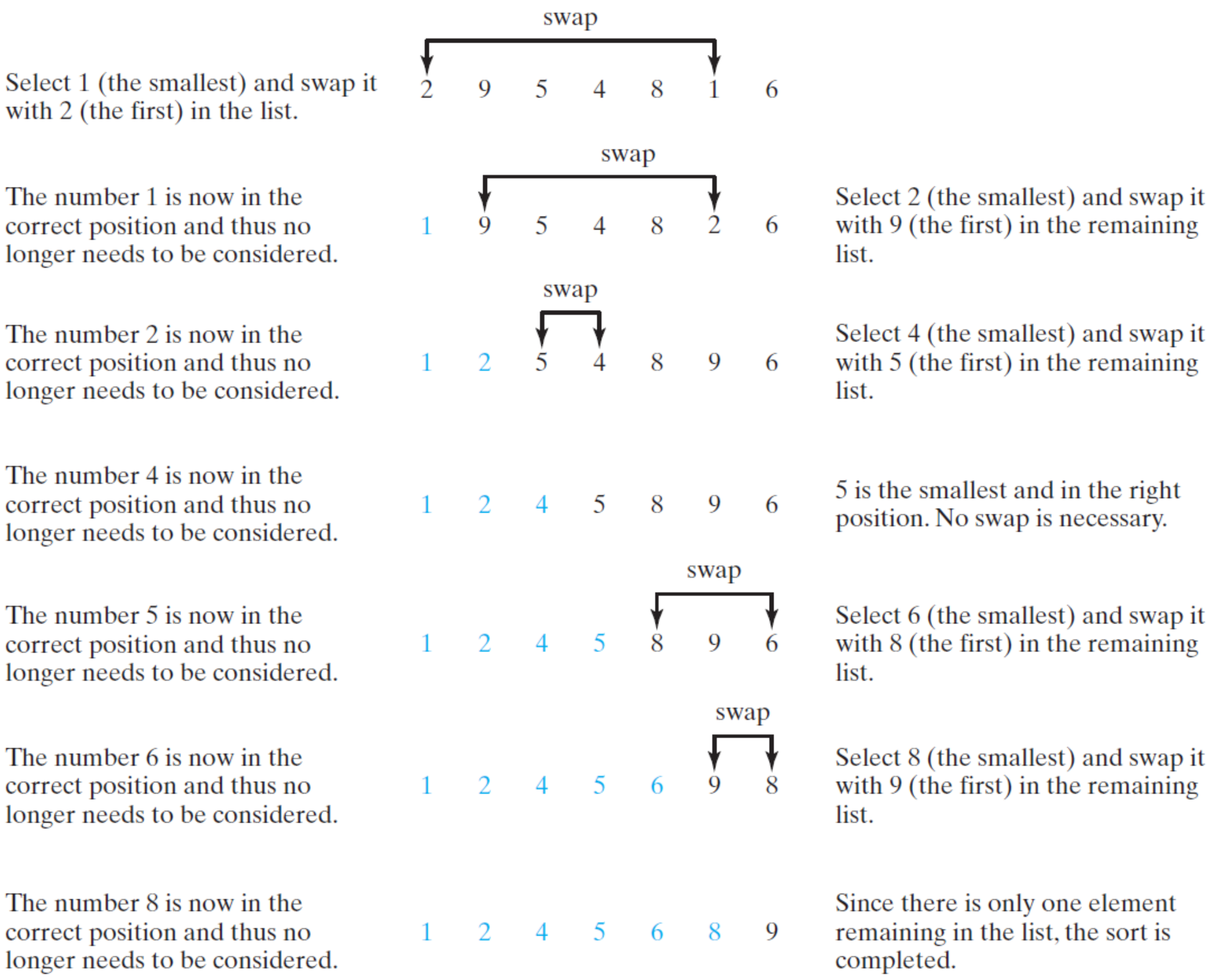

swap
Select 1 (the smallest) and swap it with 2 (the first) in the list.
2 9 5 4 8 1 6
swap
The number 1 is now in the correct position and thus no longer needs to be considered.
1 9 5 4 8 2 6
Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.
swap
The number 2 is now in the 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.

FIGURE 7.11 Selection sort repeatedly selects the smallest number and swaps it with the first number in the list.

LISTING 7.8 SelectionSort.java

```
public class SelectionSort {
  /** The method for sorting the numbers */
  public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length - 1; 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;
      }
    }
  }
}
```

- To understand this method better, trace it with the following statements:

```
double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
SelectionSort.selectionSort(list);
```

```
-4.5 1.0 4.5 5.7 6.6 9.0
```

7.12 The Array Class

- 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
```

```
Index 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)
```

```
Index is -4
```

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

- 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);
```

```
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);
```

```
4 A D F P a
```

7.13 Command-Line Arguments

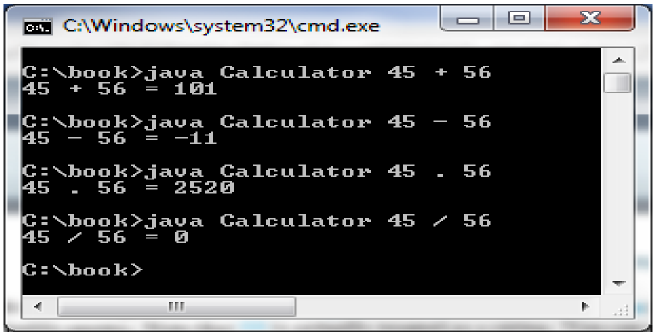
- The main method can receive string arguments from the command line.
- In the main method, get the arguments from args[0], args[1], ..., args[n], which corresponds to arg0, arg1, ..., argn in the command line.

```
java Calculator 2 + 3
```

LISTING 7.9 Calculator.java

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

```
public class Calculator {
  /** Main method */
  public static void main(String[] args) {
    // Check number of strings passed
    if (args.length != 3) {
     System.out.println(
        "Usage: java Calculator operand1 operator operand2");
      System.exit(0);
    }
   // The result of the operation
    int result = 0;

    // Determine the operator
    switch (args[1].charAt(0)) {
      case '+': result = Integer.parseInt(args[0]) +
                         Integer.parseInt(args[2]);
                break;
      case '-': result = Integer.parseInt(args[0]) -
                         Integer.parseInt(args[2]);
                break;
      case '.': result = Integer.parseInt(args[0]) *
                         Integer.parseInt(args[2]);
                break;
      case '/': result = Integer.parseInt(args[0]) /
                         Integer.parseInt(args[2]);
    }
    // Display result
    System.out.println(args[0] + ' ' + args[1] + ' ' + args[2]
     + " = " + result);
  }
}
```



```
C:\book>java Calculator 45 + 56
45 + 56 = 101

C:\book>java Calculator 45 - 56
45 - 56 = -11

C:\book>java Calculator 45 . 56
45 . 56 = 2520

C:\book>java Calculator 45 / 56
45 / 56 = 0

C:\book>
```