Chapter 7 Single-Dimensional Arrays

Dr. Abdallah Karakra
Dr. Asem Kitana

Opening Problem

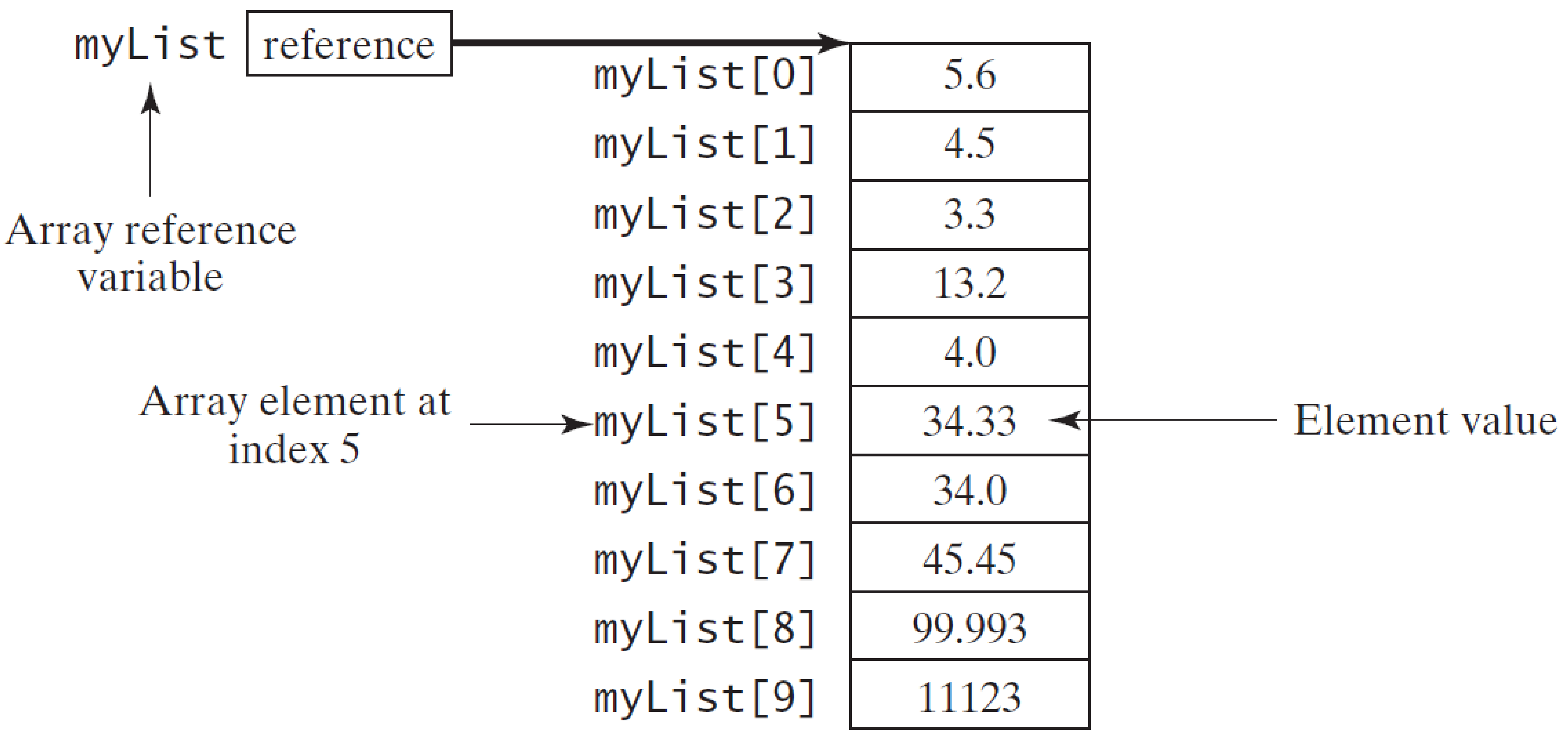
Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Arrays

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

Array stores a fixed-size sequential collection of elements of the same type.

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



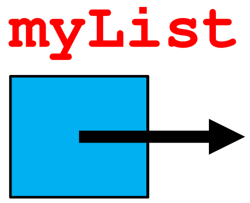
Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array and specify the array's *element type*. Here is the syntax for declaring an array variable:

- `datatype[] arrayRefVar;`**`//Reference to array`**

Example:

```
double[] myList;
```



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

Example:

```
double myList[];
```

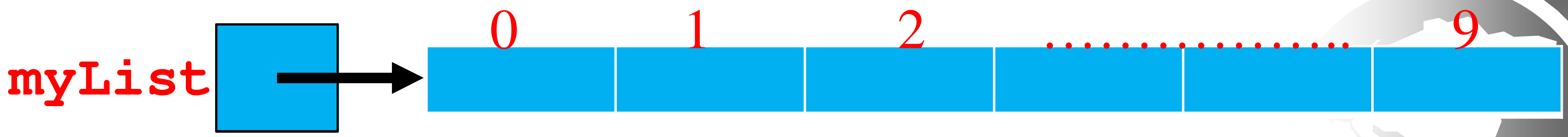
Creating Arrays

❖ We can create an array by using the new operator and assign its reference to the variable with the following syntax:

```
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

✓ Check Point

Assume int[] t = {1, 2, 3, 4, 6}. What is t.length?

Answer : 5

If you declare an array double[] list = {3.4, 2.0, 3.5, 5.5,7.0}, the highest index in array list is .

Answer : 4

How many elements are in array double[] list = new double[8]?

Answer : 8

Default Values

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

0 for the **numeric primitive data types**,

false for **boolean types**.

null for **String types.**

In Java, elements of an array are automatically initialized to some default value. What is the default value for the elements of an array of integers?

Answer : 0

✓ Check Point

If you declare an array double [] myArray=new double [10];
myArray [3] is

Answer : 0.0

If you declare an array int [] myArray=new int [10];
myArray [2] is

Answer : 0

If you declare an array boolean [] myArray=new boolean [10];
myArray [3] is

Answer : false

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.

- If myList array holds ten double values then 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];

Using Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in myList[0] and myList[1] to myList[2].

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

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:

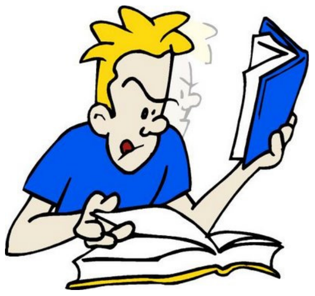
```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

CAUTION

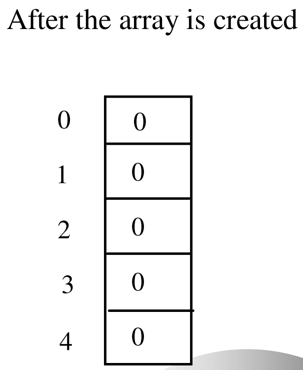
Using the shorthand notation, you have to declare, create, and initialize the array all in **one statement**. Splitting it would cause a syntax error. For example, the following is wrong:

```
double[] myList;

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

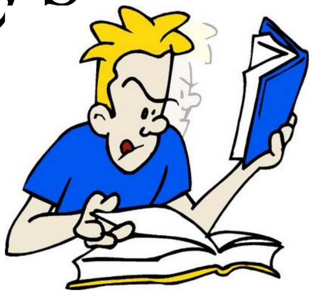
Trace Program with Arrays



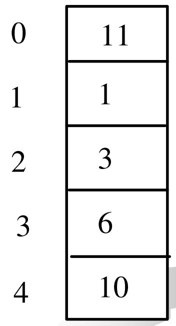
```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



Trace Program with Arrays



```
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];
  }
}
```



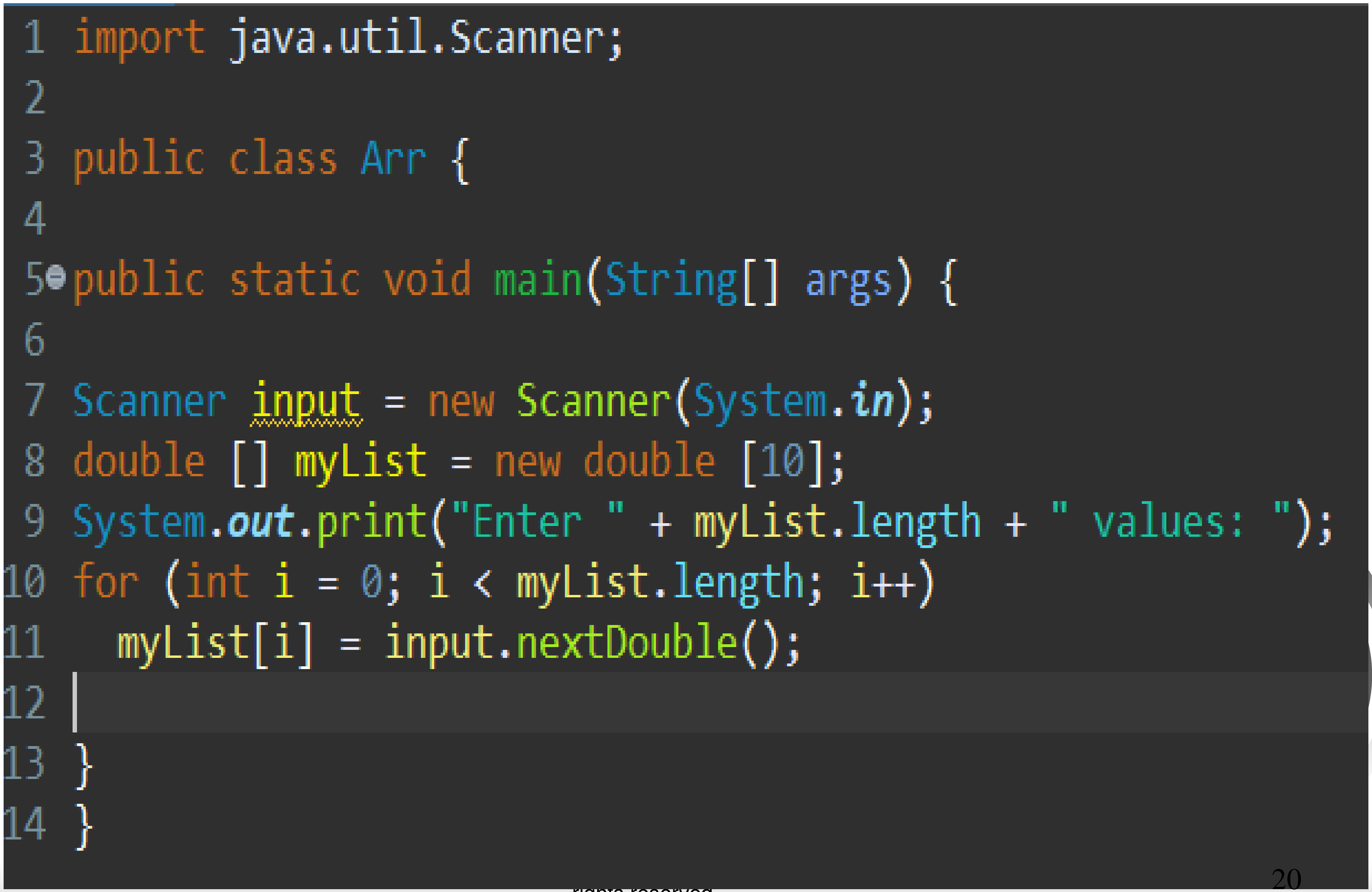
Processing Arrays

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. (*Shifting elements*)

Initializing arrays with input values

```
Scanner input = new Scanner(System.in);
double [] myList = new double [10];
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

Initializing arrays with input values



```
import java.util.Scanner;

public class Arr {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);
double [] myList = new double [10];
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();

}
}
```

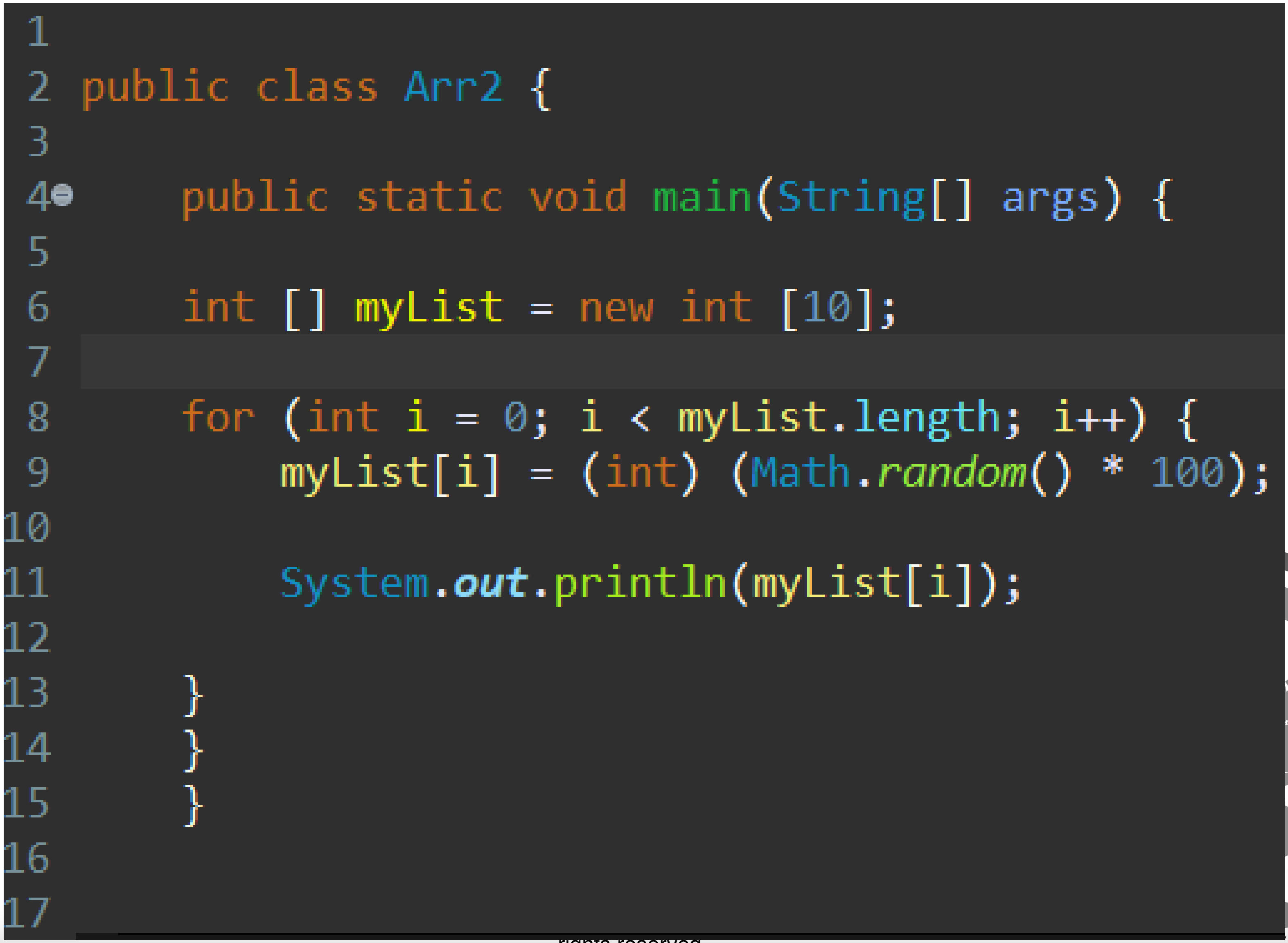
Initializing arrays with random values

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

Printing arrays

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

Initializing arrays with random values and printing arrays



```

public class Arr2 {

    public static void main(String[] args) {

    int [] myList = new int [10];

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

        System.out.println(myList[i]);

    }
    }
    }


```

Summing all elements

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

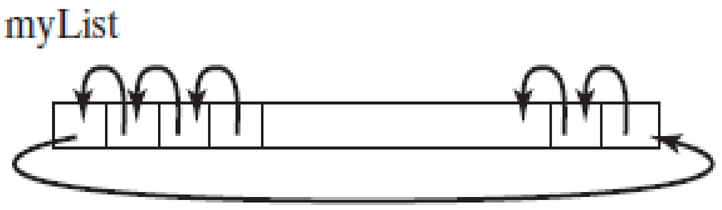
Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max)
    max = myList[i];
}
```

Shifting Elements

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

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

// Shift elements 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.

✓ Check Point

What is the output of the following code ?

```
public class Mystery {
  public static void main(String[] args) {
    double[] x = {2.5, 3, 4, 6 ,5};
    for (double value: x)
      System.out.print(value + "  ");
  }
}
```

Answer :

```
2.5  3.0  4.0  6.0  5.0
```

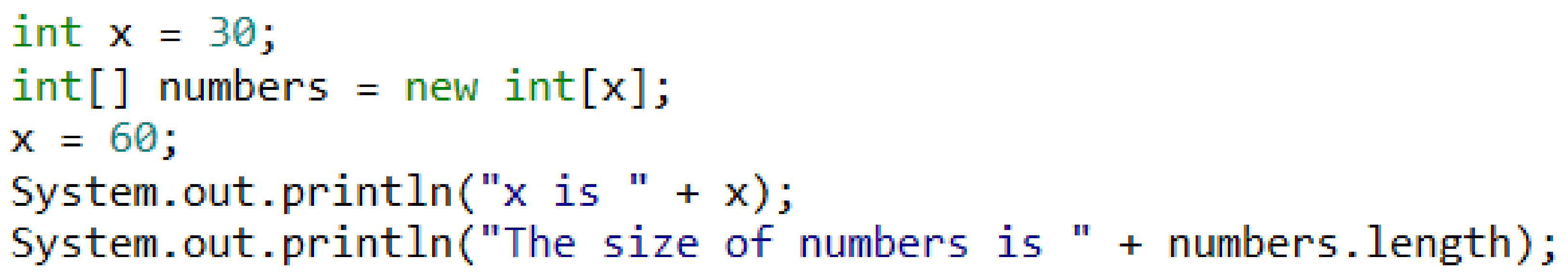
✓ Check Point

❖ Correct the following code ?

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

Answer : The **<=** should be replaced by **<**

❖ What is the output of the following code?



```
int x = 30;
int[] numbers = new int[x];
x = 60;
System.out.println("x is " + x);
System.out.println("The size of numbers is " + numbers.length);
```

Answer :

```
x is 60
The size of numbers is 30
```

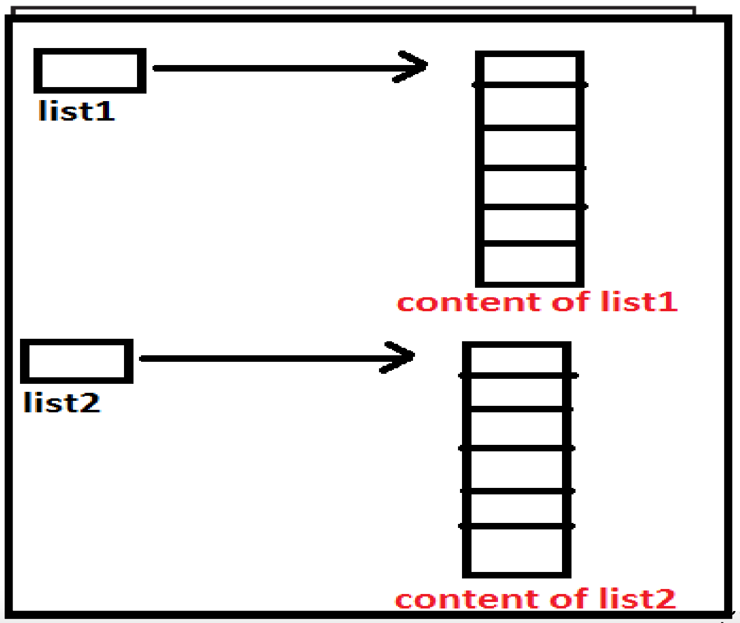
Copying Arrays

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:

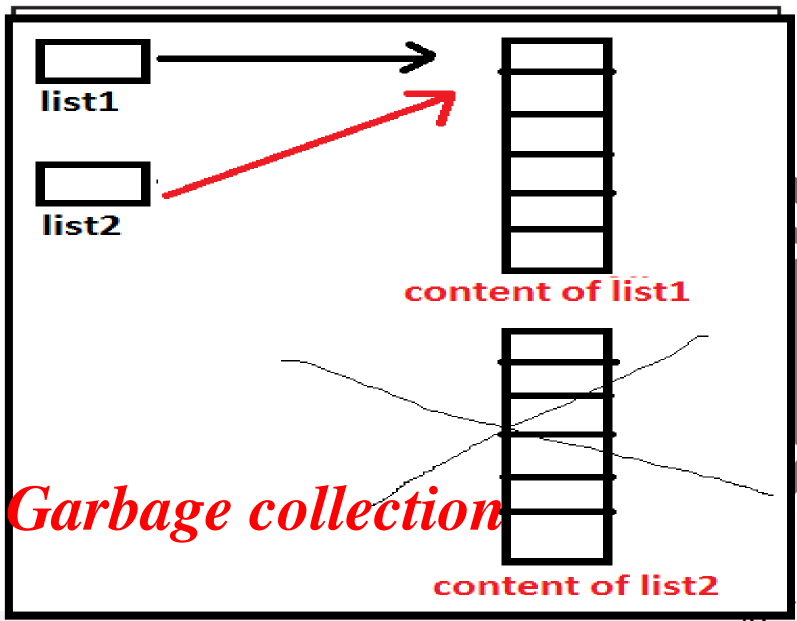
list2 = list1;

The above statement does not copy the contents of the array referenced by **list1,** but instead merely copies the reference value from **list1** to **list2.** After this statement, list1 and list2 reference the same array.

Before the assignment



After the assignment



Example

```
public class Mystery {
  public static void main(String[] args) {
    double [] x = {2.5, 3, 4, 6 ,5};
    double []y  = {1,2,3};
    y=x;
    for (double value: x)
      System.out.print(value + " ");

       System.out.println();

      for (double value: y)
      System.out.print(value + " ");
  }
}
```

```
2.5 3.0 4.0 6.0 5.0
2.5 3.0 4.0 6.0 5.0
```

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];
```

Passing Arrays to Methods

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

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

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*.

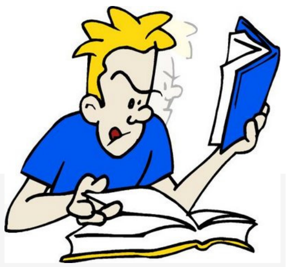
Pass By Value

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(pass-by-sharing). 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); // x is 1
    System.out.println("y[0] is " + y[0]);// y[0] is 5555
  }

  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]
  }
}
```

```
x is 1
y[0] is 5555
```

Listing 7.3 TestPassArray.java

```java
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}
```

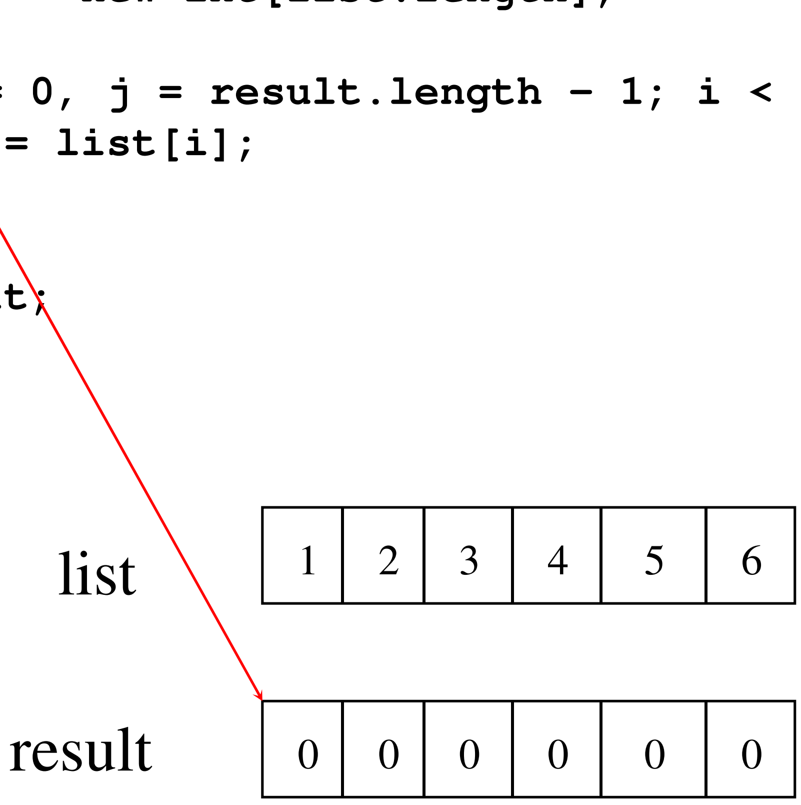
Returning an Array from a Metho

```
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;
}
```

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

Trace the reverse Method

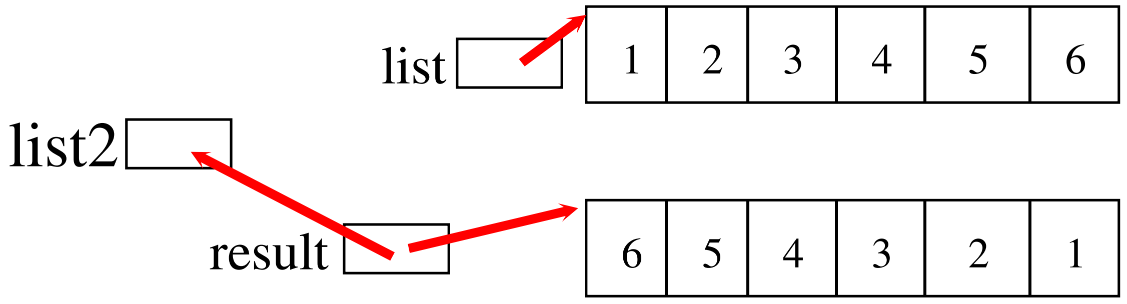
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


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

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

    return result;
  }
```



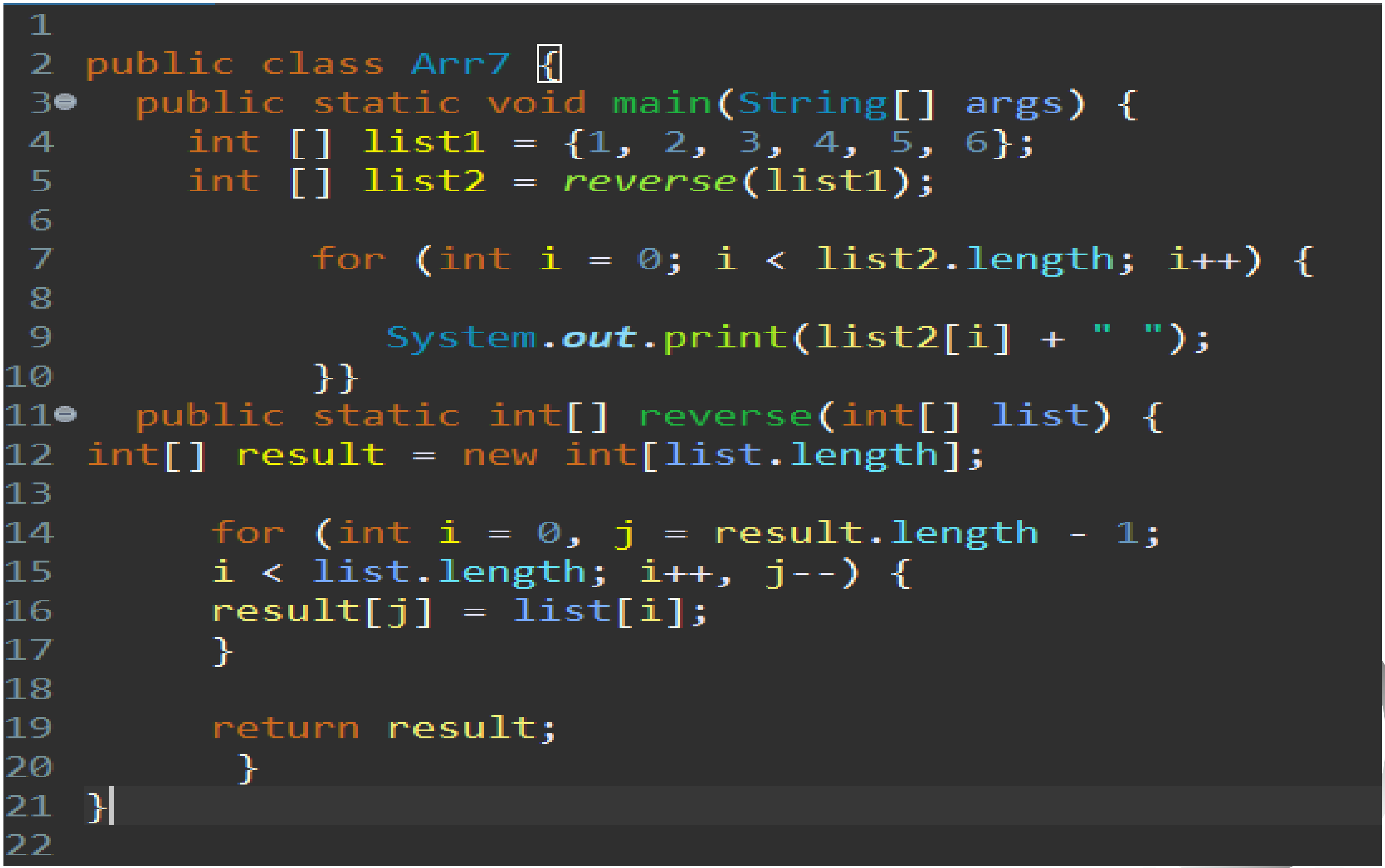
Trace the reverse Method, cont.

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

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

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

    return result;
  }
```



The reverse Method

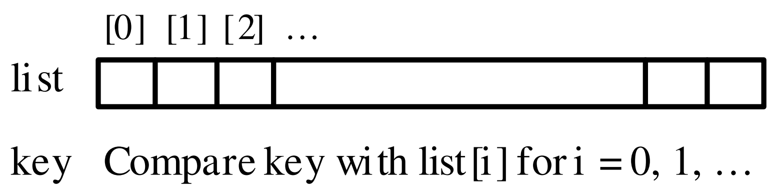


```

public class Arr7 {
   public static void main(String[] args) {
      int [] list1 = {1, 2, 3, 4, 5, 6};
      int [] list2 = reverse(list1);

          for (int i = 0; i < list2.length; i++) {

             System.out.print(list2[i] + " ");
          }}
   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;
       }
}

```

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*.

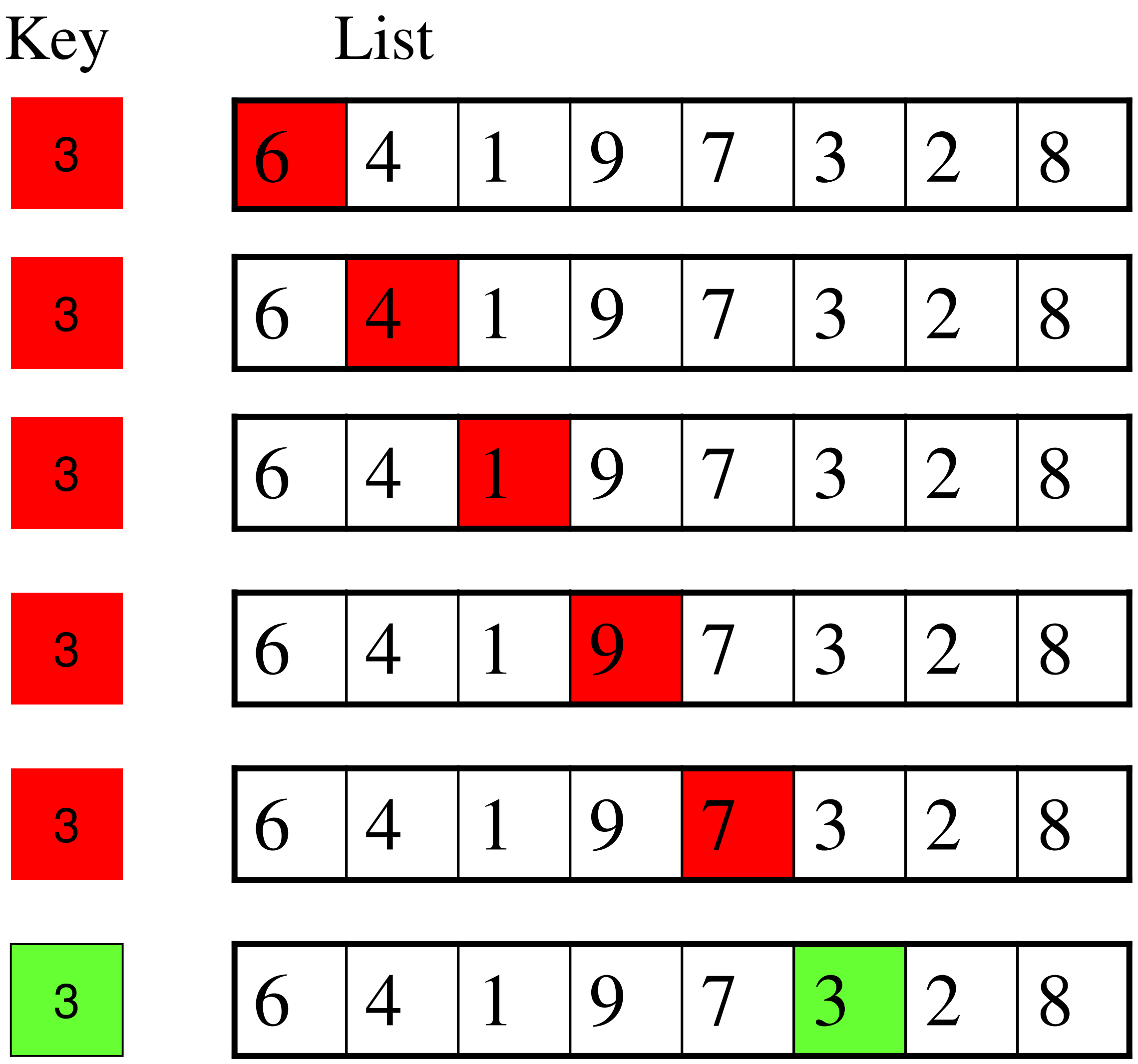
```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
}
```



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



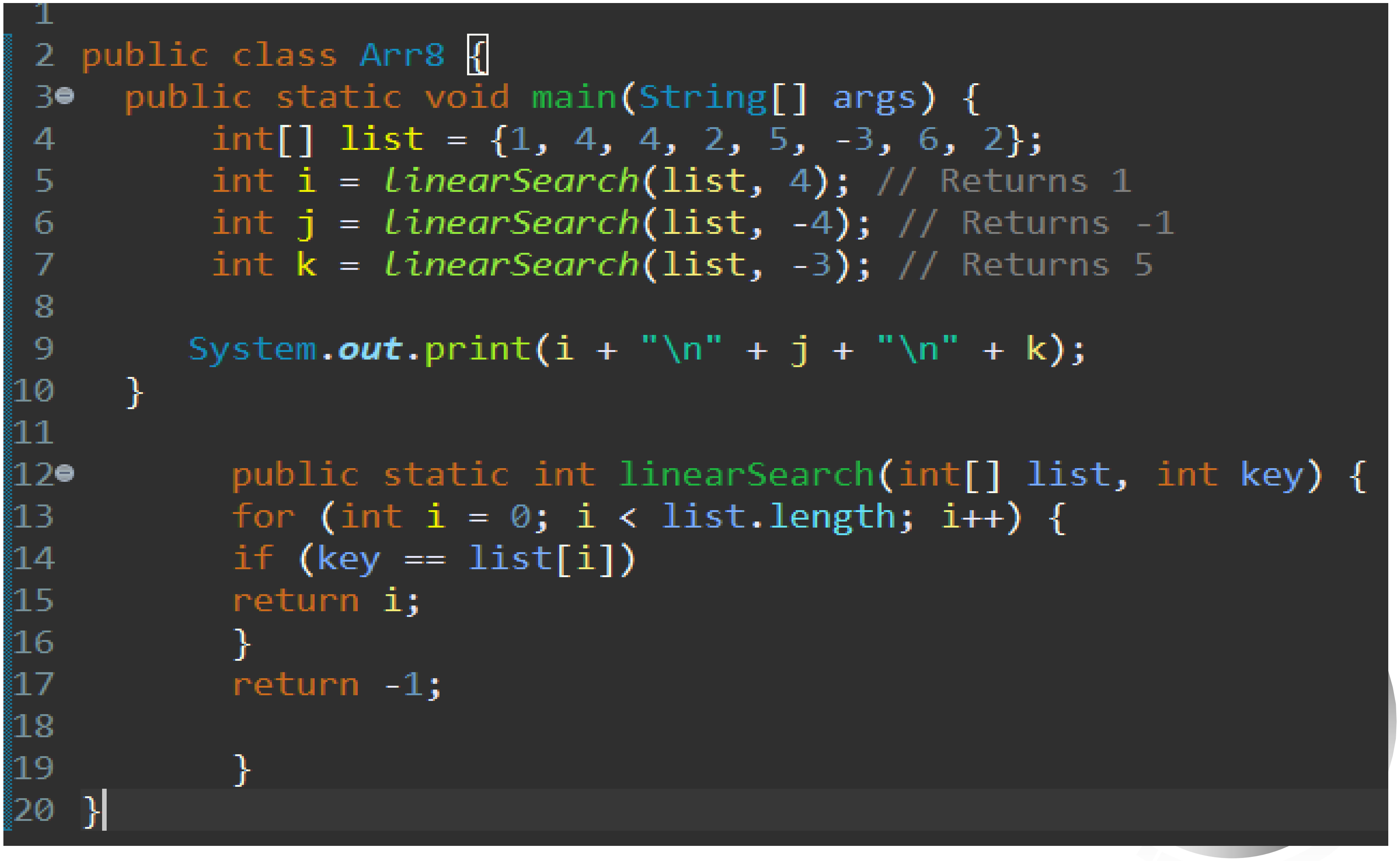
From Idea to Solution

```
/** 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;
}
```

Trace the method

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

Linear Search



```java

public class Arr8 {
  public static void main(String[] args) {
      int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
      int i = linearSearch(list, 4); // Returns 1
      int j = linearSearch(list, -4); // Returns -1
      int k = linearSearch(list, -3); // Returns 5

     System.out.print(i + "\n" + j + "\n" + k);
  }

       public static int linearSearch(int[] list, int key) {
       for (int i = 0; i < list.length; i++) {
       if (key == list[i])
       return i;
       }
       return -1;

       }
}
```

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.

Binary Search

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

- ☞ Let **low** and **high** denote, respectively, the first index and last index of the array that is currently being searched. Initially, **low** is **0** and **high** is **list.length–1**. Let **mid** denote the index of the middle element, so **mid** is **(low + high)/2**.
- ☞ Example:

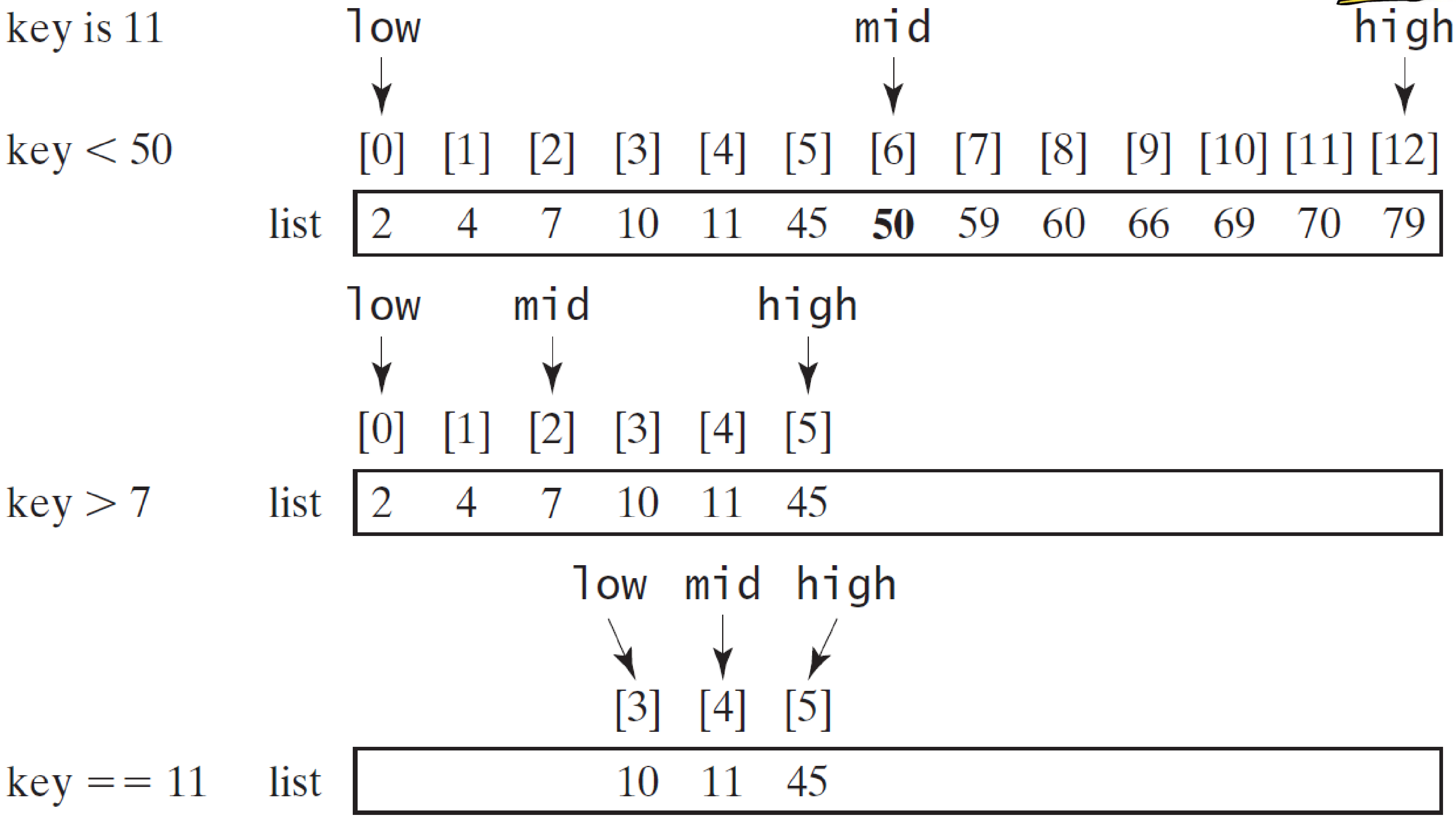
  {**2**, **4**, **7**, **10**, **11**, **45**, **50**, **59**, **60**, **66**, **69**, **70**, **79**}

Low = 0 (first index)

High = 12 (last index)

Mid = (0+12)/2 = 6 (sixth index)

Binary Search



Binary Search

The binary search 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).

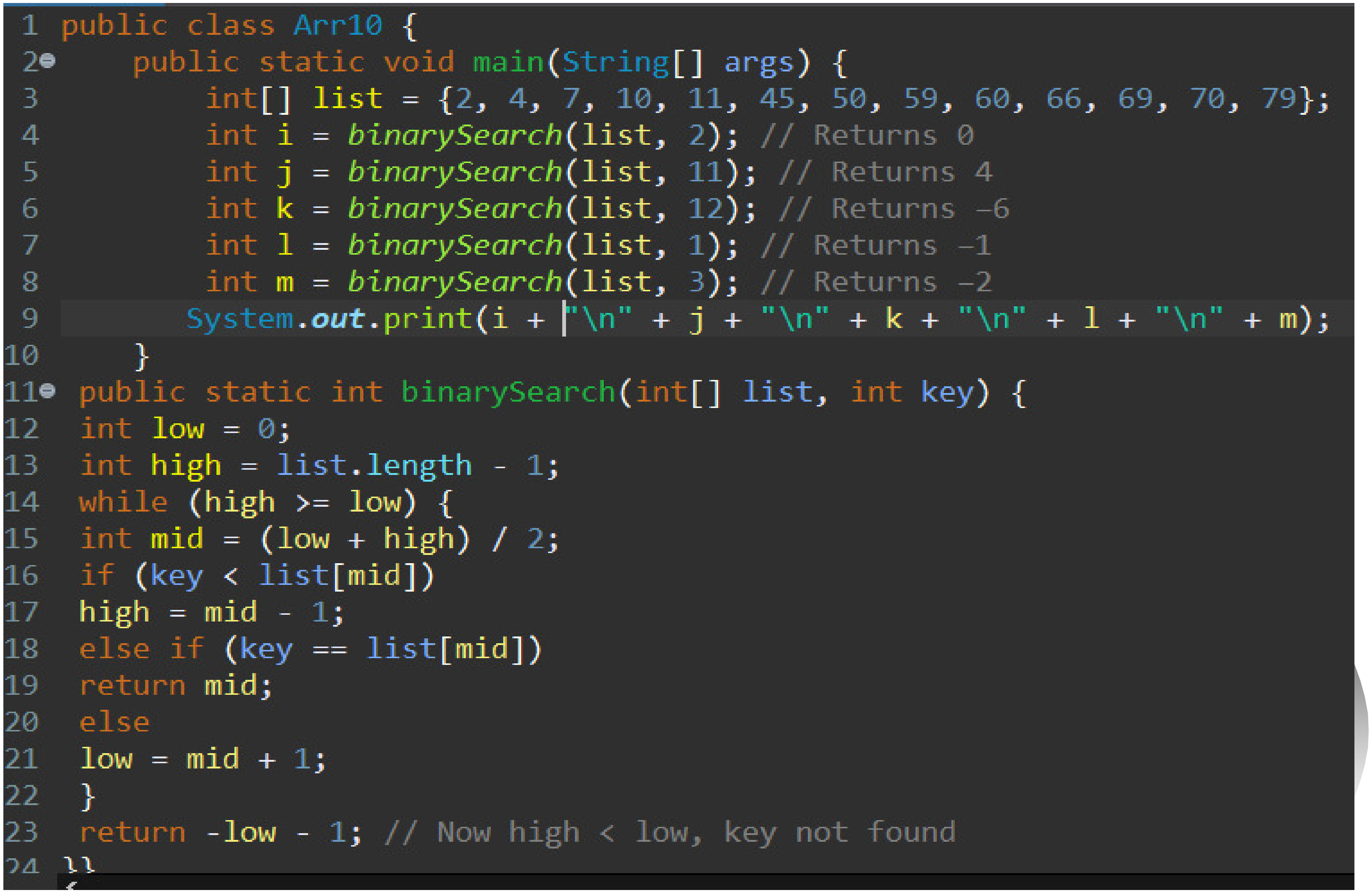
(insertion point equals the index of the "low" element in the last iteration).

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

From Idea to Soluton

```
/** 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 -1 - low;
}
```

Binary Search



```
public class Arr10 {
    public static void main(String[] args) {
        int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
        int i = binarySearch(list, 2); // Returns 0
        int j = binarySearch(list, 11); // Returns 4
        int k = binarySearch(list, 12); // Returns -6
        int l = binarySearch(list, 1); // Returns -1
        int m = binarySearch(list, 3); // Returns -2
       System.out.print(i + "\n" + j + "\n" + k + "\n" + l + "\n" + m);
    }
 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, key not found
}}
```

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 " + 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.

Example

```
int[] num = {1,2,4,5,6};
System.out.println("Index is " + java.util.Arrays.binarySearch(num, 3));
```

Return is (Index is -3)

```
int[] num = {6,5,4,2,1};
System.out.println("Index is " + java.util.Arrays.binarySearch(num, 3))
```

Not sorted in increasing order

Return is (Index is -1)

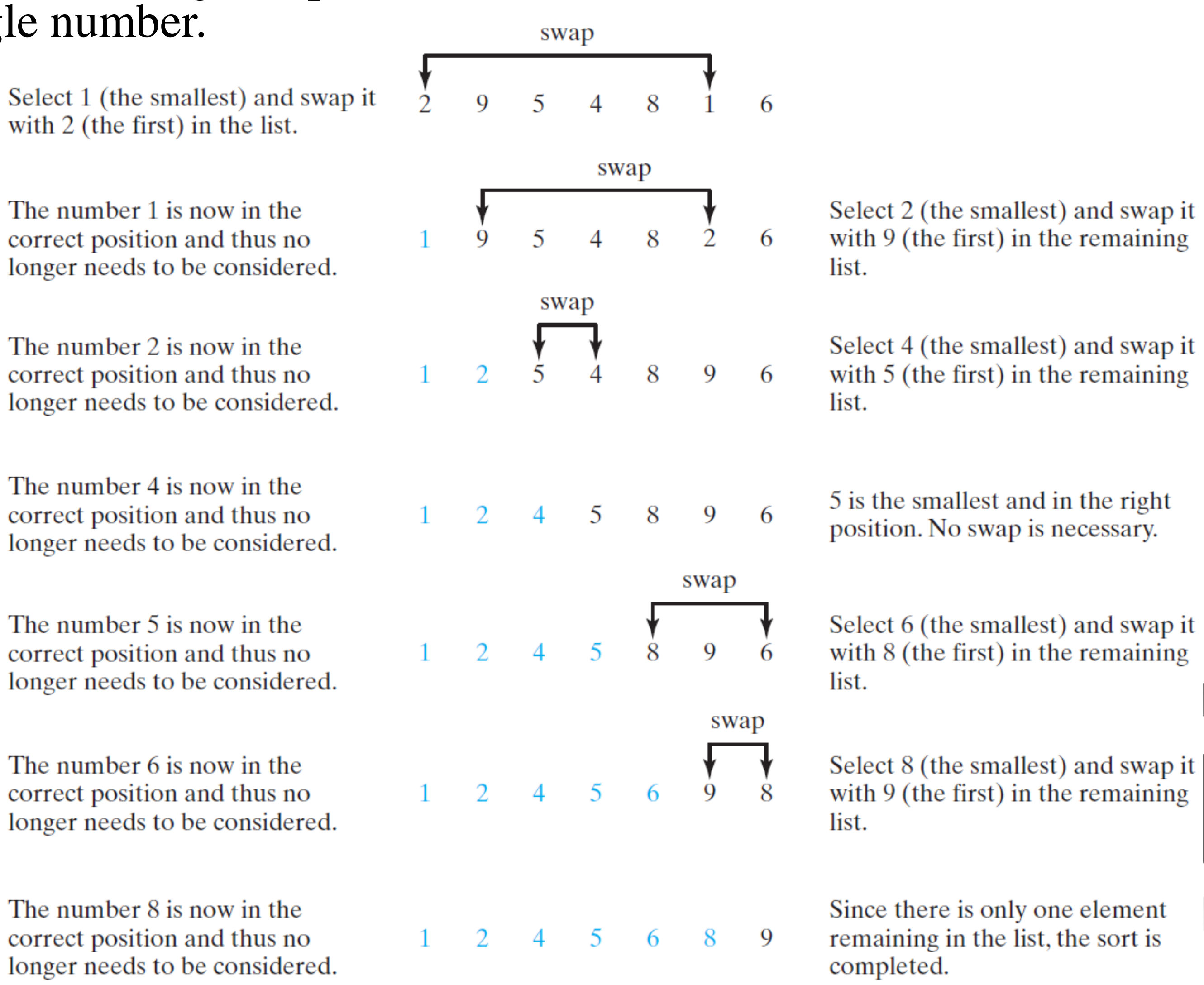
Sorting Arrays

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



Select 1 (the smallest) and swap it with 2 (the first) in the list.

The number 1 is now in the correct position and thus no longer needs to be considered.

Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

The number 2 is now in the correct position and thus no longer needs to be considered.

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.

5 is the smallest and in the right position. No swap is necessary.

The number 5 is now in the correct position and thus no longer needs to be considered.

Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.

The number 6 is now in the correct position and thus no longer needs to be considered.

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.

Since there is only one element remaining in the list, the sort is completed.

From Idea to Solution

```
for (int i = 0; i < list.length; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i+1..listSize-1]
}
```

```
list[0] list[1] list[2] list[3] ...                list[10]

list[0] list[1] list[2] list[3] ...                list[10]

list[0] list[1] list[2] list[3] ...                list[10]

list[0] list[1] list[2] list[3] ...                list[10]

list[0] list[1] list[2] list[3] ...                list[10]

                                ...

list[0] list[1] list[2] list[3] ...                list[10]
```

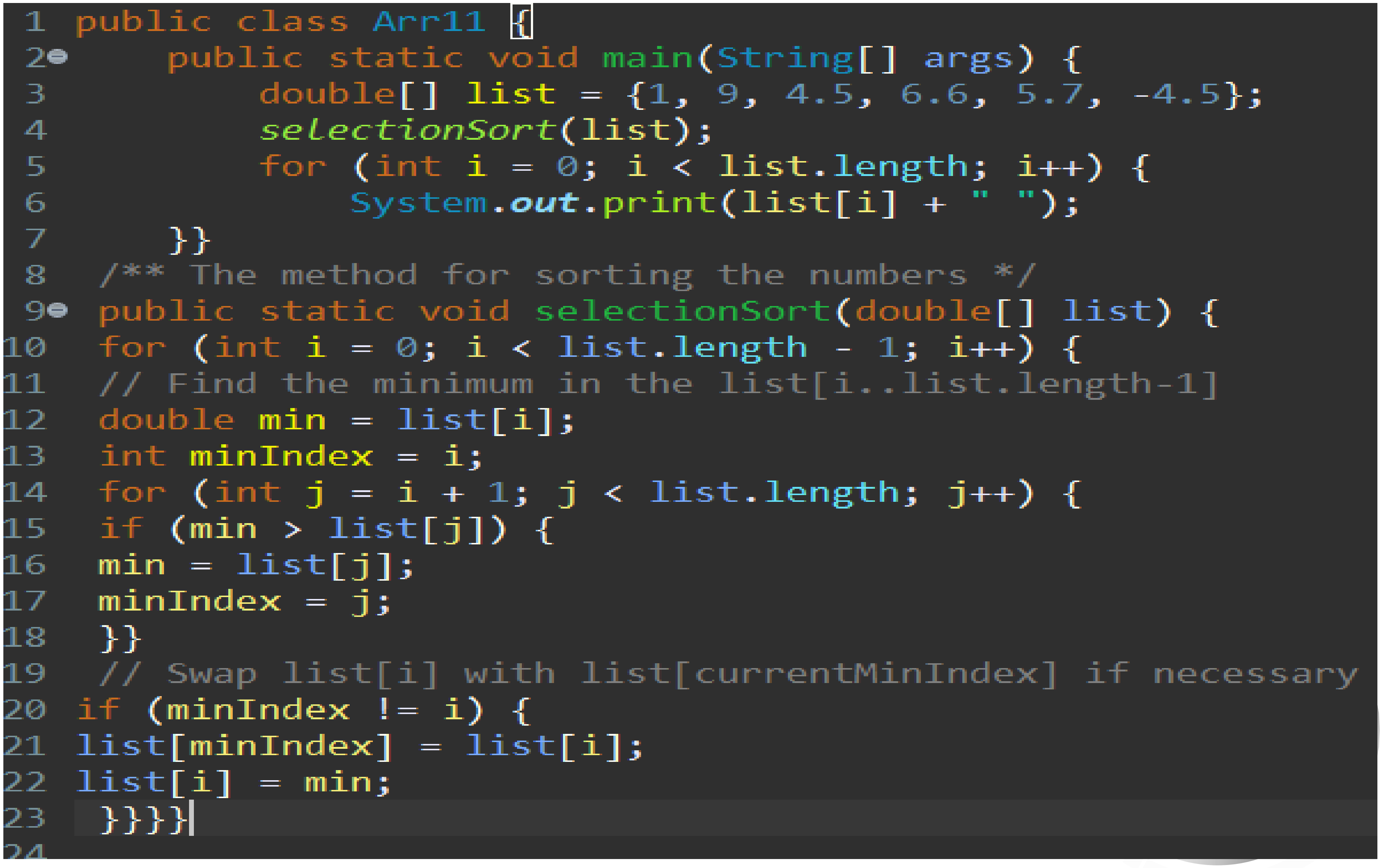
Wrap it in a Method

```
/** 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)

Selection Sort



```
public class Arr11 {
    public static void main(String[] args) {
        double[] list = {1, 9, 4.5, 6.6, 5.7, -4.5};
        selectionSort(list);
        for (int i = 0; i < list.length; i++) {
            System.out.print(list[i] + " ");
    }}
 /** 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 min = list[i];
 int minIndex = i;
 for (int j = i + 1; j < list.length; j++) {
 if (min > list[j]) {
 min = list[j];
 minIndex = j;
 }}
 // Swap list[i] with list[currentMinIndex] if necessary
if (minIndex != i) {
list[minIndex] = list[i];
list[i] = min;
 }}}}
```

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.

Example

```
public static void main(String[] args) {
      double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
      java.util.Arrays.sort(numbers);
       for (int i=0;i<numbers.length;i++)
         System.out.print(numbers[i]+" ");


  }
```

```
1.9 2.9 3.4 3.5 4.4 6.0
```

Another way

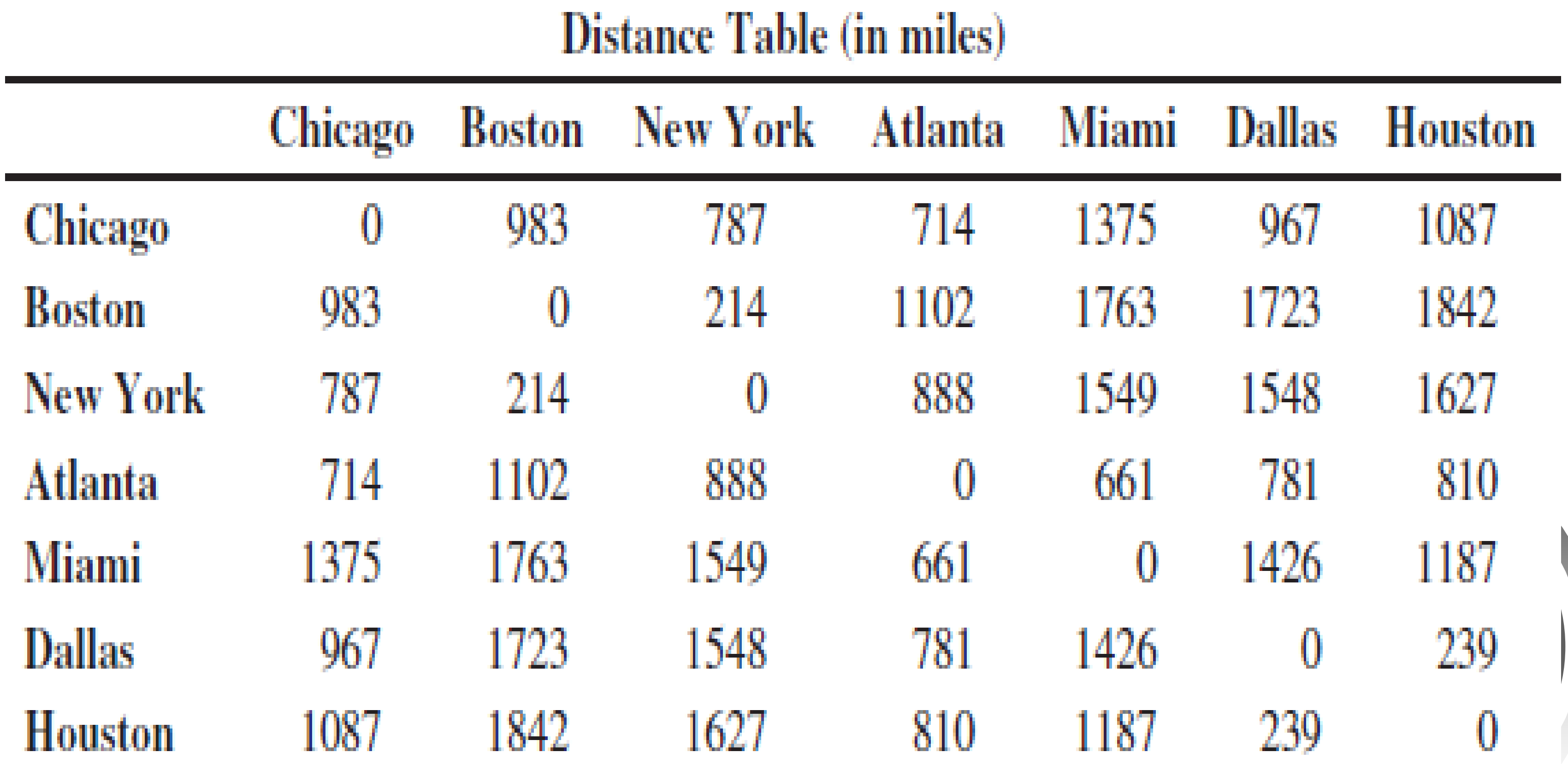
```
import  java.util.Arrays;
Arrays.sort(numbers);
```

Chapter 8 Multidimensional Arrays

- The previous chapter introduced how to use one-dimensional arrays to store linear collections of elements.
- You can use a two-dimensional array to store a matrix or a table.

Multidimensional Arrays

☞ For example, this table that lists the distances between cities can be stored using a two dimensional array named **distances.**



Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Multidimensional Arrays

```
double[][] distances = {
  {0, 983, 787, 714, 1375, 967, 1087},
  {983, 0, 214, 1102, 1763, 1723, 1842},
  {787, 214, 0, 888, 1549, 1548, 1627},
  {714, 1102, 888, 0, 661, 781, 810},
  {1375, 1763, 1549, 661, 0, 1426, 1187},
  {967, 1723, 1548, 781, 1426, 0, 239},
  {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

Declare/Create 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

```
                                   row    column
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

$$\begin{array}{c|c|c|c|c|c|}
 & [0] & [1] & [2] & [3] & [4] \\ \hline
[0] & 0 & 0 & 0 & 0 & 0 \\ \hline
[1] & 0 & 0 & 0 & 0 & 0 \\ \hline
[2] & 0 & 0 & 0 & 0 & 0 \\ \hline
[3] & 0 & 0 & 0 & 0 & 0 \\ \hline
[4] & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

```java
matrix = new int[5][5];
```

(a)

$$\begin{array}{c|c|c|c|c|c|}
 & [0] & [1] & [2] & [3] & [4] \\ \hline
[0] & 0 & 0 & 0 & 0 & 0 \\ \hline
[1] & 0 & 0 & 0 & 0 & 0 \\ \hline
[2] & 0 & 7 & 0 & 0 & 0 \\ \hline
[3] & 0 & 0 & 0 & 0 & 0 \\ \hline
[4] & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}$$

```java
matrix[2][1] = 7;
```

(b)

$$\begin{array}{c|c|c|c|}
 & [0] & [1] & [2] \\ \hline
[0] & 1 & 2 & 3 \\ \hline
[1] & 4 & 5 & 6 \\ \hline
[2] & 7 & 8 & 9 \\ \hline
[3] & 10 & 11 & 12 \\ \hline
\end{array}$$

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

(c)

matrix.length? 5

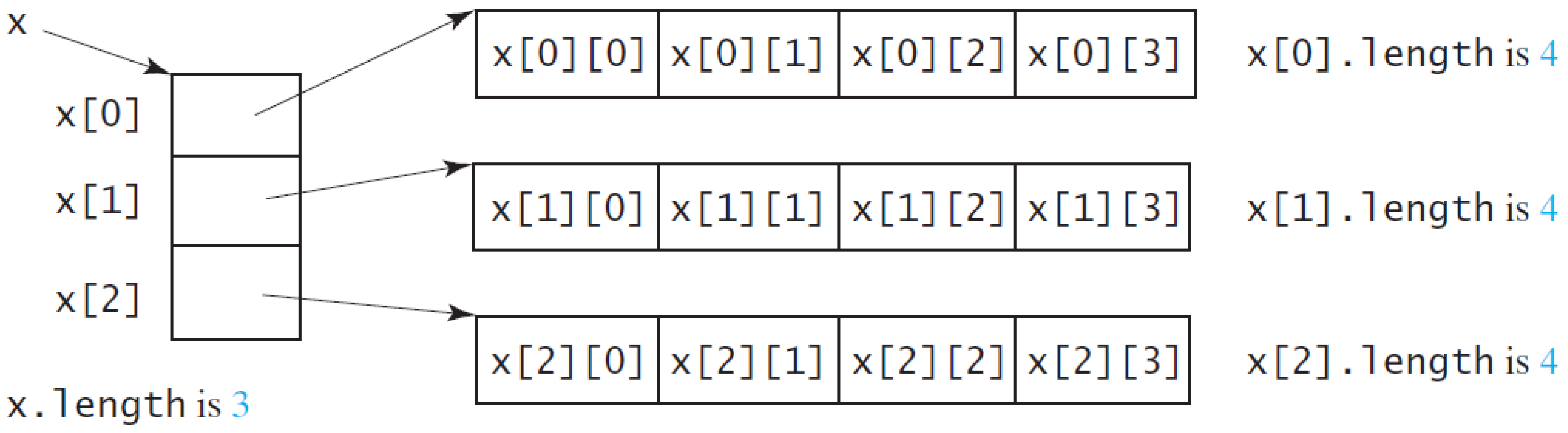
matrix[0].length? 5

array.length? 4

array[0].length? 3

Lengths of Two-dimensional Arrays

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



x
x[0]
x[1]
x[2]
x[0][0] x[0][1] x[0][2] x[0][3]
x[1][0] x[1][1] x[1][2] x[1][3]
x[2][0] x[2][1] x[2][2] x[2][3]
x[0].length is 4
x[1].length is 4
x[2].length is 4
x.length is 3

Ragged Arrays

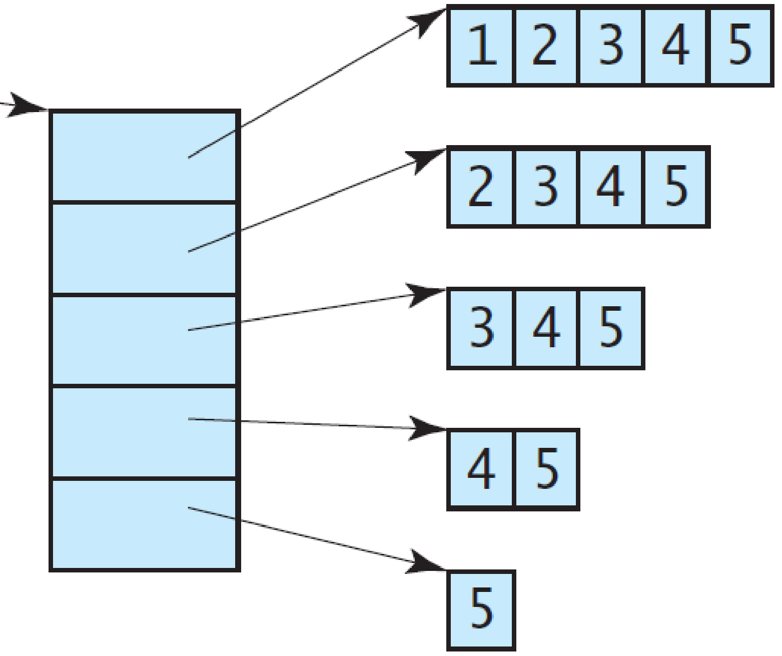
Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
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.

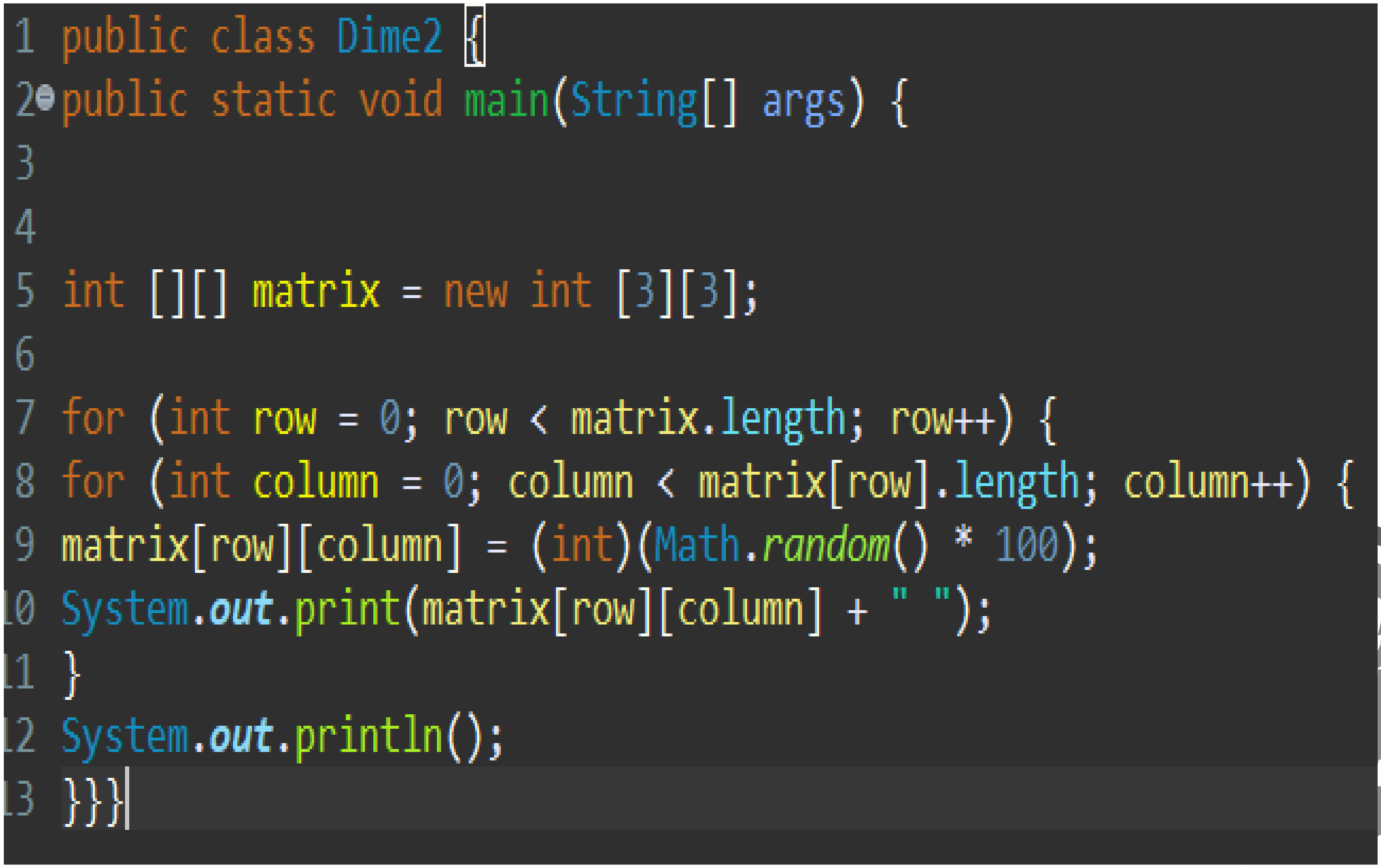
```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Initializing arrays with random values

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Two-dimensional Array with random values



```
public class Dime2 {
public static void main(String[] args) {


int [][] matrix = new int [3][3];

for (int row = 0; row < matrix.length; row++) {
for (int column = 0; column < matrix[row].length; column++) {
matrix[row][column] = (int)(Math.random() * 100);
System.out.print(matrix[row][column] + " ");
}
System.out.println();
}}}
```

Summing all elements

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

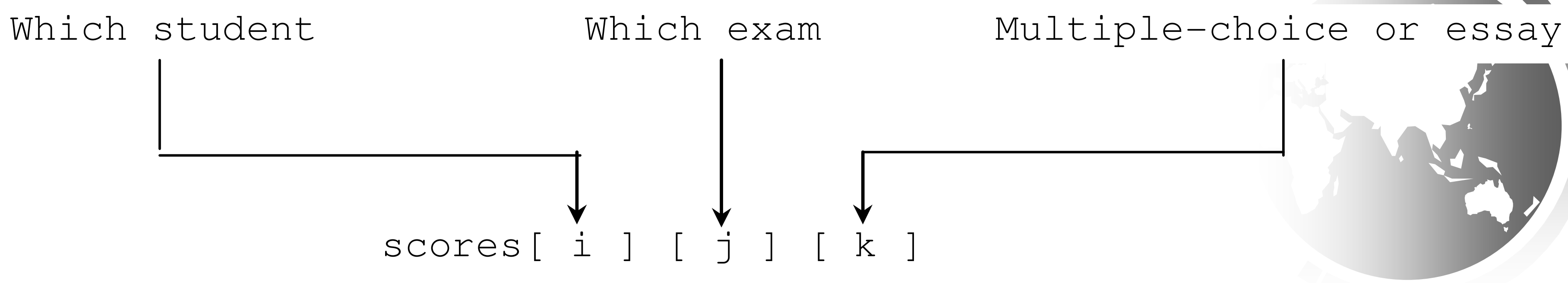
Multidimensional Arrays

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.

```
double[][][] scores = new double[6][5][2];
```

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}
};
```


Which student
Which exam
Multiple-choice or essay
scores[i] [j] [k]