# ARRAYS IN CLASSES AND METHODS

Ch 7.2

## Arrays in Classes and Methods: Outline

- Common operations: printing, average, etc
- Arrays of Strings
- Case Study: Sales Report
- Indexed Variables as Method Arguments
- Entire Arrays as Arguments to a Method
- ~~Arguments for the Method main~~
- Array Assignment and Equality
- Methods that Return Arrays

# Printing an array

- Consider an array list:

```
int list[ ] = new int[5];
list[0] = 50;
list[3] = 70;
```

- What happens if we print the array name ?

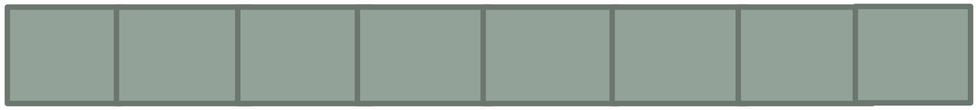
```
System.out.print(list + " ");
```

- How do we print the whole array?

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

## Common operations

```
int sales[]={                                  };
```



- Sum and average

```
int sum = 0; double average = 0.0;
for (int index = 0; index < sales.length; index++)
    sum = sum + sales[index];
if (sales.length != 0)
    average = sum / sales.length;
```

- Finding index of largest number

```
maxIndex = 0;
for (int index = 1; index < sales.length; index++)
    if (sales[maxIndex] < sales[index])
       maxIndex = index;
int largestSale = sales[maxIndex];
```

## Common operations

```
int sales[]={12, 32, 4, 55, 1, 23, 17, 30};
```

- Searching for a specific value

```
int searchItem = 10;   // what if searchItem = 4 ?
int loc = 0;
boolean found = false;

while (loc < sales.length && !found)
   if (sales[loc] == searchItem)
         found = true;
   else
         loc++;

if (found)
   System.out.print(loc);
else
   System.out.print("not found");
```

## ARRAY OF Strings

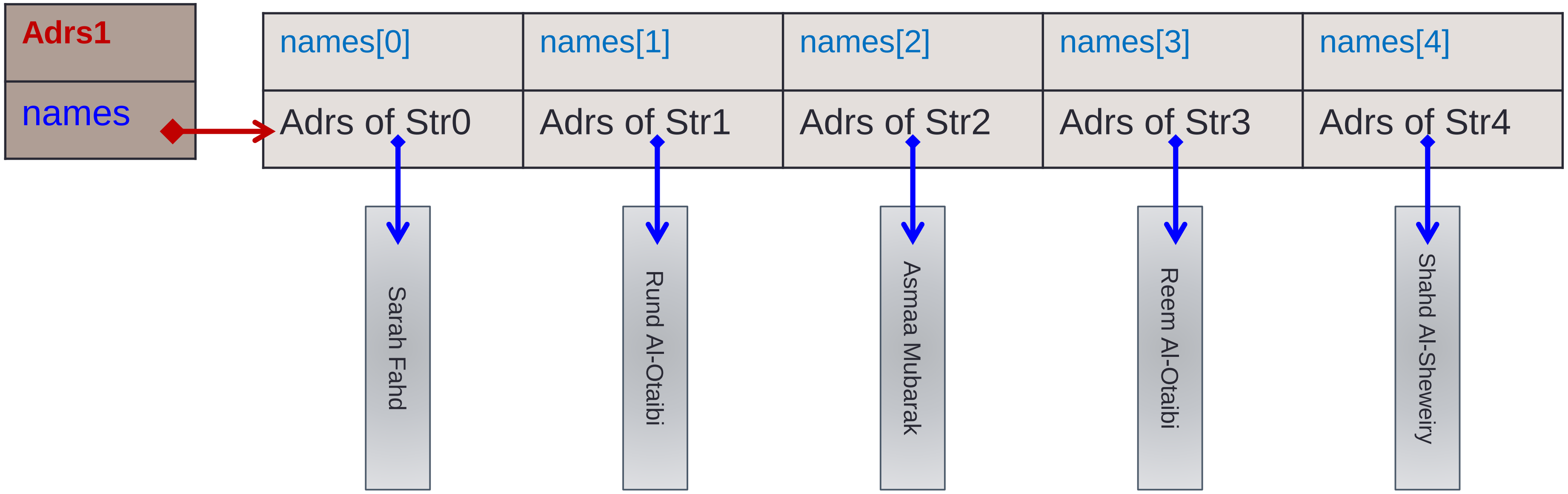
```
String[] names = new String [5];      //declaration
//Fill in the array
names[0] = “Sarah Fahd”;
names[1] = “Rund Al-Otaibi”;
names[2] = “Asmaa Mubarak”;
names[3] = “Reem Al-Otaibi”;
names[4] = “Hind Al-Tamimi”;
```

- The first statement declares an array names of size 5.
- Each element of names is a String.
- Remember that a String stores an address rather than a value.

- Naturally, all String methods that we previously studied can be applied on EACH element of the String array.

# ARRAY OF Strings

- Therefore, after the execution of the previous code segment, each array element contains an address that points to (refers to) the corresponding String.
- The memory layout will be as follows:



# ARRAY OF Strings

- String methods may be applied on each element of the String array.
- The following code segment applies a few methods on each String of the array names:

```
String[] names = new String [5];       //declaration
//Fill in the array
names[4] = "Hind Al-Tamimi";
for (index = 0; index < names.length; index++)
   {

     System.out.println (names[index]);

     System.out.println (names[index].length());

     System.out.println (names[index].substring(0, 5));

     System.out.println (names[index].toUpperCase());
   }
```

## ARRAY OF Strings

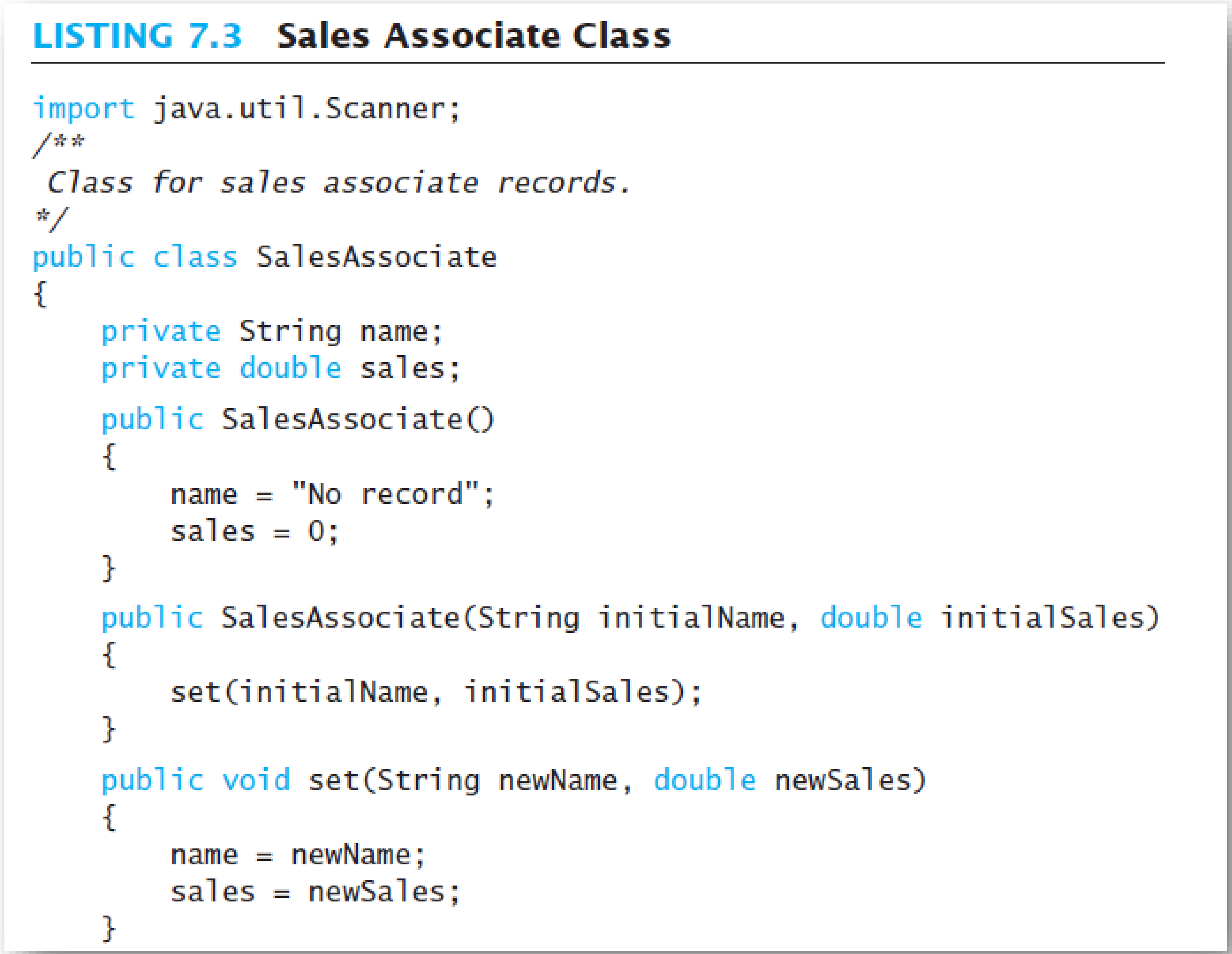
- String methods may be applied on each element of the String array.
- The following code segment applies a few methods on each String of the array names:

```
String[] names = new String [5];      //declaration
//Fill in the array
names[4] = “Hind Al-Tamimi”;
for (index = 0; index < names.length; index++)
  {
   // print the stored names
   System.out.println (names[index]);
   // print the length of each name
   System.out.println (names[index].length());
   //extracts the first four letters
   System.out.println (names[index].substring(0, 5));
   // converts the string into upper case
   System.out.println (names[index].toUpperCase());
  }
```

## Case Study: Sales Report

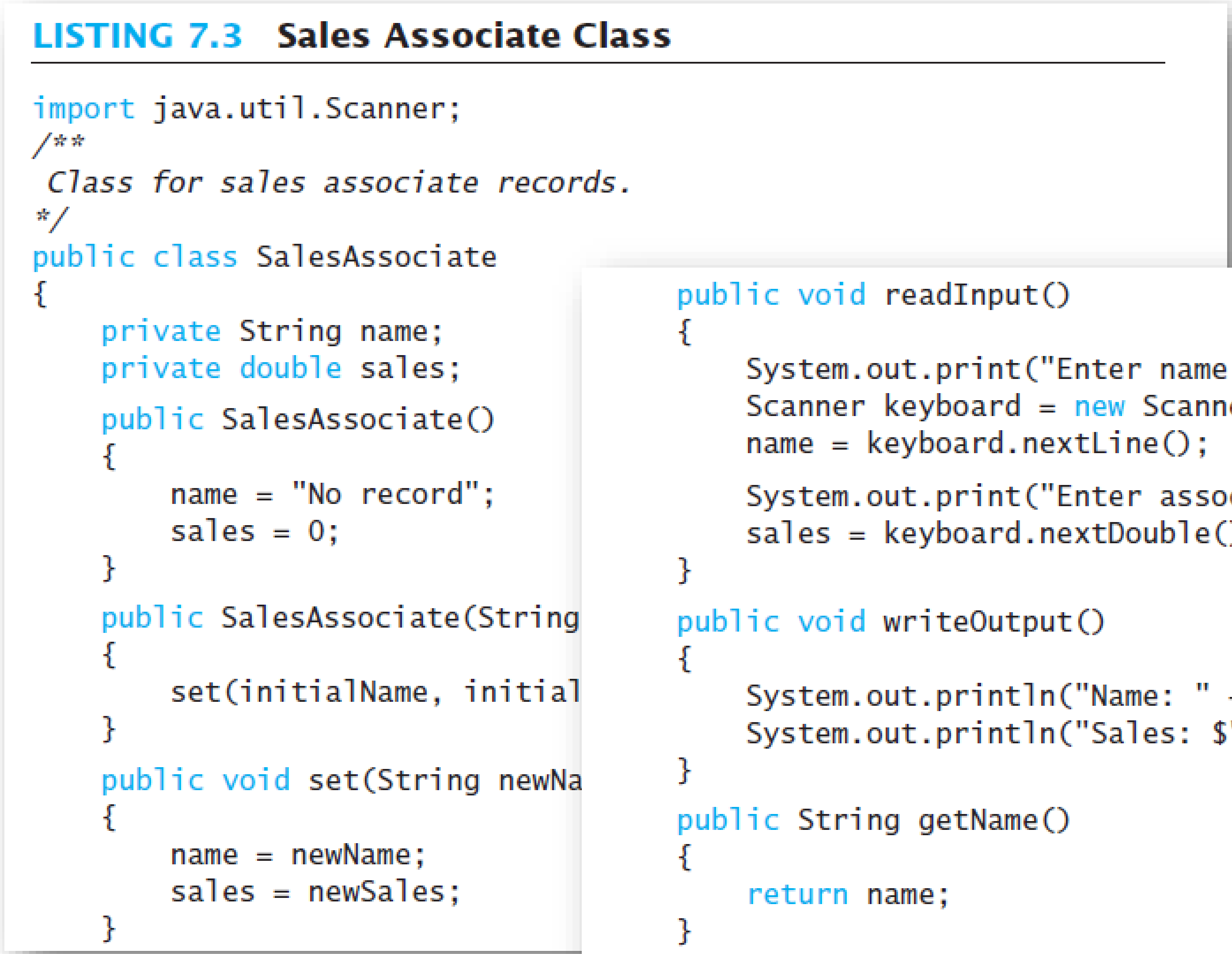
- Program to generate a sales report
- Class will contain
  - Name
  - Sales figure
- View class declaration, listing 7.3

```
class SalesAssociate
```



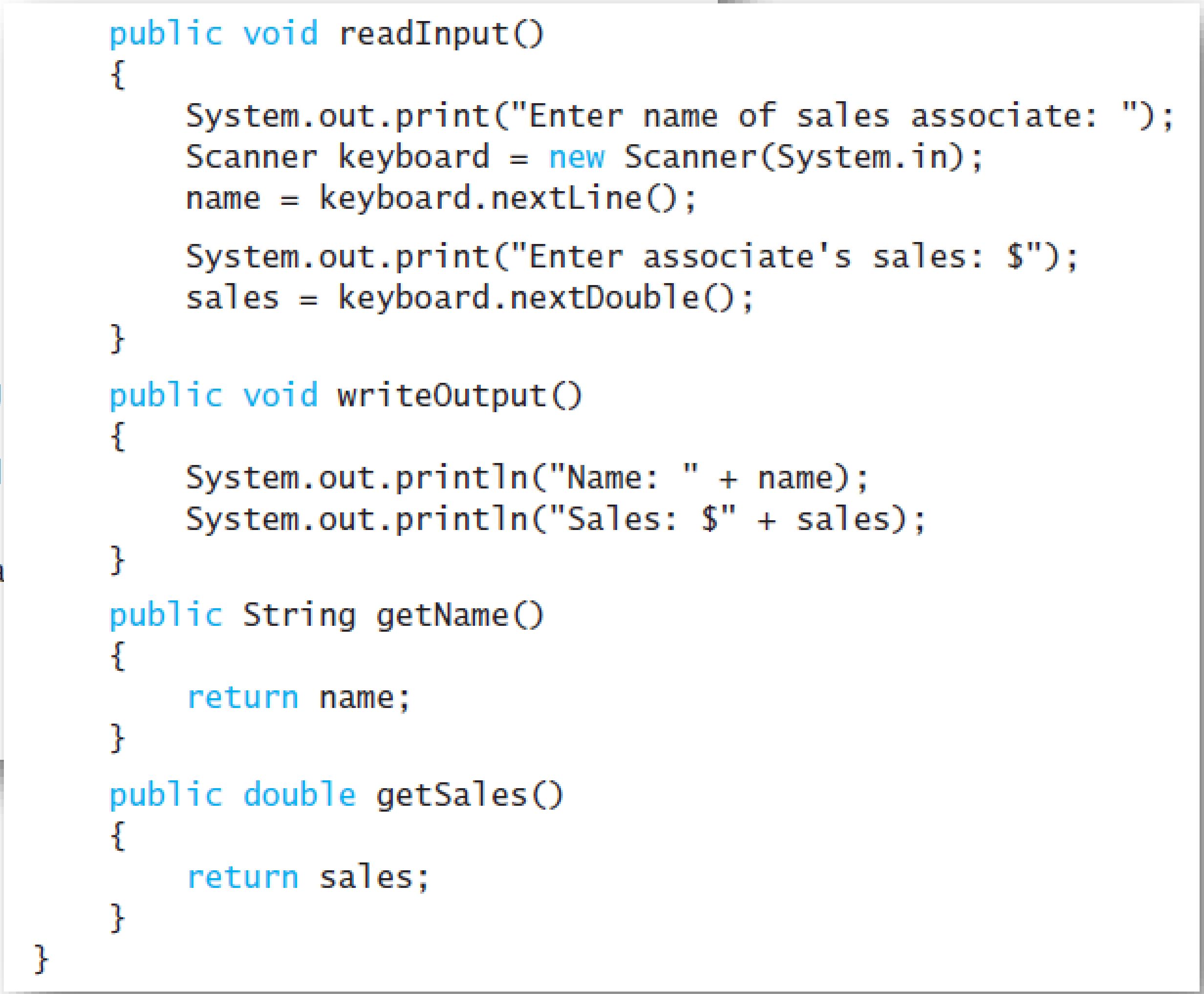
## LISTING 7.3 Sales Associate Class

```
import java.util.Scanner;
/**
 Class for sales associate records.
*/
public class SalesAssociate
{
    private String name;
    private double sales;

    public SalesAssociate()
    {
        name = "No record";
        sales = 0;
    }

    public SalesAssociate(String initialName, double initialSales)
    {
        set(initialName, initialSales);
    }

    public void set(String newName, double newSales)
    {
        name = newName;
        sales = newSales;
    }
```



**LISTING 7.3 Sales Associate Class**

```
import java.util.Scanner;
/**
 Class for sales associate records.
*/
public class SalesAssociate
{
    private String name;
    private double sales;

    public SalesAssociate()
    {
        name = "No record";
        sales = 0;
    }

    public SalesAssociate(String
    {
        set(initialName, initial
    }

    public void set(String newNa
    {
        name = newName;
        sales = newSales;
    }
```



```
    public void readInput()
    {
        System.out.print("Enter name of sales associate: ");
        Scanner keyboard = new Scanner(System.in);
        name = keyboard.nextLine();

        System.out.print("Enter associate's sales: $");
        sales = keyboard.nextDouble();
    }

    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Sales: $" + sales);
    }

    public String getName()
    {
        return name;
    }

    public double getSales()
    {
        return sales;
    }
}
```

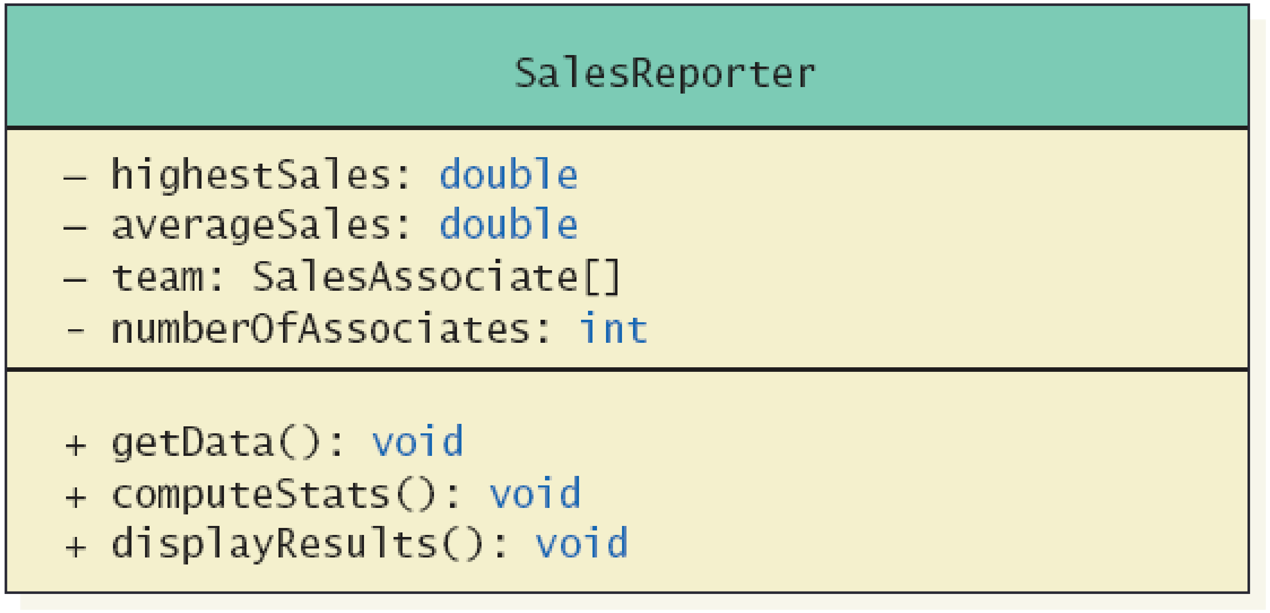
## Case Study: Sales Report

Main subtasks for our program

1. Get ready
2. Obtain the data
3. Compute some statistics (update instance variables)
4. Display the results

## Case Study: Sales Report

- Figure 7.3 Class diagram for class **`SalesReporter`**



## LISTING 7.4 A Sales Report Program *(part 1 of 3)*

```
import java.util.Scanner;
/**
 Program to generate sales report.
*/
public class SalesReporter
{
    private double highestSales;
    private double averageSales;
    private SalesAssociate[] team;   //The array object is
                                     //created in getData.
    private int numberOfAssociates; //Same as team.length
    /**
     Reads the number of sales associates and data for each one.
    */
    public void getData()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter number of sales associates:");
        numberOfAssociates = keyboard.nextInt();
        team = new SalesAssociate[numberOfAssociates + 1];
        for (int i = 1; i <= numberOfAssociates; i++)
        {
            team[i] = new SalesAssociate();
            System.out.println("Enter data for associate " + i);
            team[i].readInput();
            System.out.println();
        }
    }
```

*The `main` method is at the end of the class.*

*Array object created here.*

`SalesAssociate` *objects created here.*

```java
/**
 Computes the average and highest sales figures.
 Precondition: There is at least one salesAssociate.
*/
public void computeStats()
{
    double nextSales = team[1].getSales();
    highestSales = nextSales;
    double sum = nextSales;
    for (int i = 2; i <= numberOfAssociates; i++)
    {
        nextSales = team[i].getSales();
        sum = sum + nextSales;
        if (nextSales > highestSales)
            highestSales = nextSales; //highest sales so far.
    }
    averageSales = sum / numberOfAssociates;
}
```

*Already processed* `team[1]`, *so the loop starts with* `team[2]`.

```java
/**
 Displays sales report on the screen.
*/
public void displayResults()
{
    System.out.println("Average sales per associate is $" +
                       averageSales);
    System.out.println("The highest sales figure is $" +
                       highestSales);
    System.out.println();
    System.out.println("The following had the highest sales:");
    for (int i = 1; i <= numberOfAssociates; i++)
    {
        double nextSales = team[i].getSales();
        if (nextSales == highestSales)
        {
            team[i].writeOutput();
            System.out.println("$" + (nextSales - averageSales)
                               + " above the average.");
            System.out.println();
        }
    }
```

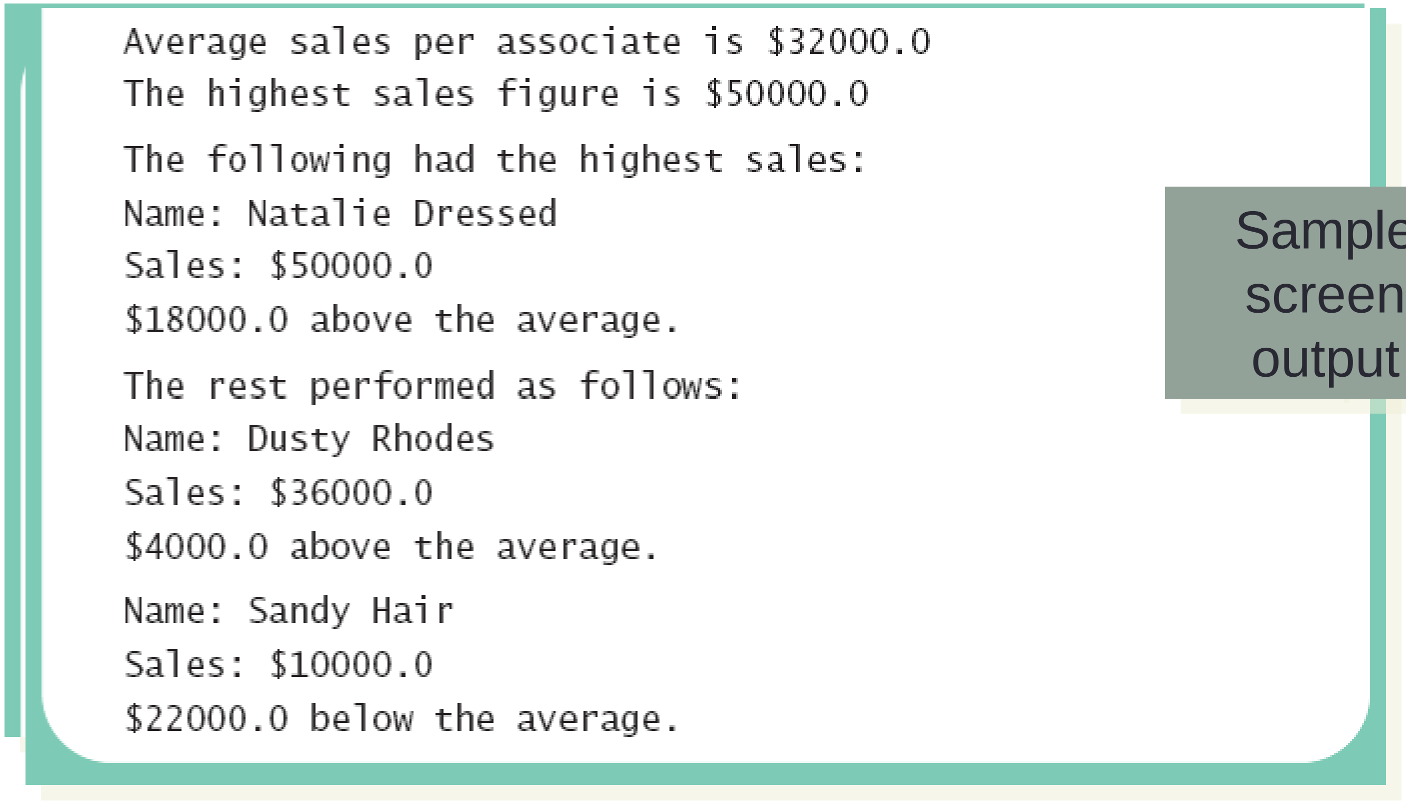
```
            System.out.println("The rest performed as follows:");
            for (int i = 1; i <= numberOfAssociates; i++)
            {
                double nextSales = team[i].getSales();
                if (team[i].getSales() != highestSales)
                {
                    team[i].writeOutput();
                    if (nextSales >= averageSales)
                        System.out.println("$" + (nextSales -
                                averageSales) + " above the average.");
                    else
                        System.out.println("$" + (averageSales -
                                   nextSales) + " below the average.");
                    System.out.println();
                }
            }
        }
        public static void main(String[] args)
        {
            SalesReporter clerk = new SalesReporter();
            clerk.getData();
            clerk.computeStats();
            clerk.displayResults();
        }
    }
```

```
/**
 Displays sales report on the screen.
*/
public void displayResults()
{
    System.out.println("Average sales per associate is $" +
                       averageSales);
    System.out.println("The highest sales figure is $" +
                       highestSales);
    System.out.println();
    System.out.println("The following had the highest sales:");
    for (int i = 1; i <= numberOfAssociates; i++)
    {
        double nextSales = team[i].getSales();
        if (nextSales == highestSales)
        {
            team[i].writeOutput();
            System.out.println("$" + (nextSales - averageSales)
                                  + " above the average.");
            System.out.println();
        }
    }
```

```
    System.out.println("The rest performed as follows:");
    for (int i = 1; i <= numberOfAssociates; i++)
    {
        double nextSales = team[i].getSales();
        if (team[i].getSales() != highestSales)
        {
            team[i].writeOutput();
            if (nextSales >= averageSales)
                System.out.println("$" + (nextSales -
                       averageSales) + " above the aver
            else
                System.out.println("$" + (averageSales -
                          nextSales) + " below the aver
            System.out.println();
        }
    }
  }
  public static void main(String[] args)
  {
      SalesReporter clerk = new SalesReporter();
      clerk.getData();
      clerk.computeStats();
      clerk.displayResults();
  }
}
```

# Case Study: Sales Report

- View sales report program, listing 7.4
  **`class SalesReporter`**



```
Average sales per associate is $32000.0
The highest sales figure is $50000.0

The following had the highest sales:
Name: Natalie Dressed
Sales: $50000.0
$18000.0 above the average.

The rest performed as follows:
Name: Dusty Rhodes
Sales: $36000.0
$4000.0 above the average.

Name: Sandy Hair
Sales: $10000.0
$22000.0 below the average.
```

Sample screen output

## Indexed Variables as Method Arguments

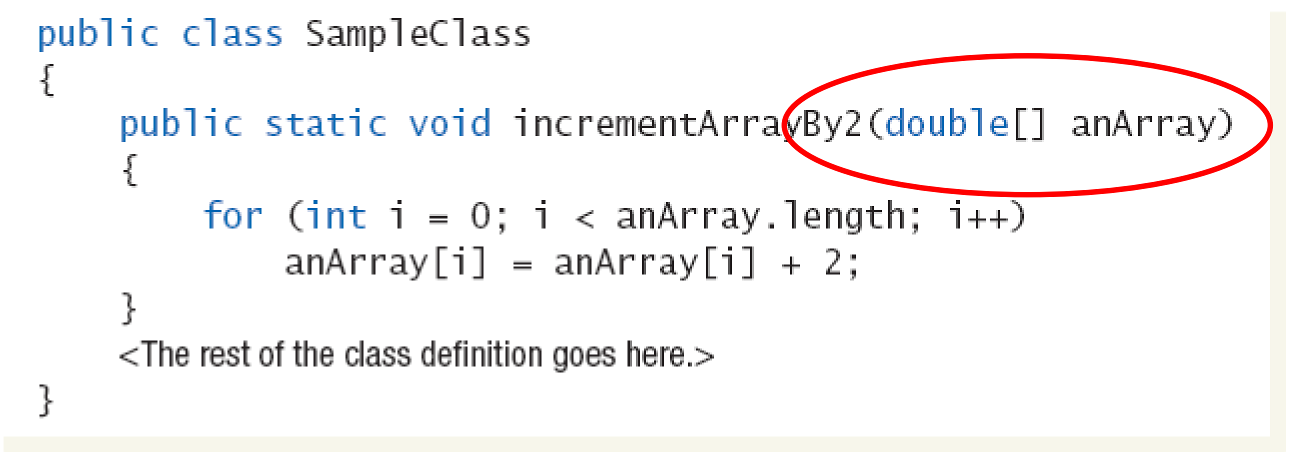
- Indexed variable of an array
  - Example … **`a[i]`**
  - Can be used anywhere a variable of the array base type can be used
- View program using indexed variable as an argument, listing 7.5
  **`class ArgumentDemo`**

## LISTING 7.5 Indexed Variables as Arguments

```
import java.util.Scanner;

/**
 A demonstration of using indexed variables as arguments.
*/
public class ArgumentDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt();
        int[] nextScore = new int[3];

        for (int i = 0; i < nextScore.length; i++)
            nextScore[i] = firstScore + 5 * i;

        for (int i = 0; i < nextScore.length; i++)
        {
            double possibleAverage =
                            getAverage(firstScore, nextScore[i]);
            System.out.println("If your score on exam 2 is " +
                                nextScore[i]);
            System.out.println("your average will be " +
                                possibleAverage);
        }
    }

    public static double getAverage(int n1, int n2)
    {
        return (n1 + n2) / 2.0;
    }
}
```

## Entire Arrays as Arguments

- Declaration of array parameter similar to how an array is declared
- Example:



```
public class SampleClass
{
    public static void incrementArrayBy2(double[] anArray)
    {
        for (int i = 0; i < anArray.length; i++)
            anArray[i] = anArray[i] + 2;
    }
    <The rest of the class definition goes here.>
}
```

## Entire Arrays as Arguments

- Note – array parameter in a method heading does not specify the length
  - An array of any length can be passed to the method
  - Inside the method, elements of the array can be changed
- When you pass the entire array, do not use square brackets in the actual parameter
  - For example:

```
double[] myArray = {1,2,3};
incrementArrayBy2(myArray);
```

## Array Assignment and Equality

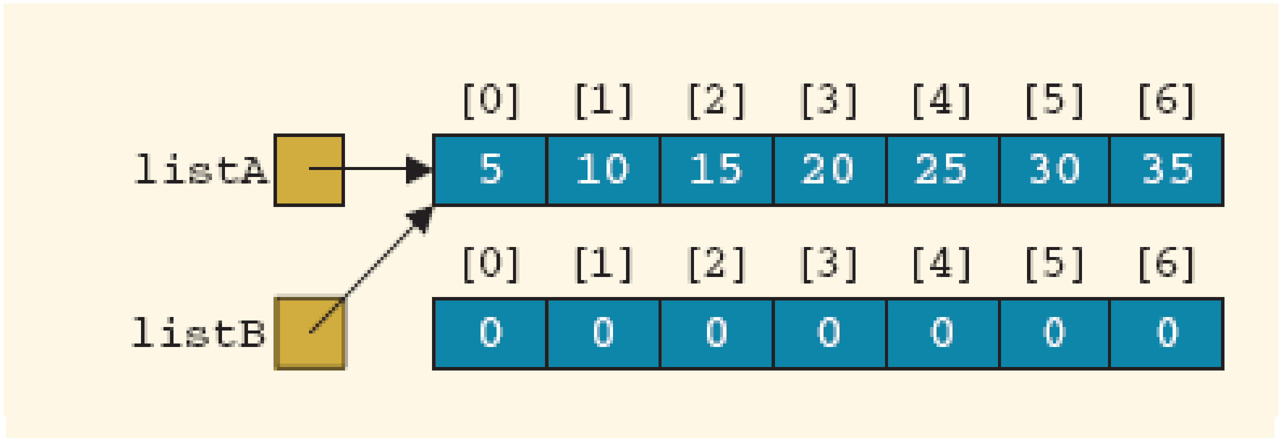
- Arrays are objects
  - Assignment and equality operators behave (misbehave) as specified in previous chapter
- Variable for the array object contains memory address of the object
  - Assignment operator **=** copies this address
  - Equality operator **==** tests whether two arrays are stored in same place in memory

## Assinging vs. Copying Arrays of same size

- Consider two arrays: `listA` and `listB` as shown
- Assume we want to make the content of `listB` a copy of `listA`
- Therefore, we wrote the following:

```
listB = listA;
```

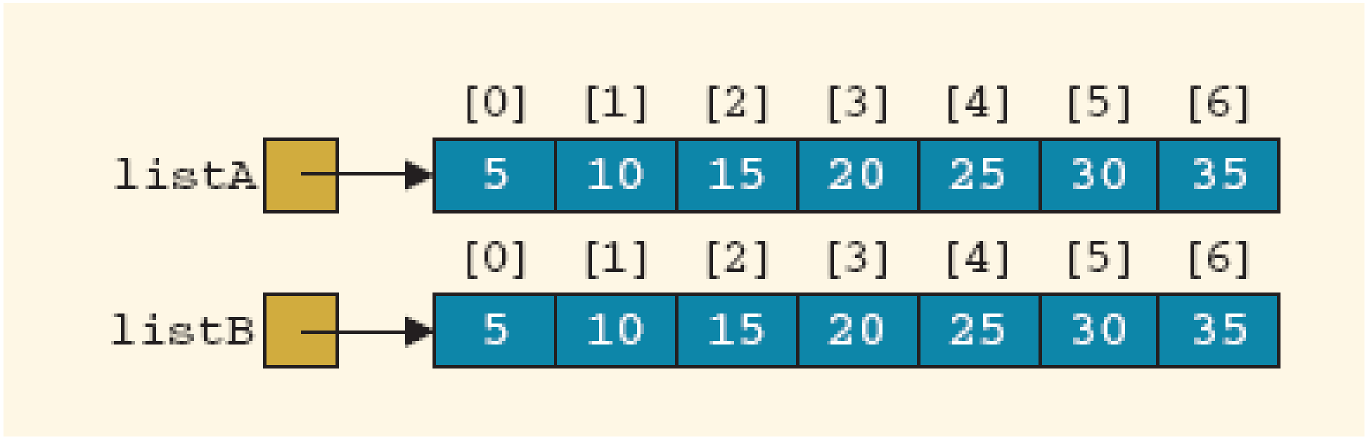
- Will that work correctly?
- No, because it only assigns the address of `listA` to `listB`



## Copying Arrays of same size

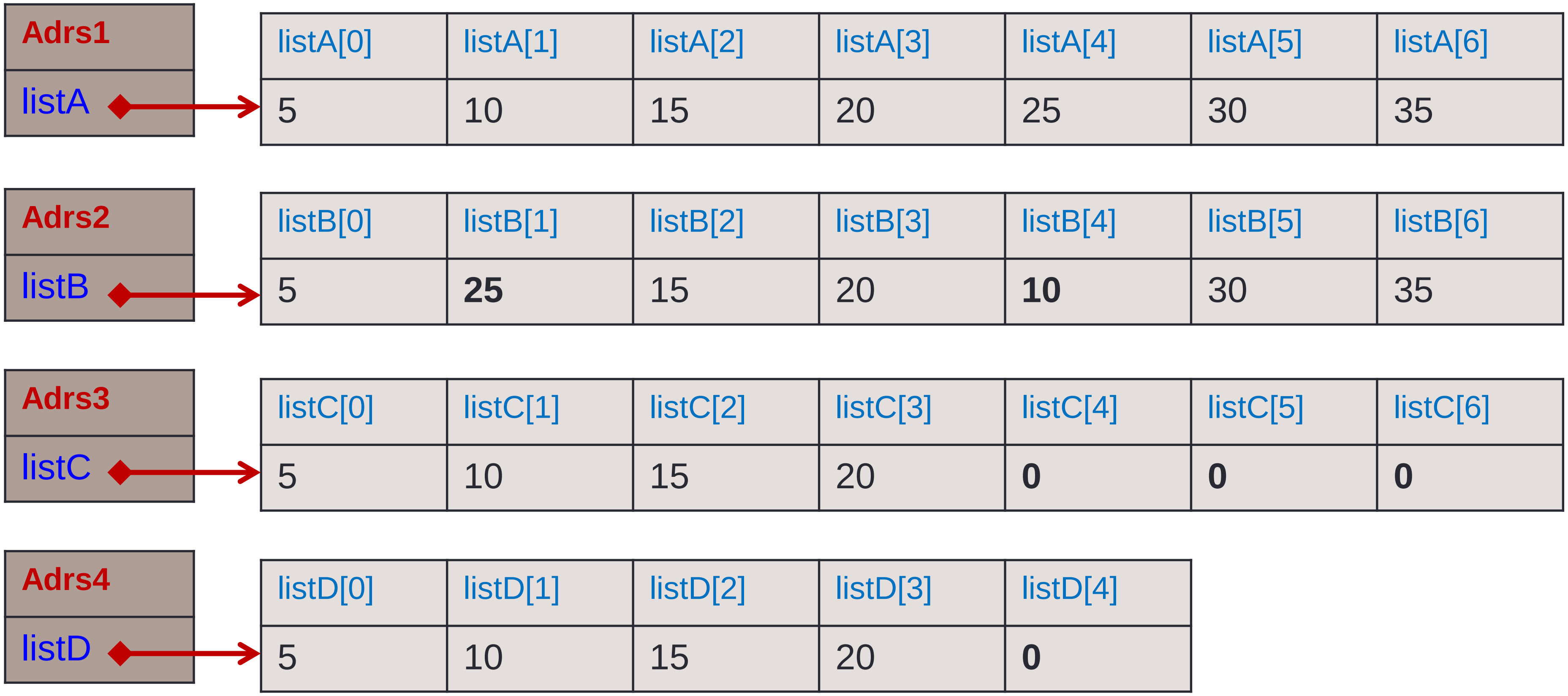
- This is the correct way to copy the contents:

```
for (int index = 0; index < listA.length; index++)
    listB[index] = listA[index];
```



## EQUALITY OF TWO ARRAYS

- Two arrays are considered equal if:
  - They have the same size, and
  - All corresponding elements in both arrays are equal.



| listA[0] | listA[1] | listA[2] | listA[3] | listA[4] | listA[5] | listA[6] |
|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | 25 | 30 | 35 |

| listB[0] | listB[1] | listB[2] | listB[3] | listB[4] | listB[5] | listB[6] |
|---|---|---|---|---|---|---|
| 5 | **25** | 15 | 20 | **10** | 30 | 35 |

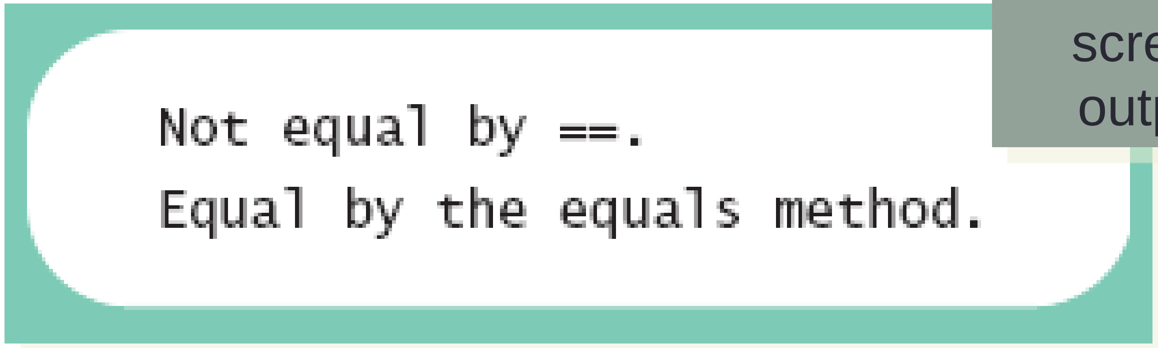
| listC[0] | listC[1] | listC[2] | listC[3] | listC[4] | listC[5] | listC[6] |
|---|---|---|---|---|---|---|
| 5 | 10 | 15 | 20 | **0** | **0** | **0** |

| listD[0] | listD[1] | listD[2] | listD[3] | listD[4] |
|---|---|---|---|---|
| 5 | 10 | 15 | 20 | **0** |

# Array Assignment and Equality

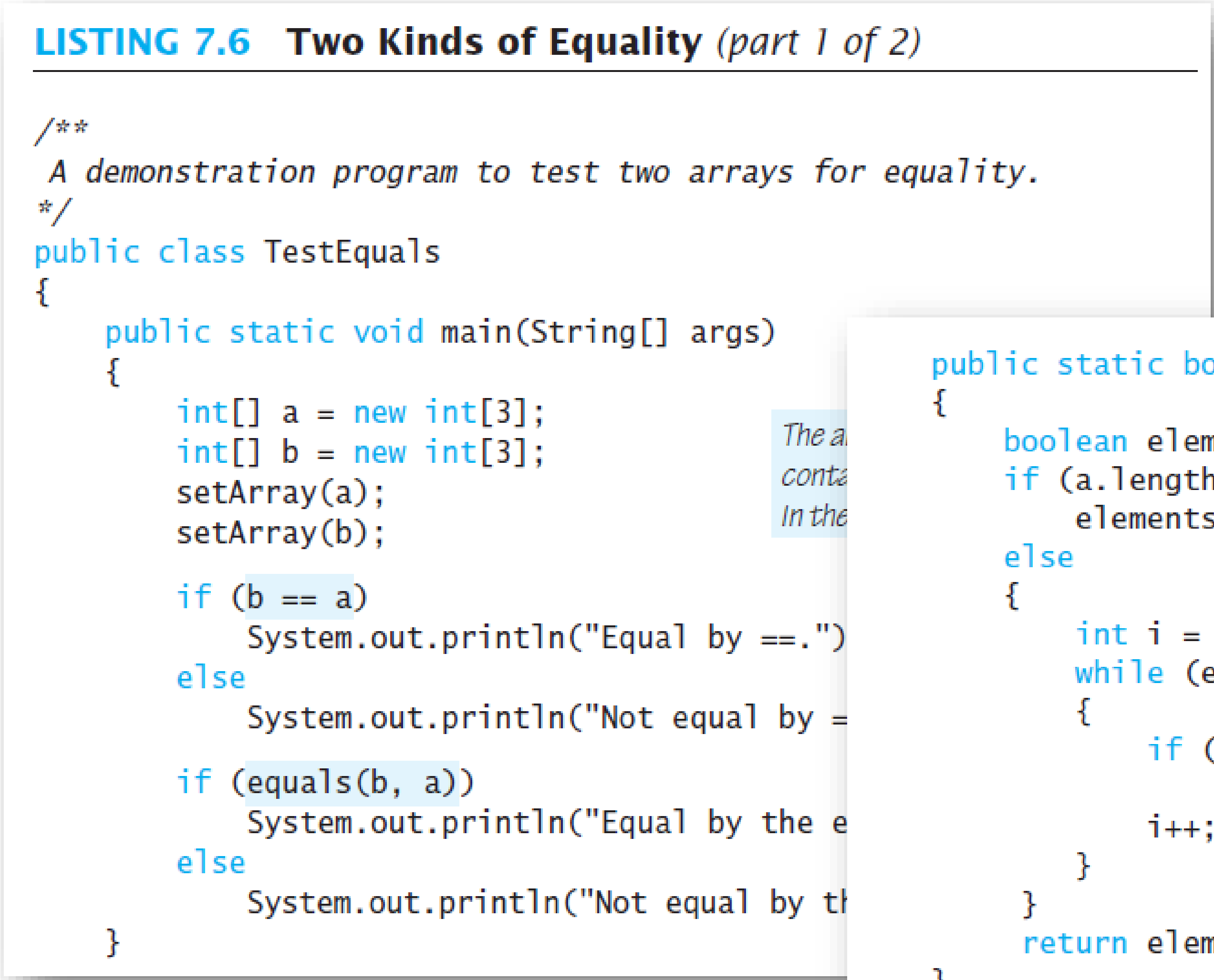
- Two kinds of equality
- View example program, listing 7.6

**`class TestEquals`**



```
Not equal by ==.
Equal by the equals method.
```

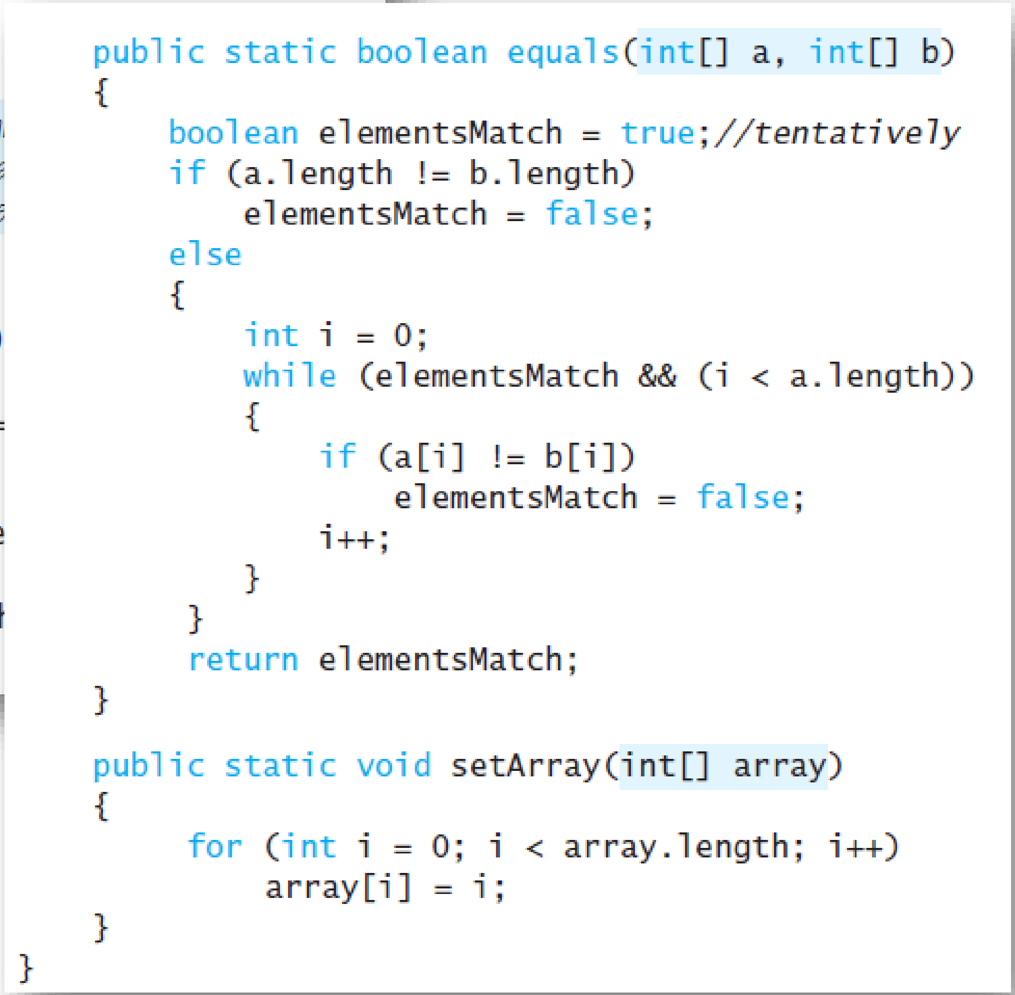
Sample screen output



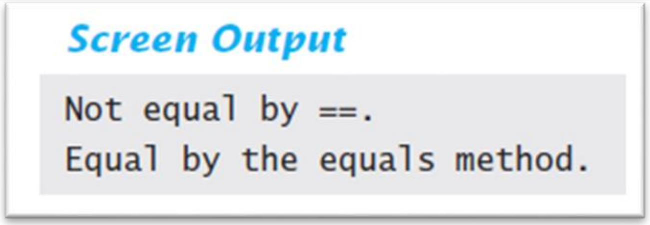
**LISTING 7.6 Two Kinds of Equality** *(part 1 of 2)*

```
/**
 A demonstration program to test two arrays for equality.
*/
public class TestEquals
{
    public static void main(String[] args)
    {
        int[] a = new int[3];
        int[] b = new int[3];
        setArray(a);
        setArray(b);

        if (b == a)
            System.out.println("Equal by ==.")
        else
            System.out.println("Not equal by =

        if (equals(b, a))
            System.out.println("Equal by the e
        else
            System.out.println("Not equal by th
    }
```

*The a*
*conta*
*In the*



```
    public static boolean equals(int[] a, int[] b)
    {
        boolean elementsMatch = true;//tentatively
        if (a.length != b.length)
            elementsMatch = false;
        else
        {
            int i = 0;
            while (elementsMatch && (i < a.length))
            {
                if (a[i] != b[i])
                    elementsMatch = false;
                i++;
            }
         }
         return elementsMatch;
    }

    public static void setArray(int[] array)
    {
         for (int i = 0; i < array.length; i++)
             array[i] = i;
    }
}
```



***Screen Output***

```
Not equal by ==.
Equal by the equals method.
```

## Array Assignment and Equality

- Note results of **`==`**
- Note definition and use of method **`equals`**
  - Receives two array parameters
  - Checks length and each individual pair of array elements
- Remember array types are reference types

## Gotcha – Don’t Exceed Array Bounds

- The code below fails if the user enters a number like 4.  Use input validation.

```
Scanner kbd = new Scanner(System.in);
int[] count = {0,0,0,0};

System.out.println("Enter ten numbers between 0 and 3.");
for (int i = 0; i < 10; i++)
{
 int num = kbd.nextInt();
 count[num]++;
}
for (int i = 0; i < count.length; i++)
   System.out.println("You entered " + count[i] + " " + i + "'s");
```

## Gotcha – Creating an Array of Objects

- When you create an array of objects Java does not create instances of any of the objects!
- For example, consider the code:

```
SalesAssociate[] team = new SalesAssociate[10];
System.out.println(team[0].getName()); // ERROR - why?
```

  - We can NOT access `team[0]` yet; it is **null**.

  - First we must create references to an object:

```
team[0] = new SalesAssociate("Jane Doe", 5000);
team[1] = new SalesAssociate("John Doe", 5000);
```

  - we can now access team[0].getName() or team[1].getSalary()

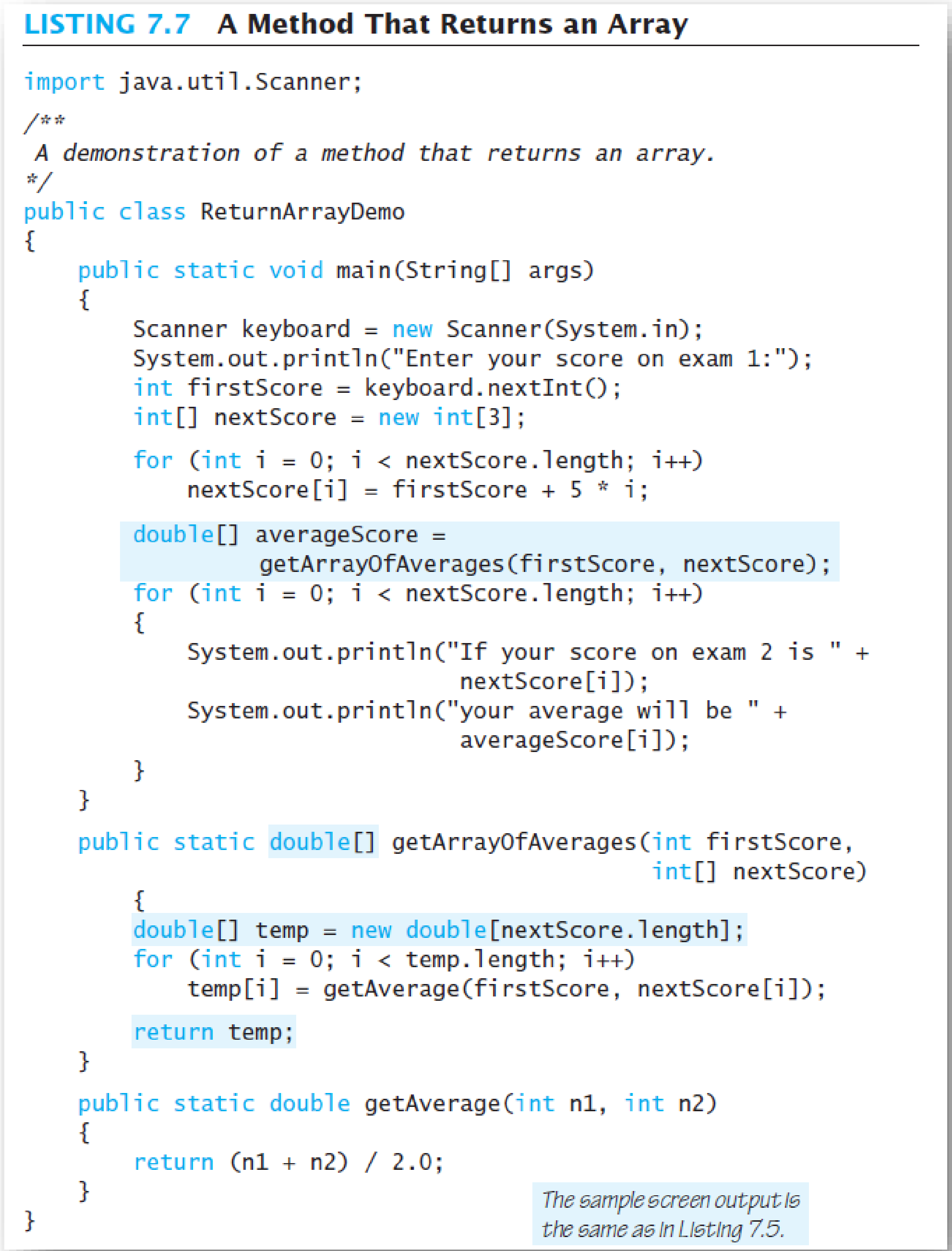
```
System.out.println(team[0].getName());    // OK - why?
System.out.println(team[1].getSalary()); // OK - why?
System.out.println(team[7].getSalary()); // ERROR - why?
```

## Self-Check Exercises

- Write a complete program:
  - That searches in an array X for all the elements that are multiples of 7 or multiples of 3
  - X is of type integer, and size 100.
  - The subscripts of the target elements are to be stored in another array Y of the same size.
  - The array X is filled by the user.
  - The array Y is initialized to -1.

## Methods that Return Arrays

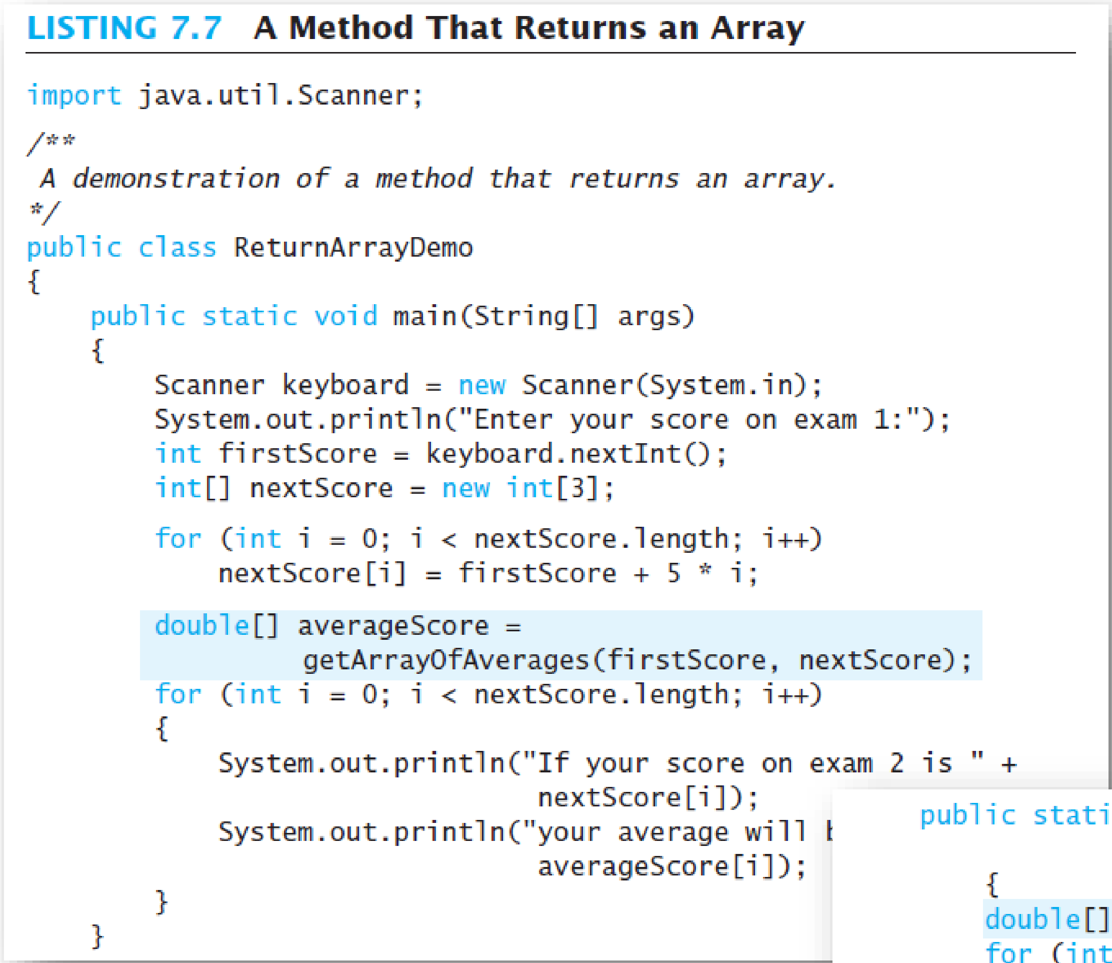
- A Java method may return an array
- View example program, listing 7.7
  `class ReturnArrayDemo`
- Note definition of return type as an array
- To return the array value
  - Declare a local array
  - Use that identifier in the `return` statement



**LISTING 7.7 A Method That Returns an Array**

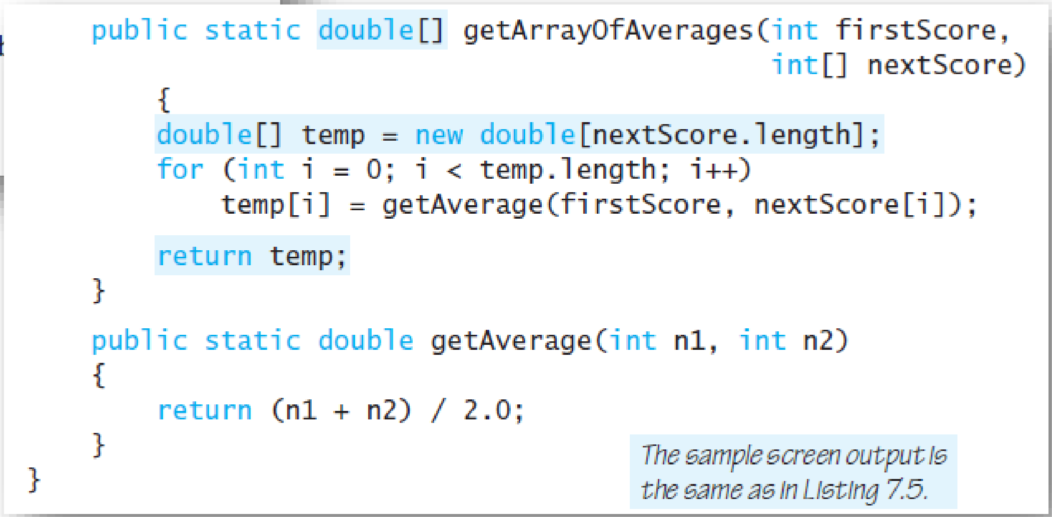
```
import java.util.Scanner;
/**
 A demonstration of a method that returns an array.
*/
public class ReturnArrayDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt();
        int[] nextScore = new int[3];
        for (int i = 0; i < nextScore.length; i++)
            nextScore[i] = firstScore + 5 * i;
        double[] averageScore =
                 getArrayOfAverages(firstScore, nextScore);
        for (int i = 0; i < nextScore.length; i++)
        {
            System.out.println("If your score on exam 2 is " +
                               nextScore[i]);
            System.out.println("your average will be " +
                               averageScore[i]);
        }
    }
    public static double[] getArrayOfAverages(int firstScore,
                                              int[] nextScore)
        {
        double[] temp = new double[nextScore.length];
        for (int i = 0; i < temp.length; i++)
            temp[i] = getAverage(firstScore, nextScore[i]);
        return temp;
    }
    public static double getAverage(int n1, int n2)
    {
        return (n1 + n2) / 2.0;
    }
}
```

*The sample screen output is the same as in Listing 7.5.*



**LISTING 7.7 A Method That Returns an Array**

```
import java.util.Scanner;
/**
 A demonstration of a method that returns an array.
*/
public class ReturnArrayDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt();
        int[] nextScore = new int[3];

        for (int i = 0; i < nextScore.length; i++)
            nextScore[i] = firstScore + 5 * i;

        double[] averageScore =
                  getArrayOfAverages(firstScore, nextScore);
        for (int i = 0; i < nextScore.length; i++)
        {
            System.out.println("If your score on exam 2 is " +
                               nextScore[i]);
            System.out.println("your average will
                               averageScore[i]);
        }
    }
```



```
    public static double[] getArrayOfAverages(int firstScore,
                                              int[] nextScore)
        {
        double[] temp = new double[nextScore.length];
        for (int i = 0; i < temp.length; i++)
            temp[i] = getAverage(firstScore, nextScore[i]);

        return temp;
    }

    public static double getAverage(int n1, int n2)
    {
        return (n1 + n2) / 2.0;
    }
}
```

*The sample screen output is the same as in Listing 7.5.*