ARRAY

MAC 190
Object Oriented Programming

ARRAY BASICS

- An array is a special kind of object
- Think of as collection of variables of same type
- Create an array with 7 variables of type double

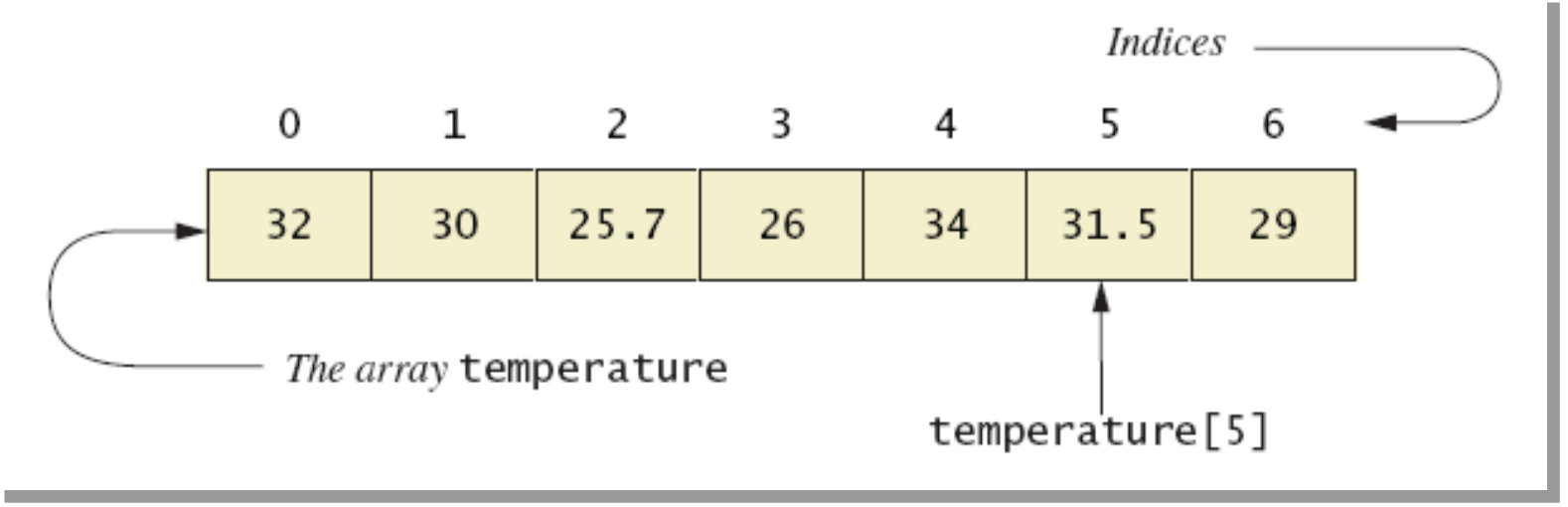
```
double[] temperature = new double[7];
```

- To access an element use
  - The name of the array
  - An index number enclosed in braces
- Array indices begin at zero

CREATING AND ACCESSING ARRAYS

double[] temperature = new double[7];

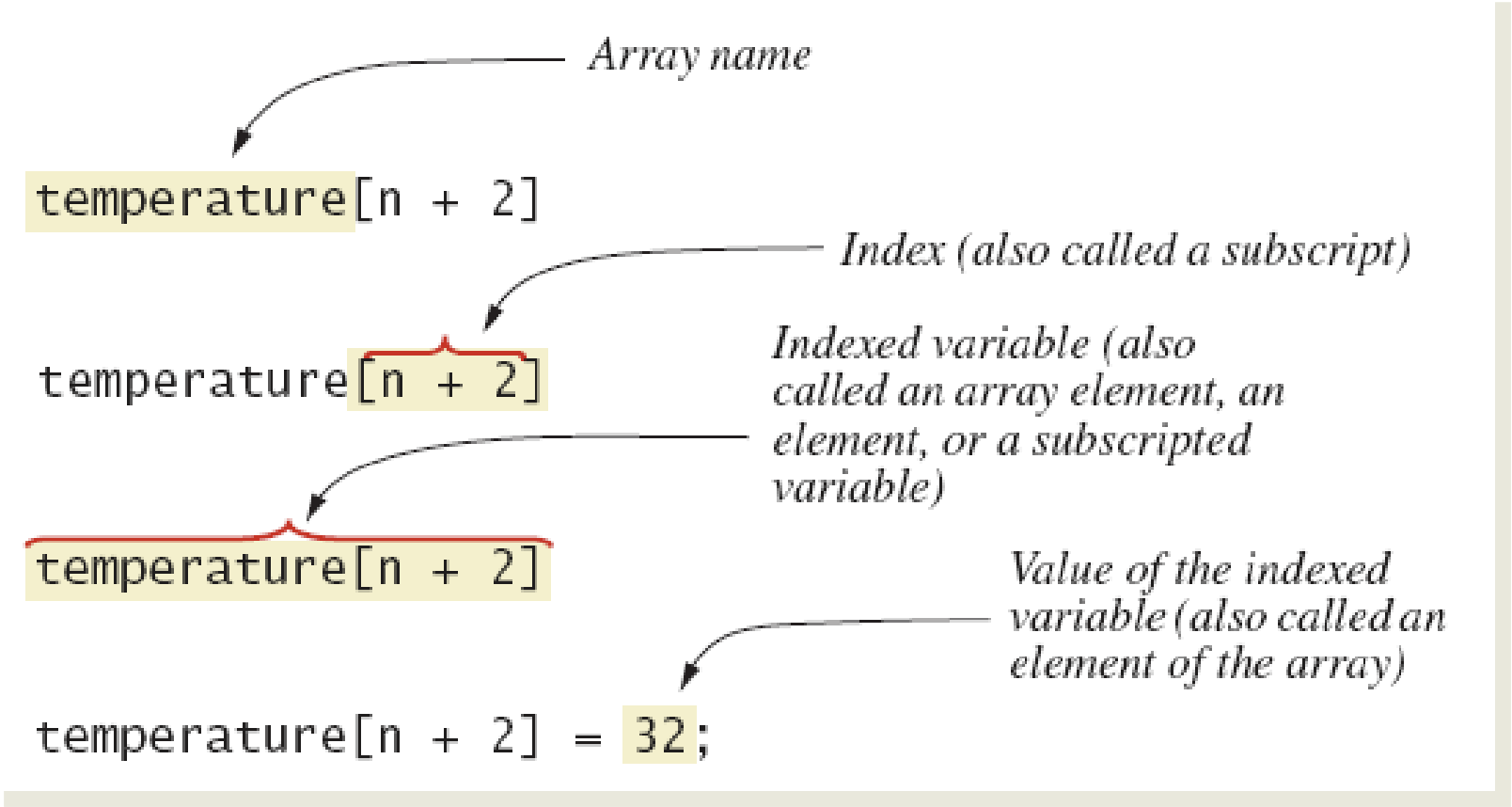
```
temperature[0], temperature[1], temperature[2], temperature[3],
temperature[4], temperature[5], temperature[6]
```



```java
import java.util.Scanner;
public class TempArray {
public static void main(String[] args) {
int index = 0;
System.out.println("Enter number of temperature reading ");
Scanner input = new Scanner(System.in);
index = input.nextInt();
double[] tempArray = new double[index];

for(int i = 0; i < index; i++){
	System.out.println("Enter temperature["+i+"]");
	tempArray[i] = input.nextDouble();
}
System.out.println("Values in temperature array: ");
for(int i = 0; i < index; i++){
	System.out.println("temperature["+i+"]: "+tempArray[i]);
}
}
}
```

```
Enter number of temperature reading
4
Enter temperature[0]
12
Enter temperature[1]
13.5
Enter temperature[2]
2.5
Enter temperature[3]
1.4
Values in temperature array:
temperature[0]: 12.0
temperature[1]: 13.5
temperature[2]: 2.5
temperature[3]: 1.4
```

ARRAY TERMINOLOGY



ARRAY INDICES

As an object an array has only one public instance variable

- Variable `length`
- Contains number of elements in the array
- It is final, value cannot be changed

- Index of first array element is 0
- Last valid Index is `arrayName.length - 1`
- Array indices must be within bounds to be valid
  - When program tries to access outside bounds, run time error occurs

INITIALIZING ARRAYS

Possible to initialize at declaration time

```
double[] reading = {3.3, 15.8, 9.7};
```

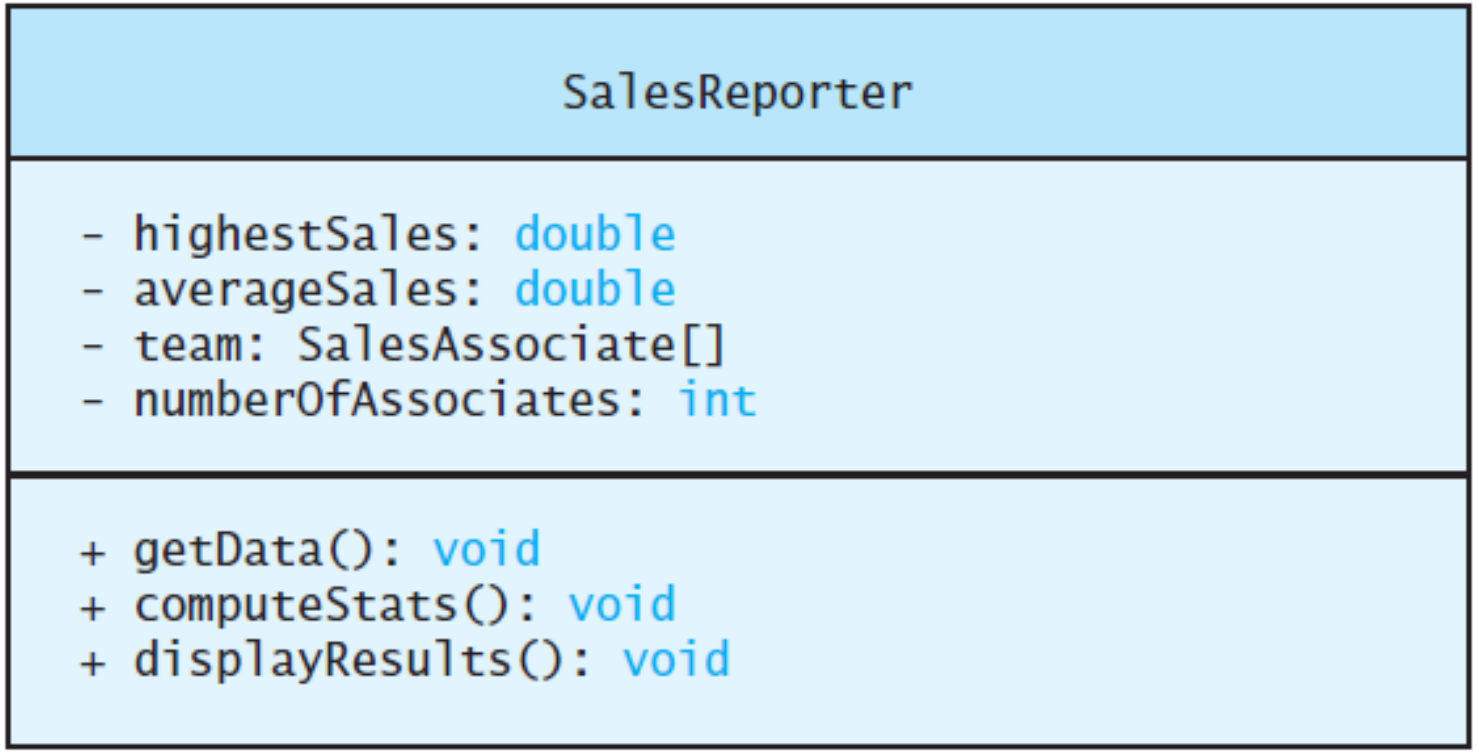
Also may use normal assignment statements

- One at a time
- In a loop

```
int[] count = new int[100];
for (int i = 0; i < 100; i++)
    count[i] = 0;
```

ARRAYS IN CLASSES AND METHODS

Class Diagram for the Class **Sales Reporter**



```java
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;
    }

    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;
    }
}
```

```
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.*

```
/**
 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]`.

```
/**
 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();
        }
    }
```

Sample Screen Output

```
Enter number of sales associates:
3
Enter data for associate number 1
Enter name of sales associate: Dusty Rhodes
Enter associate's sales: $36000
Enter data for associate number 2
Enter name of sales associate: Natalie Dressed
Enter associate's sales: $50000
Enter data for associate number 3
Enter name of sales associate: Sandy Hair
Enter associate's sales: $10000
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.
```

INDEXED VARIABLE AS METHOD ARGUMENTS

Indexed variable of an array **Example** … a[i] can be used as an argument to a method.

```
double possibleAverage = getAverage(firstScore,nextScore[i]);

double possibleAverage = getAverage(nextScore[i],firstScore);

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

ENTIRE ARRAYS AS ARGUMENTS

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

```
double[] a = new double[10];
double[] b = new double[30];
```

```
SampleClass.incrementArrayBy2(a);
SampleClass.incrementArrayBy2(b);
```

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.>
}
```

ARGUMENTS FOR METHOD MAIN

Heading of method **main**
public static void main (String[] args)

This declares an array

- Formal parameter named **args**
- Its base type is **String**

Thus possible to pass to the run of a program multiple strings

- These can then be used by the program

```
public class MainArgument {
public static void main(String[] args) {
        System.out.println("Hello "+args[0]+" "+args[1]+" Welcome to OOP.");
}
}
```

```
Hello John Doe Welcome to OOP.
```

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

```
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;
}
```

METHODS THAT RETURN ARRAYS

```
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;
}
```

METHODS THAT RETURN ARRAYS

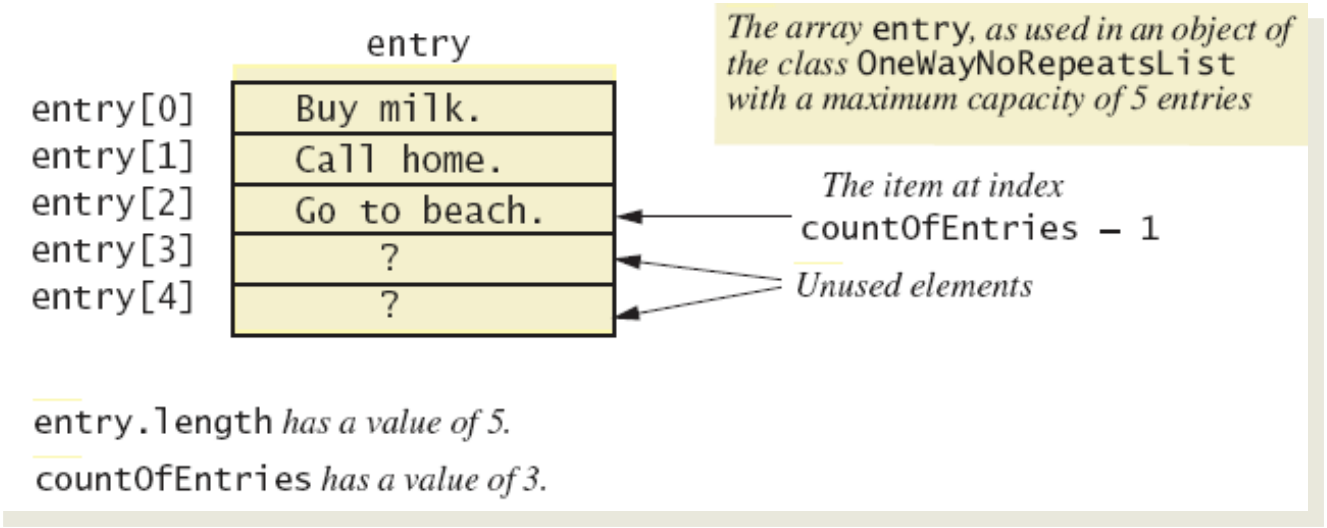
```
public static void main(String[] args)
{
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter your score on exam 1:");
    int firstScore = keyboard.nextInt();
    int[] nextScore = new int[3];

    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]);
    }
}
```

PARTIALLY FILLED ARRAYS

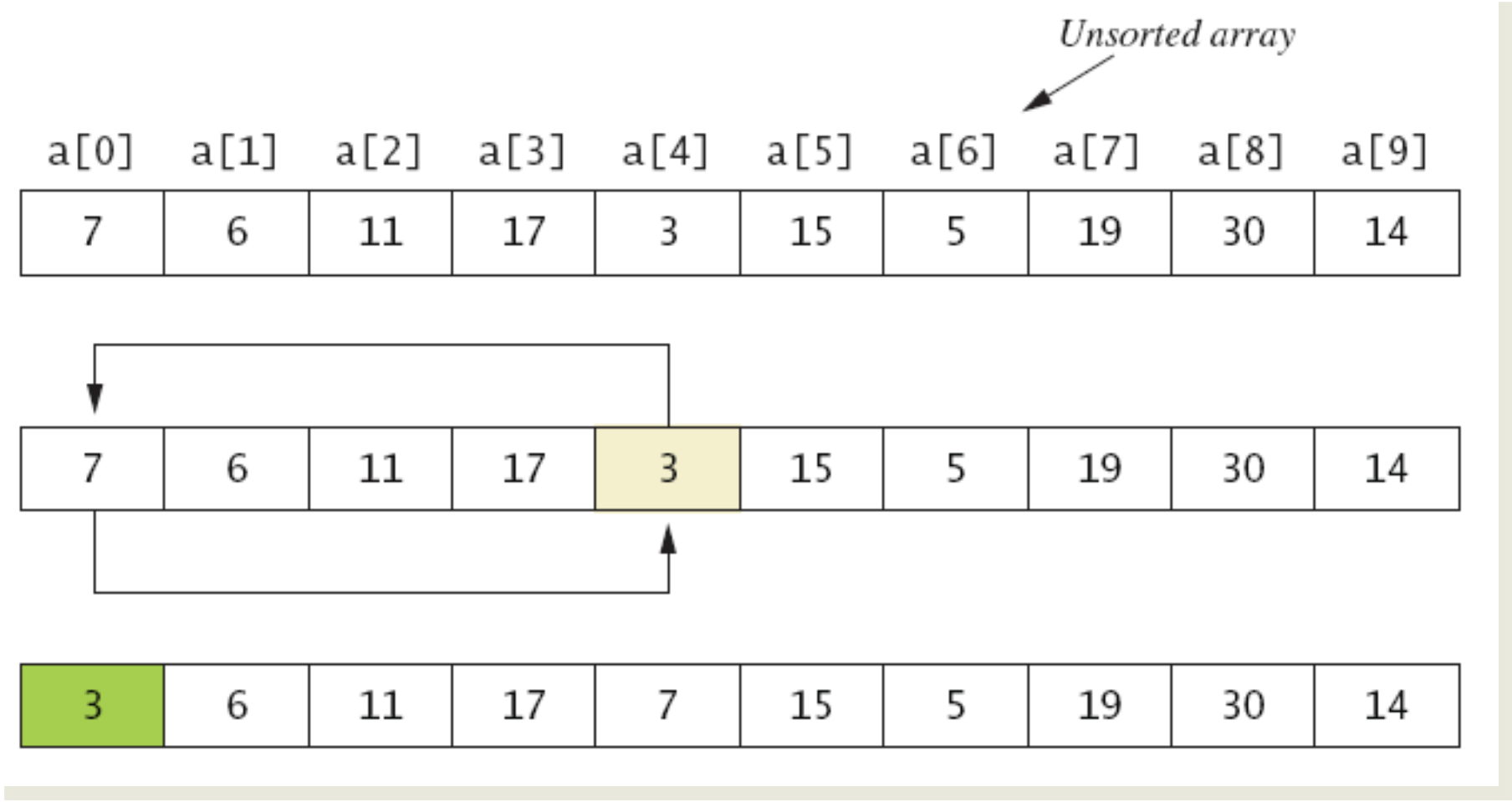
Not all elements of the array might receive values

- This is termed a partially filled array

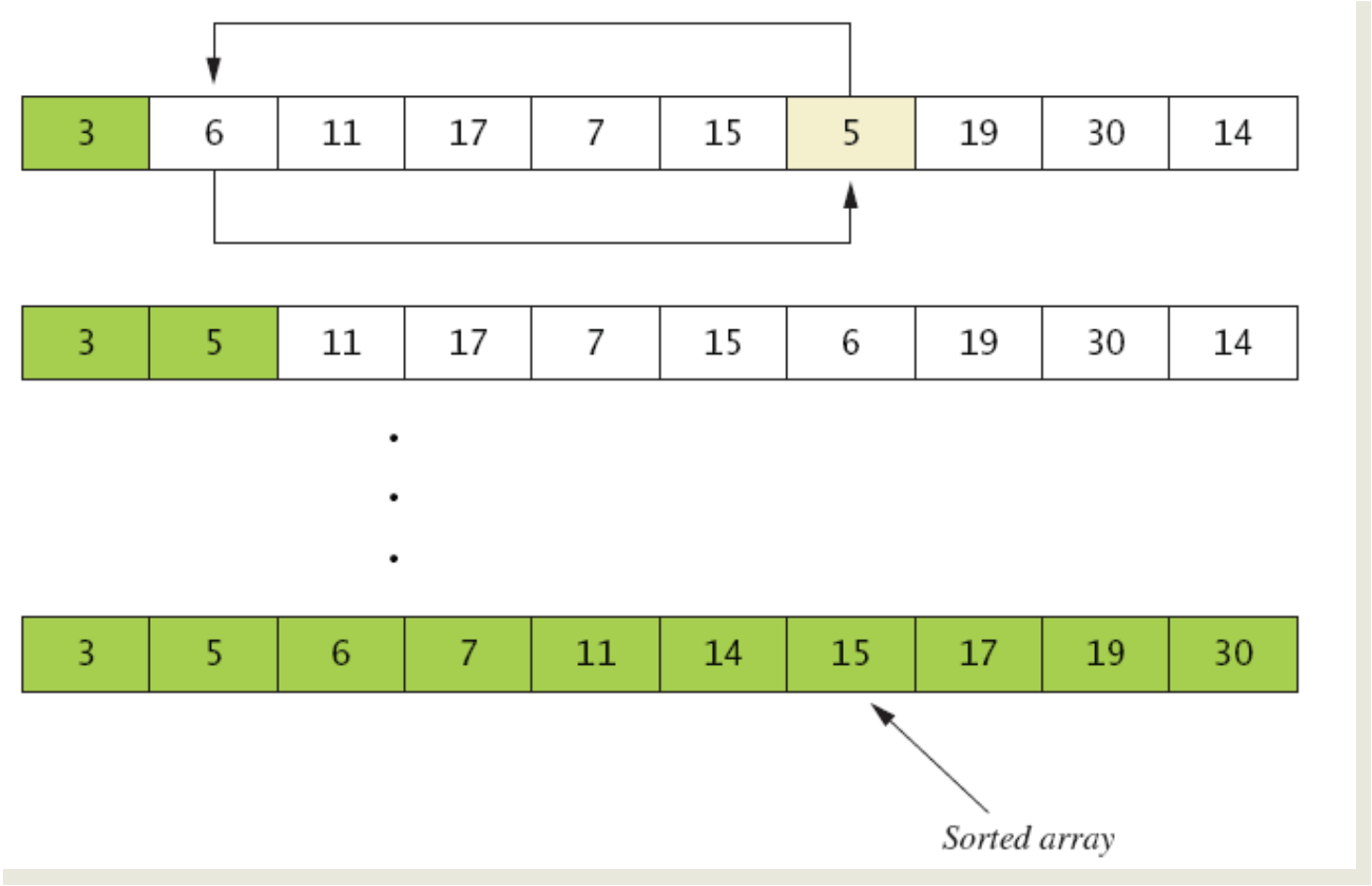
Programmer must keep track of how much of array is used



SELECTION SORT



SELECTION SORT



SELECTION SORT

Consider arranging all elements of an array so they are ascending order

Algorithm is to step through the array

- Place smallest element in index 0
- Swap elements as needed to accomplish this

Called an interchange sorting algorithm

JAVA'S REPRESENTATION OF MULTIDIMENSIONAL ARRAYS

Multidimensional array represented as several one-dimensional arrays

Given
int [][] table = new int [10][6];

Array table is actually 1 dimensional of type **int[]**

- It is an array of arrays

Important when sequencing through multidimensional array

```
for (int row = 0; row <table.length; row++)
   for (int column = 0;column< table[row].length; column++)
       table[row][column]=
           getBalance(1000.00,row + 1,(5 + 0.5 * column));
```