To store several records of an entity, an array of objects can be created.

Syntax:

<class name> [ ] <name of array>= new <class name>[size/number of records in array];

Example:

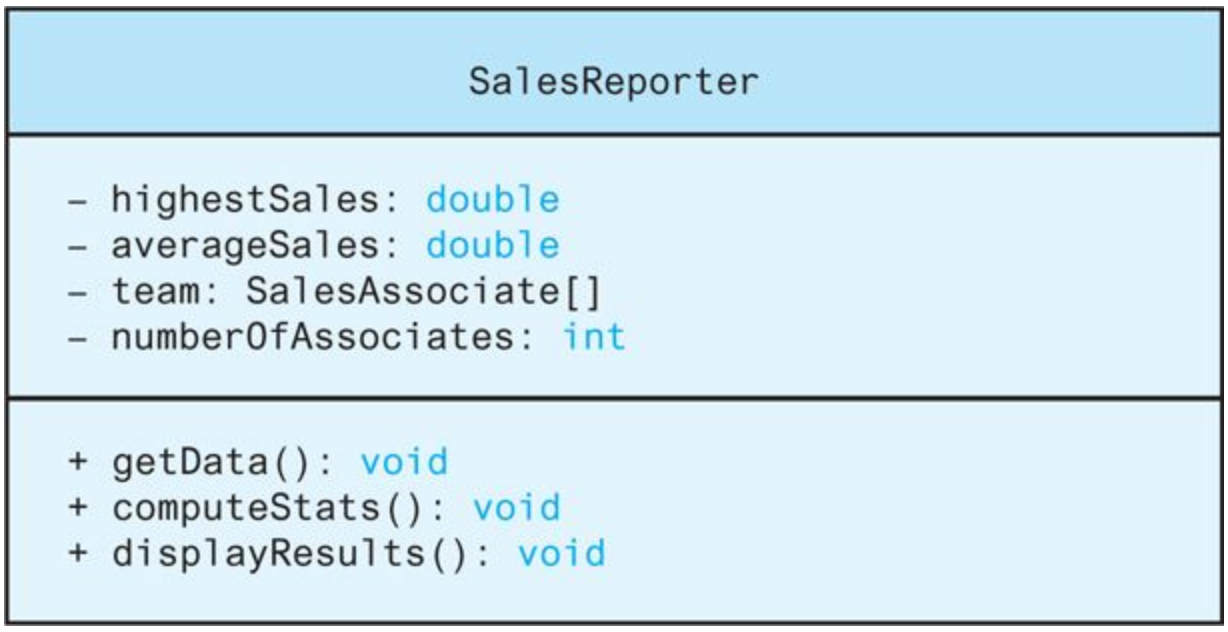
Employee [ ] E= new Employee[5];

Example Code of 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;
    }
    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;
        }
    }
```

A program will need an array to keep track of the data for all sales associates. It will also need to record the highest sales and the average sales. Therefore, we will need to design another class. We can give our new class the following instance variables to record the desired data:



The code below shows a complete example for using an array of objects. It finds the highest sales and average sales for a sales team.

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

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

Array object created here

`SalesAssociate` *objects created here*

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

Indexed Variables as Method Arguments

An indexed variable for an array a, such as a[i], can be used anywhere that you can use any other variable of the base type of the array. So an indexed variable can be an argument to a method in exactly the same way that any other variable of the array's base type can be an argument.

Example:

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

Example Code:

```
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 to a Method : A parameter can represent an entire array.

Example Code:

```
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.>
}
```

A method can change the values of the elements in its array argument.

Remember Array Parameters Do Not Specify the Array Length

An array parameter in a method's heading specifies the base type of the array, but not the length of the array.

- No square brackets are written when you pass an entire array as an argument to a method.
- An array of any length can be the argument corresponding to an array parameter.
- A method can change the values in an array argument.