Engineering Software Development in Java

Lecture Notes for ENCE 688R, Civil Information Systems

Spring Semester, 2017

Mark Austin,
Department of Civil and Enviromental Engineering,
University of Maryland,
College Park,
Maryland 20742, U.S.A.

Chapter 8

Working with Objects and Classes

8.1 Classes and Objects

Object-oriented development procedures observe that in real life:

1. Collections of objects share similar traits. They may store the same data and have the same structure and behavior.

2. Then, collections of objects will form relationships with other collections of objects.

Instead of working in terms of objects alone, it makes sense to create models that capture the common attributes, properties and behaviors shared by collection of objects.

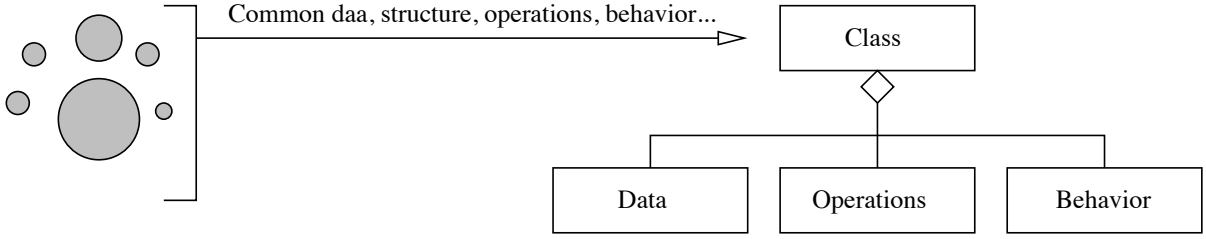

Figure 8.1. Pathway from collections of objects to classes.

As illustrated in Figure 8.1, these models are called classes. From a software perspective, a class is a ...

> **... specification or blueprint for a software object containing data and an ensemble of operations for inspecting and manipulating the data.**

In Java, these operations are computed by bodies of executable code called methods. Methods contain step-by-step instructions for computing a specific task and are the object-oriented counterpart of functions in C. Programming with classes and methods is strongly encouraged because it enables complex problems to be efficiently represented as hierarchies of simplier tasks.

Figure 8.2 shows the pathway from a class specification to the generation of families of specific objects. Each object will have its own specific data values. We say that each object is **an instance of a class**.

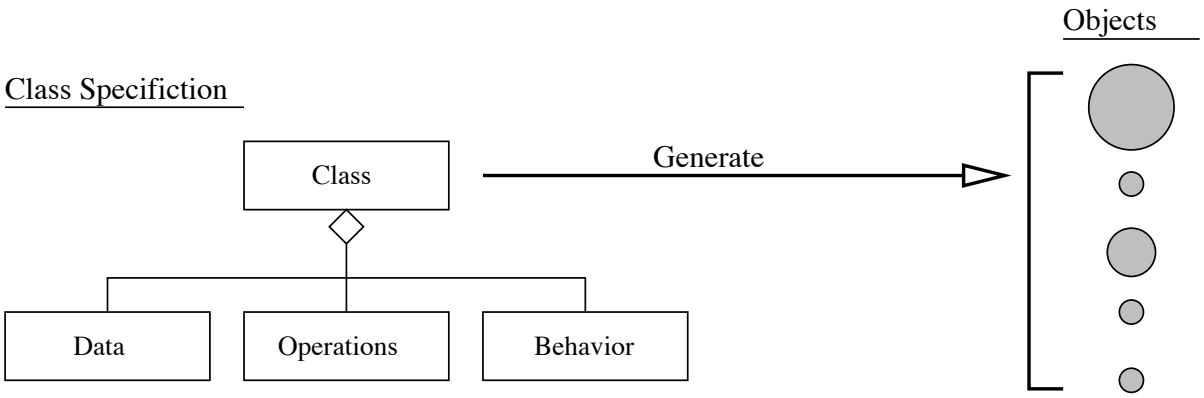

Figure 8.2. Generation of objects from a class specification.

8.2 Syntax for Class Definition

The syntax for making a class definition in Java is

```
modifier class name-of-class {   <=== beginning of the
                                      class body.

     ... variables and methods  ....

}                                <=== end of the
                                      class body.
```

The `modifier` establishes the class type and its scope (i.e., what other classes can call it). A summary of class modifiers is located in Table 8.1.

Class Modifiers

Modifier	Interpretation in Java
abstract	The class contains methods that are unimplemented. An abstract class cannot be instantiated.
final	The class cannot be subclassed.
(none)	A nonpublic class is accessible only in its package.
public	The class can be accessed by any class.
static	This is a top-level class (not an inner class).

Table 8.1. Summary of class modifiers.

The code for each class will usually be stored in a separate file called `class-name.java`. Defining an object of a certain class is called creating an instance of that class. When you want to work with a particular object, you create an instance of that class.

Example: Circle Class

We now illustrate these concepts by defining a class to represent circles. A circle can be described by the `x` and `y` position of its center and by its `radius`. See Figure 8.3.

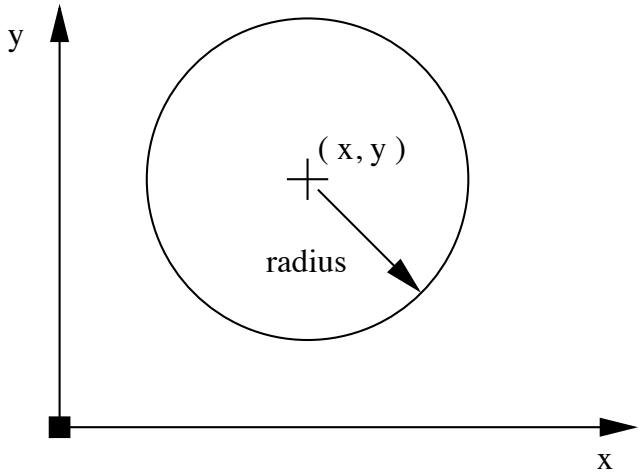

Figure 8.3. Schematic for a circle object.

There are numerous things we can do with circles – compute their circumference or perimeter, compute their area, check whether points are inside them, and so forth. Although each circle is a particular object with its own `(x,y)` coordinate and `radius`, circle is a general concept that can be captured in a "class definition."

```
public class Circle {
      public double dX, dY;
      public double dR;

      public double perimeter() {
         return 2 * 3.1415 * dR ;
      }

      public double area() {
         return 3.1415 * dR * dR ;
      }
}
```

The first line of source code uses the keyword `class` to define the class and the keyword `public` to control access to the class by other classes. Note: The name used for the class must be the same as the one used for the file name (i.e., the source code for class Circle needs to be store in a file called `Circle.java`).

The body of `Circle` contains three public instance variables, `dX`, `dY`, and `dR`, and two public methods, `perimeter()` and `area()`. Our use of the keyword `public` makes the variables and methods accessible from outside of the class. Our use of the keyword `double` serves a dual purpose. First, it specifies that variables `dX`, `dY`, and `dR` will be of type `double`. Second, it indicates that `perimeter()` and `area()` will both return a `double`. The methods `perimeter()` and `area()` do not have any arguments, but can access the values of `dX`, `dY`, and `dR` declared within the class.

8.3 Creating an Object

Now that we have created a class to represent circles, we want to be able to work with it. To be able to work with an actual object `Circle`, we need to create an instance of that class.

Creation of an object requires two steps: declaration and creation.

```
Circle smallCircle;
smallCircle = new Circle();
```

The first line is a declaration of a variable `smallCircle` that can reference an object of of type `Circle`. In the second step, the `new` operator creates an instance of class `Circle`, and a reference to the object is assigned to `smallCircle`. This two-step procedure can be shortened into one step of course:

```
Circle smallCircle = new Circle();
```

After a computer has executed these steps, the layout of memory looks similar to Figure 8.4.

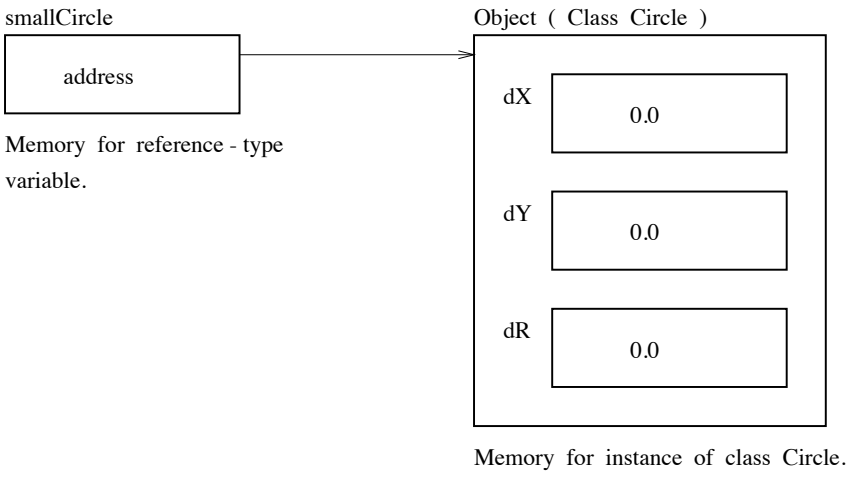

Figure 8.4. Layout of memory for circle object.

`smallCircle` is an abstract identifier that holds a reference to (or the address in memory of the object location) an object of type `Circle`. `smallCircle` does not hold the object itself. Otherwise, creating an instance of a class is like creating a copy of the code defined in the class. The

copy will have its own values for the instance variables `dX`, `dY`, and `dR`. Access will also be provided to the methods defined in the class.

No Pointers. We know that some of you will look at `smallCircle` in Figure 8.4 and think "hey, that's a pointer." It is not. Java has no pointers.

Although the Java run-time system may treat the reference as a pointer or handle, or perhaps a pointer to a pointer, all these details are implementation dependent and deliberately hidden from the Java programmer. All that a Java programmer needs to know is that ...

... the run-time system will take care of the details of referencing objects.

There are two important reasons for these restrictions:

1. Elimination of pointers simplifies the language and eliminates many notorious sources of bugs.
2. Pointers and pointer arithmetic could be used to sidestep Java's run-time checks and security mechanisms. Removing pointers allows Java to provide the security guarantees that it does.

8.4 Object Data and Methods

Accessing Object Data

Now that we have created an object, we can use its data fields. The dot operator (.) is used to access the different public variables of an object. For example

```
Circle smallCircle = new Circle();

/* Initialize the circle to have center (2,2) and radius 1.0 */

smallCircle.dX = 2.0;
smallCircle.dY = 2.0;
smallCircle.dR = 1.0;
```

sets the variables `dX`, `dY`, and `dR` to 2.0, 2.0, and 1.0, respectively.

Accessing Object Methods

This is where things get interesting. To access the methods of an object, we use the same syntax as accessing the data of the object: the dot operator (.).

```
Circle smallCircle = new Circle();
double dArea;

smallCircle.dR  = 2.5;
dArea = smallCircle.area();
```

Take a look at the last line. We did not write

```
dArea = area( smallCircle );
```

But instead, we wrote

```
dArea = smallCircle.area();
```

This is why Java is called an object-oriented language. The object is the focus, not the function call. This is probably the single most important feature of the object-oriented paradigm.

By calling `smallCircle.area()` the syntax itself implies that the object we are working on is `smallCircle`. If you remember, this object is a copy of the code of the class `Circle`, which is going to calculate the area of the circle based on its own value of the radius `dR`.

Passing Objects to Methods

When an object is passed to a method, ...

> **... it is a copy of a reference to that object that is actually passed and not a copy of the object itself.**

This means that a method can change the value of data in an external object.

8.5 Working with Constructor Methods

If we have another look at the object creation code

```
Circle smallCircle = new Circle();
```

Looks like we are calling a function named `Circle()`. Well, guess what, you are exactly right. Every Java class comes with at least one method called the **constructor method** which has the same name as the class itself. The purpose of that constructor method is to perform any necessary initialization for the new object. This method is called every time you create an instance of the class.

If you do not specify any constructor method when you write the code of a class, Java provides a default one for you that takes no arguments and performs no special initialization. We could define our own constructor method by writing

```
public class Circle {
   public double dX, dY, dR;      // Center and radius

   // Our constructor method

   public Circle( double dX, double dY, double dR ) {
      this.dX = dX;
      this.dY = dY;
      this.dR = dR;
   }
```

```
    public double perimeter() {
       return 2 * 3.1415 * dR ;
    }

    public double area() {
       return 3.1415 * dR * dR ;
    }
}
```

With this new constructor method, the initialization becomes part of the object creation step

```
Circle smallCircle = new Circle( 2.0, 2.0, 1.0 );
```

Note how we used the keyword `this` to refer to the current object. `this.dX` refers to the `dX` coordinate for the current instance of the object, whereas `dX` just refers to the variable passed as an argument. There are two important notes about naming and declaring constructor methods:

1. The constructor method name is always the same as the class name.
2. The return type is implicitly an instance of the class. No return type is specified in a constructor declaration nor is the `void` keyword used. The `this` object is implicitly returned. A constructor method should not use a return statement to return a value.

Object Destruction

Here's why Java is better than C! In standard C programming, memory management must be done (and done carefully) by the programmer. Calls to functions like `malloc()` reserve some memory for future uses. A programmer needs to remember to free this memory when it is no longer needed by the program.

In Java, you never do any explicit memory reservation with `malloc()` because creating a new object with the `new` keyword takes care of the details of memory allocation for you. Also, once a block of memory is no longer needed by a program, the Java run-time system will automatically take care of its release to the operating system.

The Garbage Collector. Java uses a technique called **garbage collection** to automatically detect objects that are no longer being used and to free them. An object is no longer in use when there are no more references to it. Remember that even though Java programmers do not use pointers, the Java system does. In a nutshell, the garbage collector uses the scope of variables and classes to determine if an object is still in use and can be released to the system. The garbage collector runs as a low-priority process and does most of its work when nothing else is going on (i.e., idle time while waiting for user input).

8.6 Working with Class Hierarchies

Inheritance Model in Java

As mentioned in our introduction to object-oriented development in Chapter 5, one one of the major benefits of this problem-solving approach is the ability to extend or subclass the behavior of an existing class, and to continue to use the code written for the original class. In other words, ...

... class hierarchies provide a means for avoiding duplication and redundancy of code.

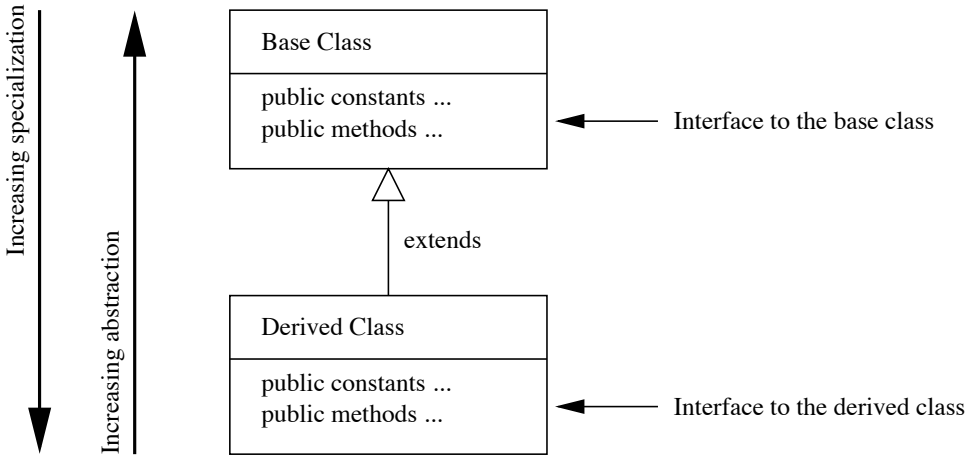

Figure 8.5. Elements of the inheritance model in Java.

As illustrated in Figure 8.5,

1. A class in the upper hierarchy is called a superclass (or base, parent class).
2. A class in the lower hierarchy is called a subclass (or derived, child, extended class).

The classes in the lower hierarchy ...

...inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies.

However, the subclasses can ...

... override the behavior of the superclass, thereby providing a mechanism for behavior to be customized within a class hierarchy.

By pulling out all the common variables and methods into the super-classes, and leave the specialized variables and methods in the subclasses, redundancy can be greatly reduced as these common variables and methods do not need to be repeated in all the subclasses.

Extending Circle to create ColoredCircle

In Java, you extend a class by using the keyword `extends` in the class declaration. To see how this works in practice, let us build on the previous `Circle` class by adding a color attribute to `Circle`.

```
public class ColoredCircle extends Circle {
      private Color color;  // The color of the circle

      // Constructor method for this class.

      public ColoredCircle() {
         super(); // Call the superclass constructor method
         this.color = Color.black;
      }

      // Set the color for the current circle.

      public void setColor(Color c) {
         color=c;
      }

}
```

Now let us see how it works. First we declare the class `ColoredCircle` as extending the class `Circle` by using the keyword `extends`. We then declare a public variable for the class, named `color` and of type `Color` to handle the color of the circle. The class hierarchy is shown in Figure 8.6.

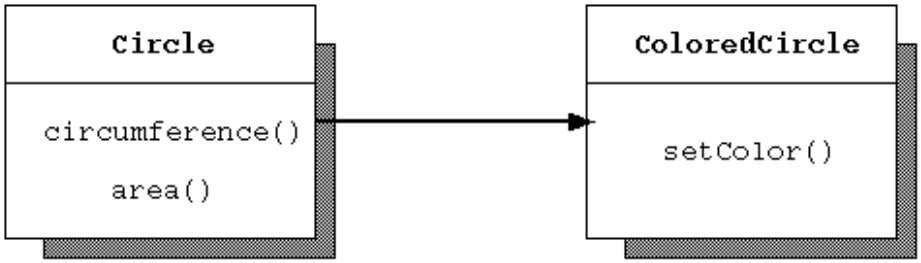

Figure 8.6. `Circle` class and `ColoredCircle` subclass

Two public methods are defined for this class:

1. `setColor.` This method takes a color as its argument and assigns this value to the color of the circle.

2. `ColoredCircle.` This method has the same name as the class itself; it is a constructor method. We first use the keyword `super`. This keyword refers to the superclass of the current class (just like `this` refers to the current class). Function calls of the type `super.method()` call the implementation of the method in the superclass. By just calling `super()`, we invoke the

constructor method of the superclass [the method `Circle()` in that case]. The next line uses the keyword `this` to refer to the public variable `color` of the current class.

Remark. One of the main differences between C++ and Java is that Java does not allow multiple inheritance (i.e., a class cannot extend more than one class). Multiple inheritance allows a designer to mix attributes from disparate classes in the class hierarchy. This restriction in Java is justified by the need to keep the language and program design simple. However, Java employs the `interface` construct to simulate multiple inheritance.

8.7 Working with Enumerated Types

An enumerated type is a type whose instances describe a finite set of values. Typically, an enumerated type is used when the most important information is the existence of the value. A static enumerated type is an enumerated type whose set of possible values is fixed, and does not vary at run time. For example, the concept of days of the week includes the set:

```
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.
```

Not only are these seven values the only possible values for the concept, but there will never be another possible value, and none of these values will ever become obsolete.

Example. Days of the Week

This program demonstrates how an enumerated data type for days of the week can be used in conjuction with selection implemented via a switch statement. The program is divided into two Java files:

DayOfWeek.java. Setup an enumerated type for days-of-the-week.

```
/*
 *  ================================================================
 *  DayOfWeek.java: Setup an enumerated type for days-of-the-week ...
 *  ================================================================
 */

public enum DayOfWeek {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

WorkSchedule.java. This program demonstrates use of enumerated data types, used in combination with switch statements.

```
/*
 *  ============================================================================
 *  WorkSchedule.java: This program demonstrates use of enumerated data types,
 *  used in combination with switch statements.
 *
 *  Written by: Mark Austin                                          June 2009
 *  ============================================================================
 */

public class WorkSchedule {
   DayOfWeek day;

   public WorkSchedule( DayOfWeek day ) {
      this.day = day;
   }

   // Create a string representation of the day ...
```

```
public String getDay() {
   String s = null;

   switch(day) {
      case SUNDAY:
           s = "Sunday";
           break;
      case MONDAY:
           s = "Monday";
           break;
      case TUESDAY:
           s = "Tuesday";
           break;
      case WEDNESDAY:
           s = "Wednesday";
           break;
      case THURSDAY:
           s = "Thursday";
           break;
      case FRIDAY:
           s = "Friday";
           break;
      case SATURDAY:
           s = "Saturday";
           break;
      default:
           break;
    }

    return s;
}

// Describe work schedule ...

public String describe() {
   String description;

   switch(day) {
      case SUNDAY:
           description = "I stay home...";
           break;
      case MONDAY:
      case TUESDAY:
      case WEDNESDAY:
      case THURSDAY:
           description = "It's a week day, so I go to work.";
           break;
      case FRIDAY:
           description = "Last day of the working week, so I leave early!";
           break;
      case SATURDAY:
           description = "It's been a long work week. I stay home.";
           break;
      default:
```

```
                description = "Something wrong Day not defined?";
        }

        return description;
    }

    // Exercise methods in WorkSchedule ...

    public static void main(String[] args) {

        // Create objects for work schedules on three different days ...

        WorkSchedule day1 = new WorkSchedule( DayOfWeek.THURSDAY );
        WorkSchedule day2 = new WorkSchedule( DayOfWeek.FRIDAY );
        WorkSchedule day3 = new WorkSchedule( DayOfWeek.SATURDAY );
        WorkSchedule day4 = new WorkSchedule( DayOfWeek.SUNDAY );

        // Describe the work schedules on each of these days ...

        System.out.printf("Today is %8s\n",         day1.getDay() );
        System.out.printf("My work schedule: %s\n", day1.describe() );

        System.out.println("");
        System.out.printf("Today is %8s\n",         day2.getDay() );
        System.out.printf("My work schedule: %s\n", day2.describe() );

        System.out.println("");
        System.out.printf("Today is %8s\n",         day3.getDay() );
        System.out.printf("My work schedule: %s\n", day3.describe() );

        System.out.println("");
        System.out.printf("Today is %8s\n",         day4.getDay() );
        System.out.printf("My work schedule: %s\n", day4.describe() );

    }
}
```

Compiling and running WorkSchedule in the usual way leads to the output:

```
Today is Thursday
My work schedule: It's a week day, so I go to work.

Today is   Friday
My work schedule: Last day of the working week, so I leave early!

Today is Saturday
My work schedule: It's been a long work week. I stay home.

Today is   Sunday
My work schedule: I stay home...
```

8.8 Application 1. Area and Volume of a Folded Box

Figure 8.7 shows a 10cm by 5cm sheet of metal that will be folded into a small open box.

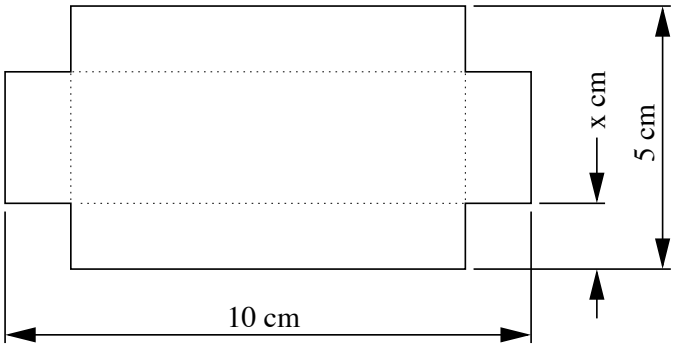

Figure 8.7. Sheetmetal schematic for a folded box.

We wish to write a Java program that will compute and print the surface area and volume of folded box objects having the following characteristics:

```
-------------------------------------------------------------
Folded Box:     Name   Length (in)  Width (in)  Height (in)
-------------------------------------------------------------
               Match           2.0         1.0          0.5
                Shoe          12.0         8.0          7.0
-------------------------------------------------------------
```

The source code is as follows:

```
/*
 *  ============================================================================
 *  FoldedBox.java: Create instances of open folded box objects.
 *                  Compute and print the volume and surface area of each box.
 *
 *  Written By: Mark Austin                                            May 2007
 *  ============================================================================
 */

public class FoldedBox {
   private String sName;
   private double dLength, dWidth, dHeight;

   // Custom constructor for creating folded box objects ....

   public FoldedBox ( String  sName, double dLength,
                      double dWidth, double dHeight ) {
      this.dLength = dLength;
      this.dWidth  = dWidth;
      this.dHeight = dHeight;
      this.sName   = sName;
   }
```

```
   // Compute box surface area and volume ....

   public double surfaceArea() {
     return dLength*dWidth + 2.0*(dLength+dWidth)*dHeight;
   }

   public double volume() {
     return dLength*dWidth*dHeight;
   }

   // Create String description of folded box characteristics ...

   public String toString() {
      String s = "FoldedBox: "  + sName + "\n";
      s = s + "Volume       = " + volume()   + "\n";
      s = s + "Surface Area = " + surfaceArea()   + "\n";
      return s;
   }

   // ============================================================
   // Exercise methods in FoldedBox.java
   // ============================================================

   public static void main ( String args [] ) {

      // Create and initialize folded box objects

      FoldedBox fbMatch    = new FoldedBox ("Match",  2.0, 1.0, 0.5 );
      FoldedBox fbShoe     = new FoldedBox ( "Shoe", 12.0, 8.0, 7.0 );

      // Print details of each folded box ...

      System.out.println(    fbMatch.toString() );
      System.out.println(     fbShoe.toString() );
   }
}
```

A script of the program input and output is as follows:

```
prompt >>
prompt >> java FoldedBox
FoldedBox: Match
Volume       = 1.0
Surface Area = 5.0

FoldedBox: Shoe
Volume       = 672.0
Surface Area = 376.0
prompt >>
```

Points to note:

1. Each box object is described by its name (i.e., String sName) and parameters for the length, width and height of the box. The box surface area and volume are computed with methods

`surfaceArea()` and `volume()`, respectively. A string description of the folded box will be created by `toString()`.

2.

3.

8.9 Application 2. Complex Number Arithmetic

Complex variables and complex variable arithmetic are used in the solution of many types of engineering problems. In electrical engineering, for example, complex numbers are a staple of circuit analysis. In structural dynamics, complex variables can be used to formulate models of damping. In geotechnical engineering, analysis with complex numbers and tranformations can be used to compute streamlines for groundwater flow.

In this section we present abbreviated Java code for complex number arithmetic and a corresponding test program. The full implementation has the methods:

```
=========================================================================
Method                  Description
=========================================================================

Complex();              Create new instance of complex number.
Negate();               Compute -ve of complex number.
Abs();                  Compute absolute value of complex number.
Add();                  Add two complex numbers.
Sub();                  Subtract two complex numbers.
Mult();                 Multiply two complex numbers.
Div();                  Divide complex numbers.
Scale();                Scale complex number by real number.
Conjugate();            Compute complex conjugate.
Sqrt();                 Compute square root of complex number.

toString();             Convert a complex number to a string.
=========================================================================
```

The skeleton class library is as follows (the algorithms for arithmetic are taken from the text "Numerical Recipies in C."

abbreviated source code

```
/*
 *  ================================================================
 *  Complex.java: A library of methods for complex number arithmetic.
 *
 *  Written By : Mark Austin                                May 2006
 *  ================================================================
 */

import java.lang.Math;

public class Complex {
   protected double dReal, dImaginary;

   // Constructor methods ....

   public Complex() {}

   public Complex( double dReal, double dImaginary ) {
      this.dReal      = dReal;
```

```
   this.dImaginary = dImaginary;
}

// Convert complex number to a string ...

public String toString() {
   String s="";

   if (dImaginary >= 0)
       s += s.format("%6.2f+%6.2fi", dReal,  dImaginary );
   else
       s += s.format("%6.2f-%6.2fi", dReal, -dImaginary );

   return s;
}

// Compute sum of two complex numbers cA + cB.....

public Complex Add( Complex cB ) {
Complex sum = new Complex();

    sum.dReal      = dReal      + cB.dReal;
    sum.dImaginary = dImaginary + cB.dImaginary;

    return (sum);
}

// Compute difference of two complex numbers cA - cB.....

public Complex Sub( Complex cB ) {
Complex diff = new Complex();

    diff.dReal      = dReal      - cB.dReal;
    diff.dImaginary = dImaginary - cB.dImaginary;

    return (diff);
}

// Compute product of two complex numbers cA * cB ... details removed ...

// Compute divisor of two complex numbers cA / cB ... details removed ...

// Scale complex number .. details removed ...

// Compute complex number conjugate ... details removed ...

// Compute absolute value of complex number ...details removed ...

// Compute square root of complex number ... details removed ...

// Exercise methods in Complex class ....

public static void main ( String args[] ) {

   System.out.println("Complex number test program");
```

```
        System.out.println("============================");

        // Setup and print two complex numbers .....

        Complex cA = new Complex( 1.0, 2.0 );
        Complex cB = new Complex( 3.0, 4.0 );

        System.out.println("Complex number cA = " + cA.toString() );
        System.out.println("Complex number cB = " + cB.toString() );

        // Test complex addition and substraction .....

        Complex cC = cA.Add( cB );
        System.out.println("Complex   cA + cB = " + cC.toString() );
        Complex cD = cA.Sub( cB );
        System.out.println("Complex   cA - cB = " + cD.toString() );
    }
}
```

and covers just the most basic parts – a constructor to create complex number objects, methods to add and substract complex numbers, a method to create a string representation, and main(), a short method to exercise the methods for arithmetic.

The program output is:

```
prompt >>
Complex number test program
============================
Complex number cA =   1.00+  2.00i
Complex number cB =   3.00+  4.00i
Complex   cA + cB =   4.00+  6.00i
Complex   cA - cB =  -2.00-  2.00i
prompt >>
```

Key points to note:

1. The toString() method creates a string representation of the data stored in a complex number. In the fragment of code:

```
String s="";

if (dImaginary >= 0)
    s += s.format("%6.2f+%6.2fi", dReal,  dImaginary );
else
    s += s.format("%6.2f-%6.2fi", dReal, -dImaginary );

return s;
```

creates an empty charater string, and then appends to it, formatted string representations of the real and imaginery components of the complex number. For the purposes of clarity in its usage, we have written,

```
System.out.println("Complex number cA = " + cA.toString() );
```

However, the toString() method is so common – every object implementation should have a toString() method – that you can also write,

```
System.out.println("Complex number cA = " + cA );
```

Java will simply assume that you want to print a string representation of the object.

2. In a procedural approach to programming we would create and sum two complex numbers by writing something like:

```
Complex cA = new Complex( 1.0, 2.0 );
Complex cB = new Complex( 3.0, 4.0 );

cC = cA + cB;
```

Since the focus in Java is on objects and the data that they store, here, instead, we write,

```
Complex cC = cA.Add(cB);
```

which can be interpreted as: to the data in object cA, add the data from object cB. Then assign the result to a new object cC. It is important to notice that within the method Add(), the statement

```
Complex sum = new Complex();
```

allocates memory for the result of the addition operation.

8.10 Application 3. Point and Line Segment Operations

Problem Statement

Figure 8.8 shows a line segment in a (x-y) coordinate frame. The segment is defined as a straight line joining two endpoints at coordinates (x_1, y_1) and (x_2, y_2).

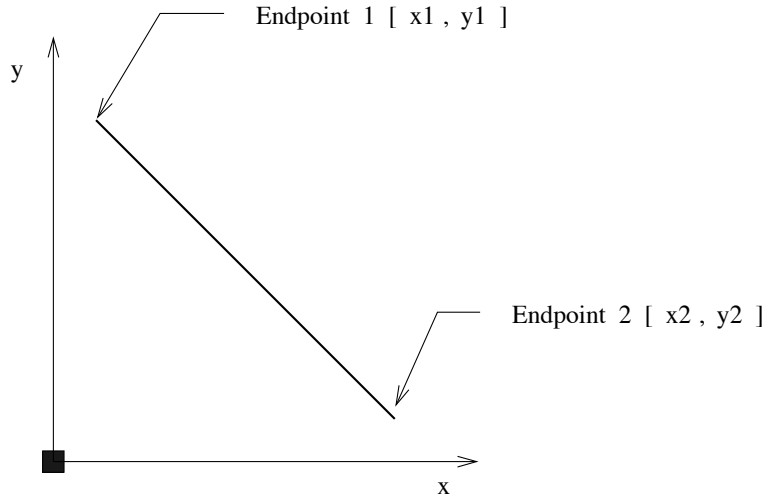

Figure 8.8. Line segment in (x-y) coordinate frame.

The following program is developed in two stages. First, we write a class `Point` for the definition and positioning of (x, y) coordinates. Then with the `Point` class in place, we write a class for defining and computing operations on line segments. Test programs are written to exercise the capabilities of both classes.

Program Modules and Class Hierarchy

The point and line segment operations are

```
=========================================================================
Point                          Line Segment
=========================================================================

Allocate a new point.          Allocate a new line segment.
Set (x,y) coordinates.         Set coordinates of line segment endpoints.
Get x coordinate.              Get endpoint 1.
Get y coordinate.              Get endpoint 2.
Print coordinates.             Print endpoints of line segment.

                               Compute and print equation of line segment
                               (not implemented in program).
                               Compute intersection of two line segments
                               (not implemented in program).

Exercise methods in Point      Exercise methods in LineSegment class.
```

```
     class.
     ========================================================================
```

The `Point` class should contain methods for constructing instances of a `point`, setting the (x,y) coordinate values, retrieving the x and y coordinates, and printing the coordinates as a string. Each of these methods can be exercised by a test program included as part of the `Point` class.

Similarly, the `LineSegment` class should contain methods for constructing instances of a `line segment`, setting the coordinate values of the endpoints, retrieving endpoints `1` and `2`, and printing the details of the line segment. An advanced implementation (these details are not included in the program) might also compute and print the equation of the line segment, and compute the intersection of two line segments. Each of these methods can be exercised by a test program included as part of the `LineSegment` class.

Program Source Code

The source code for the line segment program is partitioned into two parts. The first half contains the user-written code for the `Point` class. The second part of the source code contains details of the `Linesegment` class.

source code

```
/*
 *  ============================================================
 *  Definition of class Point : A point is represented by its
 *                              (x,y) coordinates.
 *
 *  Written By : Mark Austin                      October 1997
 *  ============================================================
 */

public class Point {
   protected double dX, dY; // (x,y) coordinates of the Point

   // Constructor method : just create instance of class.

   public Point() { }

   // Create instance of class and set (x,y) coordinates.

   public Point( double dX, double dY ) {
      setPoint( dX, dY );
   }

   // Set x and y coordinates of Point.

   public void setPoint( double dX, double dY ) {
      this.dX = dX;
      this.dY = dY;
   }

   // Get x and y coordinates.
```

```
   public double getX() { return dX; }
   public double getY() { return dY; }

   // Convert the point into a String representation

   public String toString()
      { return "[" + dX + ", " + dY + "]"; }

   // Exercise methods in point class.

   public static void main( String args[] ) {
   double dX, dY;

      System.out.println("Point test program");
      System.out.println("==============================");

      // Create two new points.

      Point point1 = new Point();
      point1.setPoint( 1.0, 1.0 );
      Point point2 = new Point( 1.5, 4.5 );

      // Print (x,y) coordinates of points.

      System.out.println("Point 1 : " + point1.toString() );
      System.out.println("Point 2 : " + point2.toString() );

      // Get x and y coordinates.

      dX = point1.getX(); dY = point1.getY();
      System.out.println("Point 1 : X coordinate = " + dX );
      System.out.println("          Y coordinate = " + dY );
      dX = point2.getX(); dY = point2.getY();
      System.out.println("Point 2 : X coordinate = " + dX );
      System.out.println("          Y coordinate = " + dY );

      System.out.println("==============================");
      System.out.println("End of Point test program");
   }
}
```

source code

```
/*
 *  ========================================================
 *  Definition of class LineSegment:
 *
 *  A line segment is defined by the (x,y) coordinates of
 *  its two end points.
 *
 *  Written By : Mark Austin                    October 1997
 *  ========================================================
```

```
 */

public class LineSegment {
   protected Point p1, p2; // points defining the LineSegment

   // Constructor method : just create instance of class.

   public LineSegment() { }

   // Create instance of class and set (x,y) coordinates.

   public LineSegment( double dX1, double dY1,
                       double dX2, double dY2 ) {

      setLineSegment( dX1, dY1, dX2, dY2 );
   }

   // Set x and y coordinates of LineSegment.

   public void setLineSegment( double dX1, double dY1,
                               double dX2, double dY2 ) {
      p1 = new Point( dX1, dY1 );
      p2 = new Point( dX2, dY2 );
   }

   // Get end points 1 and 2.

   public Point getPoint1() { return p1; }
   public Point getPoint2() { return p2; }

   // Print details of line segment.

   public void printSegment() {

      System.out.println("Line Segment");
      System.out.println("Point 1 : (x,y) = " + p1.toString() );
      System.out.println("Point 2 : (x,y) = " + p2.toString() );
   }

   // Compute length of line segment.

   public double segmentLength() {
      double dLength;

      dLength = (p1.getX() - p2.getX())*(p1.getX() - p2.getX()) +
                (p1.getY() - p2.getY())*(p1.getY() - p2.getY());

      return ((double) Math.sqrt(dLength));
   }

   // Exercise methods in line segment class.

   public static void main( String args[] ) {
   double dX, dY;
```

```
        System.out.println("LineSegment test program");
        System.out.println("================================");

        // Create two new line segments.

        LineSegment s1 = new LineSegment();
        s1.setLineSegment( 1.0, 1.0, 4.0, 5.0 );
        LineSegment s2 = new LineSegment( 1.5, 1.5, 1.5, 4.5 );

        // Print details of line segments.

        s1.printSegment();
        s2.printSegment();

        // Compute length of line segments.

        System.out.println("Segment1 has length : " + s1.segmentLength());
        System.out.println("Segment2 has length : " + s2.segmentLength());

        // End of exercise.

        System.out.println("================================");
        System.out.println("End of LineSegment test program");
    }
}
```

Compiling and Running the Program

The line segment program has source code for two classes: `Point.java` and `LineSegment.java`. The `Point` class is compiled by typing

```
prompt >> javac Point.java
```

and the `LineSegment` class is compiled by typing

```
prompt >> javac LineSegment.java
```

The Java compiler is pretty smart and, when the `LineSegment` class is being compiled, will determine the dependency of the `LineSegment` and `Point` classes and automatically compile `Point.java` into a bytecode file if it does not already exist. The list of program files before and after compilation is

FILES BEFORE COMPILATION	FILES AFTER COMPILATION	
Point.java	Point.java	Point.class
LineSegment.java	LineSegment.java	LineSegment.class

The `Point` and `LineSegment` classes both have test programs located inside their `main()` methods. Typing

```
>> java Point
```

generates the output

```
Point test program
==============================
Point 1 : [1, 1]
Point 2 : [1.5, 4.5]
Point 1 : X coordinate = 1
          Y coordinate = 1
Point 2 : X coordinate = 1.5
          Y coordinate = 4.5
==============================
End of Point test program
```

The `Point` class test program creates and initializes two new points, the first using the default constructor and the `setPoint()` method, and the second with the `Point()` constructor containing two arguments. The point coordinates are then printed by calling the `toString()` method to create a character string. Finally, the `getX()` and `getY()` methods are used to retrieve and print the individual `x` and `y` point coordinates.

Similarly, the `LineSegment` test program generates

```
LineSegment test program
==============================
Line Segment
Point 1 : (x,y) = [1, 1]
Point 2 : (x,y) = [4, 5]
Line Segment
Point 1 : (x,y) = [1.5, 1.5]
Point 2 : (x,y) = [1.5, 4.5]
Segment1 has length : 5
Segment2 has length : 3
==============================
End of LineSegment test program
```

The line segment test program begins its execution by creating two new line segments – for simplicity of implementation, the endpoint coordinates are specified in the program source code. After the details of each line segment have been printed, the length of each line segment is computed and printed.

Program Architecture

Figure 8.9 shows the class hierarchy of user-defined Java code and classes from the JDK that make up the quadratic equation solver.

The quadratic equation solver and line segment programs employ essentially the same hierarchy of classes for simple mathematical computations and program output. What is new in the point and line program is its use of two classes of user-defined Java code. A line segment endpoint is defined by the skeleton class

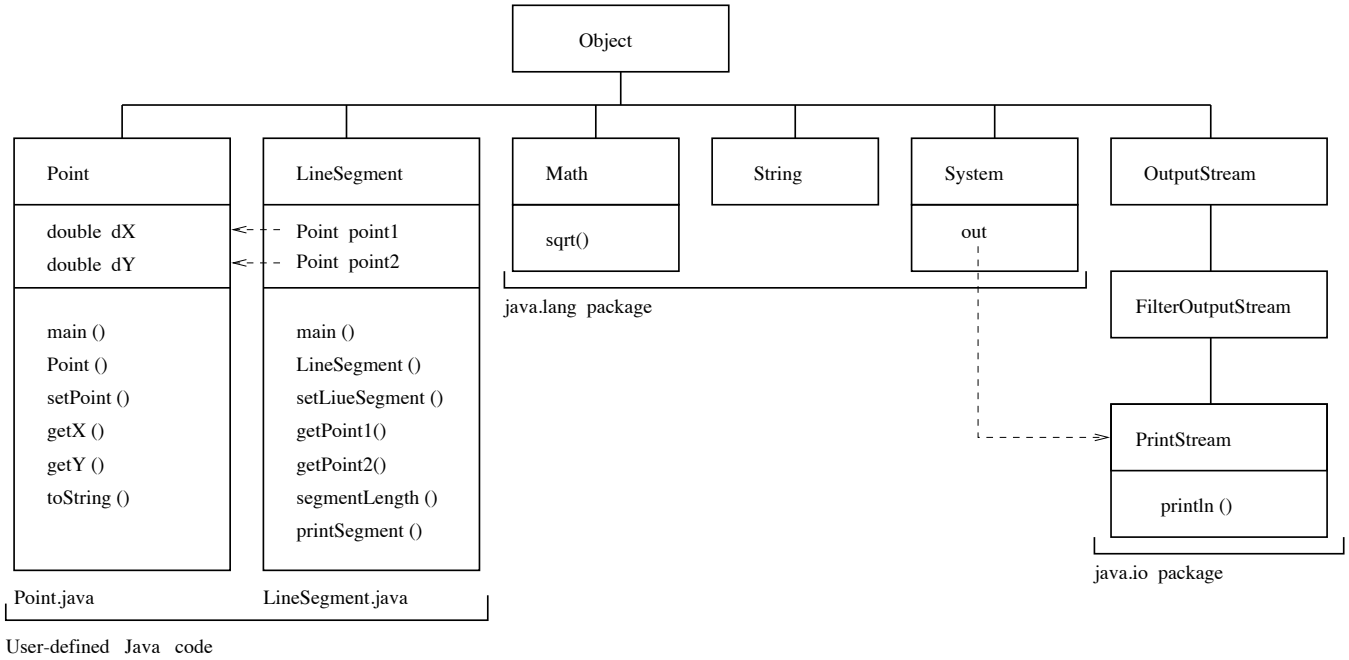

Figure 8.9. Class hierarchy for line segment program.

```
public class Point {
    protected double dX, dY;

    ..... methods for class Point...
}
```

and the line segment by

```
public class LineSegment {
    protected Point p1, p2;

    ..... methods for class LineSegment...
}
```

The connectivity between these classes is indicated by the dashed arrows in Figure 8.9.

Constructor Methods

Both the `Point` and `LineSegment` classes support multiple constructor methods. We look at the former in detail. As demonstrated in the `Point` test program, `Point` objects can be created using either of the methods:

```
// Constructor method : just create instance of class.

public Point() { }

// Create instance of class and set (x,y) coordinates.

public Point( double dX, double dY ) {
   setPoint( dX, dY );
}
```

The default constructor (i.e., `public Point()` {}) must be explicitly listed when more than one constructor is used in a class.

8.11 Application Program: Design of a Data Array

This program implements a simple data array; that is, a class that stores a one-dimensional array of floating-point numbers and its name, and methods to compute basic statistics (e.g., minimum, maximum and average array element values) and arithmetic operations (e.g., sum and difference of two data arrays).

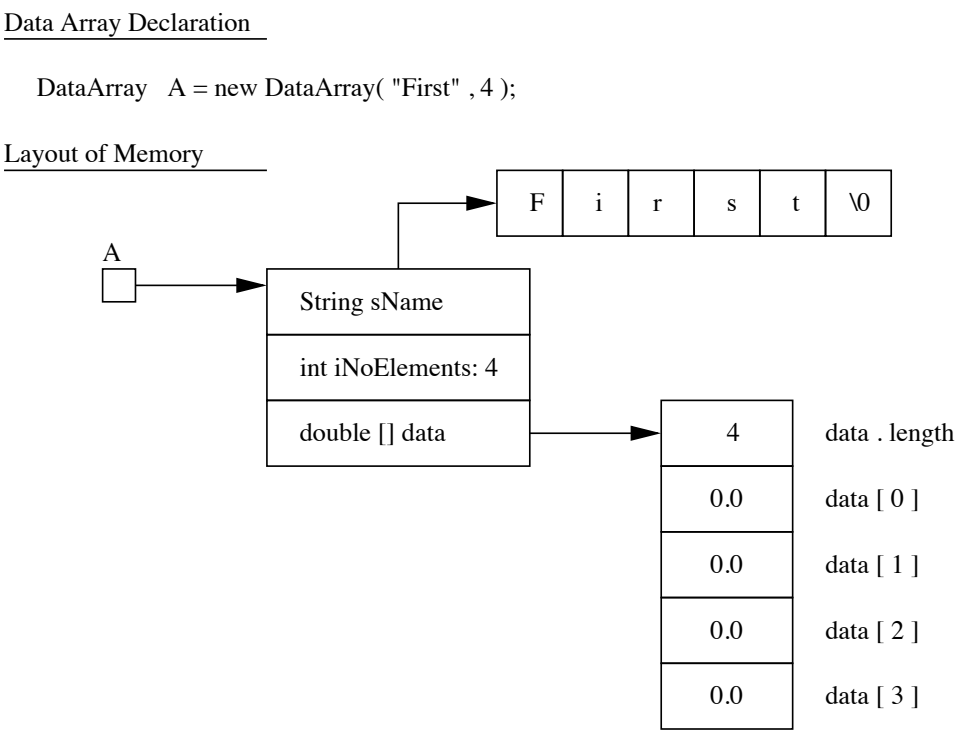

Figure 8.10. Layout of memory for a data array.

Figure 8.10 shows a typical statement for setting up a data array object, followed by the layout of memory that is generated.

source code

```
/*
 *  ====================================================================
 *  DataArray.java: Compute simple operations on a one-dimensional array
 *                  of double precision floating point numbers.
 *
 *  Written by: Mark Austin                                    April 2009
 *  ====================================================================
 */

import java.lang.Math;
import java.util.*;
import java.io.*;
import java.text.*;
```

```
public class DataArray {
    private String        sName;
    private    int  iNoElements;
    private double  data[];

    // Define NoColums to be a "constant" .... (see use below)

    public final static int NoColumns = 5;

    // Array constructors ....

    DataArray( int iNoElements )  {
        this.iNoElements = iNoElements;
        this.data = new double [ iNoElements ];

        // Initialize array elements (strictly speaking, don't need to do this)...

        for(int i = 0; i < iNoElements; i++)
            setElement( i, (double) 0.0 );
    }

    DataArray( String sName, int iNoElements )  {
        this.sName = sName;
        this.iNoElements = iNoElements;
        this.data = new double [ iNoElements ];
    }

    DataArray( String sName ) {
        this.sName = sName;
    }

    // ======================================
    // Set/get name for data array
    // ======================================

    public void setName ( String sName ) {
       this.sName = sName;
    }

    public String getName() { return this.sName; }

    // ======================================
    // Retrieve and set matrix element values
    // ======================================

    public double getElement( int i ) {
        double returnValue;

        if( i < 0 || i >= iNoElements ) throw new
            RuntimeException("*** In getElement(): Array element index out of range");

        returnValue = data[i];
        return returnValue;
    }
```

```
public void setElement( int i, double value )  {

    if( i < 0 || i >= iNoElements ) throw new
        RuntimeException("*** In setElement(): Array element index out of range");

    this.data[i] = value;
}

// ==========================================
// Convert array to string format ....
// ==========================================

public String toString() {
    String s = "Array: " + this.sName + "\n";

    for (int i = 1; i <= iNoElements; i = i + 1) {
         s += s.format(" %10.3e", data[i-1]);
         if (i % NoColumns == 0 || i == iNoElements )
             s += "\n";
    }

    return s;
}

/* ================================================= */
/* Method to read an array of data from a file       */
/* ================================================= */

public void readDataFile( String fileName ) throws IOException {
    String sLine;

    // [a] Open reader to the input file ....

    FileReader inputFile = new FileReader( fileName );
    BufferedReader    in = new BufferedReader( inputFile );

    // [b] Read file contents ...

    try {

         // [b.1] Get no of lines in input file ....

         sLine = in.readLine();
         StringTokenizer st = new StringTokenizer(sLine);
         if ( st.hasMoreTokens() == true )
            this.iNoElements = Integer.parseInt( st.nextToken() );
         else
            System.out.println("*** ERROR in input file ");

         // [b.2] Dynamically allocate memory for the array ....

         this.data = new double [ iNoElements ];

         // [b.3] Read measurements from data file
```

```
 for ( int i = 1; i <= iNoElements; i = i + 1 ) {
             sLine = in.readLine();
             StringTokenizer st1 = new StringTokenizer( sLine );
             if ( st1.hasMoreTokens() == true ) {
                double dItem = Double.parseDouble ( st1.nextToken() );
                this.setElement ( i-1, dItem );
             }
          }
    }

    catch (FileNotFoundException e){}
    catch (EOFException e){}

    // [c] Close input file

    in.close();
}

// =======================================================
// Compute sum and difference of two data arrays...
// =======================================================

public DataArray Add( DataArray dA ) {

   // Check compatibility of array lengths

   if( this.iNoElements != dA.iNoElements ) throw new
       RuntimeException("*** In Add(): Incompatible array lengths");

   // Compute sum of data arrays ....

   DataArray daSum = new DataArray( this.iNoElements );
   for(int i = 0; i < iNoElements; i++)
       daSum.data[i] = data[i] + dA.data[i];

   return (daSum);
}

public DataArray Sub( DataArray dA ) {

   // Check compatibility of array lengths

   if( this.iNoElements != dA.iNoElements ) throw new
       RuntimeException("*** In Sub(): Incompatible array lengths");

   // Compute difference of data array values ....

   DataArray daDiff = new DataArray( this.iNoElements );
   for(int i = 0; i < iNoElements; i++)
       daDiff.data[i] = data[i] - dA.data[i];

   return (daDiff);
}

// =======================================================
```

```
// Compute basic statistics on array values ....
// ====================================================

public double max() {
   double dMaxValue = data[0];

   for(int i = 1; i < iNoElements; i++)
       dMaxValue = Math.max( data[i], dMaxValue );

   return (dMaxValue);
}

public double min() {
   double dMinValue = data[0];

   for(int i = 1; i < iNoElements; i++)
       dMinValue = Math.min( data[i], dMinValue );

   return (dMinValue);
}

public double average () {
   double dSum = 0.0;

   for(int i = 0; i < iNoElements; i++)
       dSum = dSum + data[i];

   return (dSum/iNoElements);
}

public double range () {
   return (max() - min());
}

public double std() {
  double dMean = average();

  double dSum = 0.0;
  for (int i = 0; i < iNoElements; i = i + 1)
     dSum = dSum + data[i]*data[i];

  return Math.sqrt(dSum/iNoElements - dMean*dMean);
}

// ============================================
// Exercise methods in array class ....
// ============================================

public static void main(String[] args) {

   // Create, initialize and print a small array .....

   DataArray A = new DataArray("First", 4);
   A.setElement( 0, 1.0);
   A.setElement( 1, 2.0);
```

```
        A.setElement( 2, 3.0);
        A.setElement( 3, 4.0);

        System.out.println( A.toString() );
        System.out.println( A.getName() + ".(Max element value) = " + A.max() );
        System.out.println( A.getName() + ".(Min element value) = " + A.min() );
        System.out.println( A.getName() + ".(Range) = " + A.range() );
        System.out.println( A.getName() + ".(Average value)      = " + A.average() );
        System.out.println( A.getName() + ".(Standard Deviation) = " + A.std() );
        System.out.println( "" );

        // Initialize and print a second data array ...

        DataArray B = new DataArray("Second", 4);
        B.setElement( 0, 5.0);
        B.setElement( 1, 7.0);
        B.setElement( 2, 6.0);
        B.setElement( 3, 8.0);
        System.out.println( B );

        // Compute/print A + B ...

        DataArray daSum  = A.Add(B);
        daSum.setName("A+B");
        System.out.println( daSum );

        // Compute/print A - B ...

        DataArray daDiff = A.Sub(B);
        daDiff.setName("A-B");
        System.out.println( daDiff );

        // Read data from a file ....

        DataArray sensor1 = new DataArray( "Sensor 1" );

        try {
           sensor1.readDataFile( "sensor1.txt" );
        }
        catch (IOException e){}

        // Print measurements ...

        System.out.println( sensor1.toString() );
    }
}
```

Now let the contents of file `sensor1.txt` be:

```
7
0.210
0.211
0.210
```

```
0.211
0.212
0.215
0.213
```

The script of code shows the commmand input and output generated by DataArray.

```
prompt >> java DataArray
Array: First
  1.000e+00  2.000e+00  3.000e+00  4.000e+00

First.(Max element value) = 4.0
First.(Min element value) = 1.0
First.(Range) = 3.0
First.(Average value)      = 2.5
First.(Standard Deviation) = 1.118033988749895

Array: Second
  5.000e+00  7.000e+00  6.000e+00  8.000e+00

Array: A+B
  6.000e+00  9.000e+00  9.000e+00  1.200e+01

Array: A-B
 -4.000e+00 -5.000e+00 -3.000e+00 -4.000e+00

Array: Sensor 1
  2.100e-01  2.110e-01  2.100e-01  2.110e-01  2.120e-01
  2.150e-01  2.130e-01

prompt >>
```

Key points to note are as follows:

- Notice that when an object of type data array is created, the array element values are automatically set to zero. Individual array elements can be initialized with statements of the type:

  ```
  A.setElement ( 0, 1.0 );
  A.setElement ( 1, 2.0 );
  ```

  and so forth.

- The method readData() reads arrays from a file located locally on your computer.

  File format: The first line contains the number of elements in the array. Then, one array element is located on each line. See, for example, the layout of data in sensor1.txt .

8.12 Hiding Data in Object Oriented Software

As already noted, encapsulation involves separating the interface of a class from its implementation. There are two key benefits in the use of encapsulation:

1. You can't accidentally corrupt the value of a field – instead, you have to use a method to change a value.
2. The separation of interface and implementation makes it easier to modify the code within a class without breaking any other code that uses it.

In this example we create classes for circle and colored circle specifications, and then test classes to exercise the methods within each class.

The relationship between the classes is as follows:

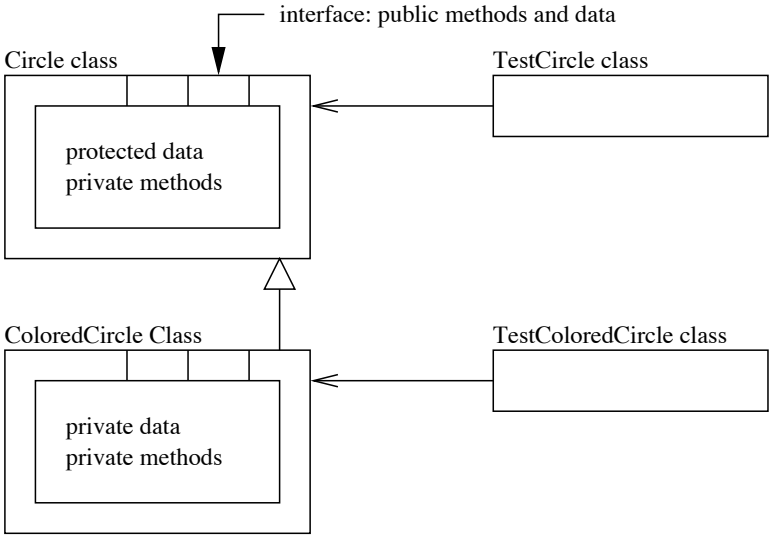

Figure 8.11. Relationship between the classes Circle and ColoredCircle, and their corresponding test classes.

As illustrated in Figure 8.11 encapsulation involves:

1. **Data Hiding.** Variables and methods internal to the object implementation will be declared as private and/or protected. Private data is only available within the class in which it is declared. Protected data is only available within the class in which it is declared, and, its subclasses.
2. **Access Methods.** Access to data/properties of the object will only be possible through setXXX() and getXXX() methods. For example, the Circle class contains the methods setRadius() and getRadius().

Source code for Circle.java

source code

```
/*
 *  ====================================================================
 *  Circle(): Implementation of the Circle class where data and circle
 *            properties can only be accessed through an interface.
 *
 *  Note. It is common programming practice to begin variable names
 *  with an underscore (e.g., _xxxx) when variables are either private
 *  or protected.
 *
 *  Written by: Mark Austin                                  April, 2009
 *  ====================================================================
 */

import java.lang.Math.*;

public class Circle {
    protected double _dX, _dY, _dRadius;

    // Constructor methods ...

    public Circle () {}

    public Circle( double dX, double dY, double dRadius ) {
        _dX = dX;
        _dY = dY;
        _dRadius = dRadius;
    }

    // Compute the circle area ....

    private double Area() {
       return Math.PI*_dRadius*_dRadius;
    }

    // Create public interface for variables and area computation....

    public void setX (double dX) { _dX = dX; }

    public double getX () { return _dX; }

    public void setY (double dY) { _dY = dY; }

    public double getY () { return _dY; }

    public void setRadius (double dRadius ) { _dRadius = dRadius; }

    public double getRadius () { return _dRadius; }

    public double getArea() { return Area(); }

    // Copy circle parameters to a string format ...

    public String toString() {
       return "(x,y) = (" + _dX + "," + _dY + "): Radius = " + _dRadius;
    }
```

```
}
```

Source code for ColoredCircle.java

source code

```
/*
 *  ====================================================================
 *  ColoredCircle(): Implementation of the ColoredCircle class where
 *                   data and circle properties can only be accessed
 *                   through an interface.
 *
 *  Written By: Mark Austin                                  April 2009
 *  ====================================================================
 */

import java.awt.Color;

public class ColoredCircle extends Circle {
    private Color _color;

    // Constructor methods

    public ColoredCircle() {
        super();
        _color = Color.blue;
    }

    public ColoredCircle( double dX, double dY, double dRadius, Color color ) {
        super();

        _dX = dX;
        _dY = dY;
        _dRadius = dRadius;
        _color   = color;
    }

    // Set and retrieve colors ....

    public void setColor( Color color ) {
       _color = color;
    }

    public String getColors() {
       return "Color (r,g,b) = (" + _color.getRed() +
              "," + _color.getGreen() + "," + _color.getBlue() + ")";
    }
}
```

Exercise the Circle Interface (TestCircle.java)

source code

```
/*
 *  ====================================================================
 *  TestCircle(): Exercise methods in the Circle class.
 *
 *  Written by: Mark Austin                                  April, 2009
 *  ====================================================================
 */

import java.lang.Math.*;

public class TestCircle {

    public static void main( String [] args ) {

       System.out.println("Exercise methods in class Circle");
       System.out.println("================================");

       Circle cA = new Circle();
       cA.setX(1.0);
       cA.setY(2.0);
       cA.setRadius(3.0);

       Circle cB = new Circle( 1.0, 2.0, 2.0 );

       System.out.printf("Circle cA : %s\n", cA.toString() );
       System.out.printf("Circle cA : Area = %5.2f\n", cA.getArea() );

       System.out.printf("Circle cB : %s\n", cB );
       System.out.printf("Circle cB : Area = %5.2f\n", cB.getArea() );
    }
}
```

The program output is:

```
prompt >>
prompt >> java TestCircle
Exercise methods in class Circle
================================
Circle cA : (x,y) = (1.0,2.0): Radius = 3.0
Circle cA : Area = 28.27
Circle cB : (x,y) = (1.0,2.0): Radius = 2.0
Circle cB : Area = 12.57
prompt >>
```

Exercise the ColoredCircle Interface (TestColoredCircle.java)

source code

```
/*
 *  ====================================================================
```

```
 *  TestColoredCircle(): Test interface methods in ColoredCircle.
 *
 *  Written By: Mark Austin                                  April 2009
 *  ============================================================
 */

import java.awt.Color;

public class TestColoredCircle {

    public static void main( String [] args ) {

       System.out.println("Exercise methods in class ColoredCircle");
       System.out.println("=====================================");

       // Create, initialize, and print circle "cA" ...

       ColoredCircle cA = new ColoredCircle( 1.0, 2.0, 3.0, Color.blue );
       cA.setX(1.0);
       cA.setY(2.0);
       cA.setRadius(3.0);
       cA.setColor( Color.blue );

       System.out.println( "Circle cA:" + cA.toString() );
       System.out.println( cA.getColors() );

       // Create, initialize, and print circle "cB" ...

       ColoredCircle cB = new ColoredCircle( -1.0, -2.0, 3.0, Color.orange );
       System.out.println( "Circle cB:" + cB.toString() );
       System.out.println( cB.getColors() );
    }
}
```

The program output is:

```
prompt >>
prompt >> java TestColoredCircle
Exercise methods in class ColoredCircle
=======================================
Circle cA:(x,y) = (1.0,2.0): Radius = 3.0
Color (r,g,b) = (00000000,0,255)
Circle cB:(x,y) = (-1.0,-2.0): Radius = 3.0
Color (r,g,b) = (255,200,0)
prompt >>
```

8.13 Exercises

This section covers Java programming with objects. It is a work in progress!

8.1 As shown in Figure 8.12 below, rectangles may be defined by the (x,y) coordinates of corner points that are diagonally opposite.

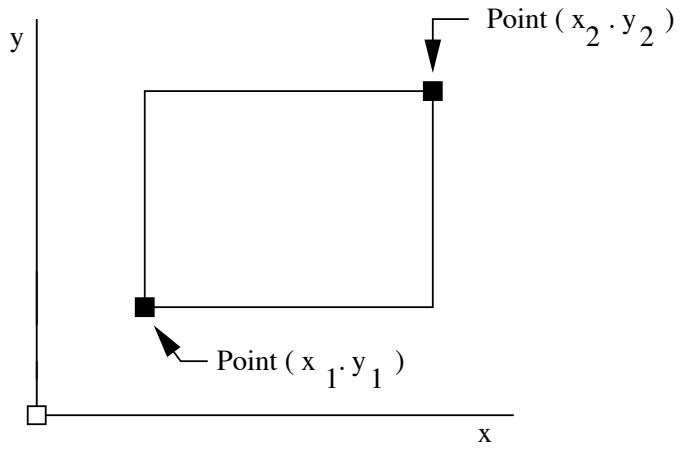

Figure 8.12. Definition of a rectangle via diagonally opposite corner points

With this definition in place, the following script of code is a very basic implementation of a class for creating and working with rectangle objects.

```
/*
 *  ============================================================================
 *  Rectangle.java : A library of methods for creating and managing rectangles
 *
 *  double      area() -- returns the area of a rectangle
 *  double perimeter() -- returns the perimeter of a rectangle
 *
 *  Written By : Mark Austin                                        November 2005
 *  ============================================================================
 */

import java.lang.Math;

public class Rectangle {
   protected double dX1, dY1;  // Coordinate (x,y) for corner 1....
   protected double dX2, dY2;  // Coordinate (x,y) for corner 2....

   // Constructor methods ....

   public Rectangle() {}

   public Rectangle( double dX1, double dY1, double dX2, double dY2 ) {
      this.dX1     = dX1; this.dY1     = dY1;
      this.dX2     = dX2; this.dY2     = dY2;
   }
```

```
   // Convert rectangle details to a string ...

   public String toString() {
          return "Rectangle: Corner 1: (x,y) = " + "(" + dX1 + "," + dY1 + ")\n" +
                 "           Corner 2: (x,y) = " + "(" + dX2 + "," + dY2 + ")\n";
   }

   // ===================================
   // Compute rectangle area and perimeter
   // ===================================

   public double area() {
       return Math.abs( (dX2-dX1)*(dY2-dY1) );
   }

   public double perimeter() {
       return 2.0*Math.abs(dX2-dX1) + 2.0*Math.abs( dY2-dY1 );
   }

   // Exercise methods in the Rectangle class ....

   public static void main ( String args[] ) {

      System.out.println("Rectangle test program     ");
      System.out.println("==========================");

      // Setup and print details of a small rectangle....

      Rectangle rA = new Rectangle( 1.0, 1.0, 3.0, 4.0 );
      System.out.println( rA.toString() );

      // Print perimeter and area of the small rectangle....

      System.out.println( "Perimeter = " + rA.perimeter() );
      System.out.println( "Area      = " + rA.area() );
   }
}
```

The script of program output is as follows:

```
Script started on Fri Apr 21 10:17:29 2006
prompt >>
prompt >> java Rectangle
Rectangle test program
==========================
Rectangle: Corner 1: (x,y) = (1.0,1.0)
           Corner 2: (x,y) = (3.0,4.0)

Perimeter = 10.0
Area      = 6.0
prompt >> exit
Script done on Fri Apr 21 10:17:39 2006
```

The Rectangle class has methods to create objects (i.e, Rectangle), convert the details of a rectan-

gle object into a string format (i.e., toString), and compute the rectangle area and perimeter (i.e., area() and permieter(), respectively). The implementation uses two pairs of doubles (dX1, dY1) and (dX2, dY2) to define the corner points.

Suppose that, instead, the corner points are defined via a Vertes class, where

```
public class Vertex {
    protected double dX, double dY

    ..... details of constructors and other methods removed ...
}
```

The appropriate modification for Rectangle is:

```
public class Rectangle {
   protected Vertex vertex1;  // First corner point....
   protected Vertex vertex2;  // Second corner point....

   ..... details rectangle removed ....
}
```

Fill in the missing details (i.e., constructors and toString() method) of class Vertex. Modify the code in Rectangle to use Vertex class. The resulting program should have essentially has the same functionality as the original version of Rectangle. **Hint.** Your implementation should make use of the toString() method in Vertex.

8.2 There are lots of problems in engineering where the position of point needs to be evaluated with respect to a shape. Evaluation procedures can be phrased in terms of questions. For example, is the point inside (or outside) the shape?; Does the point lie on the boundary of the shape?; Does the point lie above/below the shape? Does the point lie to the left or right of the shape?; How far is the point from the perimeter?

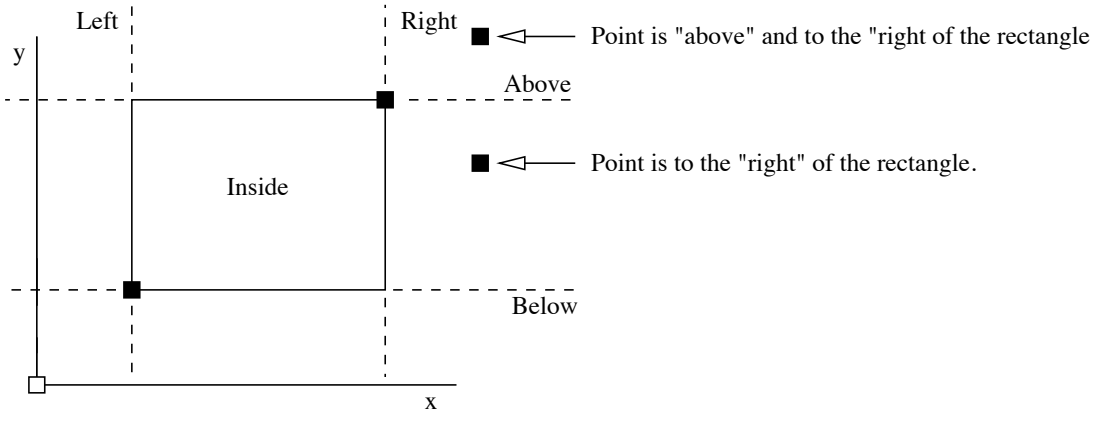

Figure 8.13. Classification of an (x,y) cooordinate relative to a rectangle

Figure 8.13 illustrates these ideas for one of the simplest cases possible, one point and a rectangle. Extend the functionality of the Rectangle class so that the possition of a point can be evaluated with respect to a specific rectangle object.

The appropriate method declarations are as follows:

```
public boolean isInside ( Vertex v ) { ... }
public boolean isOutside ( Vertex v ) { ... }
public boolean isOnPerimeter ( Vertex v ) { ... }
public boolean isAbove ( Vertex v ) { ... }
public boolean isBelow  ( Vertex v ) { ... }
public boolean isLeft ( Vertex v ) { ... }
public boolean isRight ( Vertex v ) { ... }
```

If the (x,y) coordinates of a vertex are inside a particular rectangle, then `isInside ()` should return `true.` Otherwise, it should return `false`. From Figure 8.13 is should evident that some points will result in multiple methods returning true. For example, points in the top right-hand side of the coordinate system will be outside, to the right, and above the rectangle.

Fill in the details of each method, and then develop a test program to exercise the procedures. Perhaps the most straight forward way of doing this is to write an extensive set of tests in the main() method for class Rectangle.

8.3 Figure 8.14 shows three types of spatial relationship (touching, overlap, and enclosure) between two rectangle.

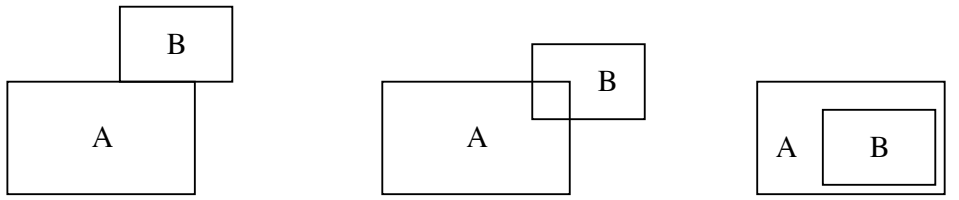

Figure 8.14. Spatial relationships between two rectangles

Extend the functionality of the Rectangle class to support the evaluation of these three types of relationships.

8.4 A data array class is a one dimensional array of floating-point numbers together with methods to compute statistics on the stored values (e.g., maximum value, minumum value). Download, compile, and run the DataArray java code from the class web site. Re-code the rainfall analysis program so that it uses the DataArray facilities.

Hint. Your solution will have two source code files, DataArray.java and RainfallAnalysis.java. Don't change any of the code in DataArray.java; just write a new version of RainfallAnalysis.java. The files before and after compilation should be:

```
                Before                          After
====================          ====================
RainfallAnalysis.java          RainfallAnalysis.java
DataArray.java                        DataArray.java
                                RainfallAnalysis.class
                                       DataArray.class
```

```
====================          ====================
```

You should find that your implementation is considerably shorter than in Problem 22.

8.5 This problem will give you practice at using the DataArray class to read, compute, and print the statistics of data collected from a structural engineering experiment in which strain measurements are recorded over an extended period of time.

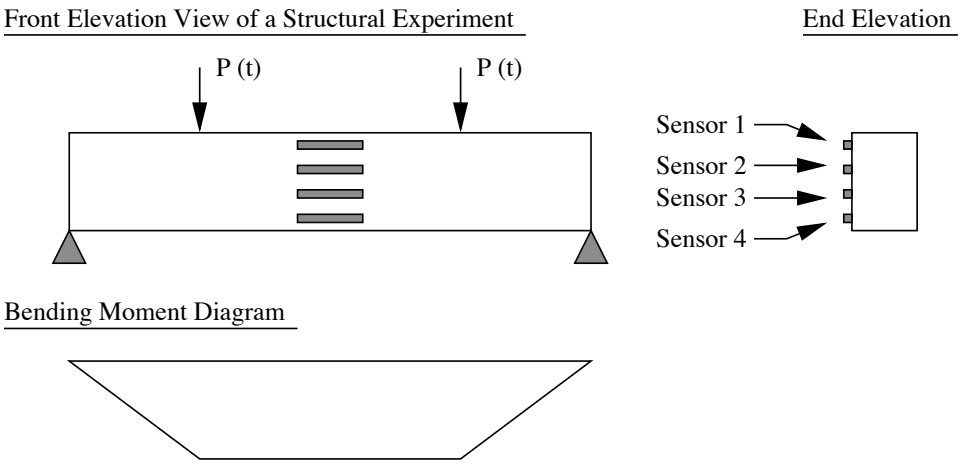

Figure 8.15. Structural experiment.

Figure 8.15 shows front and end elevation views of the experimental setup, and the bending moment diagram that will result from the symmetrically applied loading. Four sensors are attached to the center of the beam (where the bending moment will be constant. Since shear forces will be zero, any cracks that will develop will be vertical).

Now suppose that four files (i.e., `sensor1.txt`, `sensor2.txt`, `sensor3.txt` and `sensor4.txt`) contain the sensor strain measurements:

```
Sensor 1         Sensor 2          Sensor 3         Sensor 4
-------------------------------------------------------------
  -0.010          -0.0025            0.0025           0.010
  -0.011          -0.0030            0.0026           0.011
  -0.010          -0.0032            0.0027           0.010
  -0.011          -0.0034            0.0030           0.015
  -0.012          -0.0036            0.0100           0.120
  -0.015          -0.0040            0.0150           0.250
  -0.017          -0.0042            0.0250           ***** <- failed!
-------------------------------------------------------------
```

Things to do:

1. Download, compile, and run the DataArray.java program from the java examples web page.
2. Create four data files for the experimental data. I suggest that you call them `sensor1.txt` .. etc. Notice that sensor 4 fails, and hence contains an incomplete set of measurements.

3. Write a program (e.g., ExperimentalAnalysis.java) that will read and store each set of experimental measurements in a DataArray object. For each set of data, compute and print the maximum and minimum values, the range, average and standard deviation.

Your solution should have six files: ExperimentalAnalysis.java (which you will write), DataArray.java (given), and the data files `sensor1.txt` through `sensor4.txt`.

8.6 Suppose that we need to compute the sum of the geometric series,

$$\text{Sum}(c, n) = c + c^2 + c^3 + \cdots + c^n, \tag{8.1}$$

where "c" is a complex number of the form c = a + bi, and "n" is a positive integer. The sum of a geometric series can be computed efficiently in two ways; (1) Using Horner's rule to re-write the series as,

$$\text{Sum}(c, n) = c\,(1 + c\,(1 + \cdots c(1 + c)) \cdots)\,, \tag{8.2}$$

and (2) Using a geometric summation formula,

$$\text{Sum}(c, n) = c + c^2 + \cdots + c^n = \left[\frac{c - c^{(n+1)}}{1 - c}\right] \tag{8.3}$$

Starting with the Complex.java package handed out in class, write a Java program that will prompt a user for the coefficients a, b, and n, and then compute the series summation using Horner's rule and the geometric summation formula. In each case, show that

$$(1 + 2i) + (1 + 2i)^2 + (1 + 2i)^3 \quad \text{evaluates to} \quad -13 + 4i. \tag{8.4}$$

8.7 If a, b, and c are complex numbers, show that solutions to

$$a \cdot x^2 \; + \; b \cdot x \; + \; c = 0 \tag{8.5}$$

are given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{8.6}$$

Use your library of methods for complex number arithmetic and equation 8.6 to show that solutions to

$$(2.00 + 2.00i)\,x^2 \; + \; (3.00 + 3.00i)\,x \; + \; (3.00 + 5.00i) = 0.0 \tag{8.7}$$

are $x_1 = -0.5445 - 1.2164i$ and $x_2 = -0.9555 + 1.2164i$.

Check that x_1 and x_2 are in fact solutions to Equation 8.7 by evaluating each term, and arranging real and complex components in a table of output.

Hint. You need to remember that all of the arithmetic in this problem needs to work with complex numbers. Hence, suppose that you want to compute the roots to the equation:

$$2x^2 - 3x + 2 = 0.0 \tag{8.8}$$

where the coefficients are all real numbers. In this context, you need to implement the equation as if it were written:

$$(2 + 0i)x^2 - (3 + 0i)x + (2 + 0i) = 0.0 \tag{8.9}$$

Your solution should consist of two files: Complex.java and Quadratic.java. You can download Complex.java from the class web site. Quadratic.java will simply call the methods in Complex.java; therefore, there is no need to change the contents of Complex.java.

8.8 Lots of problems in Civil Engineering boil down to evaluation of some sort of geometric relationship among various kinds of objects (e.g., points, lines, areas, volumes).

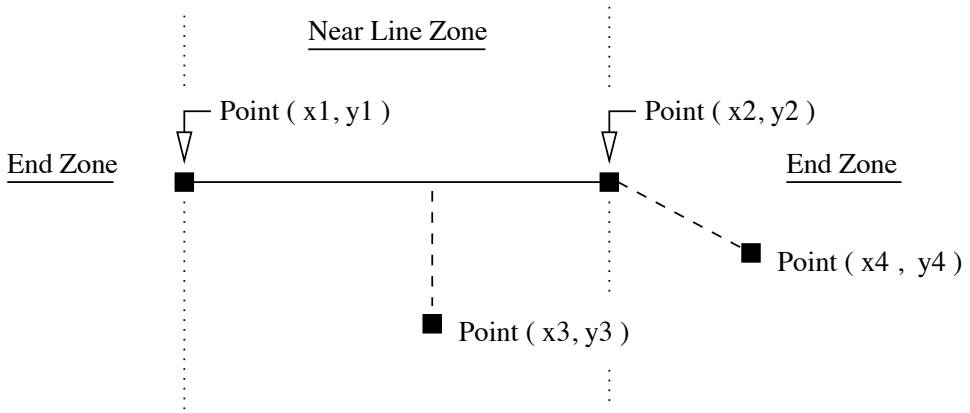

Figure 8.16. Distance of point to a line segment.

Figure 8.16 shows a line segment defined by two pairs of points (x_1, y_1) and (x_2, y_2), plus two single points located at coordinates (x_3, y_3) and (x_4, y_4).

Write a Java program that will:

1. Prompt a user for (x,y) coordinates of the line segment end points,

2. Prompt a user for (x,y) coordinates of a single point,

3. Compute and print the equation of the line segment,

4. Compute and print the distance of the single point from the line segment.

Notice that if the single point is located inside the "near line zone" then the point-to-line distance is defined by the perpendicular distance from the line to the point. This distance is defined by standard formulae. For cases where the single point lies in an "end zone" then the distance is simply the euclidean distance from the closest end point and the single point itself.