Simple Methods

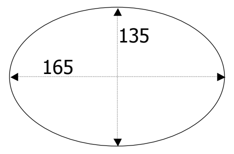
Chap. 4

Study Sections 4.1 – 4.4

Writing Reusable Formulas

Example

Last time, we designed and implemented a program to compute the area and the circumference of an *ellipse*.



These were our objects:

Description	Java Type	Kind	Name
program	**new class**	**--**	***Ellipse***
screen	`Screen`	**variable**	***theScreen***
prompt	`String`	**constant**	**--none--**
major axis	`double`	**variable**	***majorAxis***
minor axis	`double`	**variable**	***minorAxis***
keyboard	`Keyboard`	**variable**	***theKeyboard***
area	`double`	**variable**	***area***
circumference	`double`	**variable**	***circumference***
label	`String`	**constant**	**--none--**
π	`double`	**constant**	***PI***
half major axis	`double`	**variable**	***semiMajor***
half minor axis	`double`	**variable**	***semiMinor***

And these were our operations:

Description	Built-in/Class	Name
display strings	`Screen`	`print()`
read doubles	`Keyboard`	`readDouble()`
compute area		
– multiply doubles	**built-in**	`*`
compute circumference		
– multiply doubles	**built-in**	`*`
– add doubles	**built-in**	`+`
– divide doubles	**built-in**	`/`
– power	**Math**	`pow()`
– square root	**Math**	`sqrt()`
display doubles	`Screen`	`println()`

This was our algorithm:

1. Ask *theScreen* to display a prompt for the length and width of an ellipse.
2. Ask *theKeyboard* to read *majorAxis*, *minorAxis*.
3. Check validity of data (both numbers are positive).
4. Compute *semiMajor* = *majorAxis* /2.0; *semiMinor* = *minorAxis* /2.0.
5. Compute *area* = *PI* * *semiMajor* * *semiMinor*
6. Compute *circumference* = 2.0 * *PI* * sqrt(($semiMajor^2$ + $semiMinor^2$) / 2.0)
7. Display *area* and *circumference* with descriptive labels.

And this was our code:

```
/* Ellipse.java computes an ellipse's area and circumference.
 * Input:  Ellipse's length and width
 * Output: Ellipse's area and circumference
 * Written by L. Nyhoff for CPSC 185 Project 99 on 9/23/2002
 */

import ann.easyio.*;     // Keyboard, Screen

class Ellipse extends Object
{
  public static void main(String [] args)
  {
    Screen theScreen = new Screen();
    Keyboard theKeyboard = new Keyboard();

    // Get the axes
    theScreen.print("To compute the area and circumference of an "
                  + "ellipse,\n\tenter its major & minor axes: ");
    double majorAxis = theKeyboard.readDouble();
    double minorAxis = theKeyboard.readDouble();
```

```
    // Check validity of the input values
    theScreen.println("Nonegative values? " +
                      (majorAxis > 0 && minorAxis > 0) );

    // Compute area and circumference
    double semiMajor = majorAxis / 2.0;
    double semiMinor = minorAxis / 2.0;

    double area = Math.PI * semiMajor * semiMinor;
    double circumference = 2.0 * Math.PI * Math.sqrt(
         ( Math.pow(semiMajor, 2) + Math.pow(semiMinor, 2) ) / 2.0 ) ;

    // Output area and circumference
    theScreen.println("\nThe area is " + area +
                      "\nand the circumference is " + circumference);
  }
}
```

We executed it several times -- with test data and with other data; Here is the output produced to solve our original problem for an ellipse with major axis 165 and minor axis 135:

```
To compute the area and circumference of an ellipse,
        enter its major & minor axes: 165 135

The area is 17494.74408967816
and the circumference is 473.5892313120682
```

Maintenance

After a program has been developed and tested, it may be necessary sometime later to modify it by changing features or adding new ones to meet new requirements, to satisfy a customer, etc.

For example, for our problem, the values for the area and the circumference are each displayed with 13 decimal places. For this problem, this is probably more than necessary and/or may not be acceptable to the customer. (This would almost surely be the case if we were displaying monetary amounts!)

So, how can we fix this?

For output, we have used the `print()` and `println()` methods from the `Screen` class, which display numbers in a system-defined default manner. The `Screen` class also has several `printFormatted()` methods that allow us to specify our own formatting. The easiest one to use has the form:

See pp.52-54

```
theScreen.printFormatted(dubValue, decimalDigits);
```

Other forms allow one to specify the number of integer digits (before the dec. point) and to specify a fill character (e.g., '*' to fill blank spaces).

If we change our last output statement,

```
// Output area and circumference
theScreen.println("\nThe area is " + area +
                  "\nand the circumference is " + circumference);
```

to

```
// Output area and circumference
theScreen.print("\nThe area is ")
         .printFormatted(area, 2)
         .print("\nand the circumference is ")
         .printFormatted(circumference, 2)
         .println();
```

the output produced will be

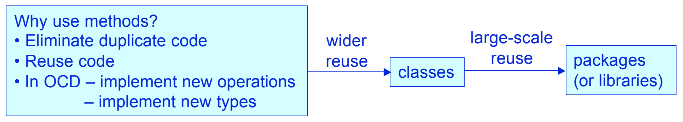
```
To compute the area and circumference of an ellipse,
        enter its major & minor axes: 165 135
Nonegative values? true

The area is 17,494.74
and the circumference is 473.59
```

Nice . . . but there's a bigger issue . . .

Problem

We worked fairly hard to create the expressions for an ellipse's area and circumference, but we have no way to directly reuse that work if we ever need these ellipse-related formulas again.

Solution: Encapsulate those formulas in *methods* (called *functions* in C++ and other languages).


Why use methods?
- Eliminate duplicate code
- Reuse code
- In OCD – implement new operations
  – implement new types

Some Terminology

We'll create our methods from the ground up, but first some definitions.

- Methods often receive values in special variables known as *parameters*. Each parameter is given a name and a type (like any other variable).
- A method can also return one value known as its *return value*. This return value also has a type. (It is `void` if no value is returned.)
- A value is returned by means of a `return` statement:

  `return expression;`

  (Not used or ***expression*** is omitted for **void** methods.)

13

Example – Version 2

```
/* Ellipse.java computes an ellipse's area and circumference.
 * Input:  Ellipse's length and width
 * Output: Ellipse's area and circumference
 * Written by L. Nyhoff for CPSC 185 Project 100 on 9/24/2002
 */

import ann.easyio.*;     // Keyboard, Screen

class Ellipse extends Object
{
  /* Method to find area of an ellipse
   * Receive: major and minor axes
   * Return:  area of ellipse
   */
  public static double EllipseArea(double major, double minor)
  {
    double semiMajor = major/2.0,
           semiMinor = minor/2.0;
    return Math.PI * semiMajor * semiMinor;
  }
```

14

```
/* Class method to find circumference of an ellipse
 * Receive: major and minor axes
 * Return:  circumference of ellipse
 */
public static double EllipseCircumference
                              (double major, double minor)
{
  double semiMajor = major/2.0,
         semiMinor  = minor/2.0;
  return 2.0 * Math.PI * Math.sqrt(
     (Math.pow(semiMajor, 2.0) + Math.pow(semiMinor, 2.0)) / 2.0 );
}


public static void main(String [] args)
{
  Screen theScreen = new Screen();
  Keyboard theKeyboard = new Keyboard();
```

15

```
  // Get the axes
  theScreen.print("To compute the area and circumference of an "
                + "ellipse,\n\tenter its major & minor axes: ");
  double majorAxis = theKeyboard.readDouble();
  double minorAxis = theKeyboard.readDouble();

  // Check validity of the input values
  theScreen.println("Nonegative values? " +
                    (majorAxis > 0 && minorAxis > 0) );

  // Compute area and circumference
  double area = EllipseArea(majorAxis, minorAxis);
  double circumference = EllipseCircumference(majorAxis, minorAxis);

  // Output area and circumference
  theScreen.println("\nThe area is " + area +
                    "\nand the circumference is " + circumference);
 }
}
```

16

Parameters

Parameters are method variables for which the caller can specify values. They are declared between the parentheses of a method's heading.

```
public static double EllipseArea(double major, double minor)
{
  double semiMajor = major/2.0,
         semiMinor  = minor/2.0;
  return Math.PI * semiMajor * semiMinor;
}
```

17

Arguments

When a method is *called* from another method, its caller can pass it values called *arguments* which are stored in the method's parameters.

```
 double area = EllipseArea(165, 135);

public static                         165          135
   double EllipseArea(double major, double minor)
   {
     double semiMajor = major/2.0,
            semiMinor  = minor/2.0;
     return Math.PI * semiMajor * semiMinor;
   }
```

The method then executes using its parameter values and sends its return value back to the caller.

18

General Form of Method Definition

p. 169

Syntax:

heading

```
modifiers returnType methodName(parameterDeclarations)
{
   statements
}
```

body

`modifiers`:	specify various features of the method (`static`, `public`, `private`, etc.)
`returnType`:	type of value returned by method, or `void` if it does not return a value
`methodName`:	identifier that names the method
`parameterDeclarations`:	list of parameters (separated by commas) for which values must be provided
`statements`:	specify the behavior of the method

19

In contrast to an *instance method*

public : other classes can access it

static: it's a class method; access it by sending message to the *class*, not to an *object*

```
public static double EllipseArea(double major, double minor)
{
  double semiMajor = major/2.0,
         semiMinor  = minor/2.0;
  return PI * semiMajor * semiMinor;
}
```

Returns a double value

Parameters: Receives two double values

A good name describes and documents what the method does

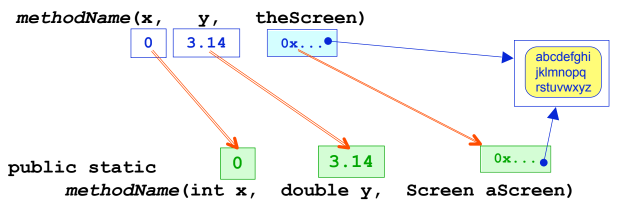
Notes

- When a method definition is placed inside a class definition, it can be called by other methods within that class with:
  `methodName(argument_list)`
- When a *public static* method definition is placed inside a class `ClassName`, it can be called by *methods in another class* with:
  `ClassName.methodName(argument_list)`
- **`void`** methods do not return a value but may have a **`return`** statement with no return value to return execution to the calling method.

Parameter Passing

- The number of arguments must match the number of parameters. The type of each argument must be compatible with (the same as or can be promoted to) the type of the corresponding parameter.
- Variables like **`semiMajor`** and **`semiMinor`** that are declared in the body of a method are called **locals**; they exist only while the **method is executing**. This means:
  - *They can only be accessed within the method.*
  - *Other methods can use the same identifiers as locals without conflict.*
- For primitive types, the value of an argument is ***copied*** to the corresponding parameter. Thus, *changing the parameter's value in the method will not change the argument in the calling method*.

- **Alias Problem**: For reference types, an argument is a ***handle***; it stores the ***address*** of the memory location
- where the object it refers to is stored. It is this address that gets copied to the corresponding parameter.



```
methodName(x,    y,    theScreen)

public static
      methodName(int x,  double y,  Screen aScreen)
```

Result: ***`aScreen`*** *and* ***`theScreen`*** *are* ***aliases*** *for the same object; i.e., they both* ***refer to the same object****.*
This means that if ***`methodName`*** *changes the object referred to by parameter* ***`aScreen`****, this also changes the object referred to by the argument* ***`theScreen`****.*

OCD with Methods

1. Describe behavior of program
2. Identify the objects in the problem
3. Identify the operations needed to solve the problem.
   If one isn't provided in the programming language:
   a. Build a **method** to perform it.
   b. Store the method in a **class** for which it represents an operation.
4. Develop an algorithm

Thus, for our ellipse problem, we really should store the area and circumference methods in a class of operations for ellipses.

An Ellipse Class

```
/* This ellipse class contains operations for ellipses:
     1. Compute area
     2. Compute circumference
     . . . other operations . . .
   Written by L. Nyhoff for CPSC 185 Project 100 on 9/24/2002
 */

class Ellipse extends Object
{
  /* Method to find area of an ellipse
   * Receive: major and minor axes
   * Return:  area of ellipse
   */
  public static double area(double major, double minor)
  {
    double semiMajor = major/2.0,
           semiMinor = minor/2.0;
    return Math.PI * semiMajor * semiMinor;
  }
```

```
  /* Class method to find circumference of an ellipse
   * Receive: major and minor axes
   * Return:  circumference of ellipse
   */
  public static double circumference(double major, double minor)
  {
    double semiMajor = major/2.0,
           semiMinor  = minor/2.0;
    return 2.0 * Math.PI * Math.sqrt(
      (Math.pow(semiMajor, 2.0) + Math.pow(semiMinor, 2.0)) / 2.0 );
  }

} // end of Ellipse class definition
```

To test our methods, we usually write a **driver program** that exercises each method with various sets of test data.

```
//--- Driver program to test the Ellipse class
//--- Written by L. Nyhoff for CPSC 185 Project 100 on 9/24/2002

import ann.easyio.*;

class EllipseDriver extends Object
{
  public static void main(String [] args)
  {
    Screen theScreen = new Screen();
    Keyboard theKeyboard = new Keyboard();

    theScreen.print("Enter major & minor axes of an ellipse: ");
    double majorAxis = theKeyboard.readDouble();
    double minorAxis = theKeyboard.readDouble();

    theScreen.println("\nThe area is " +
                      Ellipse.area(majorAxis, minorAxis) +
                      "\nand the circumference is " +
                      Ellipse.circumference(majorAxis, minorAxis));
  }
}
```

We can insert the definition of class **`Ellipse`** into our program:

```
//--- Driver program to test the Ellipse class
//--- Written by L. Nyhoff for CPSC 185 Project 100 on 9/24/2002

import ann.easyio.*;

// Insert definition of class Ellipse here

class EllipseDriver extends Object
{
  public static void main(String [] args)
  {
    Screen theScreen = new Screen();
    Keyboard theKeyboard = new Keyboard();

    theScreen.print("Enter major & minor axes of an ellipse: ");
    double majorAxis = theKeyboard.readDouble();
    double minorAxis = theKeyboard.readDouble();

    theScreen.println("\nThe area is " +
                      Ellipse.area(majorAxis, minorAxis) +
                      "\nand the circumference is " +
                      Ellipse.circumference(majorAxis, minorAxis));
  }
}
```

See Fig. 4.1

Or we can put the definition of class **`Ellipse`** in a separate file **`Ellipse.java`**, but in the same directory as the **`EllipseDriver.java`**, compile the two files separately, and then execute **`EllipseDriver`** with **`java EllipseDriver`**.

```
//--- Driver program to test the Ellipse class
//--- Written by L. Nyhoff for CPSC 185 Project 100 on 9/24/2002

import ann.easyio.*;

class EllipseDriver extends Object
{
  public static void main(String [] args)
  {
    Screen theScreen = new Screen();
    Keyboard theKeyboard = new Keyboard();

    theScreen.print("Enter major & minor axes of an ellipse: ");
    double majorAxis = theKeyboard.readDouble();
    double minorAxis = theKeyboard.readDouble();

    theScreen.println("\nThe area is " +
                      Ellipse.area(majorAxis, minorAxis) +
                      "\nand the circumference is " +
                      Ellipse.circumference(majorAxis, minorAxis));
  }
}
```

See modified (next slide) Fig. 4.5

29

Some Typos/Changes in Chapter 4:

- Page 178: Figure 4.3
  Change **`import ann.easyio.`**
  to **`import ann.easyio.*;`**

  Change **`EinsteinConvert`**
  to **`EinsteinConverter`**
- Page 190: Figure 4.5
  Delete **`import Sphere;`**

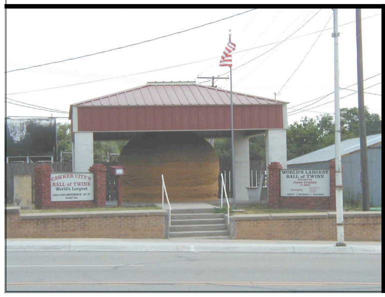
This was okay in earlier versions of Java, but in Java 1.4, **`import`** *can be used only with packages, not with class files.*

30

Work through the *Volume of a Sphere* example in Section 4.3. Not the use of OCD to design a solution to the original problem and how it is used in the same way to design the volume and mass methods.

Also read the summary on methods in Sec. 4.4 & Chap. Summary (pp.196-7)

Note the photo on p. 184



31