Chapter 4

Methods and Testing

4.1 Call by Value

Consider this code:

```
1 public static void cube(int n) {
2     n = n * n * n;
3 }
4
5 public static void main() {
6     int n = 3;
7     cube(n);
8     System.out.println("n is now " + n);
9 }
```

When you run this code, the output will be `n is now 3`—`n` will not have changed. Why? Because the *value* of an argument is assigned to the parameter. Figure 4.1 shows the call stack when the `cube` method is entered at line 1.

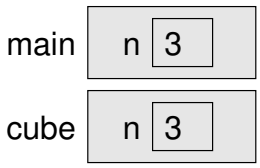

Figure 4.1: Stack diagram for code fragment at line 1.

The `n` on line 6 is local to `main`; the `n` on line 1 is local to `cube` and is a *copy* of the argument's value.

After line 2 is done executing, the stack looks like Figure 4.2. The variable `n` that belongs to `cube` has changed, but the `n` in `main` remains untouched.

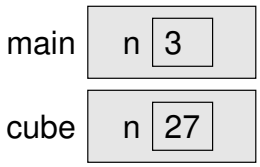

Figure 4.2: Stack diagram for code fragment after line 2.

This idea of assigning a copy of the argument's value to the parameter is known as *call by value*, and it is *always* used in Java. **Changing the value of a parameter never changes the value of the original argument.**

In order to change the value of `n` in `main`, the `cube` method must `return` a value that `main` can use to re-assign:

```
1 public static double cube(int n) {
2     n = n * n * n;
3     return n;
4 }
5
6 public static void main() {
7     int n = 3;
8     n = cube(n);
9     System.out.println("n is now " + n);
10 }
```

4.2 Exercises

Exercise 4.1 Presume you are selling a product that has a particular cost to build (say, $7.50) and a price that you charge the customer (say, $10.00). There are two ways to calculate your profit: profit off the bottom, which uses this formula:

$$bottom = 100 \cdot \frac{(sellingPrice - costToBuild)}{costToBuild}$$

You can also have profit off the top, which uses this formula:

$$top = 100 \cdot \frac{(sellingPrice - costToBuild)}{sellingPrice}$$

In our example, the profit off the bottom is $33\frac{1}{3}\%$ and the profit off the top is 25%.

Write methods named `profitOffTop` and `profitOffBottom`, both of which take the cost to build and selling price as `double` parameters and return the percentage as a `double`.

Your `main` method will ask the user for the cost to build and selling price. It will then calculate both profits and display them, properly labeled, with one digit to the right of the decimal point. Here is what the output might look like:

```
Enter cost to build: $7.50
Enter selling price: $10
The profit off the bottom is 33.3%.
The profit off the top is 25.0%
```

Hint: to output a percent sign using `printf`, use two percent signs in a row. For example:

```
double result = 0.12 * 37.0;
System.out.printf("12%% of 37 is %.1f\n", result);
```

Exercise 4.2 To determine the focal length of a camera lens, you use this formula:

$$\frac{1}{f} = \frac{1}{d_o} + \frac{1}{d_i}$$

Where d_o is the distance from the object to the lens and d_i is the distance from the lens to the camera's image sensor.

Write a program that implements a method named `reciprocal`, which takes a `double x` as its parameter and returns `1.0 / x` (the reciprocal).

The `main` method will ask the user for the distance to the object in meters, and the distance to the image sensor in centimeters. It will then calculate and display the focal length in millimeters. Use the `reciprocal` method in your calculation. Remember to do the unit conversion so that both distances are in millimeters (mm). There are 1000 mm in a meter and 10 mm in a centimeter.Here is what the program output might look like:

```
Enter distance from object to lens in meters: 3.8
Enter distance from lens to image sensor in cm: 9.5
The focal length is 92.68 mm.
```

Exercise 4.3 Write a program that calculates three different kinds of means (averages) of three numbers. This program will use the `reciprocal` method you wrote in the preceding exercise. Implement these methods:

- `reciprocal`, which takes a `double x` as its parameter and returns `1.0 / x` (the reciprocal).
- `arithmeticMean`, which takes three `double` values as parameters and returns their average:
$$\frac{a+b+c}{3}$$
- `geometricMean`, which takes three `double` values as parameters and returns the geometric mean:
$$\sqrt[3]{a \cdot b \cdot c}$$
(Hint: use `Math.pow(x, 1.0 / 3.0)` to get a cube root.)
- `harmonicMean`, which takes three `double` values as parameters and returns the harmonic mean:
$$\frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$$
Write this method to calculate the result as the reciprocal of the arithmetic mean of the individual reciprocals. (The arithmetic mean of the reciprocals is $\frac{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}{3}$; the reciprocal of *that* is the harmonic mean.) This is another instance where one method calls another method—in this case, many times—to do a calculation.

The `main` method will ask the user for three numbers and display their arithmetic, geometric, and harmonic means. Here is what the output might look like:

```
Enter first number: 1.5
Enter second number: 6.3
Enter third number: 2.4
Arithmetic mean: 3.40
Geometric mean:  2.83
Harmonic mean:   2.42
```

Exercise 4.4 In this exercise, you will write a program that prompts the user for information about a loan. The program will calculate the monthly payment for the loan, based on:

- The amount of the loan, also called the "principal".
- The annual interest rate as a percentage.
- The number of years of the loan.

Write a method named `calculatePayment` that takes the principal, annual percentage rate, and number of years as its parameters. The method calculates and returns the monthly payment on the loan with this formula:

$$payment = \frac{r \cdot p}{1 - (1 + r)^{-n}}$$

where p is the principal, r is the *monthly* interest rate as a decimal, and n is the number of *months* of the loan.

Although the preceding formula uses single-letter variables, it's preferable to use variable names like `principal`, `rate`, and `months` in your program for readability. (The annual percentage rate and months can use names like `annualRate` and `years`.)

The `main` method will prompt the user for the principal, annual percentage rate, and number of years. It will then call the `calculatePayment` method with this information and display the returned result, properly labeled. Your output should use `printf` to display the payment with exactly two digits following the decimal point.

In the following sample output, the dollar sign for the input is part of the prompt; the user does not type the dollar sign.

```
Enter principal of loan: $15000
Enter annual interest rate as a percent: 5.25
Enter number of years of the loan: 10
Your monthly payment is $160.94
```

Exercise 4.5 Consider a rectangular prism (also known as a "box") as shown in Figure 4.3.

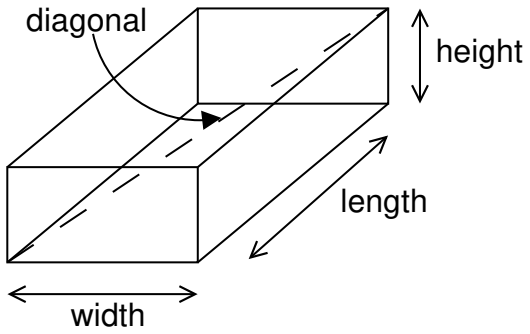

Figure 4.3: Rectangular prism with length, width, height, and diagonal labeled.

Write a program with these methods, all of which take the length, width, and height of the box as their three parameters and return a `double` result:

`getVolume`
: Returns the volume of the box.

`getSurfaceArea`
: Returns the surface area of the box.

`getDiagonal`
: Returns the diagonal distance of the box—the distance from the bottom left front corner of the box to the top right back corner.

The `main` method will prompt the user to enter the length, width, and height of a box and read them. It will then calculate and print the volume, surface area, and diagonal distance, properly labeled.

Here is an example of what the output might look like. Your program's output does not have to be identical to this, but it must reflect the same information:

```
Enter length of box: 7
Enter width of box: 3.5
Enter height of box: 4

Volume: 98.00 cubic units
Surface area: 133.00 square units
Diagonal distance: 8.79 units
```

Exercise 4.6 The *Pythagorean distance* between points (x_1, y_1) and (x_2, y_2) is calculated as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. The code to find the Pythagorean distance is in the *Think Java 2nd Edition* book. Take the code from the book—or write it yourself—as a method named `distance` that takes four parameters (the coordinates of the two points) and returns the Pythagorean distance between those points. All the parameters and the return value are `double`.

Another way to measure the distance between two points is the "city block" distance: the sum of the absolute x distance plus the absolute y distance:

$$|x_1 - x_2| + |y_1 - y_2|$$

For example, the city block distance from (1.5, 7) to (5, 2) is 8.5—the x distance is 3.5 ($|1.5 - 5|$) and the y distance is five ($|7 - 2|$).

Write a method named `cityBlockDistance` that takes the coordinates of the two points and returns the city block distance between them. All the parameters and the return value are `double`. *Hint*: Use `Math.abs` to calculate absolute value.

The `main` method will prompt the user to enter the coordinates; it will then calculate the Pythagorean distance using the `distance` method and print the returned result. Finally, `main` will call the `cityBlockDistance` method to calculate the city block distance and then print the returned result.

Exercise 4.7 The Pythagorean and city block distances are fine for a two-dimensional grid, but when you want to find the distance between two points

on the spherical earth, you don't use a straight line. Instead, you calculate the **great circle distance** from the latitude and longitude of the points.

Here is the formula where r is the radius of earth in kilometers (6371.009), the first point's latitude and longitude are lat_1, lon_1 and the second point's latitude and longitude are lat_2, lon_2:

$$d = r \cdot cos^{-1}(sin(lat_1) \cdot sin(lat_2) + cos(lat_1) \cdot cos(lat_2) \cdot cos(lon_1 - lon_2))$$

Important! The latitude and longitude in this formula are measured in *radians*, not degrees. You can use the `Math.toRadians` method to convert degrees to radians:

```
double degrees = 30.0;
double radians = Math.toRadians(degrees);
// Following line gives correct result (0.5)
System.out.println(Math.sin(radians));
```

Write a method named `greatCircleDistance` which takes the latitude and longitude of two points *in degrees* as its four parameters and returns the great circle distance, which will be in kilometers.

You will also write a method named `kmToMiles` which takes as its single parameter a number of kilometers and returns the equivalent number of miles by multiplying by 0.621371.

Here are the latitude and longitude (in degrees) of several cities:

- San José, California, USA: 37.333, -121.9
- Los Angeles, California, USA: 34.05, -118.25
- Seoul, South Korea: 37.56, 126.99
- Vienna, Austria: 48.2, 16.367

Use this information to write the `main` method, which will calculate and print, properly labeled, the distance from San José to Los Angeles and the distance from Seoul to Vienna. Print the distances in both kilometers and miles.

Here is the expected output:

```
Distance from San Jose to Los Angeles: 492 km (306 miles)
Distance from Seoul to Vienna: 8277 km (5143 miles)
```