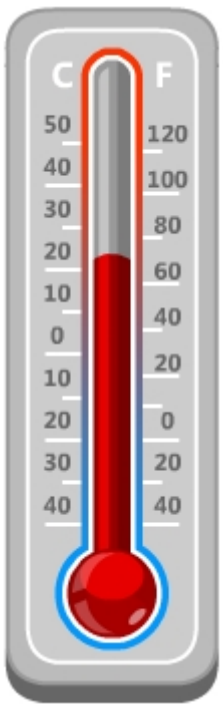
Chapter 2: Deciding on Program Description and Data Requirements

Deciding on Program Description and Data Requirements

One of the most common reasons for program problems is that someone started writing a program before understanding exactly what it should do. So before we jump in, let's lay out our goals for the program.

You're probably familiar with at least two temperature scales, Fahrenheit and Celsius, since they often appear side by side on thermometers.



A thermometer with two scales

Scientists often use a third temperature scale, Kelvin. I'd like a program that converts between all three of those scales. Here's the complete program description, and then we'll simplify it for this lesson.

Temperature Conversion Program Specification

Purpose:
Convert temperatures to Celsius, Fahrenheit, or Kelvin.

Description:
This program will accept temperatures in any of the three scales and display the equivalent values in the other two temperature scales.

Assumptions:

- Temperatures may not be below absolute zero, which is 0° on the Kelvin scale, –273.15° Celsius, and –459.67° Fahrenheit.

- C, F, and K will be the valid temperature types to specify which kind of temperature the user wants to enter.

- The program will keep performing conversions until the user enters Q as a temperature type. (Q stands for quit.)

- The conversion formulas are:
    - Celsius to Fahrenheit: F = C × 9 ÷ 5 + 32
    - Fahrenheit to Celsius: C = (F – 32) × 5 ÷ 9
    - Celsius to Kelvin: K = C + 273.15
    - Kelvin to Celsius: C = K – 273.15
    - Fahrenheit to Kelvin: K = (F – 32) × 5 ÷ 9 + 273.15
    - Kelvin to Fahrenheit: F = (K – 273.15) × 9 ÷ 5 + 32

Process:

1. Ask the user for a temperature type (K for Kelvin, C for Celsius, F for Fahrenheit, and Q to quit).
2. Read the user's entry from the keyboard.
3. If the temperature type is Q, end the program.
4. If the temperature type isn't C, F, or K, display an error message, and go back to Step 1.
5. Ask the user for the temperature to convert.
6. Read the user's entry.
7. If the temperature isn't a number, display an error, and go to step 5.
8. If the temperature is below absolute zero, display an error, and go to step 5.
9. Convert the value from the indicated temperature scale to the other two scales using the formulas from the assumptions.
10. Display the original entry and the converted results.
11. Go back to Step 1.

For this lesson, we'll do steps 5, 6, and 9.

Starting the Program

Start BlueJ. If there's a project open when you start it, choose the **Close** option from BlueJ's Project menu.

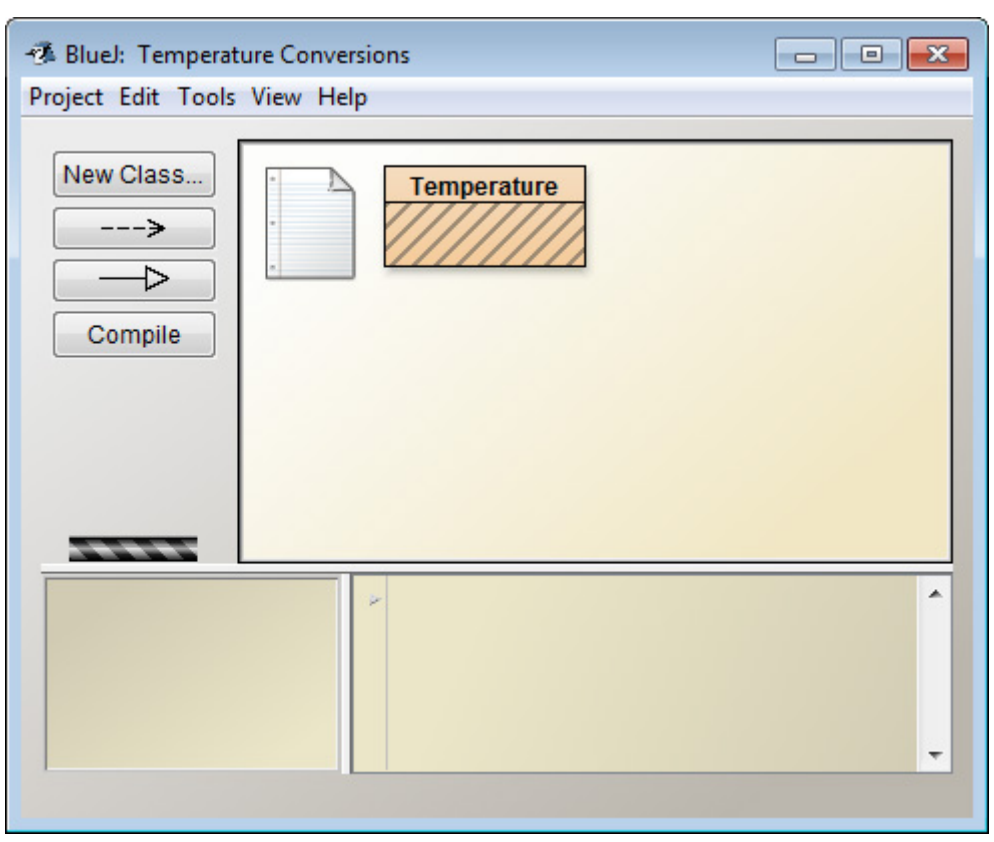
Now select the **New Project** option from the same menu. You can name your project anything that makes sense to you; I named mine "Temperature Conversions."

In object-oriented design, one of the first questions we ask is, "What *entities* (things) are involved in the project?" Since most classes in a project represent an entity, that will give us a good idea of how many classes might be involved. In this case, the only entity is the temperature we want to convert.

Let's create a new class to represent our temperature.

1. Select **New Class** from the Edit menu, or click the **New Class** button.

2. Name the class. I named mine Temperature. If you've done the same, here's what your BlueJ window will look like:



A new project with a class named Temperature

3. Double-click your new class to open the text editor, and we'll start on our code.

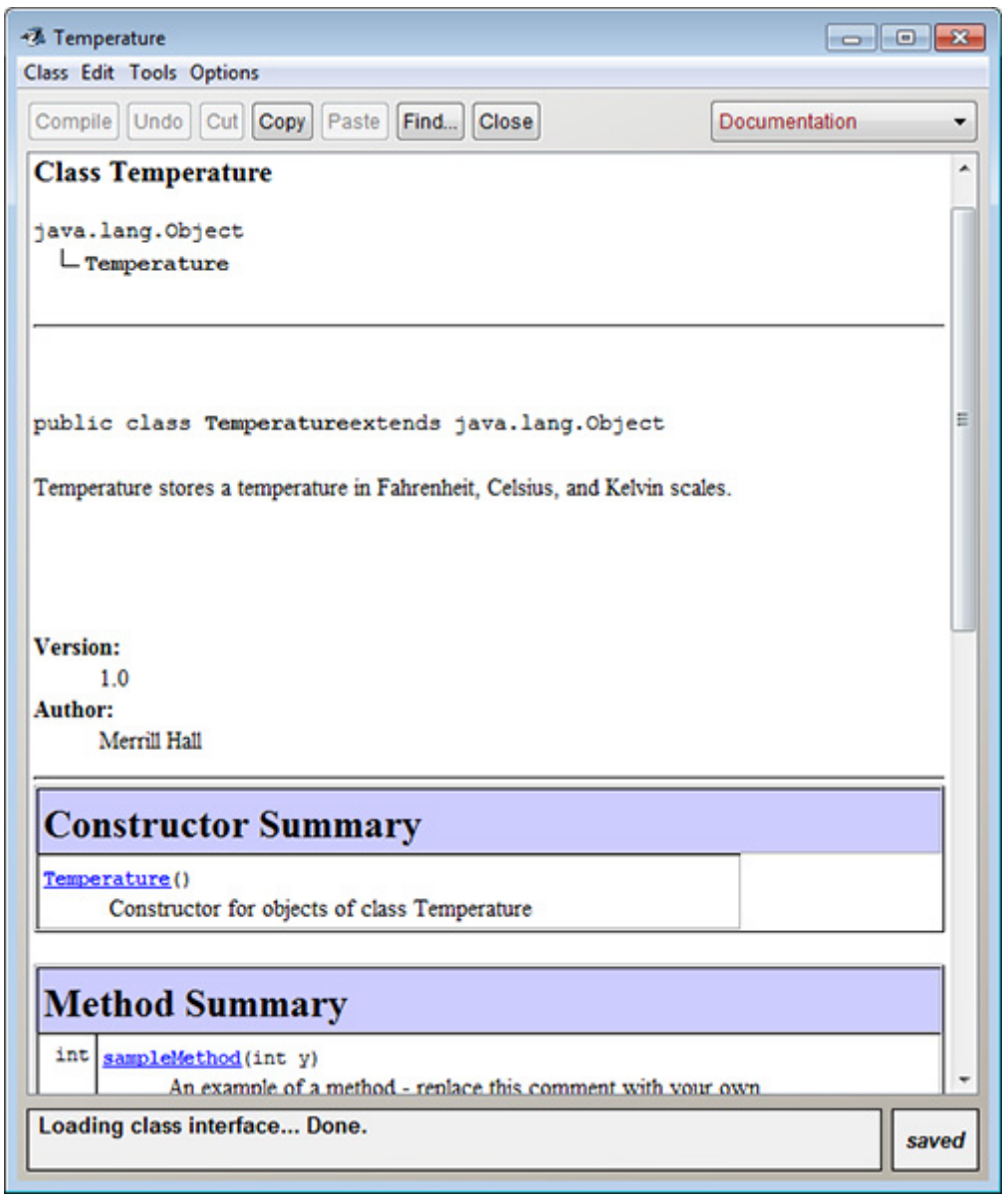
We need to update the comments at the start of the class with a description of what the class will do. I entered "Temperature stores a temperature in Fahrenheit, Celsius, and Kelvin scales." I also put in my name and a version number.

> **Why Add a Name and Version Number?** Tip
>
> I know this may seem like a waste of time, but it's best to maintain your program comments from the start so that it becomes a habit. It'll be important once you're writing more complex programs.

A widely used program called *Javadoc* produces automated documentation from Java classes and from the comments in the code. Javadoc reads all comments that start with /** and formats them for documentation. So I entered this for my comment:

```
/**
* Temperature stores a temperature in Fahrenheit, Celsius,
* and Kelvin scales.
*
* @author Merrill Hall
* @version 1.0
*/
```

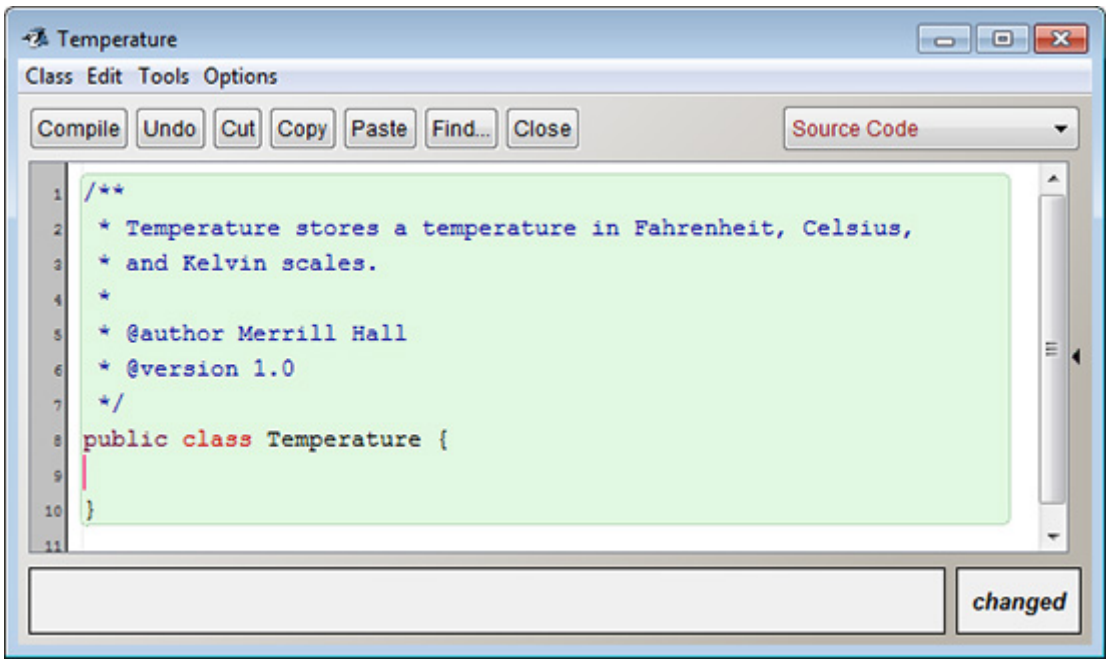
If you're curious about Javadoc, go up to the top-right corner of your editor window, and click **Documentation** in the drop-down list. BlueJ will run Javadoc on your class and produce the documentation for you to look at. It might look something like this:



Documentation view of the Temperature class

Note how Javadoc treats the keywords @author and @version. Javadoc looks for these keywords to identify specific items for the documentation.

Let's continue by deleting everything in the class except the first and last lines. We'll add back what we need as we go. Here's a picture of what that looks like just to make sure we're all on the same page.



Empty Temperature class

Selecting Your Data Requirements

The next question is, "What data elements will our class need?" BlueJ supplied an int variable x, but we just deleted that. The other questions we have to answer at this point are, "What variables do we need?" and "What type should they be?"

If we use integers, we're limited to whole numbers for temperatures, and that won't work. We know from the spec that absolute zero is –273.15° C and –459.67° F.

The two types that allow fractions are float and double. Java's default type for numbers with decimal points is double, so most programmers use that unless there's a reason to use float. Also, double values are more accurate. Java's doubles are accurate to about 15 digits, while floats are only accurate to about seven digits.

I have a choice to make. I can either store one temperature value in each of my objects and convert that value to the other two scales every time I need one of them, or I can do the conversion when I store the temperatures in the object and store all three values. Either way is valid; it just depends on the design of your project and its intended use.

If I think that users will convert each stored temperature only a few times, then it might make more sense to store one value and convert it as needed. If I think that users will frequently request each temperature in other scales, it would make more sense to store all three for easy retrieval.

I chose the second route, creating three instance variables (one for each scale). I named them degreesCelsius for Celsius, degreesFahrenheit for Fahrenheit, and degreesKelvin for Kelvin. It's possible to write a class that achieves the same results without storing all three values, but I've chosen to put them all into this class.

I now have this as my class:

```
/**
* Temperature stores a temperature in Fahrenheit, Celsius,
* and Kelvin scales.
*
* @author Merrill Hall
* @version 1.0
*/
public class Temperature {
    private double degreesFahrenheit; // Fahrenheit temperature
    private double degreesCelsius; // Celsius temperature
    private double degreesKelvin; // Kelvin temperature
}
```

I've made these private to prevent anyone from using them directly. For example, I wouldn't want anyone to set one of them to a value that wasn't calculated correctly from the others.

Now that we have our data variables, let's work on our methods.