

Chapter 4: The main() Method

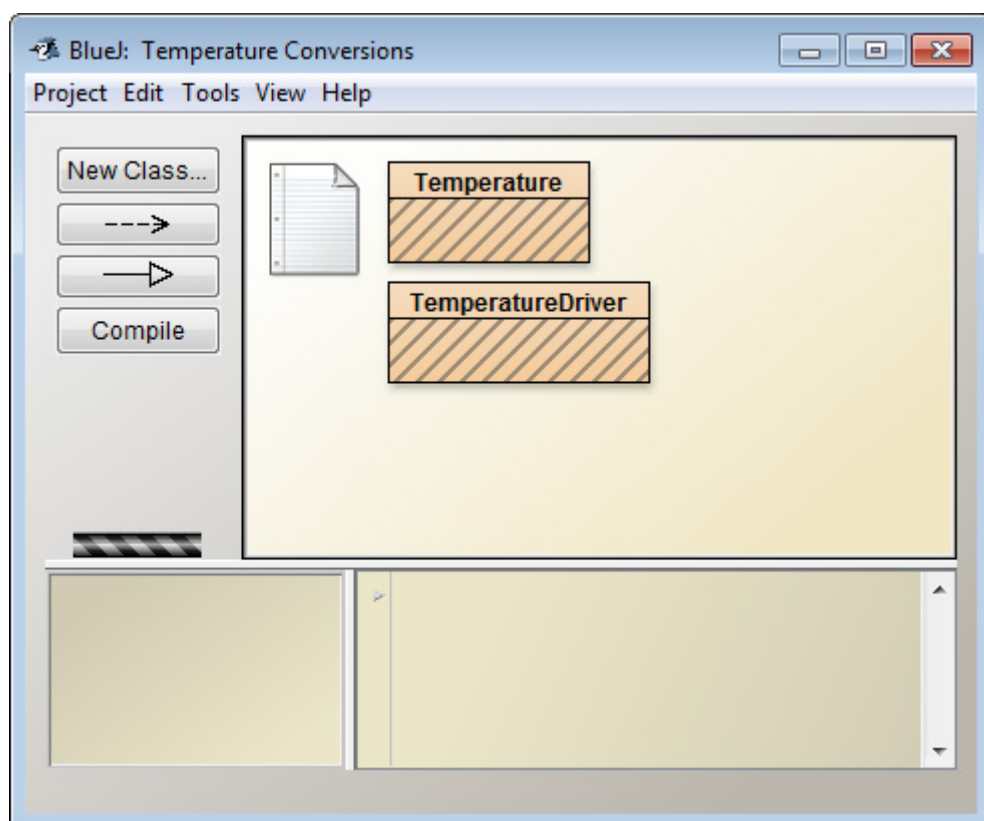
The main() Method

As with our first program, we need to have a main() method before we have a stand-alone program we can run. Our main() method will need to do several things:

- Ask its user to type in a Fahrenheit temperature.
- Read that number from the keyboard.
- Use an instance of our Temperature class to store and convert that temperature.
- Display the conversion results on-screen.

As we did with our TextWriterDriver, we're going to put our main() method in its own class.

Go back to BlueJ's main window, and create a new class there. I named mine TemperatureDriver to indicate that it has a main() method and that it uses the Temperature class. If you use the same names, your BlueJ window should look like this:



Updated temperature conversions project window

Once you've created the second class, open it for editing, and change it to look like my example below.

Here's my driver class, followed by an explanation of how it does what it does.

```

import java.util.Scanner;

/**
 * TemperatureDriver runs and tests the Temperature class.
 *
 * @author Merrill Hall
 * @version 1.0
 */
public class TemperatureDriver {
    /**
     * main() reads two Fahrenheit temperatures and
     * displays their Celsius and Kelvin equivalents.
     */
    public static void main(String[] args) {
        double inputTemperature = 0.0;
        Scanner keyboard = new Scanner(System.in);
        Temperature t1 = new Temperature();
        Temperature t2;
        System.out.print("Enter a Fahrenheit temperature: ");
        inputTemperature = keyboard.nextDouble();
        System.out.println("You entered " + inputTemperature +
            " degrees Fahrenheit");
        t1.setDegreesFahrenheit(inputTemperature);
        System.out.println("which is " + t1.getDegreesCelsius() +
            " degrees Celsius");
        System.out.println("and " + t1.getDegreesKelvin() +
            " degrees Kelvin.");
        System.out.print("Enter another Fahrenheit temperature: ");
        inputTemperature = keyboard.nextDouble();
        System.out.println("You entered " + inputTemperature +
            " degrees Fahrenheit");
        t2 = new Temperature(inputTemperature);
        System.out.println("which is " + t2.getDegreesCelsius() +
            " degrees Celsius");
        System.out.println("and " + t2.getDegreesKelvin() +
            " degrees Kelvin.");
    }
}

```

Ignore the first line (the one that starts with the keyword `import`) for now. The first lines of the class and the `main()` method are similar to our `TextWriterDriver` class. Our `main()` methods will always start this way, with the keywords `public static void` in front of the name and with `(String[] args)` as the parameter list.

In main() our variables are local variables, as they were in TextWriterDriver, and they don't have private or public designations. In this program I've created four local variables: inputTemperature, keyboard, t1, and t2.

The first declaration creates a double variable named inputTemperature and assigns it a value of zero. Whenever we read data, we need to hold it in memory so we can use it. I created this variable to give us a memory location to hold the input we get from the keyboard until we store it in a Temperature object.

```
public static void main(String[] args) {  
    double inputTemperature = 0.0;  
    Scanner keyboard = new Scanner(System.in);  
    Temperature t1 = new Temperature();  
    Temperature t2;
```

The second declaration creates a Scanner object, which we haven't seen before. I named it keyboard since its purpose is to get data from the keyboard. Scanner objects let us read text data from *input streams*. (Those are one of the ways Java gets data into programs. For now, let's just treat them as the mechanism that moves data from the keyboard to our program.)

```
public static void main(String[] args) {  
    double inputTemperature = 0.0;  
    Scanner keyboard = new Scanner(System.in);  
    Temperature t1 = new Temperature();  
    Temperature t2;
```

Java doesn't load the Scanner class automatically. We have to tell Java where to find it: in the util package of the java library.

To tell Java where the class is, we need a new statement at the top of our program:

```
import java.util.Scanner;
```

```
import java.util.Scanner;  
/**  
 * TemperatureDriver runs and tests the Temperature class.  
 *  
 * @author Merrill Hall
```

Adding that statement tells Java where to find the Scanner class. Once we tell Java that, we can use it anywhere in our program. We'll use similar import statements for any class we want to use that's not in the java.lang package unless we want to use its full name in the program. I could type java.util.Scanner every time I refer to the Scanner class, but it's easier to import it at the start so I can just refer to it as Scanner.

To finish the Scanner declaration, we need to create the Scanner object that we want to call keyboard. We do that with the code new Scanner(System.in). The argument System.in tells the constructor which input stream to read data from. For personal computers, the default input device is the keyboard, and Java uses System.in to connect to it. The object System.in represents the default input device, similar to the way System.out represents the default output device.

```
public static void main(String[] args) {  
    double inputTemperature = 0.0;  
    Scanner keyboard = new Scanner(System.in);  
    Temperature t1 = new Temperature();  
    Temperature t2;
```

Now we have our Scanner object set up so that we can use its methods to read from the keyboard.

The third declaration in main() creates a new Temperature variable named t1. It also creates a new Temperature object. Since we don't have a temperature to use yet, it uses the default constructor to create the object with a Fahrenheit value of zero.

The last declaration creates another Temperature variable named t2. I'm going to wait a bit before we create an object for this variable to refer to, so for now t2 is a *null reference*. That means it doesn't have an object to refer to. We'll change that in just a minute.

Text equivalent start.

```
Temperature t1 = new Temperature();  
Temperature t2;
```

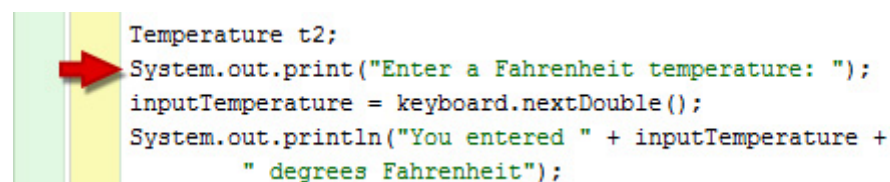
Text equivalent stop.

Now that we've set up our local variables, we're ready to start the action.

We need to get a Fahrenheit value from our user. It's always a good idea to let your users know what you expect from them, so they don't end up sitting there waiting for the program to do something. So we *prompt* (ask) the user for information.

We'll put our prompt on the screen with this statement:

```
System.out.print("Enter a Fahrenheit temperature: ");
```

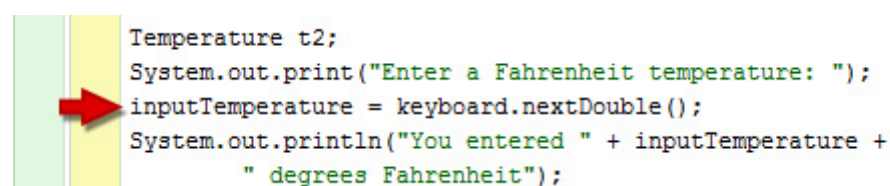


```
Temperature t2;  
System.out.print("Enter a Fahrenheit temperature: ");  
inputTemperature = keyboard.nextDouble();  
System.out.println("You entered " + inputTemperature +  
    " degrees Fahrenheit");
```

We're using the print() method instead of println() so the terminal window doesn't move to the next line. That means the user can enter the input number on the same line.

Then we follow that line with the statement

```
inputTemperature = keyboard.nextDouble();
```



```
Temperature t2;  
System.out.print("Enter a Fahrenheit temperature: ");  
inputTemperature = keyboard.nextDouble();  
System.out.println("You entered " + inputTemperature +  
    " degrees Fahrenheit");
```

This statement is an assignment that puts a value into our local variable inputTemperature. To do that, Java must first evaluate the right side of the assignment. The right side of the assignment refers to our Scanner object keyboard and calls its method nextDouble(). That method will look at the next group of characters that come from the keyboard, treat them as a double value, and return it to the caller, which is our program. Then that value goes into storage in our variable inputTemperature.

One more note about Scanner methods before we move on. Most Scanner input methods, including `nextDouble()`, will skip leading *white space* (blanks, tabs, returns, and so on) looking for input data. Once the method finds the start of the data, it reads until it finds more white space. The method considers whatever's between the white spaces (also called a *token*, which means a symbol or word) to be data, and Scanner will try to convert it to the type it's looking for.

If the method can't convert the data, it will *throw an exception*, which is Java's way of handling errors. Since we're not handling error conditions yet, that would cause our program to "blow up" or quit working. (Once we get our program going, try entering letters for a temperature to see what happens.)

Remember, our users will never make a mistake when they enter data. If we were writing this program for mere mortals, we'd make sure we found a valid number before we went any further. But we'll save that for another lesson.

Once we get a temperature, we'll display it to our user to validate what he or she did. That's what the next line of our program does. Here it is again:

```
System.out.println("You entered " + inputTemperature +  
    " degrees Fahrenheit");
```

Notice that in this output call there's a character string followed by a plus sign followed by a variable name, then another plus sign, and finally another string.

Whenever strings are involved, Java treats the plus operation as *concatenation*. That's a fancy word that means "take the second string and tack it on the end of the first string." Java doesn't let you make a string literal that's longer than one line of code, so you'll use concatenation a lot.

If one of the concatenated items isn't a string, Java will try to convert it to a string. Since we do that a lot with Java's primitive types, Java knows how to convert all of them to strings. So this line of code will give our user a message confirming what Fahrenheit temperature he or she entered.

Now we're ready to store the temperature in our Temperature object `t1`. Do you remember how to put a Fahrenheit value into it? With the `setDegreesFahrenheit()` method, of course. So the next line calls that method and gives it the user's value from `inputTemperature` as an argument.

The next two lines of `main()` display the converted Celsius and Kelvin values back to the user by getting them from the two get methods we created, `getDegreesCelsius()` and `getDegreesKelvin()`. We get the values by calling the methods, and since the methods return double values we can treat the calls just like double variables and concatenate their results into output strings. That's how those lines display their converted results.

The next five lines are almost identical to the previous five. But instead of using the `setDegreesFahrenheit()` method of an existing Temperature object, the third line of these five creates a new Temperature object using the second constructor, passing `inputTemperature` as an argument. Once that happens, the converted Kelvin and Celsius values can display just as before.

It's time to try compiling and running our program. We'll do that in the final chapter of this lesson.