Chapter 2: Class Methods

Class Methods

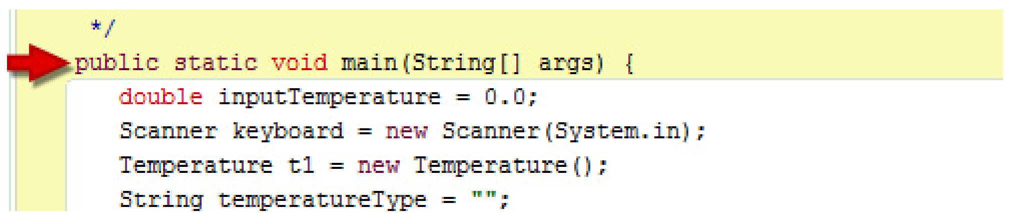
It would be helpful to know whether we have a valid temperature type and value before we create a Temperature object. It would be great to just call a method in the Temperature class that could tell us that. But all the methods we've used until now are part of objects, and it isn't possible to call them until you create an object.

Fortunately, Java gives us a way to call a method without creating an object first.

Java's static keyword, when placed at the beginning of a method declaration, tells the compiler that we want the method to belong to the class as a whole rather than being part of any particular object. Java can run a static method without creating any objects. This is called a *class method*, in contrast to the *instance methods* we've been using.

Programmers often use class methods when they need an action that isn't tied to any specific object. For example, the methods in the Math class that we use for many mathematical operations are all static class methods. If I want to find the square root of the value in a variable x, I can call Math.sqrt(x) to do that. I don't have to create a Math object to call the method. Note that in the call, the method name has a prefix that reveals the class name rather than an object name.

Java's standard main() method is static. Now you know why Java requires the static keyword for the main() method.



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

We're going to create a couple of static methods in the Temperature class. A user can call these to verify that the temperature type and temperature value are valid before the user creates a Temperature object.

These two methods will need the static keyword. The first method we'll add is one that will take a temperature type and tell the caller whether it's valid. So we need a parameter of type String to receive the type value, and we need to return true or false to tell the user whether the type is valid.

I've named my method isTypeValid() to indicate what it does. Java methods that have names starting with "is" typically return boolean values to answer the question their name asks.

Here's the method, which other than the static keyword should all be familiar to you. I won't spend a lot of time telling you more about it unless you have questions for me. Let me know in the Discussion Area if you do.

```
public static boolean isTypeValid(String temperatureType) {
    if (temperatureType.equalsIgnoreCase("C") ||
            temperatureType.equalsIgnoreCase("F") ||
            temperatureType.equalsIgnoreCase("K"))
        return true;
    else
        return false;
}
```

We can use this to simplify this part of our main() method:

```
if (temperatureType.equalsIgnoreCase("Q") ||
        temperatureType.equalsIgnoreCase("C") ||
        temperatureType.equalsIgnoreCase("F") ||
        temperatureType.equalsIgnoreCase("K")) {
    goodType = true;
}
```

by changing it to this:

```
if (temperatureType.equalsIgnoreCase("Q") ||
        Temperature.isTypeValid(temperatureType)) {
    goodType = true;
}
```

That makes things simpler, doesn't it? If we added a new temperature scale, we could just change the isTypeValid() method to check the new type and not change the driver program at all!

The second method is similar. I call it isTemperatureValid(), and here it is:

```
public static boolean isTemperatureValid(String temperatureType, double temperature)
{
    if ((temperatureType.equalsIgnoreCase("C") && temperature >= -273.15) ||
        (temperatureType.equalsIgnoreCase("F") && temperature >= -459.67) ||
        (temperatureType.equalsIgnoreCase("K") && temperature >= 0.0))
        return true;
    else
        return false;
}
```

Similarly, we can simplify this part of main():

```
if ((temperatureType.equalsIgnoreCase("C") && inputTemperature >= -273.15) ||
    (temperatureType.equalsIgnoreCase("F") && inputTemperature >= -459.67) ||
    (temperatureType.equalsIgnoreCase("K") && inputTemperature >= 0))
    goodTemperature = true;
```

like this:

```
if (Temperature.isTemperatureValid(temperatureType, inputTemperature))
    goodTemperature = true;
```

Doesn't that look better? The revised version didn't change the results, but it made the program more *encapsulated*, which is a fancy term for "self-contained." Now you or other programmers can add or remove temperature types with less work because the Temperature class handles more tasks.

Just to keep you in sync, here's a new copy of the complete classes:

TemperatureDriver Class

```java
import java.util.Scanner;
/**
 * TemperatureDriver runs and tests the Temperature class.
 *
 * @author Merrill Hall
 * @version 4.0
 */
public class TemperatureDriver {
    /**
     * main() reads a temperature type and value, then
     * converts it to the other two temperature scales.
     */
    public static void main(String[] args) {
        double inputTemperature = 0.0;
        Scanner keyInput = new Scanner(System.in);
        Temperature t1;
        String temperatureType = "";
        boolean moreTemperatures = true; // another temperature?
        boolean goodType = false; // good temperature type?
        boolean goodTemperature = false; // good temperature value?

        while (moreTemperatures) {
            System.out.print("Enter a temperature type (C=Celsius, " +
                "F=Fahrenheit, K=Kelvin, Q=Quit): ");
            temperatureType = keyInput.next();
            goodType = false;
            while ( ! goodType) {
                if (temperatureType.equalsIgnoreCase("Q") ||
                        Temperature.isTypeValid(temperatureType)) {
                    goodType = true;
                }
                else {
                    System.out.println("Invalid temperature type!");
                    System.out.println("The type must be C, F, K, or Q.");
                    System.out.print("Please enter the temperature type again: ");
                    temperatureType = keyInput.next();
                }
            }
            if (temperatureType.equalsIgnoreCase("Q")) { // quit
                moreTemperatures = false;
                System.out.println("\nProgram ended.");
            }
            else {
                goodTemperature = false;
                do {
                    System.out.print("Enter a temperature: ");
                    if (keyInput.hasNextDouble()) {
                        inputTemperature = keyInput.nextDouble();
                        if (Temperature.isTemperatureValid(temperatureType,
inputTemperature))
                            goodTemperature = true;
                        else {
                            System.out.println("You entered an invalid
temperature!");
                            System.out.println("It must be greater than absolute
zero.");
                            System.out.println("Try again.");
                        }
                    }
                    else {
                        System.out.println("You entered an invalid temperature!");
                        System.out.println("It must be a numeric value.");
                        System.out.println("Try again.");
                        keyInput.next();
                    }
                } while ( ! goodTemperature);

                t1 = new Temperature(temperatureType, inputTemperature);

                if (temperatureType.equalsIgnoreCase("F")) {
                    System.out.println("You entered " + inputTemperature +
                        " degrees Fahrenheit");
                    System.out.println("which is " + t1.getDegreesCelsius() +
                        " degrees Celsius");
                    System.out.println("and " + t1.getDegreesKelvin() +
                        " degrees Kelvin.");
                }
                else if (temperatureType.equalsIgnoreCase("C")) {
                    System.out.println("You entered " + inputTemperature +
                        " degrees Celsius");
                    System.out.println("which is " + t1.getDegreesFahrenheit() +
                        " degrees Fahrenheit");
                    System.out.println("and " + t1.getDegreesKelvin() +
                        " degrees Kelvin.");
                }
                else if (temperatureType.equalsIgnoreCase("K")) {
                    System.out.println("You entered " + inputTemperature +
                        " degrees Kelvin");
                    System.out.println("which is " + t1.getDegreesCelsius() +
                        " degrees Celsius");
                    System.out.println("and " + t1.getDegreesFahrenheit() +
                        " degrees Fahrenheit.");
                }
            }
        }
    }
}
```

Temperature Class

```
/**
 * Temperature converts temperature from Fahrenheit, Celsius,
 * or Kelvin scale to the other two scales.
 *
 * @author Merrill Hall
 * @version 4.0
 */
public class Temperature {
    private double degreesFahrenheit;  // Fahrenheit temperature
    private double degreesCelsius;  // Celsius temperature
    private double degreesKelvin;  // Kelvin temperature

    /**
     * This constructor for Temperature sets the temperature
     * values to the value from degrees, based on the type
     *
     * @param  type      temperature scale to use
     * @param  degrees   degrees Fahrenheit
     */
    public Temperature(String type, double degrees) {
        if (type.equalsIgnoreCase("C"))
            setDegreesCelsius(degrees);
        else if (type.equalsIgnoreCase("F"))
            setDegreesFahrenheit(degrees);
        else if (type.equalsIgnoreCase("K"))
            setDegreesKelvin(degrees);
    }

    public static boolean isTypeValid(String temperatureType) {
        if (temperatureType.equalsIgnoreCase("C") ||
                temperatureType.equalsIgnoreCase("F") ||
                temperatureType.equalsIgnoreCase("K"))
            return true;
        else
            return false;
    }



    public static boolean isTemperatureValid(String temperatureType, double
temperature) {
        if ((temperatureType.equalsIgnoreCase("C") && temperature >= -273.15) ||
                (temperatureType.equalsIgnoreCase("F") && temperature >= -459.67) ||
                (temperatureType.equalsIgnoreCase("K") && temperature >= 0.0))
            return true;
        else
            return false;
    }

    /**
     * The setDegreesFahrenheit method sets the Fahrenheit temperature
     *
     * @param  degrees The Fahrenheit value to store
     */
    public void setDegreesFahrenheit(double degrees) {
        degreesFahrenheit = degrees; // set Fahrenheit value
        degreesCelsius    = (degreesFahrenheit - 32.0) * 5.0 / 9.0; // set Celsius 
value
        degreesKelvin     = degreesCelsius + 273.15; // set Kelvin value

    }

    /**
     * The setDegreesCelsius method sets the Celsius temperature
     *
     * @param  degrees The Celsius value to store
     */
    public void setDegreesCelsius(double degrees) {
        degreesCelsius    = degrees;
        degreesFahrenheit = degreesCelsius * 9.0 / 5.0 + 32.0;
        degreesKelvin     = degreesCelsius + 273.15;
    }

    /**
     * The setDegreesKelvin method sets the Kelvin temperature
     *
     * @param  degrees The Kelvin value to store
     */
    public void setDegreesKelvin(double degrees) {
        degreesKelvin     = degrees;
        degreesCelsius    = degreesKelvin - 273.15;
        degreesFahrenheit = degreesCelsius * 9.0 / 5.0 + 32.0;
    }

    /**
```

```
     * getDegreesCelsius retrieves the Celsius temperature value
     *
     * @return a double value containing the Celsius temperature
     */
    public double getDegreesCelsius() {
        return degreesCelsius;
    }

    /**
     * getDegreesKelvin retrieves the Kelvin temperature value
     *
     * @return a double value containing the Kelvin temperature
     */
    public double getDegreesKelvin() {
        return degreesKelvin;
    }

    /**
     * getDegreesFahrenheit retrieves the Fahrenheit temperature value
     *
     * @return a double value containing the Fahrenheit temperature
     */
    public double getDegreesFahrenheit() {
        return degreesFahrenheit;
    }
}
```

It's unlikely that we'd want to add another temperature scale to our conversion project. But class methods and fields have many practical applications. For instance, System.in, System.out, and System.err are static class variables in the System class that provide us connections with default input and output devices. And Math.PI and Math.E are mathematical constants we can use without having to define them ourselves.

There are many examples of static class elements in Java's classes. Class elements make your code more encapsulated and less likely to throw an exception. Still, exceptions are part of every programmer's life, so let's talk about how to handle them. Please join me in Chapter 3.