Summary and Comments
Module 4 Discussion

The module 4 discussion had lots of good participation, and quite a few different ideas and solutions were posted, so I congratulate everyone on that. Here are my comments and advice.

Defining Conversion Methods

A good solution to the discussion problem re-design would be to define two static methods…one that converts Fahrenheit temperatures to Celsius temperatures, and one that converts Celsius temperatures to Fahrenheit temperatures. The code for these methods might look like this:

```
// Method to convert from Fahrenheit to Celsius
public static float fahrenheitToCelsius( float fahrTemp )
{
      return 5F/9F * ( fahrTemp – 32F );
}

// Method to convert from Celsius to Fahrenheit
public static float celsiusToFahrenheit( float celTemp )
{
      return 9F/5F * celTemp + 32F;
}
```

Notice that each method takes a single argument…the value to be converted…and simply returns the converted value based on a conversion formula. Since we are using float types for temperatures, the method arguments are of type float and the methods return a float type.

Also note the use of the fractions 5F/9F and 9F/5F. I used the 'F' suffix to explicitly specify that the numerators and denominators were float types. Similarly, I used '32F' to explicitly specify that this term in the conversion equations was a float type.

When students first tackle these methods, I see two very common mistakes being made. The first one is writing the conversion equations as follows:

```
5/9 * ( fahrTemp – 32 )

9/5 * celTemp + 32
```

Can you see the mistakes in the above conversion formulas? Take a minute to think it through and compare these formulas to the ones in the actual code. The problem is that 5/9 truncates to 0 and 9/5 truncates to 1, because the numbers are integers. The second mistake is writing the conversion equations as follows:

```
5./9. * ( fahrTemp – 32 )

9./5. * celTemp + 32
```

Summary and Comments
Module 4 Discussion

Can you see the problem here? At first glance, there doesn't seem to be one, and the truncation problem is solved…but…5./9. And 9./5. will be treated as doubles by Java, and the methods return floats. If you're not sure about either of the above, please revisit the lecture on Java type conversion.

Calling the Conversion Methods

Given the two conversion methods that were written above, the main method can be re-designed as follows:

```
import java.util.Scanner
public class TemperatureConversion
{
       public static void main( String [] args )
       {
          int userChoice = 0;                  // user selection: 1, 2, 3
          float fahrenheitTemp = 0;            // Fahrenheit temperature
          float celsiusTemp = 0;               // Celsius temperature

          Scanner input = new Scanner( System.in );

          while( userChoice != 3 )
          {
              System.out.print( "Enter 1 to convert F->C, … );
              UserChoice = input.nextInt();

              switch( userChoice )
              {
                 case 1: // convert Fahrenheit to Celsius
                         System.out.print( "Enter a Fahrenheit temperature: " );
                         fahrenheitTemp = input.nextFloat();
                         celsiusTemp = fahrenheitToCelsius( fahrenheitTemp );
                         System.out.println( fahrenheitTemp + " degrees Fahrenheit is
                                             celsiusTemp + " degrees Celsius" );
                         break;

                 case 2: // convert Celsius to Fahrenheit
                         System.out.print( "Enter a Celsius temperature: " );
                         celsiusTemp = input.nextFloat();
                         fahrenheitTemp = celsiusToFahrenheit( celsiusTemp );
                         System.out.println( celsiusTemp + " degrees celsius is
                                             fahrenheitTemp + " degrees Fahrenheit" );

                         break;

                 case 3: // end program
                         System.out.println( "Bye Bye" );
                         break;

                 default:// invalid data entered
                         System.out.println("Invalid data. You must enter 1, 2, or 3");

              } // end switch

          } // end while

       } //end main

       // … include code for conversion methods here …
}
```

Summary and Comments
Module 4 Discussion

The calls to the conversion methods are shown above in blue. I could have put the method calls right inside the print statements if I wanted to, but I didn't for purposes of clarity.

Advantages of Using Methods

Now, why is using methods to perform the temperature conversions advantageous? In this example, there are two primary advantages of using methods. First, using methods allows us to reuse the code for the conversion methods in other programs without having to re-write the conversion code. Granted, the conversion code was simple…but…suppose it wasn't. The second advantage is that using methods nicely delegates the responsibilities between the main method and the two conversion methods. The main method is responsible for doing user input and output, and the conversion methods are responsible for converting.

There can be additional advantages to using methods in your code. In situations where the logic for tasks is complicated, putting that logic inside methods can also simplify the code, making it easier to read and understand. It can also make it easier to test for correctness and pinpoint any errors.

Other Solutions

Some students suggested putting the user input and output functionality in the conversion methods themselves, like this:

```
public static void fahrenheitToCelsius()
{
      Scanner input = new Scanner( System.in );
      System.out.print( "Enter a Fahrenheit temperature: " );
      float fahrenheitTemp = input.nextFloat();
      float celsiusTemp = 5F/9F * ( fahrenheitTemp – 32F );
      System.out.println( … );  // print results
}

public static void celsiusToFahrenheit()
{
      Scanner input = new Scanner( System.in );
      System.out.print( "Enter a Celsius temperature: " );
      float celsiusTemp = input.nextFloat();
      float fahrenheitTemp = 9F/5F * celsiusTemp + 32F;
      System.out.println( … );  // print results
}
```

Summary and Comments
Module 4 Discussion

Using the above code, the main method could be re-designed to look like the following:

```
import java.util.Scanner

public class TemperatureConversion
{

       public static void main( String [] args )
       {
          Scanner input = new Scanner( System.in );

          while( userChoice != 3 )
          {
              System.out.print( "Enter 1 to convert F->C, … );
              UserChoice = input.nextInt();

              switch( userChoice )
              {
                 case 1: // convert Fahrenheit to Celsius
                         fahrenheitToCelsius();
                         break;

                 case 2: // convert Celsius to Fahrenheit
                         celsiusToFahrenheit();
                         break;

                 case 3: // end program
                         System.out.println( "Bye Bye" );
                         break;

                 default:// invalid data entered
                         System.out.println("Invalid data. You must enter 1, 2, or 3");

              } // end switch

          } // end while

       } //end main

       // … include code for conversion methods here …

}
```

So…what do you think about this solution? It certainly works, and it actually simplifies the main method code a bit. However, it is not nearly as reusable as the first solution because the conversion methods are now undertaking the responsibility for both user input/output as well as the conversions.

To see how this impacts reusability, suppose we had need for an application that read a set of, say, Fahrenheit temperatures from a file stored on a computer, and then created a file containing the equivalent Celsius temperatures…or vice versa. We wouldn't be able to reuse the second set of

conversion methods in such an application, because these methods are designed to prompt the user for keyboard input and are designed to display the converted values on the console. The first set of conversion methods could easily be used in the file input/output application, however.

Design for High Cohesion and Low Coupling

In the software engineering world[1], we have a term called **cohesion** that is used to describe how single-purposed a method is. Methods that perform very specific tasks, like the first set of methods above, are said to have *high cohesion*…which is a desirable design characteristic. Methods that do multiple tasks, and that take on more responsibilities than necessary, like in the second set of methods above, are said to have *low cohesion*. High cohesion is generally preferable to low cohesion because it helps to make methods more reusable, easier to test, and easier to maintain.

Software engineers also use the term **coupling** to describe how independent a method is with respect to other methods or program components. *Loose coupling* is good, and *tight coupling* is not so good. In our examples, the second set of conversion methods were dependent on, or coupled to, the user interface functionality, making them more tightly coupled than the first set of conversion methods. Loose coupling also helps to make methods more reusable, testable, and maintainable.

That's all for this week. I hope you found the information useful.

[1]Braude, E.J., and Bernstein, M.E., *Software Engineering: Modern Approaches*, 2nd Ed, Waveland Press, 2016.