Chapter 2: BlueJ's Debugger

BlueJ's Debugger

I'm sure you've wanted to look "under the covers" and see what's going on while a program is running. Debuggers let you do just that. Most IDEs include them, and BlueJ is no exception. Let me show you a short program, and then we'll look at the debugger window showing what's going on while it runs.

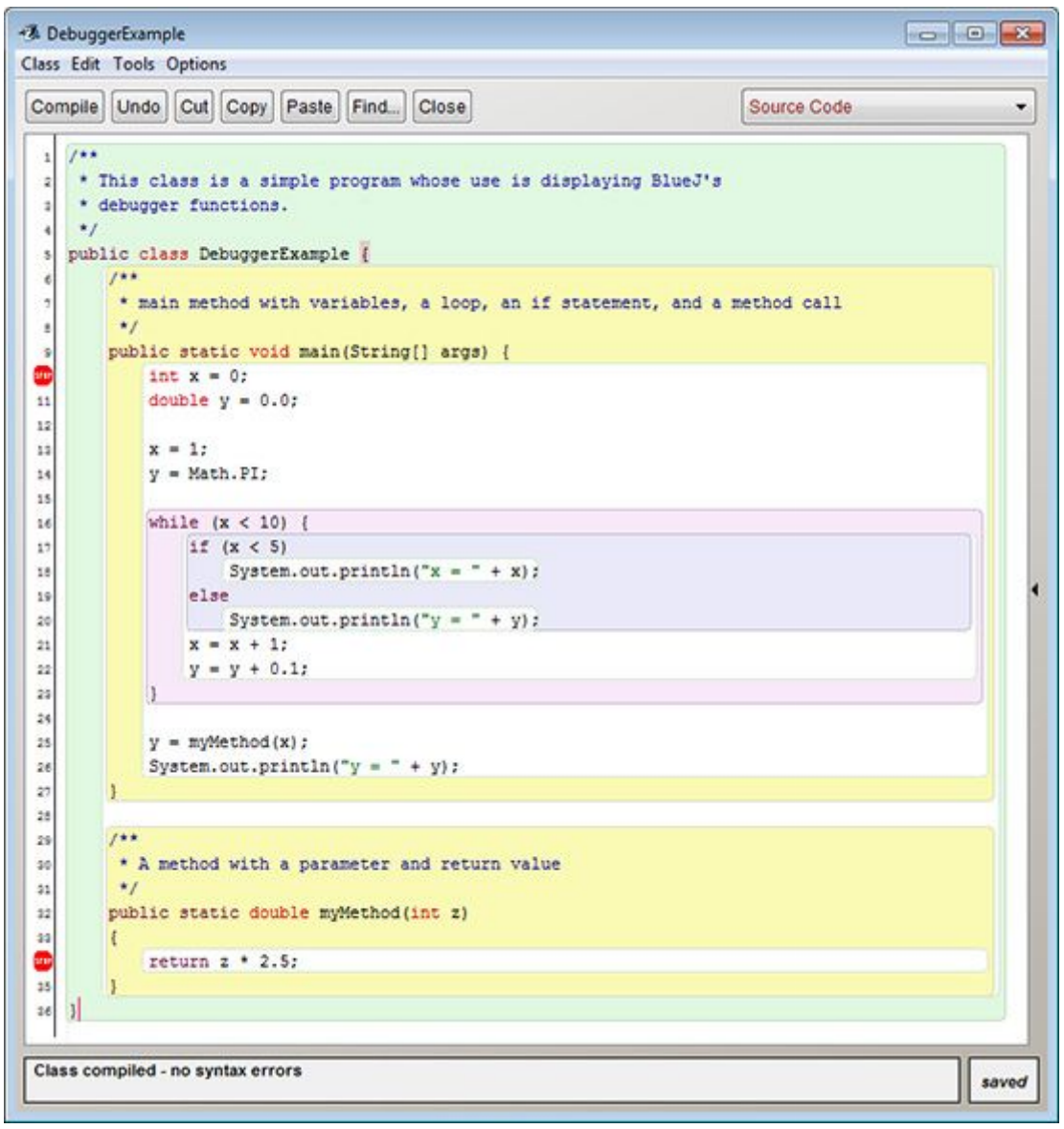
First, here's my code in a format you can copy and paste if you want to.

Hide answer

```
/**
 * This class is a simple program whose use is displaying BlueJ's
 * debugger functions.
 */
public class DebuggerExample {
    /**
     * main method with variables, a loop, an if statement, and a method call
     */
    public static void main(String[] args) {
        int x = 0;
        double y = 0.0;

        x = 1;
        y = Math.PI;

    while (x < 10) {
        if (x < 5)
            System.out.println("x = " + x);
        else
            System.out.println("y = " + y);
        x = x + 1;
        y = y + 0.1;
    }

    y = myMethod(x);
    System.out.println("y = " + y);
}

    /**
     * A method with a parameter and return value
     */
    public static double myMethod(int z)
    {
        return z * 2.5;
    }
}
```

This program is simple, but it has items we can trace in the debugger. We can watch the loop iterate, the variable values change, and the if statement make a choice. We can also observe a call to and return from a method.

Here's what my program looks like in BlueJ's editor window:

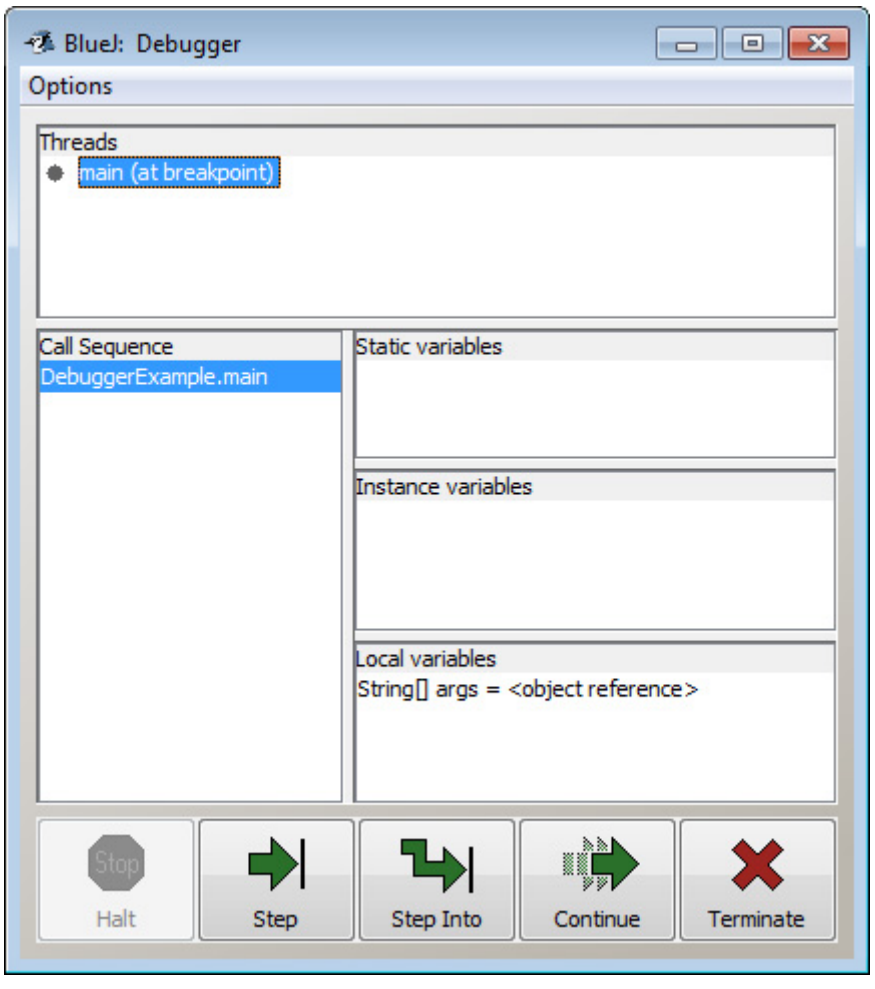


Program code with breakpoints

Notice the two miniature stop signs in the left margin where the line numbers appear. These are *breakpoints*. When the program gets to a breakpoint, execution stops, and the debugger displays the state of the program in the *debugger window*. That window is where we can watch what's going on in the program.

You can set and clear breakpoints by clicking in the left margin of an executable statement in the editor window. If you have BlueJ set to show line numbers, just click on the line number. Otherwise click where the line number would be. You will need to compile your class before you can add breakpoints.

When I ran the program above, and execution got to the first breakpoint, here's the window that opened up:



BlueJ's debugger window

We'll ignore the Threads pane of the window since we haven't discussed threads in this course. A *thread* in Java is some code that can run independently of other code. That means it can run in parallel and let the computer do more than one thing at a time. The Threads pane shows all active application threads. In this case the only thread running is our main() method.

The next pane down on the left shows the list of method calls in reverse order. The top line is the most recent call. This is the call stack we talked about in Lesson 8.

On the right side are three panes for variables. The top one is for *static variables* (if there are any) in the class containing the breakpoint. We discussed the static keyword in Lesson 8. A static variable is one that exists in the class before we create any objects.

The middle pane shows the *instance variables* for the class—the variables that are part of each object. The bottom pane lists the *local variables*—the ones defined within the currently running method. In this case the method is main, and the only local variable is the parameter named args. Args is short for "arguments" and is there to receive any arguments passed by the operating system when the program starts. Since we haven't used that parameter, it's a null reference.

Across the bottom are five buttons.

- The Halt button usually isn't available since the program is usually already stopped when this window is active.
- The Step button tells BlueJ to execute one statement and then stop again to show its results.
- The Step Into button is similar, but it does one thing differently when the statement being executed is a method call. For method calls, the Step button calls the method, runs it, and stops at the next

statement after the method call. The Step Into button calls the method and stops at the method's first line.

- The Continue button tells BlueJ to continue running the program until the next breakpoint or until the program ends.

- The last button, Terminate, tells BlueJ to stop running the program and to end the debugger.

Let me show you an example of running the debugger.

Those are the basics of how to use BlueJ's debugger. I suggest playing around with it on your own by setting breakpoints in some of your programs and then running them, and by experimenting with the debugger buttons. A debugger can be a big help when you can't find bugs by just observing a program's output.

Let's switch to a different topic in Chapter 3.