9.7 Commenting and style

Open the *calculator-engine* project to view the classes. This is Hacker's own version of the project, containing only the calculator engine and a test class, but not the user interface class. The **`CalcEngineTester`** class takes the place of the user interface at this stage of development. This illustrates another positive feature of defining interfaces between classes: it becomes easier to develop mock-ups of the other classes for the purpose of testing.

If you take a look at the **`CalcEngine`** class, you will find that its author has paid attention to some important areas of good style:

- The class has been given a multiline comment at the top, indicating the purpose of the class. Also included are annotations indicating author and version number.
- Each method of the interface has a comment indicating its purpose, parameters, and return type. This will certainly make it easier to generate project documentation for the interface, as discussed in Chapter 6.
- The layout of the class is consistent, with appropriate amounts of white-space indentation used to indicate the distinct levels of nested blocks and control structures.
- Expressive variable names and method names have been chosen.

Although these conventions may seem time-consuming during implementation, they can be of enormous benefit in helping someone else to understand your code (as we have to in this scenario), or in helping you to remember what a class does if you have taken a break from working on it.

We also note another detail that looks less promising: Hacker has not used a specialized unit test class to capture his tests, but has written his own test class. As we know about unit test support in BlueJ, we wonder why.

This does not necessarily have to be bad. Handwritten test classes may be just as good, but it makes us a little suspicious. Did Hacker really know what he was doing? We shall come back to this point a bit later.

So, maybe Hacker's abilities are as great as he thinks they are, and in that case you will not have much to do to make the class ready for integration with the others! Try the following exercises to see if this is the case.

Exercise 9.22 Make sure the classes in the project are compiled, and then create a **`CalcEngineTester`** object within BlueJ. Call the **`testAll`** method. What is printed in the terminal window? Do you believe the final line of what it says?

Exercise 9.23 Using the object you created in the previous exercise, call the **`testPlus`** method. What result does it give? Is that the same result as was printed by the call to **`testAll`**? Call **`testPlus`** one more time. What result does it give now? Should it always give the same answer? If so, what should that answer be? Take a look at the source of the **`testPlus`** method to check.