Expressions, Statements, Blocks, and Logical Branching

This exercise is made up of thirty-five small coding problems. Each problem is independent of the others, and they can be solved in any order. The exercise is intended to reinforce your understanding of expressions, statements, blocks, and logical branching.

Learning Objectives

After completing this exercise, students will understand:

- How to use expressions and statements to solve complex problems.
- How to apply comparison and logical operators to solve complex problems.
- How to organize and group statements within blocks.
- How to choose different paths within code using if/else blocks.

Evaluation Criteria & Functional Requirements

- The project must not have any build errors.
- Unit tests pass as expected.
- Appropriate variable names and data types are being used.

Getting Started

- Import the expressions-and-control-flow-exercises project into Eclipse.
- Right-click on the project, and select the **Run As -> JUnit Test** menu option.
- Click on the **JUnit** tab to see the results of your tests and which passed / failed.
- Provide enough code to get a test passing.
- Repeat until all tests are passing.

Tips and Tricks

- **Note, If you find yourself stuck on a problem for longer than fifteen minutes, move onto the next, and try again later.**
- Before each method, there is a description of the problem that needs to be solved, as well as examples with expected output. Use these examples to get an idea of the values you need to write your code around. For example, in the comments above the `more20` method, there is a section that includes the method name, as well as the expected value that will be returned for each method call. The following example signifies that when the method is called with 20, it will return false, when it is called with 21, it will return true, and when it is called with 22, it will return true:

```
more20(20) → false
more20(21) → true
more20(22) → true
```

- When you are trying to solve these sorts of problems, it is often helpful to keep track of the state of variables on a piece of paper as you are working through your code.
- The output of the test run can provide helpful clues as to why the tests are failing. Try reading the output of a failing test for more information that could be valuable when troubleshooting.
- You can also run the tests in debug mode when executing the tests. This will allow you to set a "breakpoint", which will then halt the code at certain points in the editor. You can then look at the values of variables while the test is executing, and can also see what code is currently being executed. Don't hesitate to use the debugging capabilities in Visual Studio to help resolve issues.