Using Inheritance: The Clock Example

This assignment involves the use of inheritance. It's fairly straightforward. There are three main parts, but each is an extension of the previous one, so there is only one program involved. You should also write test cases for your clock classes.

As usual it's essential that your submission works. It's also very important here that your **inheritance hierarchy is designed and implemented as cleanly as possible**.

Part 0: Creating Unit Tests

Using the test::unit framework, write unit tests for each of your Clock classes below. Testing for every possible input is impossible of course, but at least make sure that every piece of coded is executed by your tests. In the case of the AlarmClock class, it's tricky to test for the alarm going off as this uses console output. A simple way to handle this is to update the tick operation so it also returns :ALARM if the alarm goes off, and :NO_ALARM if not.

You should have a file of unit tests for each one of the Clock classes, as well as a test suite that runs all the unit tests.

Write the test cases as you go; don't leave this to the end. The best time to write your test cases is BEFORE your write the code that is to be tested. This style is called **test-driven development**, or TDD for short.

Part I: A simple Clock class

Write a class called Clock that represents the time of day using the 24-hour clock. As well as a suitable initializer method, the class Clock should contain methods:
1. `to_s`: Return a string that represents the time, e.g., 5 seconds after midnight would be: 0:0:5
2. `tick`: Increment the time by one second

Care must be taken to deal with minute, hour and 24-hour boundaries. Write a script that creates a Clock object using data from the "clock.dat" file. Then print the Clock object, invoke tick 200,000 times on the Clock object, and print the Clock object again. The "clock.dat" file contains 3 integers that represent hours, minutes and seconds. Here's a sample:
0
0
0

Part II: Creating a new AlarmClock class

Create a subclass of Clock called AlarmClock. This extends the Clock class by providing alarm clock behaviour. Whenever an AlarmClock object reaches its alarmtime, an "Alarm!" message is printed along with the current time.

Your script should now create instances of both Clock and AlarmClock. Both clocks use the same starting time, and the alarm time for the AlarmClock object is read from the "clock.dat" file as well. Again, print both the Clock objects and invoke tick 200,000 times on both the Clock objects. Finally, print both Clock objects.

A sample "clock.dat" input file could be:
0
0
0
2
31
27

Part III: Creating a new ReverseClock class

Create a subclass of Clock called ReverseClock. This is just like a Clock but whenever tick is invoked it goes back one second. So if the initial time is 0:0:0 and tick is invoked, the new time is 23:59:59.

Your main program should create an instance of each of the three Clock classes. All clocks use the same starting time, and the alarm time for the AlarmClock object is read from the "clock.dat" file as well. Again, print all the Clock objects and then invoke tick 200,000 times on all Clock objects. Finally, print all the Clock objects again.

What to Submit

Please read this carefully. Create a **zip file** of all your Ruby files. This includes the clock classes, the testcases and the testsuite that runs all the testcases. Submit this file through the submission web page.