A Time Class

Purpose:

To use basic classes and objects to explore concepts such as packaging, methods inherited from `Object`, and the difference between attributes and representation.

Due Date:

The completed lab assignment is due Wednesday 2011-01-26 *at the beginning of lecture*.

Time Class — version 1

Write a simple `Time` class with whose state is consists of three private member variables representing the hours, the minutes, and the seconds.

It should have the methods specified in Figure 1 on the following page.

⇒ Write a test class that uses the various methods of the `Time` class to show that they work. Be sure to test setting hours, minutes, or seconds outside of the usual range to see what happens.

Show that you can convert a `Time` to a `String` *without* writing additional code: for instance "`System.out.println("The time is"+t)`" should work for a `Time t`. (You get this behaviour because of the `toString()` method.)

Time Class — version 1a

Redo the previous part, but in this version put your `Time` class in a package called `version1a`. When testing this version, the test code and the time class code should be in different directories, and the test code should contain an "`import version1.Time;`" statement.

Time Class — version 2

Package this version in a package called "`version2`".

This version should have identical public method signatures and testing, but each `Time` object should have a single member variable that represents the number of seconds since midnight.

In the code comment before this `Time` class, comment on which methods are easier, and which methods are more difficult for this version.

The various classes should all have the following public methods unless otherwise specified.

- Constructors
  - `Time()` (creates midnight),
  - `Time(h,s,m)`, and
  - `Time(Time t)` (initialize from another `Time` object).
- Accessor methods
  - `getHour`,
  - `getMinute`, and
  - `getSecond`

  that return the corresponding value from the object. The hours should be between 0 and 23, and the minutes and seconds should be between 0 and 59.
- Mutator methods
  - `setHour`,
  - `setMinute`, and
  - `setSecond`

  to set the corresponding attributes of a `Time` object. These should ensure that the resulting time is legitimate. Decide and document what happens when you, say set the number of seconds to 75.
- A mutator method
  - `public void advanceBy(int seconds) { ... }`

  that changes the time by a given number of seconds.
- A method
  - `public String toString() { ... }`

  that produces a string like "`22:03:12`". The hours should be between 0 and 23, and the minutes and seconds should be between 0 and 59.
- A method
  - `public int compareTo(Time t) { ... }`

  that produces the number of seconds from `t` to `this`. That is, `t.advanceBy(this.compareTo(t))` should set `t` to the same time as `this`.
- A method
  - `public boolean equals(Time another) { ... }`

  that returns `true` if and only if the times have the same value.

Figure 1: Time class features

```
public interface TimeInterface
{
  int getHour() ;
  int getMinute() ;
  int getSecond() ;
}
```

Figure 2: Time interface specification

Sorting Experiments (`version1b` and `verion2b`)

Using your version 1a and version 2 `Time` classes, attempt to sort a `Time []` array using `java.util.Arrays.sort`. This will produce an error.

Now create new packages `version1b` and `version2b` that are the same as `version1a` and `version2` except that the class starts with

```
public class Time implements Comparable<Time> { ...
```

Repeat the sorting experiment. (It should now work).

Implementing your own interface (`version1c` and `verion2c`)

Create an interface `TimeInterface` that looks like Figure 2 in a separate `.java` file.

⇒ Write test code to determine something like

```
TimeInterface ti = new version2.Time(12,30,0) ;
```

works "out of the box". (It shouldn't.)

⇒ Create new packages `version1c` and `version2c` with `Time` classes that explicitly "`implements TimeInterface`". Now test code like

```
TimeInterface ti = new version2c.Time(12,30,0) ;
```

works. (It should.)

Can you create a `TimeInterface` array that contains a mixture of `version1c.Time` and `version2c.Time` objects?

⇒ What happens with code like

```
TimeInterface ti = new version2c.Time(12,30,0) ;
System.out.println(ti.getSecond()) ;
```

What happens with code like

```
TimeInterface ti = new version2c.Time(12,30,0) ;
ti.setSecond(12) ;
System.out.println(ti.getSecond()) ;
```

What happens with code like

```
TimeInterface ti = new version2c.Time(12,30,0) ;
System.out.println(ti) ; // Do you expect a hex address?? Why?
```

Explain your results.