UNBC Laboratory Assignment
Computer Science 101—Winter 2019

GradeBook and Time Classes

Purpose

To consolidate working with objects; in particular to understand the difference between attributes and their internal representation.

Due Date

The completed lab assignment is due Monday, 2019-02-18 *at the beginning of lecture*.

Hand-In format

In this laboratory assignment, there are multiple places where correctly completing the lab means creating code that does *not* compile. Please create and clean (`scriptfix`) a script file that shows the various compilation attempts and runs.

Put the script file in the top-level of the `.jar`-file that you submit. You may also create an `answers.txt` file if that helps communicate what you are doing.

Gradebook Class — version 3

This question relies on material in the text book. Solve Question E8.9 on *p.* 413.

Question E8.9 says

> Repeat Exercise 8.8 using multiple classes. Create a `Gradebook` class that collects objects of type `Student`.

Question E8.8 says.

> Modify the application of How-To 7.1 so that it can deal with multiple students. First, ask the user for all of the student names. Then read in scores for all of the quizzes, prompting for the scores for each student. Finally, print the names of all the students and their final scores. **Use a single class and only static methods**.

There is no need to do the part in red. Just provide the E8.9 version.

See *pp.* 330–331 for the How-To.

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. Make each of these variables bytes. You may need to review how casting works. Put your `Time` class in a package called `version1`.

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. 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;`" 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`.

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 `int` 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.

⇒ Again, write a test class that uses the various methods of the `Time` class to show that they work.

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

- Constructors
  - `Time()` (creates midnight),
  - `Time(int h, int m, int s)`, and
  - `Time(Time t)` (initialize from another `Time` object).
- `public` accessor methods
  - `int getHour   ()`,
  - `int getMinute ()`, and
  - `int 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.
- `public` mutator methods
  - `void setHour   (int h)`,
  - `void setMinute (int m)`, and
  - `void setSecond (int s)`

  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