Com S 227
Spring 2015
Assignment 1
200 points

Due Date: Thursday, February 19, 11:59 pm (midnight)
"Late" deadline (25% penalty): Friday, February 20, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Blackboard. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Note: Our first exam is Monday, February 23, which is just four days after the due date for this assignment. ***It will not be possible to have the assignments graded and returned to you prior to the exam****. The exam dates are arranged by the registrar, not the instructors. We can post a sample solution on February 21, but we strongly encourage you to test your code carefully as described in the section below, "Testing and the SpecCheckers".*

Please start the assignment as soon as possible and get your questions answered right away!

Overview

The purpose of this assignment is to give you some practice working with conditional statements before the first exam. For this assignment you'll create a class called `AlarmClock` that models an alarm clock.

Think about the way a typical alarm clock works. At any given moment, there is a *clock time* (the time of day as normally displayed), an *alarm time*, and also an *effective alarm time* that is

related to the *snooze* feature, which is described in more detail below. The clock has buttons that lets you set the clock time and alarm time. It also has a switch with which you can turn the alarm on and turn the alarm off, and at any given moment it may or may not be ringing.

Note that an AlarmClock object is just a model of some aspects of an alarm clock's behavior; it does not actually tell time or ring out loud! To simulate the passage of time, you'll create a method

```
void advanceTime(int minutes)
```

that moves the current time the given number of minutes forward. To detect whether the alarm is ringing, you'll provide a method

```
boolean isRinging()
```

that returns true if the clock is currently supposed to be ringing.

If the alarm is turned on, then when the clock time reaches or passes the effective alarm time, the clock starts "ringing". It remains in the ringing state until the alarm is turned off or the snooze button is pressed.

The snooze feature works as follows: If the snooze button is pressed while the alarm is ringing, it stops ringing and sets the *effective alarm time* to be 9 minutes after the current clock time. In this case we say that "snooze is in effect". If the alarm is not ringing, the snooze button does nothing. Snooze is canceled by either turning off the alarm, setting the clock time, or setting the alarm time. The effective alarm time is always equal to the alarm time when snooze is not in effect. Snooze being "canceled" just means that the effective alarm time is set to be equal to the alarm time again. Pressing the snooze button does not change the alarm time and does not turn off the alarm.

Example

Here is a more detailed example. Suppose the current clock time is 5:00 (that is, 5 o'clock in the morning) and the alarm time is 5:15, and now we turn on the alarm. Then suppose we advance the time by 30 minutes. The clock time is now 5:30, and the clock is ringing (in real life, it would have been ringing continuously for 15 minutes). Then we advance the time by 5 minutes

more (let the alarm ring for five more minutes), and then press snooze at 5:35. The alarm stops ringing and the effective alarm time is now 5:44 (9 minutes ahead). If we advance the time by 5 minutes, the current time is 5:40 and the alarm is not ringing. If we then advance the time by another 4 minutes, to 5:44, the alarm is ringing again. Then we advance the time by 10 minutes (let it ring for 10 minutes) and press snooze again at 5:54, so now the effective alarm time is 6:03. So if we advance the time by 30 minutes, to 6:24, the alarm will be ringing. (Most likely, we were in the shower.) If we turn off the alarm, it stops ringing. This cancels snooze and sets the effective alarm time back to the alarm time of 5:15. (That is, pressing snooze doesn't change the alarm time.) If we turn the alarm on again now, it will ring again the next morning at 5:15when the time advances another 22 hours and 51 minutes.

Also note: If the alarm is on when you set the clock time or set the alarm time, the clock doesn't start ringing. For example, if the current time is 3:00 and you set the alarm time for 3:00, it will ring 24 hours later. (Not all real alarm clocks work this way, but this assumption simplifies the implementation.)

Specification of public `AlarmClock` methods

For details on all public methods, see the javadoc online.

In addition to the public methods and two constructors, your class must define the two constants:

```
public static final int SNOOZE_MINUTES = 9;
public static final int MINUTES_PER_DAY = 1440;
```

It's about time...

Before you go any further, make sure you understand how the mod operator ("%") works, since time always "wraps around" when you start the next hour or day. For example, suppose the time is midnight, denoted 00:00 in hours:minutes form, and then 4000 minutes go by. What time is it now? Each 1440 minutes is a full 24 hours, so 4000 % 1440 is 1120, the remaining minutes. This is the number of minutes past midnight. To convert to hours:minutes form, 1120 / 60 is 18, and 1120 % 60 is 40, so the time is now 18:40 (aka 6:40 pm). In your implementation, you will want to represent time as the *number of minutes past midnight*, and convert to hours:minutes format only when necessary.

Testing and the SpecCheckers

As always, you should try to work incrementally and write tests for your code as you develop it.

Since the test code is not a required part of this assignment and does not need to be turned in, **you are welcome to post your test code on Piazza for others to check, use and discuss.**

SpecChecker 1

Your class must conform precisely to this specification. The most basic part of the specification includes the class name and package, the required constants, the public method names and return types, and the types of the parameters. We will provide you with a specchecker to verify that your class satisfies all these aspects of the specification and does not attempt to add any public attributes or methods to those specified. If your class structure conforms to the spec, you should see the following message in the console output:

```
2 out of 2 tests pass.
```

(This SpecChecker will not offer to create a zip file for you). Remember that your instance variables should always be declared **`private`**, and if you want to add any additional "helper" methods that are not specified, they must be declared **`private`** as well.

SpecChecker 2

In addition, since this is the first assignment and we have not had a chance to discuss unit testing very much, we will also provide a specchecker that will run some simple functional tests for you. This is similar to the specchecker you used in Labs 1 and 2. It will also offer to create a zip file for you to submit. *Specchecker 2 should be available around the 13th of February. Please do not wait until that time to start testing!*

Please note that we are also going to read over your code and make sure that things are done sensibly. The automated tests will count for 20% to 30% of the total points. *You may lose points even if your code passes all the specchecker's functional tests*. Hopefully you will find the feedback useful in the long run, and we will attempt to assign some partial credit even if there are mistakes.

> See the document "SpecChecker HOWTO", which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links" if don't remember how to import and run a SpecChecker.

Getting started

*Smart developers don't try to write all the code and then try to find dozens of errors all at once; they work **incrementally** and test every new feature as it's written. Here is a rough guide for some incremental steps you could take in writing this class.*

1. Review the section "How To 3.1" at the end of section 3.3 of the text, which summarizes the steps involved in implementing a class. (Note that this document plus the associated Javadoc essentially provide Steps 1 and 2 for you.)

2. Create a new Eclipse project and within it create a package **`hw1`**.

3. Go ahead and create the **`AlarmClock`** class in the **`hw1`** package. Add the two constant declarations and put in stubs for all the methods and constructors described in the Javadoc. For methods that need to return a value, just return a "dummy" value as a placeholder. At this point there should be no compile errors in the project.

4. Import and run the first specchecker. Your code should successfully pass the two tests.

5. Javadoc the classes and methods. This is a required part of the assignment, and doing it now will help clarify for you what each method is supposed to do before you begin the actual implementation.

    - Copying method descriptions from the online Javadoc is perfectly acceptable, though you are not required to be as thorough and detailed as the online documentation. A one-sentence description is sufficient. You'll need to fill in the @param and @return tags for completeness.
    - Don't forget that the class itself needs a javadoc comment. Your description of the class can be very brief, **but you *must* include an @author tag with your name.**

6. A good thing to start with might be just keeping track of time. That is, think about the methods **`advanceTime()`** (the one-parameter version), **`getClockTime()`**, and **`setTime()`**. You'll need an instance variable representing the time (as suggested above in the section "It's about time", it is easiest to represent time as the number of minutes after midnight). Start writing some test cases, for example, what should the output be for the following?

```
AlarmClock clock = new AlarmClock();
System.out.println(clock.getClockTime());
clock.advanceTime(4000);
System.out.println(clock.getClockTime());
clock.setTime(2, 30);
System.out.println(clock.getClockTime());
```

7. Note that the two-parameter **`advanceTime()`** method does not have to duplicate the code in the one-parameter version! Methods in a class can call each other, so the two-parameter version can just figure out the correct number of minutes and then *call* the one-

parameter version.

8. You can do something similar to step 6 for **`setAlarmTime()`** and **`getAlarmTime()`**. (In order to implement snooze, you will need another instance variable to keep track of the "effective alarm time" too, and for now you can just give it the same value as the alarm time.) Test your methods. Remember that the constructor needs to initialize the alarm time to 1:00 (see the javadoc for the constructors).

9. When you're testing, it might be convenient to see the clock time and alarm time as strings in the form hh:mm rather than as minutes since midnight. The methods such as **`getClockTimeAsString()`** that return a string are also pretty easy. Here is a very useful trick: once you figure out the value for the hours and minutes, you can create the a string in the form hh:mm like this:

```
String timeString = String.format("%02d:%02d", hours, minutes);
```

(The funny string with the percent signs is called a "format specifier"; if you are curious how it works, see section 4.3.2 of the text.)

10. You need some way to keep track of whether the alarm is turned on and whether the clock is ringing. Once you do that, you can easily implement **`alarmOn()`**, **`alarmOff()`**, and **`isRinging()`**. Make sure everything is correctly initialized in your constructor.

11. Now the fun part. Modify your **`advanceTime()`** method so that when the clock time reaches or passes the effective alarm time, the clock starts ringing. Don't worry about snooze right now! Start with some test cases. For example, what should the output be for the following?

```
clock = new AlarmClock();
clock.setTime(2, 0);
clock.setAlarmTime(2, 30);
clock.advanceTime(15);
System.out.println(clock.isRinging());
clock.advanceTime(15);
System.out.println(clock.isRinging());
clock.advanceTime(15);
System.out.println(clock.isRinging());
clock.alarmOff();
System.out.println(clock.isRinging());
System.out.println();

clock = new AlarmClock();
clock.setTime(2, 30);
clock.setAlarmTime(2, 30);
System.out.println(clock.isRinging());
clock.advanceTime(1440);
System.out.println(clock.isRinging());
System.out.println();
```

```
clock = new AlarmClock();
clock.setTime(23, 0);
clock.setAlarmTime(23, 30);
clock.advanceTime(60);
System.out.println(clock.isRinging());
```

12. At this point everything should be working except the snooze feature, which is definitely the hardest part. Be sure that you have unit tests for everything you've done so far, because implementing snooze will modify several existing methods and you want to check that you haven't broken the things that were working before! Be sure to keep a backup copy of what you've done so far too. *If everything is correct except the snooze feature, you will lose at most 30 points.*

13. Read the javadoc carefully and write some test cases. For example, what's the output of the following?

```
clock = new AlarmClock();
clock.setTime(5, 0);
clock.setAlarmTime(5, 15);
System.out.println(clock.getAlarmTimeAsString());
System.out.println(clock.getEffectiveAlarmTimeAsString());
clock.alarmOn();
clock.advanceTime(30);
System.out.println(clock.isRinging());
clock.advanceTime(5);
clock.snooze();
System.out.println(clock.isRinging());
System.out.println(clock.getAlarmTimeAsString());
System.out.println(clock.getEffectiveAlarmTimeAsString());
```

A key point to notice when testing is that the alarm time and the effective alarm time are different ONLY when snooze is in effect.

Style and documentation

Roughly 15% of the points will be for documentation and code style. Here are some general requirements and guidlines:

- Use instance variables only for the "permanent" state of the object, use local variables for temporary calculations within methods.
    - You will lose points for redundant instance variables. This class can be easily implemented with just five (or possibly six, depending how you think about snooze).
    - All instance variables should be `private`.
- **Accessor methods should not modify instance variables**.

- Unless explicitly specified, your code should not be producing console output. You will likely add **`println`** statements when debugging, but you need to remove them before submitting the code.

- Use the defined constants (**`SNOOZE_MINUTES`**, etc.). Don't embed numeric literals such as 9 or 1440 in your code.

- Each class, method, constructor and instance variable, whether public or private, must have a meaningful and complete Javadoc comment. Class javadoc must include the **`@author`** tag, and method javadoc must include **`@param`** and **`@return`** tags as appropriate.
    - You can copy and paste from the method descriptions online if you wish!
    - Run the javadoc tool and see what your documentation looks like! You do not have to turn in the generated html, but at least it provides some satisfaction :)

- All variable names must be meaningful (i.e., named for the value they store).

- Internal (//-style) comments are normally used inside of method bodies to explain *how* something works, while the Javadoc comments explain *what* a method does. (A good rule of thumb is: if you had to think for a few minutes to figure out how something works, you should probably include a comment explaining how it works.)
    - Internal comments always *precede* the code they describe and are indented to the same level. In a simple homework like this one, as long as your code is straightforward and you use meaningful variable names, your code will probably not need many internal comments.

- Use a consistent style for indentation and formatting.
    - Note that you can set up Eclipse with the formatting style you prefer and then use Ctrl-Shift-F to format your code. To play with the formatting preferences, go to Window->Preferences->Java->Code Style->Formatter and click the New button to create your own "profile" for formatting.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`assignment1`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`assignment1`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in, and **for this assignment you**

are welcome to post and discuss test code. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled "Official Clarification" are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the second SpecChecker. The file will be named `SUBMIT_THIS_hw1.zip`. and it will be located in the directory you selected when you ran the SpecChecker. It should contain one directory, `hw1`, which in turn contains one file, `AlarmClock.java`. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Blackboard using the Assignment 1 submission link and verify that your submission was successful by checking your submission history page. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links."

We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **hw1**, which in turn should contain the file **AlarmClock.java**. Make sure it is **AlarmClock.java** , and NOT **AlarmClock.class**. You can accomplish this easily by zipping up the **src** directory of your project. (The zip file will include the other .java files in the project, but that is ok.) The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and not a third-party installation of WinRAR, 7-zip, or Winzip.