CSE 1322L - Lab 10

Introduction

In this lab, you will write a simple program that allows the user to set up alarms that go off once their timer expires. These alarms must be run independently using multithreading.

Requirements

The features described below must be in your program:

- A class Alarm that is a subclass of Thread
  - It has 4 fields:
    - An integer called “timer”
    - A String called “name”
    - An integer called "id”
    - A static integer called “nextId”, initialized at 1
  - **Alarm(String, int)**: assigns nextId to id and increments nextId by 1. It then initializes the name and timer field as follows:
    - The timer provided in the arguments will be in seconds, but the timer field in the object must be stored in milliseconds. You must perform the appropriate conversion from seconds to millisecond before assigning the result to the timer field.
    - If the String argument is empty, the Alarm’s name must instead be set to “Alarm X”, where X is the Alarm’s id. If the String argument is not empty, set the name field using the argument.
  - **run()**: Override of its superclass method. It must do the following:
    - Every 1 second, it must decrement its timer field by 1 second
    - If there are 10 seconds remaining in the timer, it must print “X will go off in 10 seconds.”, where X is the Alarm’s name.
    - Once the timer reaches 0, it must print “X has gone off!”, where X is the Alarm’s name. The thread must then die.
    - If the thread gets interrupted before it reaches 0, it must print “X has been interrupted at Y seconds.”, where X is the Alarm’s name and Y is its remaining time. The thread must then die.
  - **String toString()**: Override of its superclass method, returning the following string (where X is the alarm’s name and Y is its remaining timer, in seconds):

```
X is currently at Y seconds
```

- Your driver must create an arraylist of Alarms. It must then implement the menu options below:
    1. **Create new alarm**: Prompts the user for a new alarm name and its timer, in seconds. It then creates an Alarm object with the information above, appends it to the arraylist, calls and prints the Alarm's toString(), and then starts the Alarm's thread.
    2. **View all alarms**: Traverses the arraylist, printing a call to the toString() only of Alarms whose threads have not died yet.
    3. **Quit**: prints "Stopping all alarms…". It then interrupts any Alarms in the arraylist that are still ongoing. Only after all alarms have been interrupted should it then print "All alarms have been stopped", after which the program must terminated.
    - Your driver must be able to recover from any explicitly thrown exceptions
    - Your driver must also be able to recover from the user entering a non-number when prompted for the number of seconds when creating an Alarm

Deliverables

- Lab10.java (driver)
- Alarm.java

Considerations

- Despite what the deliverables above say, you can submit all of your classes in a single file.
- For a thread to die, its run() must terminate its execution.
- You can find a list of useful Thread methods in the slides, or you can check its documentation here.
- Because your operating system is in charge of deciding when a thread should execute, your calls to sleep() will not be precise. sleep(5000) does not guarantee that a thread will sleep for exactly 5 seconds. Rather, the thread will sleep for at least 5 seconds. For the purposes of this lab, we will pretend that the thread does sleep for exactly 5 seconds.
- Because your program never deletes any of the expired Alarms and your user can no longer interact with them, we can say that your program leaks memory: As your program runs for longer and longer, it will consume more and more memory as you create more Alarms until your operating system refuses to supply your program with more memory.
    - We can avoid this by either allowing the user to view previously expired Alarms, or by removing Alarms from the arraylist once they expire.

Because of the nature of multithreading, your output may differ from the one below, depending on how fast you type the sample input.

Because both input and output streams are connected to the same location (the console), the output below will be intermixed with the input. Use the red bold statements to figure out the user's input.

Sample Output (user input in red)

```
[Alarm System]
1. Create new alarm
2. View all alarms
3. Quit
Enter option: 1

Enter alarm name: Rice cooker
Enter alarm timer in seconds: 3600
Rice cooker is currently at 3600 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: 1

Enter alarm name: Cheese sandwich
Enter alarm timer in seconds: 1 minute
Invalid timer: Timer must be a whole number.

1. Create new alarm
2. View all alarms
3. Quit
Enter option: 1

Enter alarm name: Cheese sandwich
Enter alarm timer in seconds: 60
Cheese sandwich is currently at 60 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: 1
```

```
Enter alarm name:
Enter alarm timer in seconds: 30
Alarm 3 is currently at 30 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: Alarm 3 will go off in 10 seconds.
1

Enter alarm name: Lunch plate
Enter alarm timer in seconds: Alarm 3 has gone off!
150
Lunch plate is currently at 150 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: 2

Here are all the alarms still running:
Rice cooker is currently at 3537 seconds
Cheese sandwich is currently at 20 seconds
Lunch plate is currently at 147 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: Cheese sandwich will go off in 10 seconds.
Cheese sandwich has gone off!
2

Here are all the alarms still running:
Rice cooker is currently at 3515 seconds
Lunch plate is currently at 125 seconds

1. Create new alarm
2. View all alarms
3. Quit
Enter option: 3

Stopping all alarms...
```

```
Rice cooker has been interrupted at 3513 seconds.
Lunch plate has been interrupted at 123 seconds.
All alarms have been stopped.
Shutting off...
```