Title
Sleeping Barber with Two Barbers

Lecture Topics Emphasized
Sleeping Barber
Locks and Conditions
Producer/Consumer
Multi-Threaded Programming Design

Introduction
The Sleeping Barber problem is a famous concurrency problem that deals with multiple customers (independent threads) with a single barber (shared resource). If the barber has no customers, he will go to sleep, requiring the next customer who enters to wake him up. If the barber is currently cutting a customer’s hair, the customer will have to wait for the barber to finish. If there are too many customers already waiting, the new customer will leave. For this lab, you will create this application.

Description
You learned about the Sleeping Barber problem in class, and there is a solution to the problem already posted on the Lectures page. For this lab, you are going to modify this solution so that instead of having one barber, there will be two barbers cutting hair simultaneously. The two barbers will serve customers from the same pool, and the number of available seats in the waiting area remains three. If both barbers are sleeping and a new customer comes in, we only want to wake up barber 1. However, if a second customer comes in, and one of the two barbers is sleeping, we want to wake up the sleeping barber and ask him to serve the new customer.

Although there is not much code to modify for this lab, the logic is extremely important. You should spend time understanding the existing code, and run it multiple times to make sure that you completely understand how it is working and when different sections are executed.

Start with creating a standalone Java project. Download the three java files (SleepingBarber.java, Customer.java, Util.java) from lecture #24 and copy them into your project.

Now we have two barbers, and we will need to uniquely identify their print statements. Modify the barber’s constructor so that it takes a string as a parameter, which is going to be the barber’s name. Add the barber’s name to the existing print statements.

Since we now have an additional barber, we will also need to pass the second barber to the customer. Make sure you modify the customer’s constructor accordingly.
The second barber is created through a new instance of the barber class. But, the two instances will have to share many resources. Identify the resources that they will need to share, and make the shared variables static, which means they are now class variables. In addition, to make the class variables thread-safe, change the ArrayList to the appropriate object.

The two barbers share the same customer base, which means the addCustomerToWaiting method should be shared across the two barbers. A class method supports the above functionality, and can be trivially achieved by adding the static keyword.

In the customer's run method, instead of always waking up the same barber, you will now need to identify the correct barber to wake up. To help you do that, add a Boolean to the barber class, and update the variable in the appropriate places to reflect if the barber is currently sleeping. Use the variable to help you decide which barber to wake up. Remember, if both barbers are sleeping, wake up barber 1. Otherwise, wake up the barber who is sleeping. You should not be waking up both barbers.

Upon the completion of all customer threads, be sure to wake up both barbers, so that they can both go home (i.e. your program should eventually end).
Grading Criteria
Labs are graded based on your understanding of the course material. To receive full credit, you will need to 1) complete the lab following the instructions above AND 2) show your understanding of the lab material by answering questions upon check-off.

If there is a discrepancy between your understanding of the material and your implementation (i.e. if your code is someone else’s work), you will receive a grade of 0 for the lab. Please note, it is the professor’s discretion to report the incident to SJACS.

Instructors, to ensure consistency across all lab sections, please strictly stick to the following criteria:

1) A second barber
   a) 0.05% - The student creates a second barber through instantiating the barber class
   b) 0% - There is NO second barber (no partial credits)

2) Print statements
   a) 0.10% - the barber constructor takes a string, and the string is added to the existing print statements
   b) 0.05% - the barber constructor takes a string, but the string is NOT added to all print statements
   c) 0% - the barber constructor takes NO parameters (no partial credits)

3) Static variables & methods: Please note, making all variables static will break the program, and will NOT get you any credit for this part
   a) 0.20% - the student has correctly identified all static variables and methods by adding the static keyword to them, and made approbate changes to the existing code to make it compatible with the keyword
   b) 0.10% - the student has correctly identified all static variables and methods by adding the static keyword to them, but FAILED to make appropriate changes to the existing code to make it compatible with the keyword
   c) 0.05% - the student has correctly identified more than HALF of the static variables and methods by adding the static keyword to them
   d) 0% - the student has identified less than HALF of the static variables and methods (no partial credits)

4) Signaling barber
   a) 0.20% - The student wakes up the appropriate barber through the use of the Boolean in the barber class
   b) 0.10% - The student correctly updates the Boolean in the barber class to reflect if the barber is currently sleeping, but is UNABLE to wake up the appropriate barber based on the Boolean
   c) 0% - The Boolean in the barber class does NOT reflect if the barber is at sleep (no partial credits)

5) Program termination
   a) 0.05% - Both barbers are able to go home (i.e. the program finishes upon the completion of all customer threads)
b) 0% - The program does NOT end *(no partial credits)*

6) Check-off Questions: Please randomly select three questions from the list. Each question is worth 0.07%, and the section is worth 0.20% total.

A) What are the static variables and methods? Why is it necessary to make them static?

B) (Potential interview question) What is the difference between instance variables and class variables? Why do we need class variables for this lab?

C) What could happen if we didn’t change the ArrayList to the appropriate object?

D) Why do we need to make the addCustomerToWaiting method static? What could go wrong if we didn’t make it static?

E) According to the implementation above, when there is no customer waiting, do we know for sure which barber does the new customer go to?

F) According to the implementation above, when there are customers waiting, do we know for sure which barber does the new customer go to (assuming there are available seats in the waiting room)?

G) How do the while loop and Thread.yield() make sure the program finishes as soon as possible?

H) In the original implementation (i.e. the code posted on the course website), when do we wake up the sleeping barber?