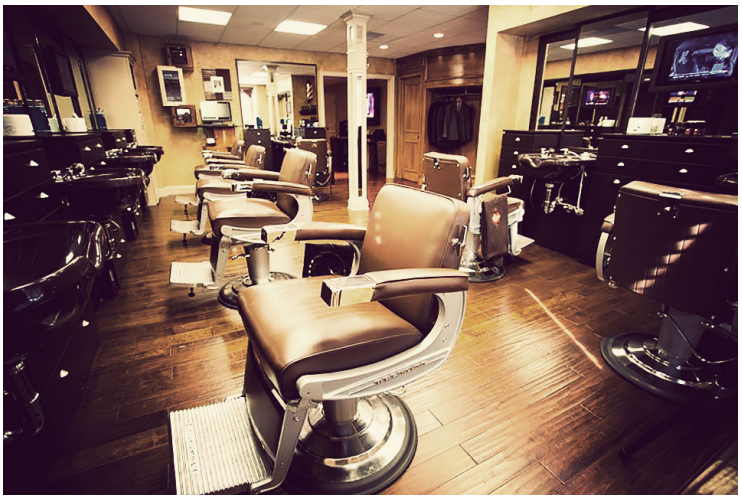
Hilzer's Barbershop with SCOOP

Riccardo Tommasini and Marco Balduini

June 9, 2016

1 Problem Description



The Hilzer's barbershop[1] consists of a shop with a waiting room of a given capacity (waiting-room-size); a sofa with a give capacity (sofa-size); a variable number of chairs and a corresponding number of barbers; a cashdesk that the barbers use to accept payments.

A Customer enters the barbershop if it has room, i.e. the total amount of customers inside is lower than the limit imposed by fire laws.

Once inside, the Customer try to take a sit on the sofa, if the sofa is at its dimension limit, he waits in the standing room.

When a barber is free, the customer that has been on the sofa the longest is served and the customer who was waiting standing the longest sits.

After the hair cut the customer can pay a free barber (anyone, not only the one who cut his hair). Since there is only one cash desk the payment happens one customer at time.

[1] Little Green Book of Semaphores, 5.3, http://www.greenteapress.com/semaphores/downey08semaphores.pdf

Notably, the barbers divide their time among cutting hair, accepting payment, and sleeping in their chair waiting for a customer.

We can elicit the following requirements: (R1) the shop has to have a size; (R2) the waiting room has to have a size; (R3) the sofa has to have a size; (R4) sum of sofa and room sizes is equal to the shop one; (R5) the entrance order has to be respected (R6) only one customer can pay at time.

2 Analysis

This problem definition of the previous section implies the implementation of some FIFO queues in order to ensure synchronization between processors. At this stage we can distinguish three queues, i.e. the Shop's waiting room, the sofa and the cash desk.

- SHOP WAITING ROOM: it is an FIFO queue that ensures the synchronization of the customers to enter;
- SOFA: it is a FIFO queue that ensure the synchronization of the customers to get an haircut.
- CASHDESK: it is an FIFO queue that ensures the synchronization for the payment process.

To design our solution we need to fulfill the requirements presented in the previous sections. Moreover, since we are dealing with concurrent programming, we have to face the following issues:

I1 Customers are a living object in our system.

I2 Customers creation must be random as well as their interaction with the system

I3 Barbers and Customers have directly interact so that the latter can cut the former's hair and/or pay.

I4 The way that queues are implemented characterizes the final execution model.

3 Design

Our problem design is Customer-centric, i.e. the customer is the only live actor in the system of separate objects and contains a direct reference to all the other involved classes.

We modeled the following classes:

- SHOP: it keeps track of how many CUSTOMERS are inside and how many are waiting in the room and make them sit on the sofa ensuring their **FIFO** ordering.
- SOFA: it keeps track of how many CUSTOMER are waiting sat and make them get and haircut ensuring their **FIFO** ordering.
- CUSTOMER: it is the agent (guides the execution) and its life-cycle is explained in the Section 3.1
- BARBER: it cuts customer hairs
- CHAIR: it is a proxy for a **QUEUE** of *separate* BARBER that allows them to be connected to the customers in a synchronized way.
- CASHDESK: it keeps track of how many CUSTOMER have to pay and make them pay ensuring their **FIFO** ordering.

Figure 1 shows an UML representation of the classes. Notably, we did implement a deferred class that defines the FIFO behavior.

All the classes are declared separate in our main class. The CUSTOMER has separate references to SHOP, SOFA, CHAIR and CASHDESK; it has a detachable separate BARBER that is used to interact with the barbers.

A Customer checks the shop in a require condition and waits until the *open* variable of the shop is False. Once the shop is open, it allows Customers to enter and start standing while waiting for a shop sit

first citizen classes since they it opens and closes; it keeps track of how many customers are inside according to a certain amount; it keeps track of how many people are waiting standing in the room.

3.1 Customer Life-Cycle

The CUSTOMER continuously cycles over the following actions, until it has reached a zero needs of haircuts:

(1) it waits for the shop to open; (2) it attempts to enter the shop, if there is room inside it enters; (3) it attempts to sit on the sofa, if not possible it waits in the standing room (waiting condition); (4) it attempts get an haircut, i.e. attempts to get a free barber, if not possible it waits on the sofa (wait condition); (5) it attempts to pay, i.e. it attempts to get a free barber, if not possible it waits in front of the cashdesk; (6) it leaves the shop.

3.2 Solving the issues

SI1 The CUSTOMER is a "living" class in our system.

SI2 The SHOP opens and acts like a barrier for the CUSTOMERs to start "living".

SI3 The CUSTOMER has a detachable separate BARBER that is used to for hair cutting and payment interaction, by the means of a method to get a valid attached instance.

SI4 SCOOP handles FIFO queuing by the means of waiting conditions, therefore we introduce classes that keeps a status used for this extent.

4 Conclusion

The results and the code are available in an online repository [2]. The hardest part of the work was to understand how to handle the waiting conditions and tame with the FIFO ordering.

Moreover, we initially thought as the problem as a producer-consumer one, where the barbers were consuming the queue of customer waiting in the sofa.

We still think that such a solution is possible, but we failed in our attempt to realize it. The blocking issue we found regards how to keep the customer waiting while the barbers are completing their life-cycles.

[2] https://github.com/riccardotommasini/hilzerbs

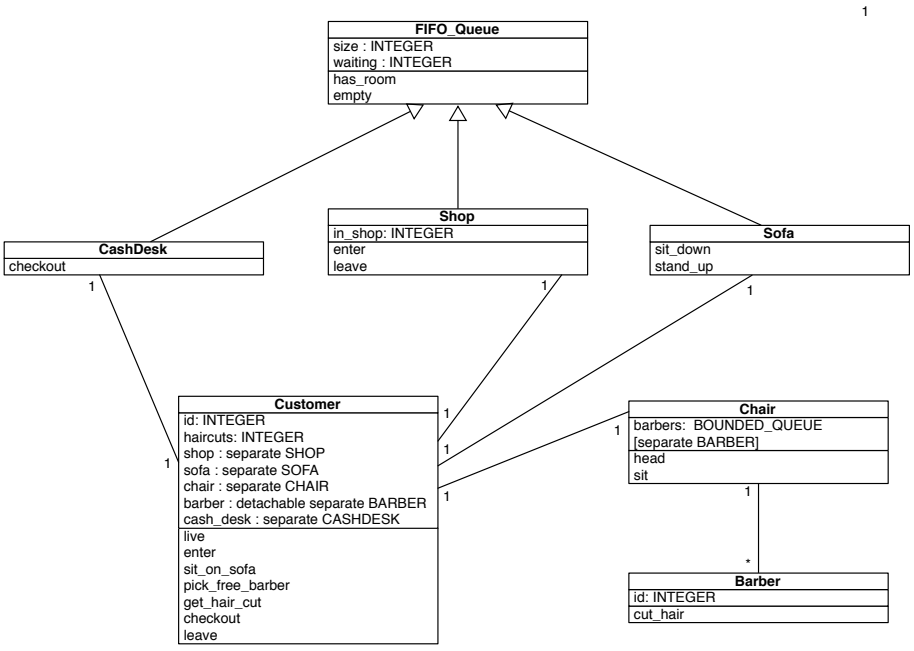


Figure 1: UML schema of proposed solution