

Contents

[Introduction 3](#_Toc41077914)

[List of Requirements Accomplished 3](#_Toc41077915)

[Concurrency Concepts and Justifications 4](#_Toc41077916)

[Waits(), Notify(), Thread.sleep(); 4](#_Toc41077917)

[Reentrant Locks 5](#_Toc41077918)

[Synchronized Block 5](#_Toc41077919)

[Volatile Keyword 6](#_Toc41077920)

[How to run program 6](#_Toc41077921)

[References 9](#_Toc41077922)

# Introduction

For my program, some of the assumptions I have changed are as such. If all seats are full, then rest of customers will exit the cafe. Any user can start first and upon calling of last orders, waiter will finish the order he has received before closing time and then leave as soon as he is done, while owner will also finish order received before closing time and then wait for customers to leave and will then leave as well. Moreover, customers who were unable to give orders before “last orders” was announced will be asked to leave and come back the next day. A full statistics report isn’t generated at the end but instead with every customer being served, it is updated to update user on how many users have been served and in the end will show total customers served during that day. Moreover, I have used a main,café and cupboard class for my resources while I have used four threads, user, clock, customer generator and customers. A linked list was also used to store 10 customers then make customers after that exit the café until one customer in linked list is served. Lastly, the timings for each resource accessed have also been modified accordingly.

# List of Requirements Accomplished

**The Café and Customers.**

* 1 table with 10 seats, if seats are full, customers will exit the shop.
* Customers served on first come, first serve basis and leave after one drink.
* There is a station with juice tap, cupboard (consisting of all the ingredients and cups)
* Taking ingredients from cupboard, Making each drink, serving and and putting them back takes a set amount of time.
* Unlimited resources, ingredients and tap separate resources and used by one person at a time.

**Clock**

* Calls last orders and café close time.

**Owner and Waiter**

* serve from one queue, one at a time.
* No more orders served after last orders.
* Waiter leave at closing time after serving order received before last orders.
* Owner leaves after last customers.

**Statistics and Configuration**

* Number of customers served shows after every customer is served.
* Cappuccino takes more time than Juice to make due to more ingredients.
* Customers can be set to zero by disabling customer generator thread in main class.
* Ratio of juice and cappuccino drinkers is decided by random generator.

# Concurrency Concepts and Justifications

## Waits(), Notify(), Thread.sleep();

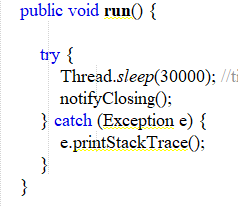
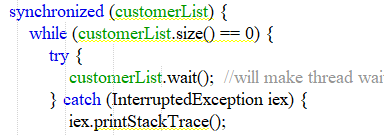


Figure 1 - use of wait for linked list (customer list) in café class.

Figure 2 - time of program duration in clock class

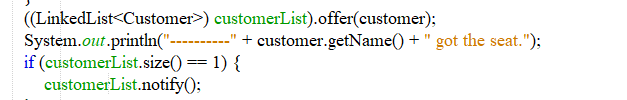


Figure 3 - use of notify in café class

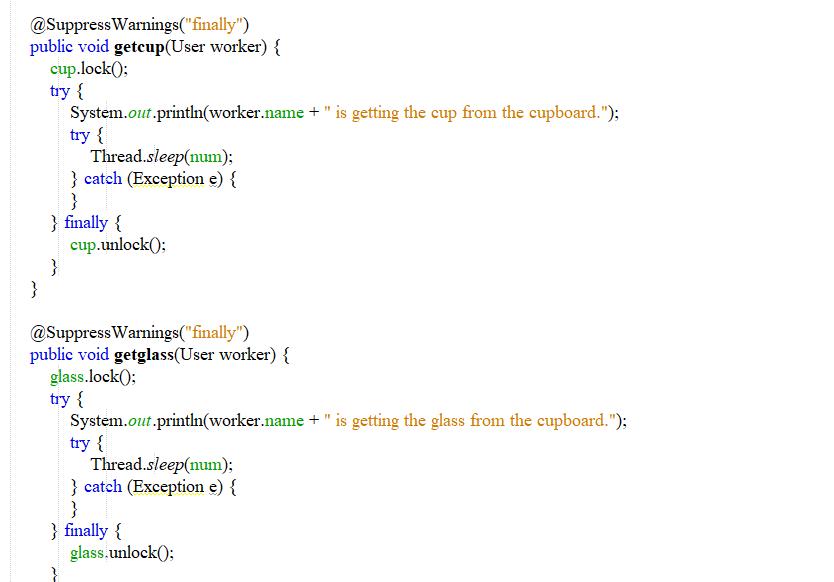
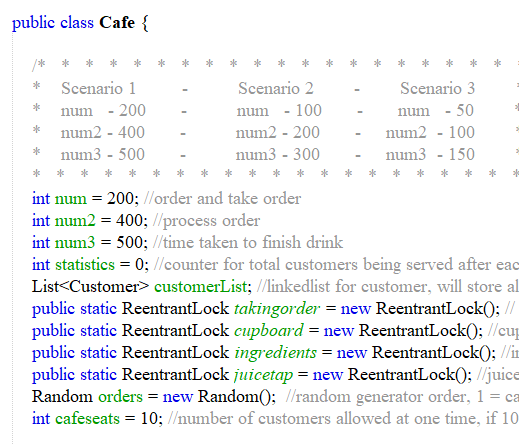
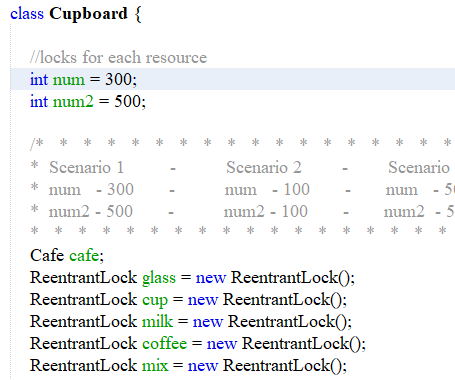


Figure 4 - use of thread.sleep in cupboard class

These are the only instances where wait and notify was used. It seemed easiers to implement Thread.sleep() than wait and notify to allow my customers to enter, drink and leave and to wake up owner and waiter since Thread.sleep() will send the thread (ex. Owner) to be a not runnable state and allow the thread to keep the locks it has. Such as when it is within a synchronized block as shown in my program. It is most suitable for the time synchronization so it will work until the time has ended. The program runs for 30 seconds, so program will stop after 30 seconds. (Gupta, n.d.)

## Reentrant Locks

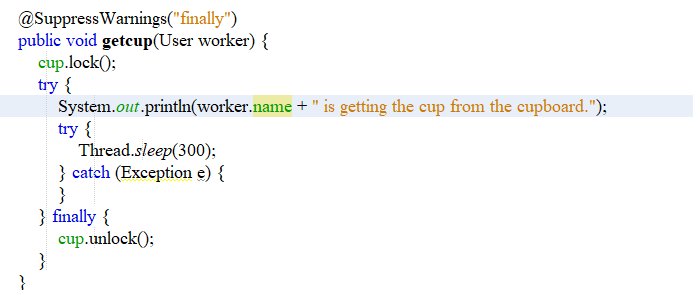


Figure ,6,7 - Reentrant Locks used in Cafe and Cupboard classes for resource accesses

A change in programming concept was using Reentrant Locks instead of synchronized keywords as it was realized that it would lead to a starvation of my items. ReentrantLock allows threads to enter multiple times into the lock on a resource. When the thread enters the at first so a lock count is set to one. The thread may be re-entered into the lock before releasing and each time the lock count is adjusted by one. For each request for release, lock count is decreased by one, and the resource is released if hold count is 0. Reentrant Locks can provide a fairness variable by which the lock would adhere by the need of the lock request i.e. after a thread unlocks the resource, the lock will go to the longest queuing thread. This method of fairness is established by transferring true to lock's constructor. (GeeksforGeeks, 2017) They were used in for my resources both in the Café class and the Cupboard class, for cupboard,milk,glass,cup,coffee,juicetap,mixing,cappuccino and order resources.

## Synchronized Block

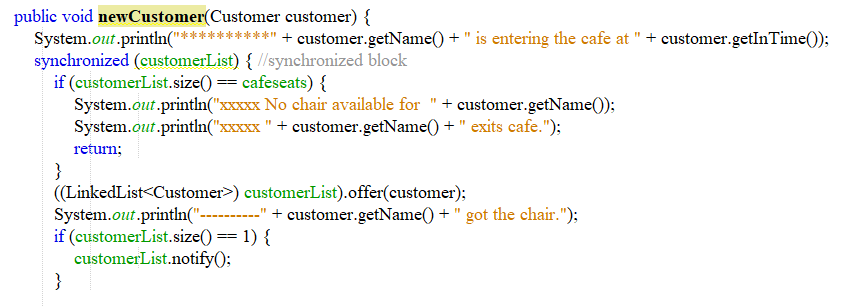


Figure 8 - synchronized block in cafe class( for new customers).

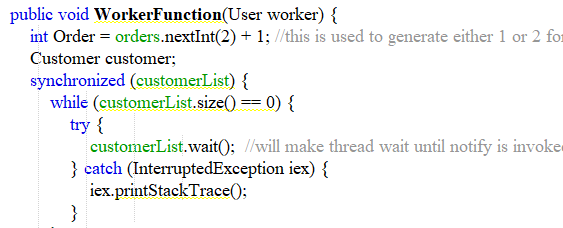


Figure 9 - synchronized block in worker function (serving customers)

I also used synchronized methods instead of synchronized methods since they perform better than the latter. It allows for other parts of the method to still be accessible to other threads. It will allows for that part of the code to be able to execute as a synchronized method. The customerList within the block will be its monitor object and an individual thread will be abllowd to execute within the block on same monitor object. (Jenkov, 2020) (Angular University, 2014)

## Volatile Keyword

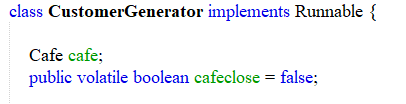


Figure 10 - use of volatile keyword in customer generator.

This keyword was used to help ensure thread safety. It was used for when we need to close the café and stop generation customers. When it will be last orders so cafeclose method will notify user threads. This keyword ensures that threads can be read from the key memory. It was implemented for closing time. (Ugarte, 2020)

# How to run program

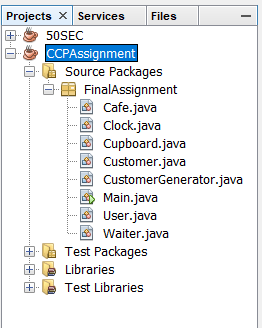


Figure 11 - project bar in netbeans

After unzipping the folder, netbeans IDE can be launched. The folder “CCPAssignment” can be simply dragged to the Project Console. Then click on “Source Packages”->Final Assignment. Here right click on Main.java and click “Run file” to run the program.

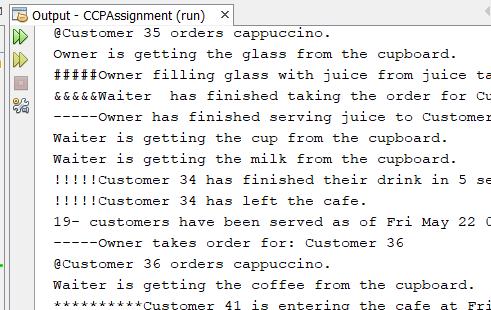
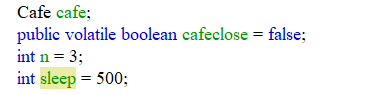


Figure 12 - Output of program

To adjust how long it takes to run the program. Double click on Clock and go to the run method.



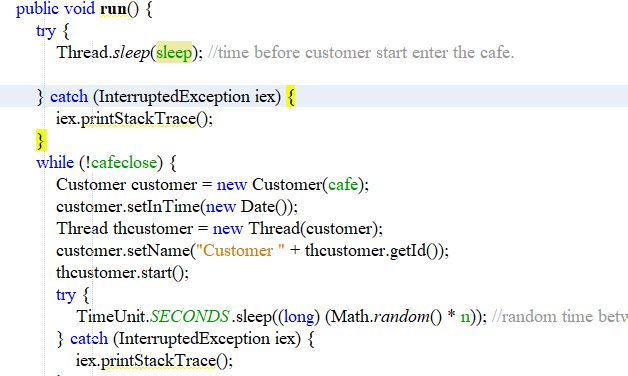


Figure 13 - Thread clock.

In Thread.sleep(sleep) has been initialized to 500 and n has been initialized to 3. This values can be adjusted as shown above where n= 3 and sleep = 500.

Resources such as getmilk can also have their timings changed the same way. Double click on Cupboard and change num values. . Class “Café”can also have their timings modified such as when they are taking orders from customers.

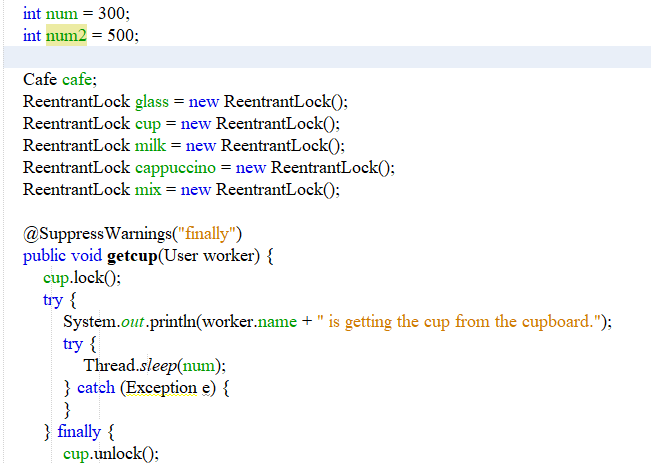


Figure 14 - Change it from 300 to whatever is preferred by User.

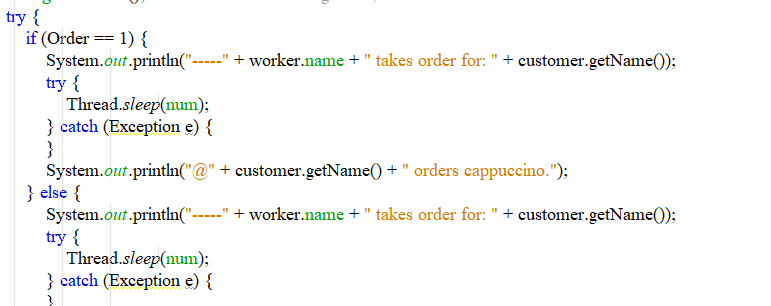


Figure 15 - Thread.sleep() when order is taken from customer.

# References

Angular University, 2014. Why Is Synchronized Block Better Than Synchronized Method?. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/20906548/why-is-synchronized-block-better-than-synchronized-method> [Accessed 20 May 2020].

GeeksforGeeks. 2017. Reentrant Lock In Java - Geeksforgeeks. [online] Available at: <https://www.geeksforgeeks.org/reentrant-lock-java/> [Accessed 21 May 2020].

Gupta, L., n.d. Difference Between Sleep() And Wait() In Java - Howtodoinjava. [online] HowToDoInJava. Available at: <https://howtodoinjava.com/java/multi-threading/sleep-vs-wait/> [Accessed 21 May 2020].

Jenkov, J., 2020. [online] Tutorials.jenkov.com. Available at: <http://tutorials.jenkov.com/java-concurrency/synchronized.html> [Accessed 21 May 2020].

Ugarte, A., 2020. What Is Thread-Safety And How To Achieve It | Baeldung. [online] Baeldung. Available at: <https://www.baeldung.com/java-thread-safety> [Accessed 21 April 2020].