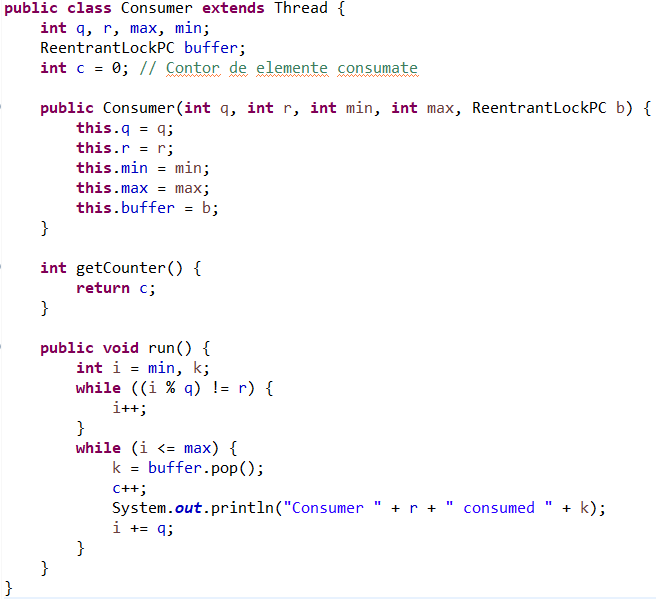
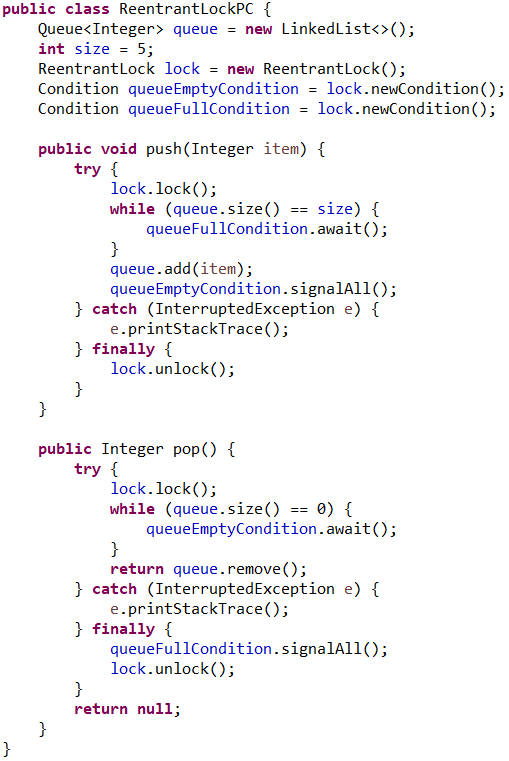
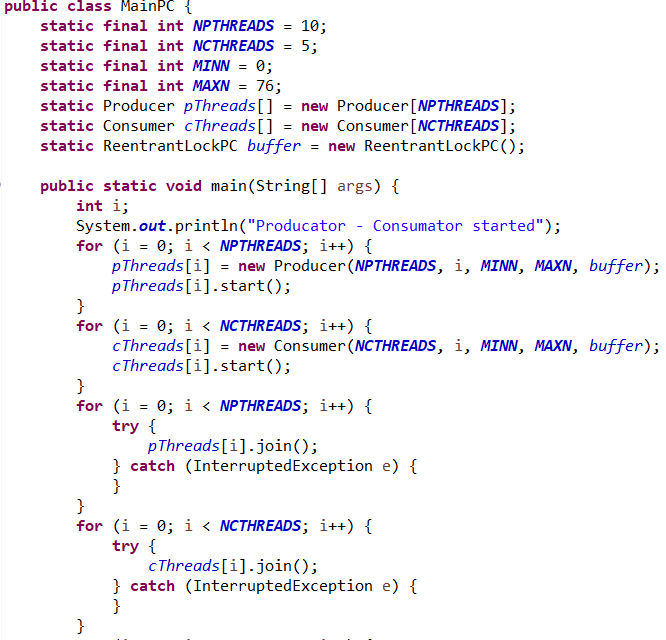
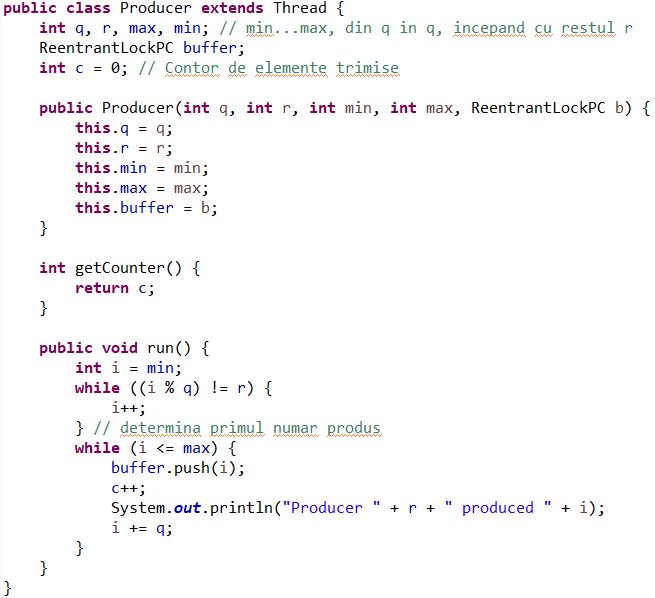
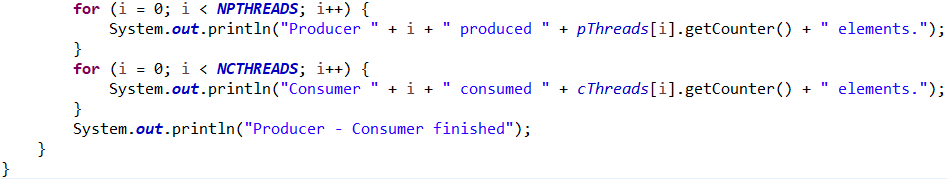
**Laboratory 7**

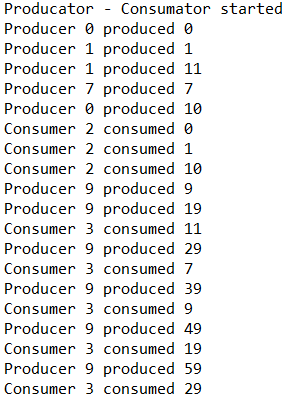
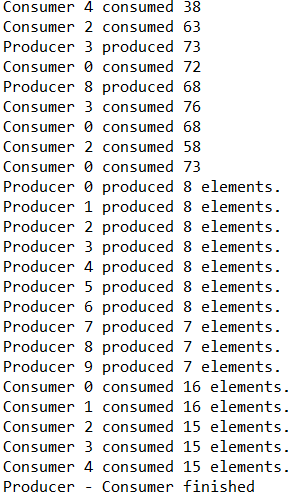
1. I implemented the Producer-Consumer example using locks as follows:







The result of running this code is:

 ………. 

The previous code has 10 producers and 5 consumers, each of them with its own task.

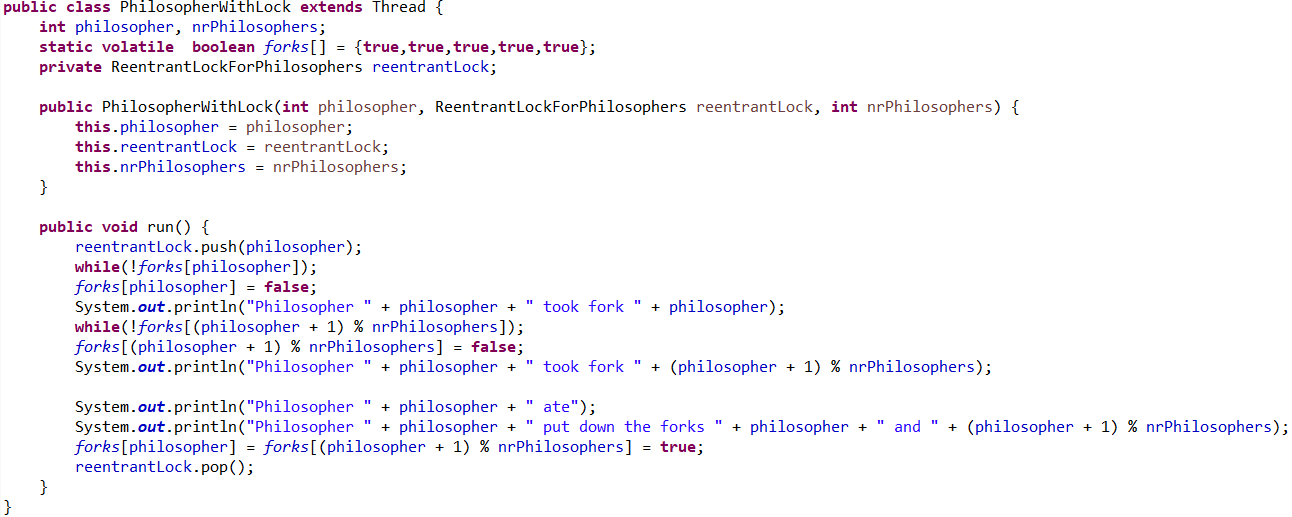
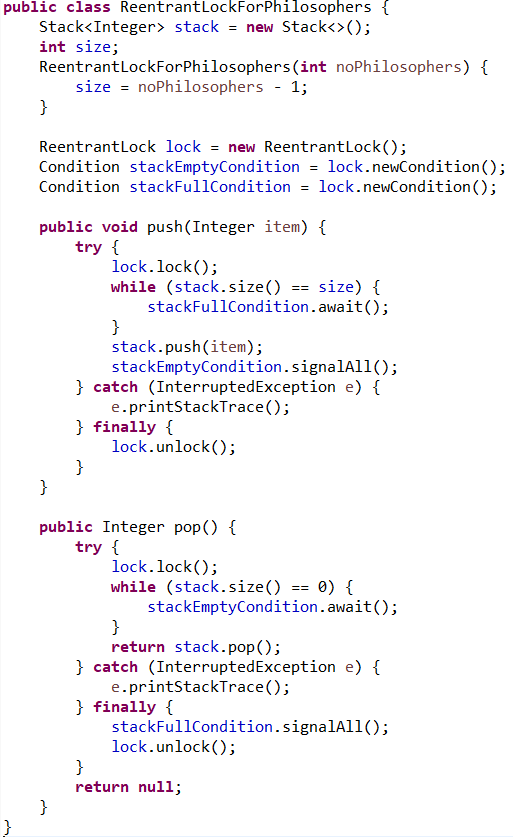
Each producer is creating an integer that has the rest of division by the number of producers equal to their ID number, and puts the generated integer in a queue of size 5, common for all producers and consumers. If this queue is fully occupied, it will wait for the last entity to be consumed before putting the newly created one in its place.

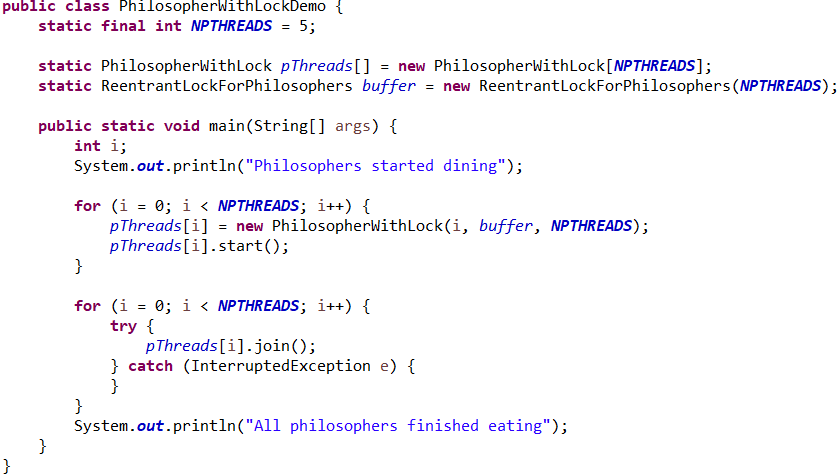
Each consumer will count from a minimum given number up to a maximum given number, and every time the counter is a number that divided by the total number of consumers creates the rest equal to the consumer id, the consumer will try to consume the oldest integer generate by the producers in the queue. If the queue is empty, the consumer will wait till a new entity will be placed in the queue.

In the end all the producers will have similar number of integers produced (the range of the interval divided by number of producers), each of them creating an integer by the algorithm explained above, resulting in all the elements in the given range being produced only once.

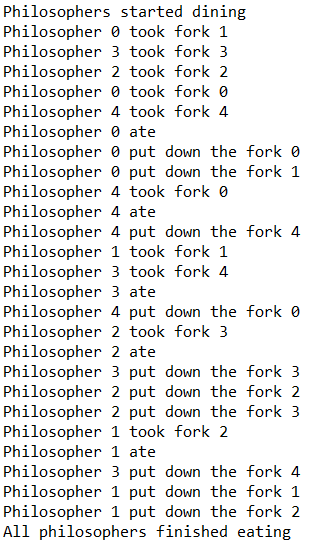
The consumers will also have similar number of integers consumed (the range of the interval divided by number of consumers), but those will simply consume the oldest entity produced in the queue.

1. I implemented the “Dining philosopher’s problem” with locks as follows:





The output for the previous code is:



As it can be observed, I used for the implementation the variant 2 possible solutions for this problem. I reduced the number of philosophers that can eat at the same time by one, to assure that we will not end up in an interlocking situation.

I set all philosophers to pick first the left fork, then the right one, but only 4 will be able to try to eat at a given moment. In this way, I will assure that always at least one philosopher can get both forks to eat, solving the situation when there could be all the philosophers stuck with only one fork.

The **push** method will wait till there are fewer than 4 philosophers eating at the table, so that the current one can join, after one seat is available for dinner, the philosopher will be added in the queue.