CS307 HOMEWORK 2 TUĞCAN BARBİN 25168

Firstly, I wanted to point out that I could not properly understand the homework document and I had no time to ask TAs due to exam week and hard work we had. These are some of the context I could not understand and I think not clear;

1-) philosophers must come to table around 1 to 10 seconds randomly. However, in the sample run video, it takes around 20 seconds to come all the philosophers to the table. Also, due to sleep(5000) in philosopher creating part that you have provided us, the randomized walking does not show itself because there is a huge creation delay between philosophers. And as I understand from document we don't need to and should not change anything related the GUI. So, we can implement and change some of thread functions? So, I did. Changed the sleep to 1 seconds. So we can clearly see the randomized walking procedure.

2-) I used java first time and in documentation, we said to implement our functions to run part. However, there was two run part and I thought that we need to implement @override run part.

However, There was a huge problem I had at the beginning of my homework which occurred as

Thread 0, had a problem while using putPlate_GUI because the table pointer is null (smth like that). But then thread 1 2 3 4 was working fine. I could not fix this and I changed the actual run part. Put the table creation part to outside of run function and it worked perfectly I do not know if it is a problem or not. But I tested like 1000 times with long periods and it did not crash anything.

That's all, these were my problems if I done anything wrong about the changes I wanted to inform you about situation.

Code;

1-) I used semaphore barriers, to implement barrier to wait all philosopher until everybody put the plate on the table. If a philosopher come to table increments all semaphores and then decrements its own 5 times which is N. It provided me everybody has to come to table for each philosopher to continue because otherwise the semaphore remains negative. When everybody comes, everyone's semaphore incremented 5 times and becomes 0. So, they can continue

2-) For the dining part, I kept track of each id of philosopher's right, right's right, left and left's left. I always decremented mutex, if I changing any state. After they become hungry, if a neighbor is not eating then current can EAT and increments its semaphore to be available to eat. Decrements it before starting eating. Then when the eating is done locks mutex, checks its neighbors to understand if they can eat by looking neighbors' neighbors. If they are available then increments their semaphore to stop their waiting. Because they waited since first check was false and their semaphores are not incremented and waited for incrementation. After that opens mutex.