Middle East Technical University | Department of Computer Engineering

CENG 443

Introduction to Object-Oriented Programming Languages and Systems

Spring '2020-2021
Assignment 2

Due date: May 30, 2021, Sunday, 23:59

1 Introduction

In this assignment, you will build a small musical chairs game simulation using thread barriers.

Kids are bored in the kindergarten, and the preschool teacher offers to play musical chairs game. The game starts with N students and N-1 chairs. The teacher starts the music, and the students walk around the chairs. As soon as the teacher stops the music, the students sit on chairs, and one who cannot get a chair is eliminated at that round. Then the teacher moves one chair away, and the game continues in this fashion until one student wins the game. After the game finishes, the teacher asks the students how they did in the game.

Keywords: *java, concurrent programming, thread, synchronization, barrier*

2 Specifications

In this simulation, there are five classes: **MusicalChairsGame, Teacher, Student, Chair, Barrier**. The application will get the student names as command line arguments.

Students will wait until the Teacher starts the round. The Teacher will warn about the beginning of the new round, and the Students will say if they are ready. After the warning, she will wait for 2 seconds before she starts the music. When the music is played, the Students will walk around the chairs. After a random amount of time, between 4 to 9 seconds, she will stop the music. Each Student thread will print out if she/he can get a chair or not. The Teacher will remove one chair and the loser from the game. This process will be repeated with the Student and Teacher threads until there is a single Student who is the winner. When the game is over, the Teacher will ask about the game history to each Student thread, and they will print out the summary. Since there is no guarantee about the order, each run might give a different output which is the intended behavior.

A barrier is a kind of synchronization method which allows multiple threads to wait for each other. You can visit **here** to get more information about the Barrier approach.

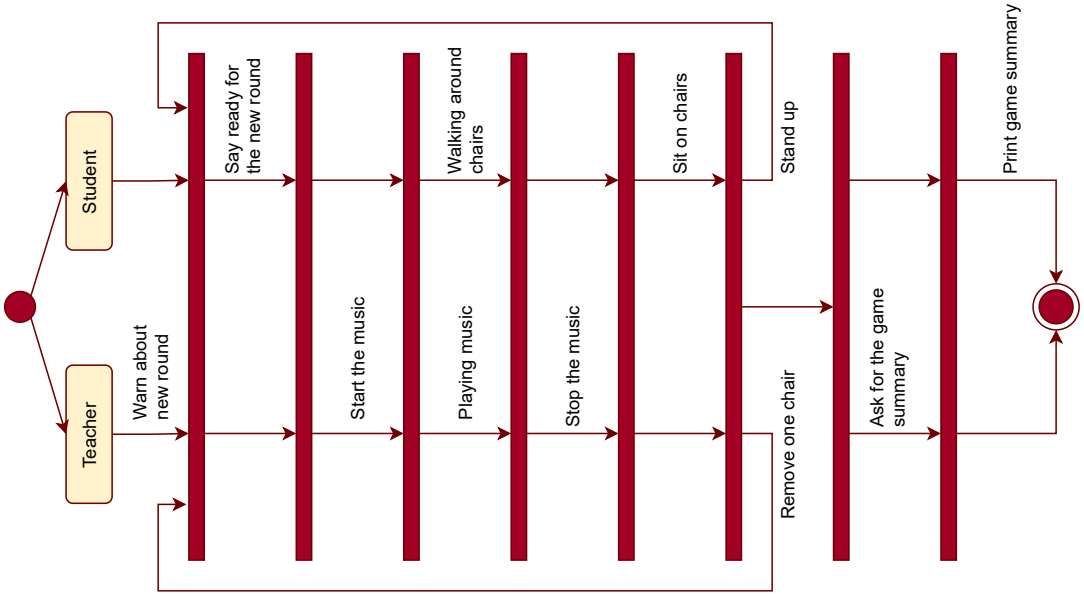

Figure 1: The schematic model of the application that uses barriers.

- **MusicalChairsGame:** Singleton class that is the responsible class for storing the required entities.
- **Teacher:** The class, extending *Thread*, that contains the main function. It manages the game, i.e. starts the rounds, plays the music for random amount of time, removes the student who couldn't sit in the round, stands all students up and removes one random chair from the game.
- **Student:** The class, extending *Thread*, for representing the players in the game.
- **Chair:** The class for representing the the chair.
- **Barrier:** It is the main class for implementing the barrier. Its constructor gets number of threads to reach barrier point as parameter. The await method notifies all threads that are waiting if the waiting number of threads reach the given nubmer.

You are given the stub code to fill and missing parts are indicated with **TODO** keyword. You will only use threading primitives: synchronized, wait, notify, notifyAll. Do not use high level concurrency constructs and do not add new class or change the existing ones. Some of the function signatures are missing for you to complete. You should be able to hear the music when your implementation is ready.

3 Sample output

Sample output with 3 students is below. Additionally, you can watch the sample video **here** to get an idea about the final application.

```
>> javac *.java
>> java Teacher Student1 Student2 Student3
Teacher: Be ready! Round 1 is about to begin.
Student3: I'm ready for the new round.
Student1: I'm ready for the new round.
Student2: I'm ready for the new round.
Teacher: The music is started!
Teacher: The music will be stopped in random amount of time...
Student3: I'm walking around.
Student2: I'm walking around.
Student1: I'm walking around.
Teacher: The music is stopped!
Student2: I got chair 1.
Student1: I got chair 2.
Student3: I couldn't get any chair.
Teacher : I am removing one chair.
--------------------
Teacher: Be ready! Round 2 is about to begin.
Student1: I'm ready for the new round.
Student2: I'm ready for the new round.
Teacher: The music is started!
Teacher: The music will be stopped in random amount of time...
Student1: I'm walking around.
Student2: I'm walking around.
Teacher: The music is stopped!
Student1: I got chair 2.
Student2: I couldn't get any chair.
--------------------
Teacher: You did great job kids. The game is over. How was the game?
Student2: I was eliminated in round 2! I sat on chair 1
Student3: I was eliminated in round 1! I never got any chairs.
Student1: I won the game :) I sat on chairs 2, 2.
```

4 Regulations

1. **Programming Language:** Java
2. **Late Submission:** A penalty of $5 \times \text{Lateday}^2$ will be applied for submissions that are late at most 3 days.
3. **Cheating:** We have zero tolerance policy for cheating. Your solution must be original. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. You can discuss algorithmic choices, but sharing code between groups or using third party code is strictly forbidden. In case a match is found, this will also be considered as cheating. Even if you take only a "part" of the code from somewhere or somebody else, this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar.
4. **Newsgroup:** You must follow the ODTUCLASS forum for discussions, possible updates, and corrections.
5. **Submission:** Submission will be made via ODTUCLASS. Create a "zip" file named "hw2.zip" that contains all your source code files inside "Source" directory and readme, javadoc in "Docs" directory. Please provide a readme which describes how to compile and run your code.
6. **Evaluation:** Your codes will be evaluated manually; small differences might be tolerated. In addition, your code will be examined to check the correctness of the implementation. You may get partial grades, so even if you are not able to implement all requirements, please submit the completed code. However, be sure that your submission can be compiled. Therefore, if you have differences when running the program, that will not be a problem.