Middle East Technical University

Department of Computer Engineering

CENG 443 Intro. to Object-Oriented Prog. Lang. and Systems

Spring 2021-2022

Homework#2 - A Day in Caketown

Version 1.0

Due date: May 29, 2022, 23:59

1. Introduction

During a typical day in Caketown, people all bake, serve and eat cakes:

- Suppliers bring cake stands to the cake ground. One Supplier at a time can enter the cake ground to place a stand inside and mark its availability on the board at the entrance.
- CakeBakers bake sliced cakes, carry them to the entrance of the cake ground, and then wait there by checking the board till a cake stand becomes available. One CakeBaker at a time can enter the cake ground to put the cake on an empty stand and unmark its availability on the board.
- CakeMonsters always wants to eat cakes. A CakeMonster directly runs to a stand randomly inside the cake ground and waits at that stand for a cake. Early-bird CakeMonsters, if any, must wait until the first stand Is put inside. One CakeMonster at a time can take a cake slice from one particular stand. Of course, there can be many CakeMonsters taking slices from different stands concurrently.
- No ordering or priority is specified about which Supplier or CakeBaker will enter the cake ground and which CakeMonster will get a cake slice.

2. Class to Implement

In this homework, you are expected to write the CakeStand class with the methods below to model this environment. Your class must be thread-safe, conforming to the specifications above and not limiting any other potential concurrency. You can only use the Java concurrency constructs covered.

- ***public static void supplyStand()***

This method should be called by a Supplier to perform its tasks listed above. Every new stand brought to the cake ground must be given a sequential id number starting from one. Only this method instantiates a new stand, and no other external code is allowed to do so.

It must print

```
<Supplier name> brought a new stand.
```

when the Supplier brought it to the cake ground;

`<Supplier name> added Stand#<id no>.`
when the Supplier put it there.

- ***public static void putCake(int slices)***

It should be called by a CakeBaker to carry out its tasks explained before.

It must print

`<CakeBaker name> baked a cake with <number of slices> slices.`
when The CakeBaker baked and brought the cake to the entrance of the cake ground;

`<CakeBaker name> put the cake with <number of slices> slices on Stand#<id no>.`
when The CakeBaker put the cake on an available stand.

- ***public static CakeStand randomStand()***

This method returns a random stand from the cake ground.

- ***public void getSlice()***

This method should be called by a CakeMonster to model its behavior given above.

It must print

`<CakeMonster name> came to Stand#<id no> for a slice.`
when the CakeMonster joins the cake stand;

`<CakeMonster name> got a slice from Stand#<id no>, so <number of slices> left.`
when the CakeMonster got a cake slice from the cake stand.

3. Additional Details

- Messages must be printed exactly as specified. Otherwise, your solution may fail the black-box tests.
- You will submit this CakeStand class to be tested by multi-threaded driver codes, which create and manage some number of threads playing the roles of Suppliers, CakeBakers and CakeMonsters. Hence, you will not do anything regarding thread management.
- You can simply print the name of the calling thread in your CakeStand class using ***Thread.currentThread().getName()*** for <Supplier name>, <CakeBaker name> and <CakeMonster name>.

4. Grading

- Clean implementation with JavaDoc/Java comments explaining your code…… 20%
- Tests to check whether your implementation matches the specifications listed…… 80%

5. Submission

Submission will be made via ODTUClass. You will submit your class file called CakeStand.java (or a zip file named hw2.zip containing this class and your helper classes if any). A penalty of 5 x LateDay^2 will be applied for submissions that are late at most three days.

Your submission must be your original solution. The regulations will be applied in case of high similarity scores.