CS242: Advanced Programming Concepts in Java

Fall 2016

Optional Assignment 5: Worth maximum 3 points bonus

In this assignment, you will be using multithreading to implement a mock-up of service at a restaurant. This is an extension of the Producer-Consumer problem we saw in class, as it involves multiple producers, or waiters, and multiple consumers, or customers. You will also be incorporating IO.
This is a single-person assignment. Do not submit copied code.

Our restaurant is an upscale *à la carte* restaurant, that allows customers to order one of several three course menus. This restaurant has one or more waiters. Each waiter waits on one or more customers. Each waiter serves each customer a three-course menu (an appetizer, a main course, and a dessert).

The waiter must serve only one course at a time, i.e., the waiter must wait for the customer to complete one course before serving another course. Once the customer completes a course, he/she must wait for the waiter to serve the next course. This concept is similar to the sushi chef-customer example in the notes.

The 'buffer' object in this assignment is a table. Each waiter serves one or more tables. Each customer eats at a single table allotted to that customer.

Your implementation must take into account the concepts we studied about **multithreading**, **synchronization**, and **deadlock**.

In this implementation, you will provide **four** classes: Table, Waiter, Customer, and Restaurant. The main class (described later) is Restaurant.

You must implement the following class 'Table' as a buffer. Use the description in the listing below to implement the Table class.

class Table

Data
private String course; // the name of the course
private boolean isEmpty; // a flag used to see if the table is empty or is not empty
(i.e., has an unfinished course)
private Object obj; // synchronization object, **not strictly necessary**
(i.e., depends on the synchronization method you use)

Constructor public Table(); // default constructor

Methods
public void serve(***String course***); // implements the Waiter serving a course <— CHANGED TO
// TAKE IN COURSE AS ARGUMENT
public String eat(); // implements the Customer eating a course

You must implement the following two classes, Waiter and Customer, as Runnable. Use the description in the listings below to create your classes.

class Waiter

Data

```
private final static int MAX_WAITER_MILLIS = 4000; // must wait for between 0 and 4 seconds.
private final static int N_COURSES = 3; // number of courses is exactly three.

private Table[] tables; // array of Table objects this Waiter waits on
private string waiterName; // name of this Waiter
private String[] customerNames; // names of Customers served by Waiter
private String[][] courses; // multi-dimensional array of courses for each Customer of this Waiter.
                            (courses[i][j] has the j-th course for the i-th Customer of this Waiter)
```

Constructor

```
public Waiter( Table[] tables, String waiterName, String[] customerNames, String[][] courses );
// initializes the data
```

Method

```
public void run(); // For each customer,
                   a thread on this Waiter object serves the three courses in the correct order
                   by calling the serve() method in the corresponding Table,
                   prints out what course is served to which Customer,
                   and sleeps for a random time between 0 & 4 seconds to mimic time taken in serving.
```

class Customer

Data

```
private final static int MAX_CUSTOMER_MILLIS = 4000; // must wait for between 0 and 4 seconds.

private Table table; // Table object that this Customer sits at
private string customerName; // name of this Customer
```

Constructor

```
public Customer( Table table, String customerName );
// initializes the data
```

Method

```
public void run(); // For each customer,
                   a thread on this Customer object eats the three courses in the correct order
                   by calling the eat() method in the corresponding Table,
                   prints out what course this Customer is eating,
                   and sleeps for a random time between 0 & 4 seconds to mimic time taken to eat.
```

For information on multi-dimensional arrays (i.e., to understand how you can populate the String[][] courses array), visit: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Your main class, i.e., **Restaurant**, needs to implement IO to read in waiter and customer information from a file. At the very least, Restaurant must contain the main() method. Your program must ask the user to enter the name of a file, and allow the user to enter the file name at System.in. Given the file name, the

program must open the file, and read in the number of waiters, waiter names, number of customers for each waiter, customer names, and customer courses from the file.

The program should read a file with the following format:
The first line of the file contains a single number representing the number of waiters.
The second line contains the name of the first waiter, the number of customers, the name of the first customer of the first waiter, his/her appetizer, his/her main course, his/her dessert, the name of the second customer, his/her appetizer, his/her main course, his/her dessert, and so on for all customers. Each item is separated from the prior item by spaces. Different words in a single course are separated by underscore, e.g., ice_cream.
The next few lines contains similar information for the second waiter, third waiter, fourth waiter, etc.
There must be a new line after the last line.

Given all the waiter and customer information, your program must create all waiter and customer threads with Waiter and Customer Runnable objects, and the program must start all threads.

Below is the input in "general.txt" (provided with the assignment, here W means waiter, C means customer, A means appetizer, M means main course, and D means dessert, W_1 means waiter 1, C2_1 means customer 2 of waiter 1):

```
2
W_1 3 C1_1 A1_1 M1_1 D1_1 C2_1 A2_1 M2_1 D2_1 C3_1 A3_1 M3_1 D3_1
W_2 2 C1_2 A1_2 M1_2 D1_2 C2_2 A2_2 M2_2 D2_2
```

This program must create and start 2 waiter threads + 3 customer threads for the first waiter + 2 customer threads for the second waiter. Following is an example run of the program using "general.txt"

```
Enter the name of the file to test: <—- prompt for the user to enter name of file
general.txt                         <—- name of the file entered at standard input
W_1 serves C1_1  A1_1               <—- output from run() method in Waiter object
C1_1 is eating: A1_1                <—- output from run() method in Customer object
W_2 serves C1_2  A1_2
C1_2 is eating: A1_2
W_1 serves C2_1  A2_1
C2_1 is eating: A2_1
W_1 serves C3_1  A3_1
C3_1 is eating: A3_1
W_1 serves C1_1  M1_1
C1_1 is eating: M1_1
W_1 serves C2_1  M2_1
W_2 serves C2_2  A2_2
C2_2 is eating: A2_2
C2_1 is eating: M2_1
W_2 serves C1_2  M1_2
C1_2 is eating: M1_2
W_1 serves C3_1  M3_1
C3_1 is eating: M3_1
W_2 serves C2_2  M2_2
C2_2 is eating: M2_2
W_1 serves C1_1  D1_1
C1_1 is eating: D1_1
W_1 serves C2_1  D2_1
C2_1 is eating: D2_1
W_2 serves C1_2  D1_2
C1_2 is eating: D1_2
W_2 serves C2_2  D2_2
```

```
C2_2 is eating: D2_2
W_1 serves C3_1  D3_1
C3_1 is eating: D3_1
```

Your output **will not necessarily look exactly** like the above output, as the waiter or customer threads may be ordered differently, e.g., W_1 may serve C2_1 before C1_1. The main point to note is that each customer must eat their particular main course after their particular appetizer, and their dessert after their main course. Example: the output "C1_1 is eating: M1_1" comes after "C1_1 is eating: A1_1", and the output "C1_1 is eating: D1_1" comes after "C1_1 is eating: M1_1". Also, the output of the customer eating the course must show up after the output of the waiter serving the course.

The assignment also has attached an example output from the file "minions.txt".

You must handle exceptions that arise because the user provided a non-existent file, or the formatting of the file is incorrect (wrong number of lines or tokens, non-integral numbers for the number of waiters or customers). You are permitted to terminate your program in the event of an exception if you feel that you cannot handle the exception to continue program execution. It is up to you whether you to decide whether you need the user to enter a filename with the .txt extension, or you can handle the .txt extension yourself, but make sure to prompt the user correctly either way.

Your program will be tested on two additional files not provided here, so you should make your program general enough to handle files of the type discussed.

Submission:
Your submission must be uploaded to Moodle. You can submit in one of the following two ways:

1) A .zip file containing your entire code (all .java files), HW5report.txt or HW5report.pdf, images if any, executable JAR file, and Javadoc doc/ folder. Please do not include supporting project directories that arise when you use an IDE. Your code may either be in the main folder of the zip file, or in an src/ folder. Your .class files may be in the main folder or a bin/ folder.
2) A .txt file with a link to a GitHub location, if you prefer to use GitHub. Your GitHub folder must contain the entire code, (all .java files), executable JAR file, HW5report.txt or HW5report.pdf, images if any, and Javadoc doc/ folder. Please do not include supporting project directories that arise when you use an IDE. Your .java files may either be in the main GitHub folder, or in an src/ folder. Your .class files may be in the main GitHub folder or a bin/ folder.

Grading Rubric (out of 100, scaled to 2):

Grading Item	**Score**
Code should compile and run correctly with the "general.txt", "minions.txt", and two additional files.	30
Correct implementation of Table class	10
Correct implementation of Waiter class	10
Correct implementation of Customer class	10
Correct implementation of reading file name from System.in, and reading in waiter and customer information from file in Restaurant class.	20
Correct creation of waiter and customer thread objects in Restaurant class.	10
Exception handling for bad IO	10