A very successful bakery is famous for their special pastries made according to elaborate recipes created by the owner of the bakery. There are three steps in making a pastry: making the dough, making the filling, wrapping the filling with dough. Finally the pastries are baked in the ovens. To improve productivity and meet the huge number of orders from clients, the owner of the bakery decides to divide the workers into three groups with each group only in charge of one step. To pipeline the work, the dough-making group and the filling-making group work in parallel; the wrapping group assembles the pastries when the dough and filling are ready and then puts them into the oven. Due to the limited space in the kitchen and the limited number of tools, each group can only work on a certain number of pastries at the same time. To make the worker’s life more complicated, from time to time, the owner of the bakery is in a creative mood and wants to change the recipe for the filling of the pastry. To avoid confusion, he will wait until the filling-making group finishes its current work. The filling-making group will wait until the owner finishes the recipe modification before they resume their work.

**Project:**

This project is to simulate a simple version of this bakery problem using Java. (Note that this is not the same as Lamport’s Bakery Algorithm discussed in class.) The problem specifications are as follows:

a. Assume clients submit orders (the unit of work is number of orders) in the morning when the bakery opens.

b. There are four worker groups: dough-making group, filler-making group, assembling group and baking group. Each group only works on its specific tasks and will not share the burden of other groups.

c. Each group other than the baking group can handle no more than 5 orders at the same time. It takes the dough making group 15 minutes, the filling making group 25 minutes and the wrapping group 10 minutes to fulfill one order.

d. There are 3 ovens in the kitchen. Each has the capacity of one order. Therefore the baking group can only work on 3 orders at the same time. It takes 40 minutes to bake the pastry.

e. The order of the working procedure for fulfilling one order is as following: dough-making group and filler-making group can work in parallel, however the assembling group has to wait until the two groups are done with an order to start
working. Only when the assembling group has finished assembling all the pastries in one order, can the baking group put them into oven. (See Fig1.)

![Fig1 Diagram](image)

f. Assume the owner periodically changes the recipe for the filling of the pastry. Every hour the owner changes the recipe and it takes the owner 10 minutes to finish the modification. The owner cannot modify the recipe, when someone in the filler-making group is making filler using the recipe book. Also, the fillermaking group will stop working when the recipe book is taken away by the owner to be modified.

Your Java program must use semaphores to ensure concurrent constraints and the order in the procedure specified in the above project specification.

The following files are provided. They are under /nfs/cs/classes/www/06W/cis415/program1 or you can read them at http://www.cs.uoregon.edu/classes/06W/cis415/program1/
DO NOT modify any of them.

Semaphore.java: It simulates the semaphores.
OSProcess.java: It simulates how operating systems manage the processes, which provide p() and v() for you to call.
Bakery.java: It is the GUI for the project. User inputs the number of orders and the progress bars on the GUI show the progressing of the computation.
FillingRecipe.java: It is the class for the recipe. Method “modify(long time)” is used by the owner to change the recipe.

To complete the project, you can modify the following file according to your design. You SHOULD NOT change the current content; JUST ADD NEW CODE. The current file is in the same folder as the previous ones.
WorkEnv.java: This class is a container of all the constants and the semaphores used by all the threads (not necessary all the semaphores for this project). The partially completed file contains the time constraints and also the instance of worker groups and the owner.

You need to implement the following:
WorkerGroup.java: It simulates the behavior of the different worker groups and the oven. One method make( ) in the class simulates the work and the different amount of
time taken by each group to do their work.

**Task.java:** This class simulates different steps in the work (making dough, filling, wrapping or baking the pastry in oven). It uses semaphore to enforce order of the tasks and to synchronize the tasks.

**PastryOrder.java:** It simulates a pastry order. As soon as the order is created, it starts four threads and each represents one task: making dough, making filling, assembling and baking.

**ShopOwner.java:** It simulates the behavior of the owner of the shop.

**Extra Credit:**

a. Add a label in the GUI class (Bakery.java), which shows the total running time for finishing the amount of orders entered.

b. Assuming each worker in the bakery can work on one task for one order at a time. The number of orders which one group except for the baking group can work on concurrently equals the number of workers in that group. For example, if 7 workers are assigned to the assembling group, then the assembling group can work on 7 orders at the same time. The other specification remains the same. Add three text fields in the GUI and change your programs accordingly, so the user can assign different number of workers (maximal number is 10) to different groups except for the baking group (since adding workers but not ovens to the baking group will not improve productivity).

c. In the survey sheet described below, answer the following additional question by using results from your project. Assume there are a total of 24 workers in the groups except for the baking group and there are still just 3 ovens. The owner does not modify the recipe and bother his workers at this time. What would be the best assignment in order to achieve the maximal productivity? You can use mathematics (the formula to maximize the rate flow of completed orders to the oven) to verify the results.

**Survey Sheet (hand-in in class to Prof. Lo)**

a. A program survey sheet will be available on the class website. On the survey, indicate which parts of your program are functioning correctly and the parts that are not finished. If any part does not work properly, state the reason in the comments section. You are responsible to fill in the form so that it truly reflects the quality of your work.

b. A short essay which should answer the following questions:
   1. The race conditions in this assignment and how you solved them. Show the initial value of the semaphores and p/v operation on each semaphore.
   2. Whether the synchronization problem in the project are similar to some classical synchronization problems. If so, what are those classical synchronization problems.

**Project submission (turn-in online by NOON on the due date)**

Turn in your implementation of the four classes, along with WorkEnv.java (source files). We will compile your classes along with the classes provided to test your project.
To submit a program:

1. Create a tar file of your program's files. Assuming you are in the directory where your files are stored, you can run the following command:

   `gtar -czvf pa1.tar.gz *`

2. Submit the compressed tar file:

   `/cs/classes/www/06W/cis415/turnin/turnin -p 8888 -s ix.cs.uoregon.edu pa1 pa1.tar.gz`

You should be able to go to [http://www.cs.uoregon.edu/classes/06W/cis415/turnin/](http://www.cs.uoregon.edu/classes/06W/cis415/turnin/) and see the names, sizes, and dates of the files you have submitted, but NOT their contents.