Assignment 8

Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor or teaching assistants).

Write the following Java programs and upload the files to Blackboard when the assignment is complete.

[40 points] Create a Java program that uses monitors to control access to a FoodBank object. You will create a FoodBank class that has a food amount and methods to give and take food. You will create two Thread classes for this program that will either put food into the food bank or take food from the food bank. Food cannot be taken if there is no food available to take. This is not a true producer/consumer problem. You only have one condition, which is to wait if there is no food available to take. Both giving and taking food must involve locking the FoodBank object and unlocking it when done and the FoodBank object must never enter an invalid state (a negative amount of food) The methods to take and give food must print out exactly what is happening with each action in the method, i.e. "Waiting to get food", "Taking 10 items of food, the balance is now 20 items".

A class FoodBank

FoodBank will have a single instance variable named food of type int. FoodBank will define a default constructor which initializes food to zero. FoodBank will have two methods: giveFood and takeFood. Both methods will have a single parameter of type int. giveFood will add the value of the parameter to the food instance variable, takeFood will subtract the value.

A class FoodProducer

FoodProducer will have a single instance variable named bank of type FoodBank. FoodProducer will have a parameterized constructor with a single parameter of type FoodBank. The parameterized constructor will initialize the value of bank to the single parameter. FoodProducer will extend the Thread class and override Thread's run method. FoodProducer's run method will loop infinitely. On each loop iteration run will generate a random number from 1-100 and add that much food to the bank instance variable. After adding food, the thread will sleep for 100 milliseconds.

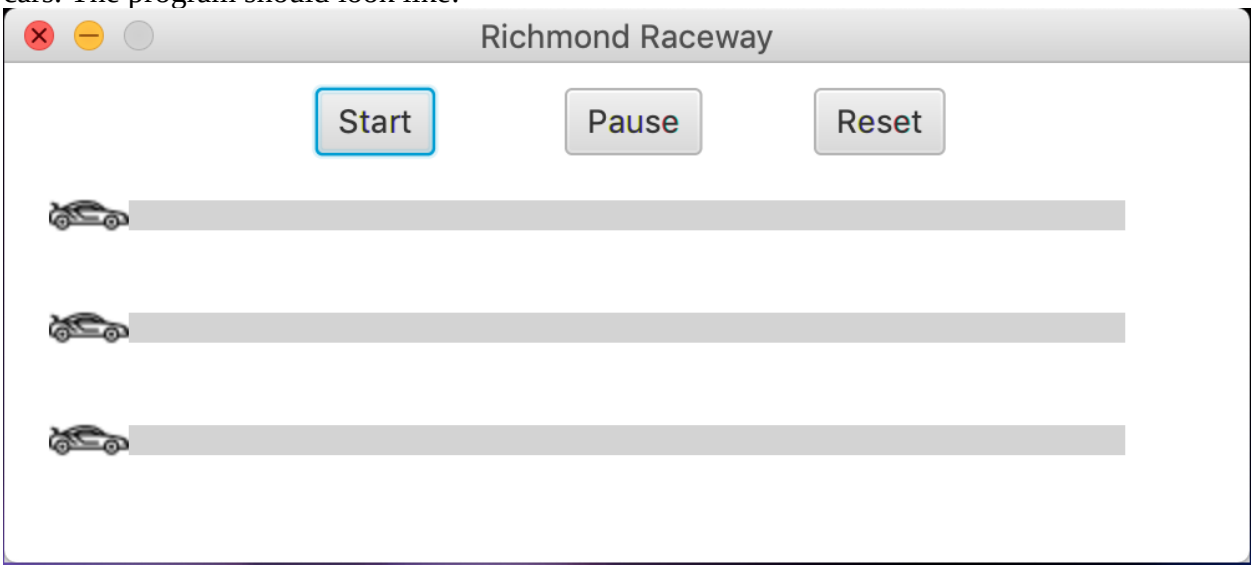
A class FoodConsumer

FoodConsumer is identical to FoodProducer except that the random number generated in run will be removed from the FoodBank object.

A class FoodBankPatrons

FoodBankPatrons will have a main method in which a FoodBank, FoodProcuder, and FoodConsumer object are created. The FoodProcuder and FoodConsumer must share the same FoodBank object. Once created, the main method starts these threads.

[60 points] Create a Java program which will use multiple threads to simulate a race between cars. The program should look like:



Create a class RaceTrack which uses JavaFX (subclasses the Application class) and overrides its start method to do the required drawing. Each car must be advanced by a separate Thread. On each thread iteration, each car should advance a random 0-10 pixels forward. Once advanced, a thread should sleep for 50 miliseconds. The threads will execute until a car reaches the end of the track. Once this occurs, an Alert should be spawned alerting the user of the winner and all cars stop. The primaryStage will be 500px by 200px and is not be resizable.

HINT: Updating the GUI from a non-GUI thread is considered bad practice as it leads to race conditions and UI errors. If you need to update a GUI component from a non-GUI thread, please use the runLater method defined in the Platform class. Don't be dissuaded by the "future time" a runnable object passed to runLater will be executed. We can expect that any runnable object passed to runLater will be executed quickly enough for the purposes of this project.

Grading Rubric for FoodBank:

Category	Points
Classes defined correctly	20
Correctly uses Java object monitors to enforce mutual exclusion	10
FoodBank state remains valid even with multiple Consumer threads active	10
Total	40

Grading Rubric for RaceTrack:

Category	Points
GUI Implemented correctly	15
Each car is run by a separate thread	15
Start button works as described	10
Winner is random and stops race from continuing	10
Pause button works as described	5
Reset button works as described	5
Total	60