CSE1322L Assignment 7 - Fall 2024

Introduction:

It is not uncommon for live events to feature some sort of merchandise store, granting event attendees the opportunity to get themselves some memorabilia from the event, and the event organizers an additional source of revenue. As goods are sold, the event organizers will want to keep track of not only how many items were sold, but also how much they were sold for.

In this assignment, we'll write a very simplified version of this system. All of our merchandise will be stored in a centralized location so we can more easily manage it (both physically and logically). However, we will have 3 different store fronts to keep the lines manageable. The idea is that whenever a store wishes to make a sale, it will pull out the first item available in the store and sell it to the customer. The stores will then update the event's ledger, keeping track of the revenue being generated and number of items sold.

This assignment makes use of Threads for parallelism and Stacks for data storage. Make sure to check the slides and videos on how those work.

Requirements

The features described below must be in your program.

- A total of 4 classes: Store, Merchandise, MerchandiseStorage, and the Driver.
- The Merchandise class has two public fields:
  - A string named merchandise
  - A Merchandise named next
    - Merchandise is supposed to work like a Node in a LinkedList. Check the course material to understand what this means.
- Merchandise only has an overloaded constructor which takes in a string, assigning it to the appropriate field, and sets next to null.
- The MerchandiseStorage class has a single field called "top", which is of type Merchandise.
  - This class should behave like a stack, with the "top" field being treated as the top of a stack.
- MerchandiseStorage has the following methods:
  - addMerchandise(): takes in a string and returns nothing. This method behaves like the push() method of a stack: It must create a Merchandise object with the string in the argument, link it to the current top, and then make the newly created object the top
  - retrieveMerchandise(): takes in nothing and returns a string. This method behaves like the pop() method of a stack, returning the current top's string after the current top has been replaced by its next.

        - If MerchandiseStorage is empty of Merchandise, it should return an empty string
- The Store class has the following fields:
    - A MerchandiseStorage field called “pile”
    - A static integer called “totalRevenue”, initialized at 0
    - A static integer called “itemsSold”, initialized at 0
    - A static integer called “nextId”, initialized at 1
    - An integer field called “id”
- Store has only an overloaded constructor, which takes in a MerchandiseStorage, assigning it to the appropriate field. It then assigns nextId to id and increments nextId by 1.
- Store has a getter for totalRevenue and itemsSold.
- Store has a method called run(), which takes in no arguments and returns nothing
    - <u>**JAVA ONLY: You must make Store a subclass of Thread for run() to be an override**</u>
    - run() retrieves a merchandise from “pile”
    - If the retrieved merchandise is an empty string, print “Store {id} is done selling” and then terminate the method
    - Otherwise, increase itemsSold by 1 and increment totalRevenue as follows:
        - “keychain”: +5
        - “t-shirt”: +30
        - “plush”: +50
    - <u>run() must keep retrieving merchandise from “pile” until it retrieves an empty string.</u>
- In your Driver, do the following:
    - Create a MerchandiseStorage object
    - Prompt the user for the number of keychains to add
    - In a loop, call addMerchandise() with “keychain” as an argument however many times the user entered above
        - For example, if the user enters they want to sell 50 keychains, you must call addMerchandise() with “keychain” as the argument 50 times.
    - Prompt the user for the number of t-shirts to add
    - In a loop, call addMerchandise() with “t-shirt” however many times the user entered above
    - Prompt the user for the number of plushies to add
    - In a loop, call addMerchandise() with “plush” however many times the user entered above
    - Create three threads, one for each Store.
    - Start all three threads
        - <u>Only start the threads AFTER you’ve created all three of them</u>
    - Wait for all 3 threads to finish working
        - Check the last 2 pages for details on how to do this
    - Print the following information:

"Total revenue: ${Store.totalRevenue}
Number of items sold: {Store.itemsSold}
The show was a success!"

Considerations

- Remember that you will get partial credit for partial work. Try to deliver as much of the assignment as you can.
- You may add any helper methods you believe are necessary, but you will not get points for them.
- While catching expected exceptions is good practice, there will be no rubric items checking for exception handling.
- The MerchandiseStorage class is probably the hardest in the assignment, as it needs to behave like a proper stack to work correctly.
    - Write the constructor, the push(), and pop() methods one at a time and test them individually. Use the debugger to help you troubleshoot their behavior.
- There will be no rubric items evaluating if any race conditions have been prevented.
- Check the last page of the assignment sheet for multi-thread implementation details in your particular language.
- **Due to the nature of parallelism, the stores may finish their work in a different order every time you run the program.**

```
Example 1: [User input in red]

[Store Management System]
How many keychains are being sold? 2000
How many t-shirts are being sold? 1000
How many plushies are being sold? 3000
Storage has been stocked. Press any key to start selling...

Store 3 is done selling.
Store 2 is done selling.
Store 1 is done selling.

Total revenue: $190000
Number of items sold: 6000
The show was a success!

Example 2: [User input in red]

[Store Management System]
How many keychains are being sold? 14500
How many t-shirts are being sold? 6500
How many plushies are being sold? 15000
Storage has been stocked. Press any key to start selling...

Store 1 is done selling.
Store 2 is done selling.
Store 3 is done selling.

Total revenue: $1017500
Number of items sold: 36000
The show was a success!
```

Submitting your answer:

Please follow the posted submission guidelines here:
https://ccse.kennesaw.edu/fye/submissionguidelines.php

Ensure you submit before the deadline listed on the lab schedule for CSE1322L here:
https://ccse.kennesaw.edu/fye/courseschedules.php

Java Threads:

Defining Thread objects: You will be required to either have Store be a subclass of Thread or to implement the Runnable interface.

Defining Thread behavior: You will need to override the run() method, which is inherited from Thread. The code which you wish to be run in parallel with other threads must be added in run(). As such, most of your Store logic will be inside of run().

Creating Thread objects: Assuming you extend the Thread class, you would create a Store object like any other:

- Store s = new Store();

Starting Threads: Once you instantiate your thread object, you can tell the computer to start executing it by calling start():

- s.start();

If you have several threads you wish to start, they need to be started individually:

```
Store s1 = new Store ();
Store s2 = new Store ();
Store s3 = new Store ();
s1.start();
s2.start();
s3.start();
```

Waiting on a thread to finish: You can call the join() method to wait for a thread to finish executing before you continue to execute your current thread. Assuming we are inside of main():

```
Store s1 = new Store();     //Creates a new store
s1.start();                         //Starts the store thread
s1.join();                          //main() will wait until s1 is done executing
                                            //before continuing
```

<u>Notice that, since main() is also a thread which will be running in parallel with your Stores, if you do not instruct main() to wait for **ALL** the Stores to finish executing, main() might terminate before all the merchandise has been sold.</u>

C# Threads:

Defining Thread behavior: You will create a method called run(). The code which you wish to be run in parallel with other threads must be inside this method. As such, most of your Store logic will be inside of run(). Keep in mind that, in theory, you can call this method anything you like, but <u>the rubric will be specifically be looking for a method called run()</u>. Make sure all of your parallel processing logic is in it.

Creating Thread objects: To create a thread object, you must pass to its constructor the name of the method you wish to run in a thread. Notice that you are only passing the name of the method, but no parameters (not even parenthesis):

```
Store s = new Store();
Thread t = new Thread(s.run);        //Notice the lack of parenthesis when passing run()
```

Starting Threads: Once you instantiate your thread object, you can tell the computer to start executing it by calling Start():

- t.Start();

If you have several threads you wish to start, they need to be started individually:

```
Thread t1 = new Thread(s1.run);
Thread t2 = new Thread(s2.run);
Thread t3 = new Thread(s3.run);
t1.Start();
t2.Start();
t3.Start();
```

Waiting on a thread to finish: You can call the Join() method to wait for a thread to finish executing before you continue to execute your current thread. Assuming we are inside of Main():

```
Store s1 = new Store();              //Creates a new Store
Thread t = new Thread(s1.run);       //Creates a new thread
t.Start();                           //Starts the thread
t.Join();                            //main() will wait until the thread is done executing
                                             //before continuing
```

<u>Notice that, since Main() is also a thread which will be running in parallel with your Stores, if you do not instruct Main() to wait for **ALL** the Stores to finish executing, main() might terminate before all the merchandise has been sold.</u>