[**Department of Computer Science**](http://www.cs.umd.edu/)

[**CMSC132:**](http://www.cs.umd.edu/class/fall2016/cmsc132-04XX/) Fall 2016

**Project:** Orders Processor (Concurrency)

**Due Date:** Tuesday Nov 29, 11:00pm

## Overview

For this project you will implement a program that processes files that represent purchase orders. The program can complete the processing by using a single thread or multiple threads.

For this project there is no CVS repository distribution. The code distribution can be found at [OrdersProcessor.zip](http://docs.google.com/OrdersProcessor.zip). Import the project to Eclipse as described in  [Importing Project](http://www.cs.umd.edu/eclipse/other.html#import).

## Objectives

Practice the design and implementation of concurrent systems.

## Grading

* (38%) Public Tests
* (32%) Secret Tests
* (10%) Sychronization requirement (avoiding data races)
* (10%) Style (avoiding code duplication, indentation, good variable names)
* (10%) Report

## Clarifications

## Code Distribution

The project's code distribution has the following files/folders:

* example1.txt → A sample order.
* example2.txt → A sample order.
* example3.txt → A sample order.
* itemsData.txt → Information about the items that can be purchased.
* resultsExample.txt → Results of processing the above example1.txt, example2.txt, and example3.txt orders.
* processor package → This is where all the files associated with your implementation will reside.
* report package → This is where the Report.doc file can be found. Remember to refresh your project after editing this file (or overwriting it).

## Specifications

### Processing of Orders (Files)

Your program will process a set of files (e.g., [example1.txt](http://docs.google.com/sampleFiles/example1.txt)) each representing a purchase order. Each file lists the items bought and the date of purchase. The possible items that can be purchased (along with the item's price) can be found in a item's data file (e.g., [itemsData.txt](http://docs.google.com/sampleFiles/itemsData.txt)). The program you need to write will generate a summary for each order (file). The summary includes the client id and a **sorted list (by item's name)** of each item bought. The list will include the item's name, the cost per item, the quantity of items bought, and the total cost associated with the item's purchase. After the sorted list, an order's total will be displayed. See the [resultsExample.txt](http://docs.google.com/sampleFiles/resultsExample.txt) file for an example of the data format.

In addition to a report for each order, the program will generate a summary of all orders. The summary will display a sorted list (by item's name) providing information about the total number of items sold, and total revenue (see [resultsExample.txt](http://docs.google.com/sampleFiles/resultsExample.txt)).

### Threaded Processing

Your program will allow users to process all the orders using a single thread or one thread per order (file). For simplicity, all the orders will use the same base filename (e.g., **example** in the files above). The user will provide a filename for the results.

In order to see the advantages of threading, your program needs to print (to standard output) the time (in msec) it took to process orders. You can compute the time as follows:

long startTime = System.currentTimeMillis();  
/\* TASK YOU WANT TO TIME \*/  
long endTime = System.currentTimeMillis();  
System.out.println("Processing time (msec): " + (endTime - startTime));

### Driver

Your are free to define any number of classes/interfaces you understand you need, however, you need to provide a class called **OrdersProcessor** in the processor package. This class can have as many methods as you want, but it must have a main method that allow us to configure/run the processing of orders.

Your program will ask users how to configure a particular processing of orders by using standard input and output. The following represents a run of the OrdersProcessor main method that computed the results you see in [resultsExample.txt](http://docs.google.com/sampleFiles/resultsExample.txt) (code in italics represents user's input).

Enter item's data file name: ***itemsData.txt***   
Enter 'y' for multiple threads, any other character otherwise: ***y***  
Enter number of orders to process: ***3***  
Enter order's base filename: ***example***  
Enter result's filename: ***resultsExample.txt***  
Reading order for client with id: 1003  
Reading order for client with id: 1001  
Reading order for client with id: 1002  
Processing time (msec): 51  
Results can be found in the file: resultsExample.txt

### Report

Using the [data.zip](http://docs.google.com/data.zip) file, create a table that illustrates the time it takes to process data using a single thread and multiple threads. For example, the table can have three columns where the first indicates the number of orders, the second the time it took using a single thread, and the third the time is took using multiple threads. Provide a brief explanation (no more than a paragraph) of your results. Put your table and explanation in a file named **Report.doc** you will find in the report package.

## Requirements

* You need to define a map/ArrayList that is shared and modified by all threads. How you use the map is up to you.
* Each order has an item name, followed by the date (e.g. 6/13) the item was ordered. That date is not used in your program.
* For the multiple threads option, one thread will be created for each order (file) to be processed.
* You will start the timing process immediately after reading the name of the results file and will stop the timing process before printing the message indicating where results can be found. Make sure you close the file output stream for the results file before you stop the timing process.
* Your program will print (to standard output, NOT to the results file) the message "Reading order for client with id: " followed by the client id, before reading a particular order's file.
* Notice that the reports appear sorted by client id, but the processing of each order (when using multiple threads) is up to the Java scheduler.
* The summary for all the orders only includes items that have been bought.
* You may not use classes that are automatically synchronized (e.g., Vector).
* Synchronizing individual methods is fine (in this case you many not need to use an explicit lock object), however, be careful as you don't want to limit concurrency while trying to avoid data races. Instead of defining a method as synchronized, you should surround the critical section with a lock.
* What we are expecting for the synchronization part of this project, is that any object (e.g., map, set, whatever you use) that is being shared is not associated with a data race.
* **IMPORTANT:** Before uploading your project to the submit server, remove all large data order files (.txt). Usually these are files you utilized for performance experiments. If you have student tests that rely on .txt files, keep those text files small.
* Feel free to define as many classes and/or interfaces you understand are necessary.
* Your program must be able to process any number of orders (not just 3).
* Do not use static variables; do not use static objects. We want to avoid the JUnit problems discussed in class. Notice that using static methods is OK.
* You are not required to write student tests for this project, but you are strongly encourage to do so. Notice that you can write student tests where main is executed (by calling OrdersProcessor.main(null);) after you have redirected input. The file [TestingSupport.java](http://www.cs.umd.edu/class/summer2014/cmsc132/resources/TestingSupport.java) has tools for writing tests (including input/output redirection). Notice you don't need to use these tools.
* Your code must be efficient and you must avoid code duplication.
* Notice we plan to modify the contents of the item's data file while grading your program. That means we plan to use items other than the ones you see in the sample file.
* You can assume all order files will be placed in the same folder where example1.txt, example2.txt, etc. reside.
* The report for each client must be sorted by client id.
* You can assume that the specified output file (e.g., [resultsExample.txt](http://docs.google.com/sampleFiles/resultsExample.txt)) will be overwritten (no appending of results) if the same name is used several times.
* We will use different names for output files (not just resultsExample.txt).
* There could be items in the items' data file that no one buys.
* You can assume orders will only have items present in the items' data file (no invalid data).
* The file names will always start at 1 up to including the number of orders.
* Remember that we are planning to use base file names different than order or example. For example, we can provide files names myNewOrder1.txt, myNewOrder2.txt, etc.
* To compare files in Eclipse, you can right-click on a file and select "Compare With".
* The performance of your program using multiple threads should be better than using a single thread.
* Remember to follow exactly (except for spaces) the format associated with the file [resultsExample.txt](http://docs.google.com/sampleFiles/resultsExample.txt).
* To print currency use **NumberFormat.getCurrencyInstance().format(AMOUNT\_HERE)**.
* Remember to refresh your project after editing/copying the Report.doc file.
* See [Style Guidelines](http://www.cs.umd.edu/class/fall2016/cmsc132-04XX/content/resources/StyleGuidelines.html) for information regarding style.

## Submission

To submit your project, you will upload a zip file with your project to the submit server as follows:

* Refresh your Eclipse project (right-click "Refresh").
* Export your project as described in [Exporting Project](http://www.cs.umd.edu/eclipse/other.html#export).
* Upload your zip file to the submit server.
* Verify (by downloading from the submit server) that your submission is correct.

## Academic Integrity

Please make sure you read the academic integrity section of the syllabus so you understand what is permissible in our programming projects. We want to remind you that we check your project against other students' projects and any case of academic dishonesty will be referred to the [Office of Student Conduct](http://www.jpo.umd.edu/).