CSCD 467/567 Lab2

Signaling with Thread Interruption in Java

**Rules:** Your code must use Java Language. If your program shows a compilation error, you get a zero for this lab assignment. To avoid compatibility issues, I encourage you to upgrade your JRE to latest version of SE 1.7 or 1.8.

**Submission:** Wrap up all your java files and a ReadMe text file into a single zip file. Name your zip file as *FirstInitialYourLastName*CSCD467Lab2.zip. For example, if your legal name is Will Smith, you should name your zip file as wSmithCSCD467Lab2.zip.

You are required to submit the ReadMe text file along with all your java code. In the ReadMe file you should put your legal full name, description about how to compile and how to run your program on COMMAND LINE. An example of ReadMe file should look like the following:

(This only serve as an Example. Your ReadMe file should contain the similar content.)

Name: Will Smith

Description: unzip the submitted wSmithCSCD467Lab2.zip, you get a folder named smithLab2.

To Compile: cd into folder smithLab2,

javac \*.java

To Run

java myLab2 7 9

**Before you leave the laboratory, please show the TA or the instructor how your program works, they will give you a score for this Lab assignment.**

**For archive purpose, please also submit your single zip file on EWU Canvas by following CSCD467-01 Course 🡪Assignments🡪Lab2🡪 Submit Assignment to upload your single zip file.**

**Problem Description:**

Recall that interrupt() method in a Thread class may not necessarily used to terminate a program. Instead, we could use it as a signal that one thread sends to another. In that case, we have to provide our own code to respond to the interruption, if the thread being interrupted does not frequently call methods that throws InterruptedException. That is, we have to check whether the interrupt status is set or not, if so, we provide handler code to respond to the interrupt. Today we write an application that creates **two** separate Java Threads in the main thread. You are required to implement these features. (**GUI is not required in this lab**.)

1, You have to create two threads in the main program, named as the Waiter and the Printer. Then we start both threads one after another (without other statement between two start() method call).

2, The Printer thread is designed to print out an arbitrary message to the standard output for 50 times in a loop.

3, The Waiter thread will print out a message “Printer has already got his work half done!” **only** **after** Printer has finished printing its 25th message. In other words, the Waiter thread is okay to print out the message “Printer has already got his work half done!” after Printer’s 25th, or 26th message or 28th message, or 40th message is printed out. Before the Printer finishes its 25th message outputs, the Waiter thread cannot output any message. Before Waiter terminates, it outputs another message “Waiter has done all its work, terminating.”

4, Only after both the Printer finishes its 50th message then terminates and the Waiter terminates, the main thread output a message “Both Waiter and Printer have finished their work!”.

5, Please do not use the mutex lock and wait() method calls in this lab, which we have not formally learned.

Figure 1 shows one of the correct output sequences for Printer, Waiter and main Thread.

Message i = 1, from Thread Printer

Message i = 2, from Thread Printer

Message i = 3, from Thread Printer

Message i = 4, from Thread Printer

Message i = 5, from Thread Printer

Message i = 6, from Thread Printer

Message i = 7, from Thread Printer

Message i = 8, from Thread Printer

Message i = 9, from Thread Printer

Message i = 10, from Thread Printer

Message i = 11, from Thread Printer

Message i = 12, from Thread Printer

Message i = 13, from Thread Printer

Message i = 14, from Thread Printer

Message i = 15, from Thread Printer

Message i = 16, from Thread Printer

Message i = 17, from Thread Printer

Message i = 18, from Thread Printer

Message i = 19, from Thread Printer

Message i = 20, from Thread Printer

Message i = 21, from Thread Printer

Message i = 22, from Thread Printer

Message i = 23, from Thread Printer

Message i = 24, from Thread Printer

Message i = 25, from Thread Printer

Message i = 26, from Thread Printer

Printer got his work half done!

Message i = 27, from Thread Printer

Waiter has done its work, terminating.

Message i = 28, from Thread Printer

Message i = 29, from Thread Printer

Message i = 30, from Thread Printer

Message i = 31, from Thread Printer

Message i = 32, from Thread Printer

Message i = 33, from Thread Printer

Message i = 34, from Thread Printer

Message i = 35, from Thread Printer

Message i = 36, from Thread Printer

Message i = 37, from Thread Printer

Message i = 38, from Thread Printer

Message i = 39, from Thread Printer

Message i = 40, from Thread Printer

Message i = 41, from Thread Printer

Message i = 42, from Thread Printer

Message i = 43, from Thread Printer

Message i = 44, from Thread Printer

Message i = 45, from Thread Printer

Message i = 46, from Thread Printer

Message i = 47, from Thread Printer

Message i = 48, from Thread Printer

Message i = 49, from Thread Printer

Message i = 50, from Thread Printer

Both Waiter and Printer have finished their work!