2025.1 Multicore Computing, Project #2

(Due : 11:59pm, May 11)

Submission Rule

1. Create a directory "proj2".
2. In the directory "proj2", create sub-directories "prob1", "prob2", "prob3", and "prob4".
2-1. Insert into "prob1" directory: (i)JAVA source code "ParkingBlockingQueue.java", (ii) a PDF or TXT document file that contains two examples of execution results (output)
2-2. Insert into "prob2" directory: (i) JAVA source code "ParkingSemaphore.java", (ii) a PDF or TXT document file that contains two examples of execution results (output)
2-3. insert into "prob3" directory: (i) a document (report) PDF file that contains your explanations and source codes for problem3, (ii) source files (ex1.java, ex2.java, ex3.java, ex4.java, ex5.java)
2-4. insert into "prob4" directory: (i) a demo video file (.mp4 format) that shows compilation and execution of your source files including execution results (ParkingBlockingQueue.java, ParkingSemaphore.java, ex1.java, ex2.java, ex3.java, ex4.java, ex5.java). The size of the demo video file should be less than 50MB.
3. zip the directory "proj2" into "proj2.zip" and submit the zip file into eClass homework board.

[Problem 1] We studied a garage parking problem in lecture note 4-1. The JAVA source code for the problem, **ParkingGarageOperation.java**, which is available in our class webpage "project 2" announcement, uses **wait()/notify()** to implement the garage parking problem.

[What you need to do for this project]: Write a JAVA code generating results that are equivalent (i.e. similar) to the results of the original JAVA code **ParkingGarageOperation.java** using **ArrayBlockingQueue** and **BlockingQueue** in **java.util.concurrent** package instead of using **wait()/notify()**. This means you should not use **wait()/notify()** functions in your JAVA program. You may start from **ParkingGarageOperation.java** and modify it. See the **ArrayBlockingQueueExample.java** (available on our class webpage) that can be helpful for doing this project. Please assume that the number of free parking places is 7 and the number of cars is 10.

In the results, you should print "Car #: trying to enter" just before entering and print "Car #: entered" just after actual entering in order to show whether the entering car is waiting for empty place or not. Do the similar processing for leaving cars. Please see the output example below. Your program output should be similar to the output example. The name of the JAVA code you create should be **ParkingBlockingQueue.java**.

Output example :

```
==========================================
$ java ParkingBlockingQueue
Car 4: trying to enter
Car 4: just entered
Car 10: trying to enter
Car 10: just entered
Car 8: trying to enter
Car 8: just entered
Car 7: trying to enter
Car 7: just entered
Car 7:                          about to leave
Car 7:                          have been left
Car 5: trying to enter
Car 5: just entered
Car 6: trying to enter
Car 6: just entered
Car 3: trying to enter
Car 3: just entered
Car 9: trying to enter
Car 9: just entered
Car 4:                          about to leave
Car 4:                          have been left
```

```
Car 1: trying to enter
Car 1: just entered
Car 2: trying to enter
Car 7: trying to enter
Car 5:                                  about to leave
Car 5:                                  have been left
Car 2: just entered
Car 8:                                  about to leave
Car 8:                                  have been left
Car 7: just entered
Car 4: trying to enter
Car 6:                                  about to leave
Car 6:                                  have been left
Car 4: just entered
Car 1:                                  about to leave
Car 1:                                  have been left
```

[Problem 2] Modify the original JAVA code **`ParkingGarageOperation.java`** (available in our class webpage project announcement) using **`Semaphore`** class in **`java.util.concurrent`** package such that it generate the results that are equivalent to the results of the original JAVA code **`ParkingGarageOperation.java`**. This is similar to [problem 1], but the difference is that you need to do modification using **`Semaphore`** (i.e. counting semaphore class) instead of using **`ArrayBlockingQueue`**. The name of the JAVA code you create should be **`ParkingSemaphore.java`**.

[Problem 3] Visit the website http://tutorials.jenkov.com/java-util-concurrent/index.html that introduces and describes JAVA concurrency utilities: **`java.util.concurrent`** package.

3-1. You are supposed to study and summarize following classes that are important and useful for concurrent programming in JAVA. What you need to do in this problem is to write and submit a document (report) in PDF file format that includes:
(i)-a. Explain the interface/class **`BlockingQueue`** and **`ArrayBlockingQueue`** with your own English sentences. (DO NOT copy&paste)
(i)-b. Create and include (in your document) your own example of multithreaded JAVA code (**`ex1.java`**) that is simple and executable. (DO NOT copy&paste) Your example code should use **`put()`** and **`take()`** methods. Also, include example execution results (i.e. output) in your document.
(ii)-a. Do the things similar to (i)-a for the class **`ReadWriteLock`**.
(ii)-b. Do the things similar to (i)-b for **`lock()`**, **`unlock()`**, **`readLock()`** and **`writeLock()`** of **`ReadWriteLock`**. (**`ex2.java`**)
(iii)-a. Do the things similar to (i)-a for the class **`AtomicInteger`**.
(iii)-b. Do the things similar to (i)-b for **`get()`**, **`set()`**, **`getAndAdd()`**, and **`addAndGet()`** methods of **`AtomicInteger`**. (**`ex3.java`**)
(iv)-a. Do the things similar to (i)-a for the class **`CyclicBarrier`**.
(iv)-b. Do the things similar to (i)-b for **`await()`** methods of **`CyclicBarrier`**. (**`ex4.java`**)
(v)-a. Do the things similar to (i)-a for **`ExecutorService`**, **`Executors`**, **`Callable`**, **`Future`**.
(v)-b. Do the things similar to (i)-b
for **`newFixedThreadPool()`**, **`submit()`**, **`call()`** methods (**`ex5.java`**) . I recommend to implement the program that computes the number of prime numbers between 1 and 200000 using the thread pool as your own example **`ex5.java`**.

3-2. Submit the source files **`ex1.java`**, **`ex2.java`**, **`ex3.java`**, **`ex4.java`**, **`ex5.java`** as well as the document described above.

[Problem 4] Create a demo video file (.mp4 format) that shows compilation and execution of your source files (`ParkingBlockingQueue.java`, `ParkingSemaphore.java`, `ex1.java`, `ex2.java`, `ex3.java`, `ex4.java`, `ex5.java`). The size of the demo video file should be less than 50MB. [An audio explanation in your demo video would be appreciated, but it is not required.]