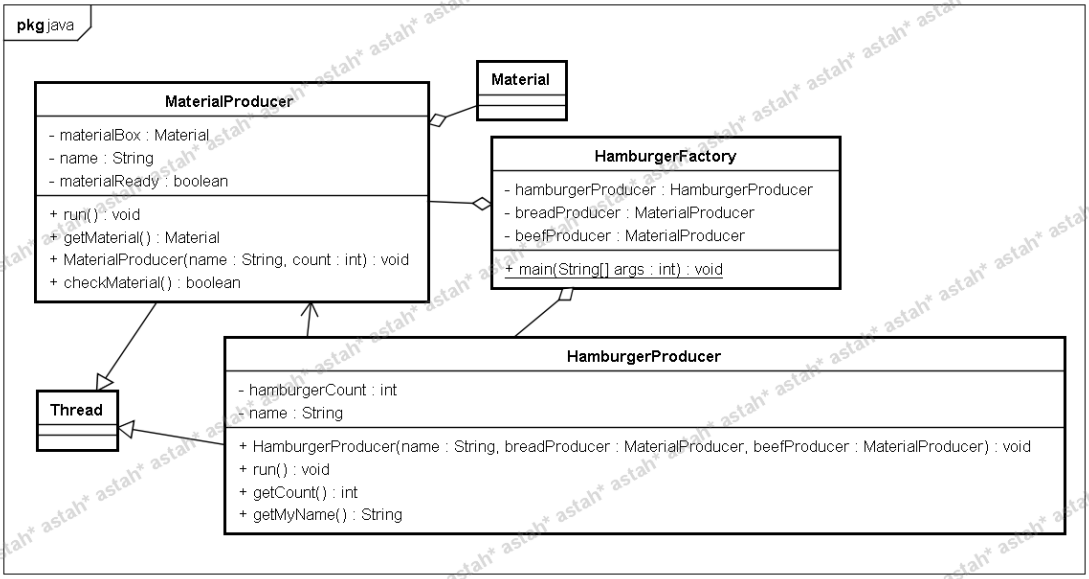
Java Mini Lab 2 Thread and Synchronization

- 作業簡介

請用 Java Thread 模擬一個漢堡工廠的作業流程，其中必須正確地使用 Java 提供的 Synchronization 機制確保程式正確性。

- 工廠介紹與 Class Diagram

HamburgerFactory 中有兩個 MaterialProducer，分別負責烤麵包(breadProducer)與煎牛肉(beefProducer)，而 HamburgerProducer 會分別向 breadProducer 和 beefProducer 索取麵包和牛肉之後，疊起來做成一個好吃的漢堡。



- 注意事項

1. 程式中只會有一個 breadProducer thread 與一個 beefProducer thread，但可能有數個 HamburgerProducer thread
2. breadProducer 與 beefProducer 建構時必需設定預計生產的數量，bread 與 beef 的生產數量一定是相等的
3. hamburgerProducer 並不知道自己該做幾個漢堡才能下班，必須透過 main function 呼叫 hamburgerProducer.interrupt()，他們才知道已經可以結束了

- 作業要求

1. 修改範例程式中的 MaterialProducer.run()與 HamburgerProducer.run()，使得程式在超過 2 個 hamburgerProducer thread 時仍然可以正確執行
2. 不可修改 MaterialProducer.run()與 HamburgerProducer.run()以外的地方
3. 程式必須可以正常結束，同時製作的漢堡總數要跟生產的麵包或牛肉總數相同

- 作業繳交

請將 eclipse 整個 project directory 壓縮成一個檔案，寄到下面這個信箱

CS340100@gmail.com

Deadline: 2010/5/28 23:59pm

- 作業提示

1. MaterialProducer 產生 material 以及 HamburgerProducer 取得 material 時都需要對 MaterialProducer 中的 materialBox member 作存取，必須用一個 lock 保證這兩件事不會交錯進行。用 synchronized(object){}可確保區塊內的程式碼不會跟其他同樣對 object 作 synchronize 的區塊交錯執行。
2. MaterialProducer 取得 lock 後，卻發現 materialBox 內仍有原料，可用 lock.wait()暫時讓出 lock，確定 materialBox 內已空之後再生產原料放入 materialBox，並且呼叫 lock.notifyAll()讓其他呼叫 wait()的人可以繼續執行。
3. HamburgerProducer 取得 lock 後，卻發現 materialBox 內沒有原料，可用 lock.wait()暫時讓出 lock，確定 materialBox 內有原料再叫 getMaterial()，並且呼叫 lock.notifyAll()讓其他呼叫 wait()的人可以繼續執行。
4. 執行一次沒問題不一定程式就沒問題，Thread synchronize 有錯的話，不一定每次執行都會錯。