TD 1 - Java concurrency: synchronizers

To set up for the practical exercises go to CELENE and download `code.zip`. Create a Java project in your favorite IDE (e.g., Eclipse, Netbeans) and import the contents of `code.zip` into your project. You should now have 6 packages, one per exercise, called `polytech.tours.di.parallel.td1.exo#` where `#` is the number of the exercise. You're now good to go.

1 Thread interference

The objective of this first exercise is to see thread interference in action. Study the three classes in package `polytech.tours.di.parallel.td1.exo1`, namely, `Counter`, `ParrallelCounting`, and `Tester`. Run the `main()` method of class `Tester` several times. Do you observe any abnormal behavior? if so, can you explain it?

2 Synchronize methods and sections

Refactor the code in package `polytech.tours.di.parallel.td1.exo2` so the thread interference is avoided.

Hint: remember the synchronize methods and synchronize sections we discussed in class.

3 Explicit locks

We saw in class that `java.util.concurrent.ReentrantLock` provides a ready-to-go implementation of a re-entrant lock for thread synchronization. For learning purposes we will reinvent the wheel and code our own implementation (but please in real applications use the one provided by Java).

Code a class called `polytech.tours.di.parallel.td1.exo3.Lock` implementing two methods `lock()` and `unlock()`. The `lock()` method locks the `Lock` instance so that all threads calling `lock()` are blocked until `unlock()` is executed.

Hint: remember the `wait()` and `notify()` methods we studied in class.

To test your `Lock`, refactor class `polytech.tours.di.parallel.td1.exo3.Counter` so it uses an instance of your `Lock` to prevent memory inconsistency errors and thread interference. You can use `polytech.tours.di.parallel.td1.exo3.Tester` to conduct the experiments.

Hint: we saw an example of these guarded blocks in class.

4 Is your `Lock` re-entrant?

Study class the `polytech.tours.di.parallel.td1.exo4.ReentrantTask`. Implement a class called `polytech.tours.di.parallel.td1.exo4.Tester` with a `main()` method that launches the execution of an instance of `ReentrantTask` in a `Thread`. What happens? why?

5 Making our `Lock` re-entrant

Implement a class called `polytech.tours.di.parallel.td1.exo5.ReentrantLock` that solves the reentrance problem. Use `polytech.tours.di.parallel.td1.exo5.Tester` to test your solution.

Hint: remember that a thread may try to obtain the same lock more than twice.
Hint: remember than a thread executing a task is nothing but an instance of class `Thread`.

6 A short case study

You are designing a parallel application that computes the credit score of a large set of costumers based on their most recent payments. The payment information of the customers is stored in a database. You access that database through class `polytech.tours.di.parallel.td1.exo5.DBConnection`. Note that `DBConnection` implements the Singleton design pattern to forbid constructing multiple instances of the class. Class `DBConnection` provides two main methods: `getCustomerRecords()` and `setCustomerScore()`. As you may guess, the former queries the database to retrieve the payment records of a customer, while the latter updates the credit score of the customer on the database. As mentioned earlier, the credit score is computed based on the customer's n most recent payments. When you query the database to get the customer records, you obtain an array of booleans where a true value represents a timely payment and a false value represents a delayed payment. The score is nothing but the probability that the customer's next payment is on-time.

Since the number of customers may be very large, the software architect suggested that you define a task as the score computation for one customer and that you concurrently run the tasks on different threads. To make your life easier, the architect provided you with a template of class `ScoreComputation` that you need to complete in order to implement the logic the parallel task. The only problem with this design, is that there is a limit on the number of database transactions that you can simultaneously execute. You have to come up with a strategy to guarantee that this constraint is satisfied and implement methods `getCustomerRecords()` and `setCustomerScore()` accordingly. You can use class `polytech.tours.di.parallel.td1.exo5.Tester` to test your code.