Thread Pool Test
==========

Implement a generic resource pool (i.e. a pool of instances of type R where R is a generic type). The pool allows resources to be acquired (checked out of the pool) and then later released (checked back into the pool). New resources may be added to the pool, and existing resources may be removed from the pool. **The resource pool must be thread-safe.** This means that multiple threads must be able to acquire, release, add or remove resources simultaneously. Two threads should not be able to acquire the same resource simultaneously. Use a defensive strategy (as opposed to design by contract) when implementing your resource pool as you do not know how your pool will be used or if it will be used correctly.

The resource pool must implement at least the following methods:

- void open()
- boolean isOpen()
- void close()
- R acquire()
- R acquire(long timeout,
  java.util.concurrent.TimeUnit timeUnit)
- void release(R resource)
- boolean add(R resource)
- boolean remove(R resource)

These methods may throw any exceptions you determine to be appropriate. The following rules govern the behavior of these methods:

- The pool shall not allow any resource to be acquired unless the pool is open.
- Resources can be released at any time.
- Resources can be added or removed at any time.
- The acquire() method should block until a resource is available.
- If a resource cannot be acquired within the timeout interval specified in the acquire(long, TimeUnit) method, either a null can be returned or an exception can be thrown.
- The add(R) and remove(R) methods return true if the pool was modified as a result of the method call or false if the pool was not modified.
- The pool main use is going to be via acquire and release calls, add and remove calls do not happen that often.
- The *remove(R)* method should be such that if the resource

that is being removed is currently in use, the remove operation will block until that resource has been released.
- Add a new method, *boolean removeNow(R resource)* that removes the given resource immediately without waiting for it to be released. It returns true if the call resulted in the pool being modified and false otherwise.
- The *close()* method should be such that it blocks until all acquired resources are released.
- Add a new method, *void closeNow()*, which closes the pool immediately without waiting for all acquired resources to be released.

Please prepare notes on why you made design decisions that you did, and how you would improve it, given more time. Please describe how you know it is thread safe, and how you would test it to make sure (implement a test if you have time). Please submit your implementation in the form of one or more .java files so that we can make reference to particular line numbers in our evaluation.

The coding test should probably represent an effort of 4-5 hours.
For the candidate's reference, code will be evaluated as follows:

1) It works according to specification. We will, most probably, run some tests against it, read and come up with regular and edge case scenarios. The correctness of the solution will obviously have a high weight on the evaluation. We will look in particular to potential synchronization issues.
2) Overall solution approach (use of appropriate libraries, java constructs, complexity of solution, performance of solution).
3) Coding style, cleanliness of delivery.
4) Design decisions explanations.