# Data Structures Exam Retake: ThreadExecutor

## Description

Mr. Man is making a multi-threaded application for a client and has too much on his head. He politely asked you to offload some of his work to you, so he can focus more on specific tasks. You have your own problems, but because you can’t refuse to people you accepted. Now you have to live up to his standards and write a “state of the art” code. You are well aware that writing a good concurrent or non-blocking code takes time, in-depth knowledge and a lot of effort. Your task is to implement a “thread pool” type data structure. This data structure consists of ‘n’ threads that execute ‘Tasks’ concurrently.

* Execute(Task) – Add the new manufactured Product in stock. You will need to implement the Count property as well. If the task id already exists, throw ArgumentException/**IllegalArgumentException**
* Count – Returns the number of tasks currently in the pool
* Contains(Task) – Checks whether the passed argument exists in the pool.
* GetById(int) – Returns the task with the particular id from the pool. \*Keep in mind that the task id is always unique. If such task is not present in the pool, throw ArgumentException/**IllegalArgumentException**
* GetByIndex(int) – Returns the **N**-th task that was added to the pool of threads. The index is based on insertion order in the data structure and starts from 0. If the given index is not present in the pool, throw ArgumentOutOfRangeException in C# or IndexOutOfBoundsException in java
* Cycle(int) – Returns the number of tasks completed with the current cycle. When cycle is called the threads inside the pool start operating and execute the tasks. Each task has consumption count. Every time cycle is called, the consumption of each task is reduced and when it reaches zero or negative you must remove it from the thread pool**,** because it is considered completed.

Throws ArgumentException/**IllegalArgumentException** if the pool is empty.

In example: The pool has 5 tasks in it and each task has 5 consumption. After Cycle(3) is called nothing happens. If Cycle with integer larger than 1 is called you must remove each task from the pool (completed).

* GetByConsumptionRange(int,int,bool) – Returns all tasks within given **consumption** range with inclusiveness as optional parameter. They should be ordered **by consumption and then by priority descending**. Keep in mind that **Cycle calls count**. If no tasks are found return an **empty collection.**
* GetByPriority(Priority) – Returns all tasks in the pool with given **priority** ordered by id descending or an **empty** collection if none were found
* ChangePriority(int, Priority) – Changes the priority of the task with the given id. If such id doesn’t exist, throw an ArgumentException/**IllegalArgumentException.** GetByPriorityAndMinimumConsumption(Priority, int) – Returns all the tasks in the pool which have the given priority and their consumption is larger or equal to the passed one. Order them by id descending. If none were found, return an empty collection.
* GetEnumerator<Task>() – Returns all tasks in the pool ordered by insertion.

Duplicate references of the task class **are not allowed**. The id is always unique and the consumption of multiple objects might be the same **(It is not acceptable for the consumption to be 0).**

## Input / Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **IntelliJ Java project** holding the interface IScheduler, the unfinished classes ThreadExecutor and Task. **Tests** covering the ThreadExecutor **functionality** and **performance**.

Your task is to **finish this class** to make the tests run correctly.

* You are **not allowed to change the tests**.
* You are **not allowed to change the interface**.
* You can add to the Task class, but don't remove anything.
* You can edit the ThreadExecutor class if it implements the IScheduler interface.

## Interface

The interface IScheduler in C# looks like the code below:

|  |
| --- |
| public interface IScheduler : IEnumerable<Task>  {  int Count { get; }  void Execute(Task task);  bool Contains(Task task);  Task GetById(int id);  Task GetByIndex(int index);  int Cycle(int cycles);  void ChangePriority(int id, Priority newPriority);    IEnumerable<Task> GetByConsumptionRange(int lo, int hi, bool inclusive);  IEnumerable<Task> GetByPriority(Priority type);  IEnumerable<Task> GetByPriorityAndMinimumConsumption(Priority priority, int lo);  } |

The interface Scheduler in Java looks like the code below:

|  |
| --- |
| public interface Scheduler {  int getCount();  void execute();  boolean contains(Task t);  Task getById(int id);  Task getByIndex(int index);  int cycle(int cycles);  void changePrioirty(int index, Priority newPriority);  Iterable<Task> getByConsumptionRange(int lo, int hi, boolean inclusive);  Iterable<Task> getByPriority(Priority priority);  Iterable<Task> getByPriorityAndMinimumConsumption(Priority priority, int lo);  } |

## Submission

Submit an archive (.zip) of the source code. Your code **mustn't** contain namespaces/packages.

## Scoring

Each implemented method brings you a specific amount of points, some of the points are awarded for correct behavior, others for performance. **The performance tests might not work on your PC**. You need to cover all tests in each group to receive points. Bellow is a breakdown of all points by methods:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Correct Behavior | Performance | Total |
| Overall | 100 | 100 | 200 |