

# Домашнее задание 9. Multithreading from scratch.

Срок сдачи: 15 мая

Прежде чем приступить к написанию, прочтите раздел **Замечания**.

## 1 Условие

Еще немного попрактикуемся в многопоточном программировании на Java.

**При выполнении этого задания нельзя пользоваться классами из `java.util.concurrent`!**

Сегодня наша задача – создать распределенный сервис для выполнения вычислительно сложной задачи увеличения целого числа на 1.

Ключевые элементы системы:

- Класс *StupidChild*, представляющий клиентские потоки, не способные самостоятельно прибавить 1 и жаждущие воспользоваться сервисом
- Класс *DistributedIncrementor* с методом *int increment(int i) throws InterruptedException*, представляющий наш сервис.
- Класс *Worker*, представляющий рабочие потоки, в которых будут производиться вычисления.
- Класс *Task*, представляющий отдельные задачи для *Worker*-ов (можете попробовать без него, но я бы не рискнул...)

Схематически, все должно происходить следующим образом:

- *StupidChild* генерирует числа и “просит” *DistributedIncrementor* их увеличить.
- *DistributedIncrementor* создает соответствующий *Task* и помещает его в очередь задач.
- Клиентский поток “ожидает”, пока результат вычисления не будет готов.
- *Worker* достает задачи из очереди одну за другой и выполняет их.
- Если очередь пуста, то рабочий поток “ожидает” до появления в очереди новых задач.

Каждому *StupidChild* в конструкторе передается его `id`, количество случайных целых чисел, которое он должен сгенерировать и интервал, в котором должны находиться эти числа.

Для каждого сгенерированного числа, *StupidChild* выводит в консоль свой `id`, сгенерированное число и результат функции *increment* для этого числа.

*Main* должен создать 5 рабочих потоков, 5 клиентских потоков, каждый из которых сгенерирует, скажем, по  $10^4$  чисел в интервале от  $[1, 1000]$ , и все это запустить.

## 2 Замечания

1. При выполнении этого задания нельзя пользоваться классами из `java.util.concurrent`!
2. Отдавайте предпочтение *implements Runnable* перед *extends Thread*
3. *LinkedList<T> implements Queue<T>*
4. Не забудьте, что если вы крутитесь в основном цикле потока и получаете на какой-то из итераций `InterruptedException`, то нужно сделать *break* (и тем самым поскорее свернуть работу потока).
5. Естественно, потокам запрещено находиться в состоянии “активного” ожидания (используйте *notify/wait*).
6. Рабочие потоки являются отличными кандидатами в `daemon threads`!
7. Задача минимум – обеспечение заявленного функционала. За удачные попытки создания более общего дизайна будут начисляться дополнительные баллы.
8. На будущее: писать в консоль при тестирования многопоточных приложение – не лучшая идея, т.к. это сильно увеличивает кол-во точек синхронизации между потоками
9. Не забываем защититься от `spurious wake-ups`!