СПб АУ НОЦНТ РАН

Java 01

08.02.2016

Barrier

Необходимо создать класс *Barrier*, реализующий следующий интерфейс:

```
public class Barrier {
    public Barrier(int parties);
    public void await();
}
```

Пока количество потоков, вызвавших *await*, меньше *parties*, каждый из них блокируется.
В тот момент, когда *await* вызывает поток с номером *parties*, заблокированные потоки продолжают свое исполнение.
Вышеупомянутый и все последующие потоки не блокируется.
Требования:

1. Maven/Gradle проект
2. Тесты

Домашнее задание

```
public interface Lazy<T> {
    T get();
}
```

Необходимо создать класс “LazyFactory” с тремя статическими методами с сигнатурой типа *Lazy<T> createLazy(Supplier<T>)*, возвращающими разные реализации “Lazy”

1. Простая версия с гарантией корректной работы в однопоточном режиме *(без синхронизации)*
2. Гарантия корректной работы в многопоточном режиме, вычисление не должно быть вызвано > 1 раза *(вспомним Singleton)*
3. То же, что и *2* но lock-free. При этом вычисление может быть вызвано > 1 раза (из разных потоков), но “get” должен возвращать один и тот же объект во всех вызовах *(см. AtomicReference/AtomicFieldUpdater)*

Домашнее задание

Дополнительные условия:

- Ограничение по памяти на каждый lazy-объект: не больше двух ссылок
- Вычисление “Supplier” может вернуть null (подумайте, почему это важно)
- Решение должно быть выполнено в виде полноценного maven-проекта
- Необходимо покрыть последовательными (однопоточными) тестами, учитывая различные вырожденные случаи
- Также нужно написать несколько тестов, пытающихся выявить наличие гонок данных (т.е. они должны работать в несколько потоков)
- Срок сдачи до 23:59 15.02

Формат сдачи

- Создаем свой репозиторий на github
- Выполняем задание в отдельной ветке
- Делаем pull request в ветку master своего же репозитория
- Тема PR имеет такой же формат, как и в прошлом семестре:
  Java02. ДЗ 0_ , <фамилия> <имя>, подгруппа _
- В комментарии к PR упоминаем username преподавателя с “@”
- На всякий случай дублируем в таком же формате на e-mail
- Нарушение формата сдачи — это печально

На оценку влияет

- Соблюдение формата сдачи
- Соответствие кода стандартным Java coding conventions
- Выполнение формальных требований задания
- Общая *аккуратность* решения
- Повторение предыдущих ошибок
- Количество итераций сдачи