Практика по Java, часть 2

Пара 1: Разминка

Юрий Литвинов
yurii.litvinov@gmail.com

16.02.2017г

Правила игры

- Как обычно, куча домашек, немного контрольных, баллы и дедлайны, HwProj
  - Будет три крупные задачи (разбитые на части) и несколько мелких
- Для зачёта надо набрать 75% баллов по домашкам и 50% баллов по контрольным
- Тех, кто не наберёт баллы к зачёту, ждёт Что-То Ужасное в конце (например, ещё задачи)

Напоминание про штрафы

Пропущенный дедлайн	-10
Задача на момент дедлайна не реализует все требования условия	пропорционально объёму невыполненных требований
Замечания не исправлены за неделю после их получения	-2
Неумение пользоваться гитом	-2
Проблемы со сборкой (в том числе, забытый org.jetbrains.annotations)	-2
Отсутствие JavaDoc-ов для всех классов, интерфейсов и паблик-методов	-2
Отсутствие описания метода в целом (в том числе, комментарии, начинающиеся с @return)	-1
Слишком широкие области видимости для полей	-2
if (...) return true; else return false;	-2
Комментарии для параметров с заглавной буквы	-0.5

Список может расширяться!

Задача

Многопоточный Lazy

Реализовать следующий интерфейс, представляющий ленивое вычисление:

Java

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

- Объект *Lazy* создаётся на основе вычисления (представляемого объектом *Supplier*)
- Первый вызов *get()* вызывает вычисление и возвращает результат
- Повторные вызовы *get()* возвращают **тот же** объект, что и первый вызов
- В однопоточном режиме вычисление должно запускаться не более одного раза, в многопоточном — как получится (см. далее)

LazyFactory

Создавать объекты надо не вручную, а с помощью класса *LazyFactory*, который должен иметь три метода с сигнатурами вида *public static <T> Lazy<T> createLazy(Supplier<T>)*, возвращающих три разные реализации *Lazy<T>*:

- Простая версия с гарантией корректной работы в однопоточном режиме (без синхронизации)
- Гарантия корректной работы в многопоточном режиме; вычисление не должно производиться более одного раза
  - Что-то наподобие многопоточного синглтона
- То же, что и предыдущее, но lock-free; вычисление может производиться более одного раза, но при этом *Lazy.get* всегда должен возвращать один и тот же объект
  - см. *AtomicReference*/*AtomicReferenceFieldUpdater*

Чтобы было не скучно

- Ограничение по памяти на каждый *Lazy*-объект: не больше двух ссылок
- *Supplier.get* вправе вернуть *null*

Дополнительные требования

- Gradle/Maven
- CI, на котором проходят ваши тесты
- Тесты
  - Однопоточные, на разные хорошие и плохие случаи
  - Многопоточные, на наличие гонок
- Дедлайн: до 23:59 02.03
- Пуллреквест к себе в репозиторий, отметка о сдаче на HwProj