Пара 1: Разминка

Итак, начинается вторая часть изучения программирования на примере языка Java. Пары в этом семестре будут логическим продолжением того, что было в предыдущем семестре — немножко больше многопоточности, программирования сетевых приложений, многопоточных И сетевых приложений, приложений с пользовательским интерфейсом наконец-то, и даже немного веб-приложений, если мы до них дойдём. Как я понял, так или иначе почти всё затрагивалось при программировании под android, но в этот раз всё будет более системным и обстоятельным.

Сначала, как водится в начале семестра, формальности. В конце вас, как обычно, ждёт зачёт (или незачёт), чтобы получить зачёт, надо, как обычно, качественно и вовремя делать домашки, писать контрольные, сдавать их через HwProj[1], делая пуллреквест в свой репозиторий и кидая в HwProj ссылку на пуллреквест. Домашек будет меньше, чем в прошлом семестре, но они будут более объёмными. Будет три больших работы (своя система контроля версий, свой FTP-сервер и свой torrent-клиент) разбитых на несколько заданий, поэтому если не сделать вовремя начало, то не получится сдать и конец, так что лучше не запускать. Для зачёта, как в прошлый раз, надо будет набрать 75% баллов за домашки и 50% баллов за контрольные (которых, наверное, тоже будет две, но мы ещё не решили). При сдаче домашки в любой момент можно “сдаться” и не исправлять замечания, тогда вы получите столько баллов, на сколько насдавали, но можно исправить и получить больше. Те, кто не нарешает задач на нужное количество баллов, будут, как в прошлом семестре, делать ещё что-то — дорешивать старые задачи и делать дополнительные, а поскольку условия дополнительных задач будут известны только на зачётной неделе и неопределённость пугает, это должно мотивировать. Списывать, как обычно, нельзя, и вообще, имеет смысл стараться решать задачи самостоятельно, курс ориентирован на сольное прохождение, к тому же, если что — всегда можно спросить у меня.

Напомню про то, за что снимались баллы в прошлом семестре:

[1] Система сдачи домашних заданий HwProj, URL: http://hwproj.me/ (дата обращения: 13.02.2017г.)

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

Исправления до дедлайна всегда бесплатно, но не обещаю, что буду до дедлайна проверять. После дедлайна обнаружение ошибок из этого списка сразу влечёт снятие баллов за задачу, даже без права исправить ошибку. На остальные ошибки я буду указывать и будет возможность их поправить, но наиболее распространённые ошибки будут пополнять этот список, так что имеет смысл ходить на пары, чтобы вовремя узнать, что меня в очередной раз ужаснуло и заставило расширить список “плохих” ошибок. Табличку с оценками я выложу на вики, но вообще, авторы HwProj обещали сделать отображение оценок прямо на странице с задачами.

На этом с формальностями всё, и можно переходить к задаче, которую желательно сделать за пару, ну либо начать её тут делать и продолжить дома. Чтобы было не так скучно, решение (или его часть) можно показать прямо здесь, и я его прямо здесь зачту (ну или его часть). Вообще, я в этом семестре буду меньше рассказывать и больше именно устраивать практику. Кстати, если хотите, чтобы я что-нибудь рассказал, предложения принимаются, в таком случае задачи надо будет делать дома.

Собственно, задача состоит в том, чтобы реализовать интерфейс *Lazy<T>*, представляющий ленивое вычисление. Интерфейс очень простой:

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

У класса, который реализует этот интерфейс, должен быть конструктор, который принимает объект типа *Supplier* (например, лямбду, которая ничего не принимает и возвращает значение типа *T*). Этот *Supplier* и будет представлять ленивое вычисление — вычисление *Supplier* не происходит до того, пока кто-нибудь не вызвал *get()*. Как только кто-то вызывает *get()*, вычисление выполняется и его результат каким-то образом запоминается в *Lazy*, так что все последующие вызовы *get()* возвращают уже его, не вызывая *Supplier* (это важно, потому что вычисление может быть трудоёмким или иметь побочные эффекты). При этом возвращать надо именно тот же объект (то есть равный со ссылочной точки зрения тому, что *get* вернул в первый раз). В однопоточном режиме вообще вычисление, хранимое в *Supplier*, должно выполняться строго не более одного раза, в многопоточном — может и несколько раз, но в результате объект должен получиться один.

Да, это задача прежде всего на многопоточность, потому что для одного потока эта

задача на несколько минут. Вообще, реализаций должно быть несколько, в зависимости от того, в какой ситуации планируется применять *Lazy*:

- Простая версия с гарантией корректной работы в однопоточном режиме (без синхронизации вообще);
- Реализация с гарантией корректной работы в многопоточном режиме, при этом вычисление *Supplier* не должно производиться более одного раза;
  - Тут может помочь воспоминание о том, как в Java пишется аккуратный многопоточный синглтон;
- То же, что и предыдущее, но lock-free; вычисление может производиться более одного раза (из нескольких потоков одновременно, как это в lock-free принято), но при этом *Lazy.get* всегда должен возвращать один и тот же объект;
  - см. *AtomicReference*[2]/*AtomicReferenceFieldUpdater*[3]

Для определённости также давайте считать, что экземпляры этих трёх классов-реализаций создаются не вручную, а классом *LazyFactory*, который имеет три метода такого вида:

```
public static <T> Lazy<T> createLazy(Supplier<T>)
```

Пока что условие тоже не должно казаться особо страшным (разве что надо вспомнить, что такое lock-free, synchronized и т.д.), но, поскольку у нас уже вторая часть курса по Java, создадим немного проблем, которые сделают неподходящими наивные решения:

- Каждый *Lazy*-объект имеет жёсткие ограничения по памяти — не больше двух ссылок внутри объекта. При этом статических полей у класса может быть сколько угодно, и этим можно пользоваться.
- *Supplier.get()*, естественно, вправе вернуть *null*, это может быть вполне валидным значением вычисления. Угадайте, почему это важно.

В общем-то, с условием задачи всё, можно начинать делать. Как обычно, хочется сборку из консоли, комментарии, тесты (причём, как однопоточные на общую работоспособность и выполнение всех требований условия, так и многопоточные, на гонки), дедлайн, как обычно, через две недели. Из новых требований — давайте считать обязательным использование Continuous Integration-сервера (например, Travis[4]. Надо, чтобы там проект собирался и прогонялись все тесты, что вы написали. Ещё было бы нелишне прикрутить к нему checkstyle[5], но пока не обязательно. Помните, tools are not important, tools are everything!

Кстати, конспект оформлен примерно как научная статья неспроста — весной в городе довольно много студенческих конференций (точно будут 3), на которые можно подать годные НИРы, сделать доклады и получить публикации, что необходимо для научной карьеры

[2] AtomicReference, URL: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicReference.html (дата обращения: 13.02.2017г.)
[3] AtomicReferenceFieldUpdater, URL: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicReferenceFieldUpdater.html (дата обращения: 13.02.2017г.)
[4] Travis homepage, URL: https://travis-ci.org/ (дата обращения: 13.02.2017г.)
[5] Checkstyle homepage, URL: http://checkstyle.sourceforge.net/ (дата обращения: 13.02.2017г.)

и не необходимо, но круто для карьеры технической. Если кто-то не в курсе, но хочет поучаствовать, я могу на следующей паре отдельно рассказать про конференции, но дедлайн по первой из них (и самой, на мой взгляд, престижной, SEIM[6] уже 5-го марта (в этом году будет проводиться в офисе JetBrains в БЦ “Таймс”). Впрочем, это не последняя, я надеюсь, конфа, на неё можно и через годик, а в этот раз на СПИСОК или на конференцию в Политехе.

[6]Software Engineering and Information Management conference, URL: http://2017.seim-conf.org/ (дата обращения: 13.02.2017г.)