Экстерн-проекты по курсу Java

Для участия в проекте, заполните **форму**.

Внимание: актуальная версия документа (2025 год) – здесь: https://docs.google.com/document/d/1D1caFW9VPJiQtd9YEJL86uDCmp0woMgF-79BrDY_OCU/edit?tab=t.0#heading=h.n0oljbb3mjh6

NB: По hurdy-gurdy сейчас все озвученные задачи решены, откроем проект когда появятся новые

~~Hurdy-gurdy: генератор Java/Kotlin кода на базе OpenAPI спецификации~~

~~2 человека~~

~~OpenAPI – широко распространённый формат специфицирования Restful-сервисов на базе YAML.~~

~~Для генерации клиентского и серверного кода на разных языках программирования на базе этого формата применяется проект https://github.com/OpenAPITools/openapi-generator , написанный на Java. Достоинствами этого проекта является широкая известность и активное развитие. Недостатком данного проекта является нацеленность сразу на все языки программирования, попытка выработать единообразный подход к кодогенерации через **шаблонизатор** (mustache). В свою очередь, шаблонизаторы, подходящие для генерации кода на языках разметки (в первую очередь HTML), не вполне подходят для генерации кода на языках общего назначения. Пример: для использования библиотечного класса в Java, его необходимо объявить в imports в начале модуля и по месту использования, что означает вставку в шаблон в двух местах и усложняет процесс кодогенерации.~~

~~Как результат, не все языковые модули в проекте Openapi-generator производят одинаково пригодный для использования результат. Модификация вывода Openapi-generator для нужд конкретного проекта может быть очень трудной.~~

~~С другой стороны, если сузить задачу генерации кода только для Java/Kotlin, можно воспользоваться двумя библиотеками:~~

- ~~Swagger-parser – Java-парсер для OpenAPI формата (используемый в том числе в качестве зависимости в openapi-generator)~~
- ~~Javapoet – библиотека для генерации кода на Java~~
- ~~Kotlinpoet – библиотека для генерации кода на Kotlin~~

~~Комбинацией этих библиотек можно довольно эффективно, малым количеством простого кода решить задачу генерации кода на Java и Kotlin.~~

~~Первая версия генератора (hurdy-gurdy) расположена здесь: https://github.com/CourseOrchestra/hurdy-gurdy~~

~~Задачи:~~

- ~~Поддержать наследование и полиморфизм OpenApi спецификации https://swagger.io/docs/specification/data-models/inheritance-and-polymorphism/~~
- ~~Поддержать работу с несколькими исходными файлами~~
- ~~Реализовать вывод в Kotlin~~
- ~~По итогам работы – публикация в хабр~~

Jedis-mock: тестовый мок Redis на Java

2-3 человека

Система Redis (in-memory кэш с продвинутыми структурами данных) исключительно популярна (60k звёзд на гитхабе, огромное коммьюнити пользователей). Библиотека jedis-mock предназначена для тестирования Java-кода, взаимодействующего с Redis, без запуска “настоящего” Redis. Jedis-mock (https://github.com/fppt/jedis-mock) по сути является частичной реимплементацией Redis на Java без претензий на highload, для тестовых нужд. Проект имеет более 140 звёзд и пользовательскую базу. Бывший мейнтейнер, Filipe Teixeira, не занимается этим проектом и отдал проект нам.

1. Недостатком такого подхода по сравнению с запуском настоящего Redis для тестов является то, что в моке реализованы не все возможности настоящего Redis
2. Достоинством этого подхода является простота (достаточно втянуть jedis-mock в качестве библиотеки в тестовый проект) и скорость выполнения тестов.

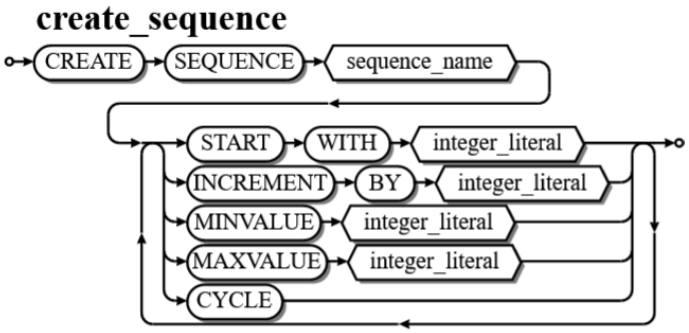
Сам по себе проект очень хорош как полигон для практики.

1. В процессе работы с проектом неизбежно осваивается Redis, что само по себе имеет ценность)
2. Мы, по сути, занимаемся реимплементацией Redis на Java, что также имеет практическую ценность (становится понятна возможная архитектура систем, подобных Redis)
3. При этом Redis для нас -- эталон имплементации. При тестировании jedis-mock используется методика comparison-тестов, когда один и тот же код прогоняется на Redis и Jedis и проверяется, что результаты одинаковы в обоих случаях. В процессе работы с проектом познакомимся с продвинутыми практиками использования JUnit5. Также при тестировании используются “родные” тесты самого Redis, написанные на языке TCL.

Задачи 2024-весна

- Есть запрос пользователя на **keyspace notifications** (https://github.com/fppt/jedis-mock/issues/406)
- Имплементация текущих issues в системе, работа с пользователями.

JSyntrax: визуализация формальных языков в виде railroad-диаграмм



1 человек

JSyntrax -- генератор синтаксических диаграмм, визуализирующий формальные грамматики (пример того, как это выглядит на практике см. здесь)

Студенческий проект JSyntrax по реимплементации старого проекта Syntrax с Python на Java был удачен, была проделана вся основная работа. Также проект был интегрирован в экосистему Asciidoctor.

Однако до необходимого уровня качества проект ещё не доведён, прежде всего из-за встречающихся багов в расчёте размеров фигур (https://github.com/atp-mipt/jsyntrax/issues). Особенно это касается вывода в PNG.
Задачи 2024-осень

- JSyntrax проинтегрирован с asciidoctor-diagram. И он привносит с собой проблемы :-) Например, https://github.com/asciidoctor/asciidoctor-diagram/issues/467 Как обсуждалось с мейнтейнерами AsciiDoctor, лучше всего избавиться от Groovy и создать свой собственный парсер DSL, например, с помощью JavaCC
- В настоящий момент JSyntrax использует шрифты системы. Для того, чтобы обеспечить попиксельную воспроизводимость диаграмм на всех платформах, небоходимо включить в проект JSyntrax open-source шрифты и сделать вывод в PNG стабильным на всех платформах.

~~LJV: Lightweight Java Visualizer~~

~~**1-2 человека.** Мотивация -- глубже разобраться с низкоуровневыми деталями устройства Java-библиотеки, Reflections API, JOL. Также нужен человек, готовый разбираться с визуализацией в формате SVG и анимацией (см. две последние задачи из списка).~~

~~Проект “оживления” LJV (https://github.com/atp-mipt/ljv), начатый магистрантами ФМБФ Ильёй Селивановым и Нурасом Ногаевым, оказался очень удачным (250 звёзд), но нужно двигаться дальше.~~

~~В настоящий момент блокером по дальнейшему развитию LJV является задержка со слиянием с проектом JOL (Java Object Layout) с целью использования внутренних API JOL. Без этого слияния LJV не будет иметь возможности работать с новыми версиями JAVA из-за закрытия доступа Reflection API к библиотечным классам. Ждём решения вопроса.~~

~~Задачи 2022~~

- ~~После влития проекта в JOL необходимо будет в первую очередь заняться рефакторингом и использованием внутреннего GraphWalker JOL для обхода дерева объектов.~~
- ~~Подумать о том, как на диаграмме отображать адреса объектов / раскладку объектов в адресном пространстве. Возможно, надо будет подключить альтернативную диаграмму (не на базе DOT, может быть, генерировать SVG напрямую)~~
- ~~Подумать о том, как можно применить SVG-анимацию для представления нескольких, друг за другом следующих состояний объекта~~

~~Оживление проекта Reflections~~

~~**2-3 человека.** Мотивация: глубже изучить организацию работы над серьёзным OpenSource проектом, обеспечение качества, системную разработку на Java (Reflections API), изучить работу проекта с большим количеством пользователей (где невозможно менять API и допускать регресс).~~

~~Библиотека Reflections (https://github.com/ronmamo/reflections) является широко используемой, но в настоящее время не поддерживаемой её основным автором. Предыдущий релиз был сделан в апреле 2020 года, у неё огромное количество незамёрженных пулл-реквестов и issues. В README написано: *We're looking for maintainers to assist in reviewing pull requests and managing releases, please reach out.*~~

Чтобы взять эту библиотеку “под своё крыло”, необходимо произвести первоначальную “санацию”:

- Определить покрытие тестами (создать и проанализировать JaCoCo отчёт, внедрить формирование JaCoCo отчёта в процесс сборки)
- По необходимости -- рефакторинг тестов и допокрытие
- Проект на travis-ci.org -- провести миграцию на gh actions
- Провести обновление maven-плагинов
- Провести статический анализ кода (SpotBugs, PVS)

-- на данном этапе мы будем уверены, что можем контролировать качество изменений, вносимых в библиотеку --

- (опционально) Миграция тестов на JUnit5

Связаться с ronmamo и предложить помощь. Дальше возможно два варианта: либо нам будет предоставлена возможность коммитить в upstream, либо нет :-) Во втором варианте мы создаём собственный форк reflections в организации atp-mipt, публикуем его на Maven Central и агитируем народ пользоваться нашим форком (в частности, проект LJV мы переводим на него). Агитацию эффективно было бы проводить путём мёржа существующих PR-ов в наш форк и отписку авторам PR-ов, что их изменения -- у нас.

Пока на паузе, задачи решены, проект работает

Xylophone 2.0 -- продолжение

Проект Xylophone (https://github.com/CourseOrchestra/xylophone) решает задачу автоматического создания красиво отформатированных Excel-документов на основе данных, представленных в XML. Это нужно для многих организаций, генерирующих печатные формы, отчёты и тому подобное. Преобразование XML-данных осуществляется с помощью так называемого дескриптора, который также пишется на XML, и на базе Excel-шаблона. Проект использует Apache POI: библиотеку, осуществляющую reverse-engineering бинарного формата файлов Microsoft Office. В проекте был заявлен план поддержать как формат Excel, так и формата ODS (OpenOffice/LibreOffice), но поддержка ODS так и не была реализована (хотя спрос на ODS в последнее время возрос в связи с общей тенденцией ухода от зависимости от офисных продуктов Microsoft). Сейчас проект практически не поддерживается: есть issues на гитхабе, но их никто не решает.

Что сделано в прошлом году:

- Проект переведён на Java11
- Реализован JSON-формат для данных, YAML для дескриптора
- Переработано тестирование, используются Approval-тесты
- Создан задел для вывода в ODS-формат, но, вероятно, была выбрана неправильная библиотека для этой цели.
- Полностью переработана логика обхода данных, исправлены застарелые ошибки
- В целом была выпилена часть функциональности Xylophone, в частности, “sax-режим” для больших входных данных.

На данный момент Xylophone2 **не доведён до рабочего состояния**. Необходимо разбираться и доводить проект до ума, улучшать поддержку ODS, сравнивать функциональность старого и нового Xylophone, писать документацию.

Результат опубликовать в Maven Central.