Содержание

1. Цель работы

Разработать standalone приложение, которое имеет следующие возможности:

1. Принимает на вход проект на языке Kotlin в каком-то виде (папка, набор папок, файл конфигурации системы сборки, ...).

2. Строит для данного проекта абстрактные синтаксические деревья (АСД) входящих в него файлов/классов

3. Считает для полученных АСД следующие метрики (метрики должны выводиться как в консоль, так и в заданный пользователем файл в машинно-читаемом формате XML/JSON/YAML/...)

    - Максимальная глубина наследования
    - Средняя глубина наследования
    - Метрика ABC
    - Среднее количество переопределенных методов
    - Среднее количество полей в классе

4. Протестировать результат на не менее чем трех проектах с GitHub и проанализировать результаты в свободном формате в виде человеко-читаемого документа.

2. Ход работы

На вход программы подается два аргумента - путь к папке с проектом и путь к файлу JSON (если указать несуществующий JSON, то он создастся автоматически). Строить АСД было решено, используя библиотеку antlr4 и словарь из их репозитория на GitHub: https://github.com/antlr/grammars-v4/tree/master/kotlin/kotlin После того, как было построено дерево, стояла задача проанализировать его и посчитать требуемые метрики. Для этого я считывал токены, которые мы получаем после парсинга файла и с помощью условий if пошагово прибавлял значения к той или иной метрике. После передаем все данные в метод класса Metrics. После считывания всех файлов считаем метрики. Каждая метрика считалась на весь проект сразу, однако при необходимости легко модифицировать код так, чтобы метрики считались для каждого файла или класса. Процесс подсчета метрик:

- Максимальная глубина наследования - при выявлении наследования пара добавлялась в Map. После чтения всех файлов я искал максимальную глубину наследования пробегаясь по ключам, и если они != null, то делаем прошлое зачение ключом, прибавляем к глубине 1 и пытаемся получить следующее значение. Поиск происходит снизу вверх (ключ - наследник, значение - наследуемый класс), так как несколько классов могут быть наследниками одного и того же, а уникальный ключ в Map может быть всего один. Если класс явно ни от кого не наследуется, то значение для него в Map = (То есть его глубина = 1). Если класс наследуется от класса, которого нет в нашей программе (получили из библиотеки), то к его глубине прибавляется 1.

- Средняя глубина наследования - считается как глубина для каждого класса / количество классов (во всей программе).

- Метрика ABC

  - Прибавляем к A при следующих условиях:
    * Появление оператора присваивания (исключая объявления констант и назначения параметров по умолчанию) (=, * =, / =,
    * Возникновение оператора инкремента или декремента (++, –).
  - Прибавляем к B при следующих условиях:
    * Возникновение вызова функции или вызова метода класса.
    * Вызов конструктора.
  - Прибавляем к C при следующих условиях:
    * Возникновение условного оператора (===, !==, <,>, <=,> =, ==,! =).
    * Появление следующих ключевых слов (' else ', ' case ', ' default ', ' ? ','?:', ' Try ', ' catch ', 'when').
    * Возникновение унарного условного оператора .

- Среднее количество переопределенных методов - считается количество переопределенных методов и делится на количество классов.
- Среднее количество полей в классе - считаются все поля в классе (вне тел функций и методов) и делится на количество классов. Параметры функций и классов пропускаются.

Далее программа выводит метрику в консоль и в JSON файл.

3. Классы приложения

- Main - принимает аргументы, передает их в класс Parse, вызывает общий подсчет метрики и ее вывод.
- Parse - ищет файлы котлин в указанной директории и выполняет их парсинг, отправляет данные в класс Metrics.
- Metrics - суммирует данные для получения итоговых метрик программы, выводит их в консоль и в JSON файл.
- PrintTree - печатает AST.

4. Анализ проектов с GitHub

Для тестов были выбраны три проекта с GitHub:

- https://github.com/r4v3n6101/rtw
- https://github.com/r4v3n6101/lovely-bones
- https://github.com/r4v3n6101/Raven-API

Для каждого из этих проектов успешно сформированы JSON файлы со следующим содержаним

- {"maxDepth":2,"averageDepth":1.0769230769230769,"ABC":162.3145095177877, "averageFields":0.8461538461538461,"averageOverride":1.3846153846153846} (rtw)

- {"maxDepth":3,"averageDepth":1.5,"ABC":396.91056927222286, "averageFields":0.2608695652173913,"averageOverride":0.9130434782608695} (lovely-bones)
- {"maxDepth":2,"averageDepth":1.3214285714285714,"ABC":1096.7556701471847, "averageFields":2.0,"averageOverride":2.6875} (Raven-API)

При ручном подсчете значения совпадают. Однако важно учитывать, что я мог не правильно понять какие то из условий (метрики для всей программы, для кажого файла или для каждого класса?). При уточнении задания это легко исправить вызовом подсчета метрики после прочтения и парсинга каждого файла/класса программы и изменением порядка записи в JSON файл.

5. Выводы

На основе словаря для языка kotlin библиотеки antlr4 было создано standalone приложение, которое создает AST представление файлов программы, считает требуемые в задании метрики и выводит их в JSON файл. Программа работоспособна и считает метрики для всей программы, однако возможны ошибки в интерпретации условий подсчета метрик (для всей программы, для кажого файла или для каждого класса?).