Домашние задания

Домашнее задание 1. Обход файлов

1. Разработайте класс `Walk`, осуществляющий подсчет хеш-сумм файлов.
    1. Формат запуска:

       ```
       java Walk <входной файл> <выходной файл>
       ```

    2. Входной файл содержит список файлов, которые требуется обойти.
    3. Выходной файл должен содержать по одной строке для каждого файла. Формат строки:

       ```
       <шестнадцатеричная хеш-сумма> <путь к файлу>
       ```

    4. Для подсчета хеш-суммы используйте алгоритм SHA-256 (поддержка есть в стандартной библиотеке).
    5. Если при чтении файла возникают ошибки, укажите в качестве его хеш-суммы все нули.
    6. Кодировка входного и выходного файлов — UTF-8.
    7. Размеры файлов могут превышать размер оперативной памяти.
    8. Пример

       Входной файл

       ```
       samples/1
       samples/12
       samples/123
       samples/1234
       samples/1
       samples/binary
       samples/no-such-file
       ```

       Выходной файл

       ```
       6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b samples/1
       6b51d431df5d7f141cbececcf79edf3dd861c3b4069f0b11661a3eefacbba918 samples/12
       a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3 samples/123
       03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4 samples/1234
       6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b samples/1
       40aff2e9d2d8922e47afd4648e6967497158785fbd1da870e7110266bf944880 samples/binary
       0000000000000000000000000000000000000000000000000000000000000000 samples/no-such-file
       ```

2. Сложный вариант:
    1. Разработайте класс `RecursiveWalk`, осуществляющий подсчет хеш-сумм файлов в директориях.
    2. Входной файл содержит список файлов и директорий, которые требуется обойти. Обход директорий осуществляется рекурсивно.
    3. Пример:

       Входной файл

       ```
       samples/binary
       samples
       samples/no-such-file
       ```

       Выходной файл

       ```
       40aff2e9d2d8922e47afd4648e6967497158785fbd1da870e7110266bf944880 samples/binary
       6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b samples/1
       6b51d431df5d7f141cbececcf79edf3dd861c3b4069f0b11661a3eefacbba918 samples/12
       a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3 samples/123
       03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4 samples/1234
       40aff2e9d2d8922e47afd4648e6967497158785fbd1da870e7110266bf944880 samples/binary
       0000000000000000000000000000000000000000000000000000000000000000 samples/no-such-file
       ```

3. При выполнении задания следует обратить внимание на:
    - Дизайн и обработку исключений, диагностику ошибок.
    - Программа должна корректно завершаться даже в случае ошибки.
    - Корректная работа с вводом-выводом.

    - Отсутствие утечки ресурсов.
    - Возможность повторного использования кода.
4. Требования к оформлению задания.
    - Проверяется исходный код задания.
    - Весь код должен находиться в пакете `info.kgeorgiy.ja.фамилия.walk`.

Репозиторий курса

Домашнее задание 2. Множество на массиве

1. Разработайте класс `ArraySet`, реализующий неизменяемое упорядоченное множество.
    - Класс `ArraySet` должен реализовывать интерфейс SortedSet (простой вариант) или NavigableSet (сложный вариант).
    - Все операции над множествами должны производиться с максимально возможной асимптотической эффективностью.
2. При выполнении задания следует обратить внимание на:
    - Применение стандартных коллекций.
    - Избавление от повторяющегося кода.

Домашнее задание 3. Студенты

1. Разработайте класс `StudentDB`, осуществляющий поиск по базе данных студентов.
    - Класс `StudentDB` должен реализовывать интерфейс `StudentQuery` (простой вариант) или `GroupQuery` (сложный вариант).
    - Каждый метод должен состоять из ровно одного оператора. При этом длинные операторы надо разбивать на несколько строк.
2. При выполнении задания следует обратить внимание на:
    - применение лямбда-выражений и потоков;
    - избавление от повторяющегося кода.

Домашнее задание 4. Implementor

1. Реализуйте класс `Implementor`, генерирующий реализации классов и интерфейсов.
    - Аргумент командной строки: полное имя класса/интерфейса, для которого требуется сгенерировать реализацию.
    - В результате работы должен быть сгенерирован java-код класса с суффиксом `Impl`, расширяющий (реализующий) указанный класс (интерфейс).
    - Сгенерированный класс должен компилироваться без ошибок.
    - Сгенерированный класс не должен быть абстрактным.
    - Методы сгенерированного класса должны игнорировать свои аргументы и возвращать значения по умолчанию.
2. В задании выделяются три варианта:
    - *Простой* — `Implementor` должен уметь реализовывать только интерфейсы (но не классы). Поддержка generics не требуется.
    - *Сложный* — `Implementor` должен уметь реализовывать и классы, и интерфейсы. Поддержка generics не требуется.
    - *Бонусный* — `Implementor` должен уметь реализовывать generic-классы и интерфейсы. Сгенерированный код должен иметь корректные параметры типов и не порождать `UncheckedWarning`.

Домашнее задание 5. Jar Implementor

Это домашнее задание **связано** с предыдущим и будет приниматься только с ним. Предыдущее домашнее задание отдельно сдать будет нельзя.

1. Создайте `.jar`-файл, содержащий скомпилированный `Implementor` и сопутствующие классы.
    - Созданный `.jar`-файл должен запускаться командой `java -jar`.
    - Запускаемый `.jar`-файл должен принимать те же аргументы командной строки, что и класс `Implementor`.
2. Модифицируйте `Implemetor` так, чтобы при запуске с аргументами `-jar имя-класса файл.jar` он генерировал `.jar`-файл с реализацией соответствующего класса (интерфейса).

3. Для проверки, кроме исходного кода так же должны быть представлены:
    - скрипт для создания запускаемого `.jar`-файла, в том числе исходный код манифеста;
    - запускаемый `.jar`-файл.
4. **Сложный вариант**. Решение должно быть модуляризовано.

Домашнее задание 6. Javadoc

Это домашнее задание **связано** с двумя предыдущими и будет приниматься только с ними. Предыдущие домашнее задание отдельно сдать будет нельзя.

1. Документируйте класс `Implementor` и сопутствующие классы с применением Javadoc.
    - Должны быть документированы все классы и все члены классов, в том числе `private`.
    - Документация должна генерироваться без предупреждений.
    - Сгенерированная документация должна содержать корректные ссылки на классы стандартной библиотеки.
2. Для проверки, кроме исходного кода так же должны быть представлены:
    - скрипт для генерации документации;
    - сгенерированная документация.

Домашнее задание 7. Итеративный параллелизм

1. Реализуйте класс `IterativeParallelism`, который будет обрабатывать списки в несколько потоков.
2. В простом варианте должны быть реализованы следующие методы:
    - `minimum(threads, list, comparator)` — первый минимум;
    - `maximum(threads, list, comparator)` — первый максимум;
    - `all(threads, list, predicate)` — проверка, что все элементы списка, удовлетворяют предикату;
    - `any(threads, list, predicate)` — проверка, что существует элемент списка, удовлетворяющий предикату.
    - `count(threads, list, predicate)` — подсчёт числа элементов списка, удовлетворяющих предикату.
3. В сложном варианте должны быть дополнительно реализованы следующие методы:
    - `filter(threads, list, predicate)` — вернуть список, содержащий элементы удовлетворяющие предикату;
    - `map(threads, list, function)` — вернуть список, содержащий результаты применения функции;
    - `join(threads, list)` — конкатенация строковых представлений элементов списка.
4. Во все функции передается параметр `threads` — сколько потоков надо использовать при вычислении. Вы можете рассчитывать, что число потоков относительно мало.
5. Не следует рассчитывать на то, что переданные компараторы, предикаты и функции работают быстро.
6. При выполнении задания **нельзя** использовать *Concurrency Utilities*.