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 64-битную версию алгоритма PJW.
    5. Если при чтении файла возникают ошибки, укажите в качестве его хеш-суммы 0000000000000000.
    6. Кодировка входного и выходного файлов — UTF-8.
    7. Если родительская директория выходного файла не существует, то соответствующий путь надо создать.
    8. Размеры файлов могут превышать размер оперативной памяти.
    9. Пример

        Входной файл

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

        Выходной файл

        ```
        0000000000000031 samples/1
        0000000000003132 samples/12
        0000000000313233 samples/123
        0000000031323334 samples/1234
        0000000000000031 samples/1
        005501015554abff samples/binary
        0000000000000000 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

        ```
        005501015554abff samples/binary
        0000000000000031 samples/1
        0000000000003132 samples/12
        0000000000313233 samples/123
        0000000031323334 samples/1234
        005501015554abff samples/binary
        0000000000000000 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.
4. Требования к оформлению задания.
    - Проверяется исходный код задания.
    - Весь код должен находиться в пакете info.kgeorgiy.ja.фамилия.walk.

Тесты к домашним заданиям

Домашнее задание 2. Множество на массиве

1. Разработайте класс ArraySet, реализующие 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, который будет генерировать 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

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. Данное домашнее задание сдается только вместе с предыдущим. Предыдущее домашнее задание отдельно сдать будет нельзя.
5. **Сложный вариант**. Решение должно быть модуляризовано.

Домашнее задание 6. Javadoc

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.
3. Данное домашнее задание сдается только вместе с предыдущим. Предыдущее домашнее задание отдельно сдать будет нельзя.

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.
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 не велико.
5. Не следует рассчитывать на то, что переданные компараторы, предикаты и функции работают быстро.
6. При выполнении задания **нельзя** использовать *Concurrency Utilities*.
7. Рекомендуется подумать, какое отношение к заданию имеют моноиды.

Домашнее задание 8. Параллельный запуск

1. Напишите класс ParallelMapperImpl, реализующий интерфейс ParallelMapper.

    ```
    public interface ParallelMapper extends AutoCloseable {
        <T, R> List<R> map(
            Function<? super T, ? extends R> f,
            List<? extends T> args
        ) throws InterruptedException;

        @Override
        void close() throws InterruptedException;
    }
    ```

    - Метод run должен параллельно вычислять функцию f на каждом из указанных аргументов (args).
    - Метод close должен останавливать все рабочие потоки.
    - Конструктор ParallelMapperImpl(int threads) создает threads рабочих потоков, которые могут быть использованы для распараллеливания.
    - К одному ParallelMapperImpl могут одновременно обращаться несколько клиентов.
    - Задания на исполнение должны накапливаться в очереди и обрабатываться в порядке поступления.
    - В реализации не должно быть активных ожиданий.
2. Доработайте класс IterativeParallelism так, чтобы он мог использовать ParallelMapper.
    - Добавьте конструктор IterativeParallelism(ParallelMapper)
    - Методы класса должны делить работу на threads фрагментов и исполнять их при помощи ParallelMapper.
    - При наличии ParallelMapper сам IterativeParallelism новые потоки создавать не должен.
    - Должна быть возможность одновременного запуска и работы нескольких клиентов, использующих один ParallelMapper.

Домашнее задание 9. Web Crawler

1. Напишите потокобезопасный класс WebCrawler, который будет рекурсивно обходить сайты.
    1. Класс WebCrawler должен иметь конструктор

        ```
        public WebCrawler(Downloader downloader, int downloaders, int extractors, int perHost)
        ```

        - downloader позволяет скачивать страницы и извлекать из них ссылки;
        - downloaders — максимальное число одновременно загружаемых страниц;
        - extractors — максимальное число страниц, из которых одновременно извлекаются ссылки;
        - perHost — максимальное число страниц, одновременно загружаемых с одного хоста. Для определения хоста следует использовать метод getHost класса URLUtils из тестов.
    2. Класс WebCrawler должен реализовывать интерфейс Crawler

        ```
        public interface Crawler extends AutoCloseable {
            Result download(String url, int depth);

            void close();
        }
        ```

        - Метод download должен рекурсивно обходить страницы, начиная с указанного URL на указанную глубину и возвращать список загруженных страниц и файлов. Например, если глубина равна 1, то должна быть загружена только указанная страница. Если глубина равна 2, то указанная страница и те страницы и файлы, на которые она ссылается и так далее. Этот метод может вызываться параллельно в нескольких потоках.
        - Загрузка и обработка страниц (извлечение ссылок) должна выполняться максимально параллельно, с учетом ограничений на число одновременно загружаемых страниц (в том числе с одного хоста) и страниц, с которых загружаются ссылки.
        - Для распараллеливания разрешается создать до downloaders + extractors вспомогательных потоков.
        - Загружать и/или извлекать ссылки из одной и той же страницы в рамках одного обхода (download) запрещается.
        - Метод close должен завершать все вспомогательные потоки.
    3. Для загрузки страниц должен применяться Downloader, передаваемый первым аргументом конструктора.

        ```
        public interface Downloader {
            public Document download(final String url) throws IOException;
        }
        ```

        - Метод download загружает документ по его адресу (URL).
        - Документ позволяет получить ссылки по загруженной странице:

            ```
            public interface Document {
                List<String> extractLinks() throws IOException;
            }
            ```

            Ссылки, возвращаемые документом, являются абсолютными и имеют схему http или https.
    4. Должен быть реализован метод main, позволяющий запустить обход из командной строки
        - Командная строка

            ```
            WebCrawler url [depth [downloads [extractors [perHost]]]]
            ```

        - Для загрузки страниц требуется использовать реализацию CachingDownloader из тестов.
2. Версии задания
    1. *Простая* — не требуется учитывать ограничения на число одновременных закачек с одного хоста (perHost >= downloaders).
    2. *Полная* — требуется учитывать все ограничения.
    3. *Бонусная* — сделать параллельный обод в ширину.

Домашнее задание 10. HelloUDP

1. Реализуйте клиент и сервер, взаимодействующие по UDP.
2. Класс HelloUDPClient должен отправлять запросы на сервер, принимать результаты и выводить их на консоль.
    - Аргументы командной строки:
        1. имя или ip-адрес компьютера, на котором запущен сервер;
        2. номер порта, на который отсылать запросы;
        3. префикс запросов (строка);
        4. число параллельных потоков запросов;
        5. число запросов в каждом потоке.
    - Запросы должны одновременно отсылаться в указанном числе потоков. Каждый поток должен ожидать обработки своего запроса и выводить сам запрос и результат его обработки на консоль. Если запрос не был обработан, требуется послать его заново.
    - Запросы должны формироваться по схеме <префикс запросов><номер потока>_<номер запроса в потоке>.
3. Класс HelloUDPServer должен принимать задания, отсылаемые классом HelloUDPClient и отвечать на них.
    - Аргументы командной строки:
        1. номер порта, по которому будут приниматься запросы;
        2. число рабочих потоков, которые будут обрабатывать запросы.
    - Ответом на запрос должно быть Hello, <текст запроса>.
    - Если сервер не успевает обрабатывать запросы, прием запросов может быть временно приостановлен.

Домашнее задание 11. Физические лица

1. Добавьте к банковскому приложению возможность работы с физическими лицами.
    1. У физического лица (Person) можно запросить имя, фамилию и номер паспорта.
    2. Локальные физические лица (LocalPerson) должны передаваться при помощи механизма сериализации.
    3. Удалённые физические лица (RemotePerson) должны передаваться при помощи удалённых объектов.
    4. Должна быть возможность поиска физического лица по номеру паспорта, с выбором типа возвращаемого лица.
    5. Должна быть возможность создания записи о физическом лице по его данным.
    6. У физического лица может быть несколько счетов, к которым должен предоставляться доступ.
    7. Счету физического лица с идентификатором *subId* должен соответствовать банковский счет с *id* вида *passport*:*subId*.
    8. Изменения, производимые со счетом в банке (создание и изменение баланса), должны быть видны всем соответствующим RemotePerson, и только тем LocalPerson, которые были созданы после этого изменения.
    9. Изменения в счетах, производимые через RemotePerson, должны сразу применяться глобально, а производимые через LocalPerson – только локально для этого конкретного LocalPerson.
2. Реализуйте приложение, демонстрирующее работу с физическим лицами.
    1. Аргументы командной строки: имя, фамилия, номер паспорта физического лица, номер счета, изменение суммы счета.
    2. Если информация об указанном физическом лице отсутствует, то оно должно быть добавлено. В противном случае – должны быть проверены его данные.
    3. Если у физического лица отсутствует счет с указанным номером, то он создается с нулевым балансом.
    4. После обновления суммы счета новый баланс должен выводиться на консоль.
3. Напишите тесты, проверяющее вышеуказанное поведение как банка, так и приложения.
    - Для реализации тестов рекомендуется использовать JUnit (Tutorial). Множество примеров использования можно найти в тестах.
    - Если вы знакомы с другим тестовым фреймворком (например, TestNG), то можете использовать его.
    - Jar-файлы используемых библиотек надо класть в каталог lib вашего репозитория.
    - Использовать самописные фреймворки и тесты запускаемые через main нельзя.
4. **Сложный вариант**
    1. Тесты не должны рассчитывать на наличие запущенного RMI Registry.
    2. Создайте класс BankTests, запускающий тесты.
    3. Создайте скрипт, запускающий BankTests и возвращающий код (статус) 0 в случае успеха и 1 в случае неудачи.
    4. Создайте скрипт, запускающий тесты с использованием стандартного подхода для вашего тестового фреймворка. Код возврата должен быть как в предыдущем пункте.

Домашнее задание 12. HelloNonblockingUDP

1. Реализуйте клиент и сервер, взаимодействующие по UDP, используя только неблокирующий ввод-вывод.
2. Класс HelloUDPNonblockingClient должен иметь функциональность аналогичную HelloUDPClient, но без создания новых потоков.
3. Класс HelloUDPNonblockingServer должен иметь функциональность аналогичную HelloUDPServer, но все операции с сокетом должны производиться в одном потоке.
4. В реализации не должно быть активных ожиданий, в том числе через Selector.
5. Обратите внимание на выделение общего кода старой и новой реализации.
6. *Бонусный вариант*. Клиент и сервер могут перед началом работы выделить O(число потоков) памяти. Выделять дополнительную память во время работы запрещено.

Домашнее задание 13. Статистика текста

1. Создайте приложение TextStatistics, анализирующее тексты на различных языках.
    1. Аргументы командной строки:
        - локаль текста,
        - локаль вывода,
        - файл с текстом,
        - файл отчета.
    2. Поддерживаемые локали текста: все локали, имеющиеся в системе.
    3. Поддерживаемые локали вывода: русская и английская.
    4. Файлы имеют кодировку UTF-8.
    5. Подсчет статистики должен вестись по следующим категориям:
        - предложения,
        - слова,
        - числа,
        - деньги,
        - даты.
    6. Для каждой категории должна собираться следующая статистика:
        - число вхождений,
        - число различных значений,
        - минимальное значение,
        - максимальное значение,
        - минимальная длина,
        - максимальная длина,
        - среднее значение/длина.
    7. Пример отчета:

        ```
        Анализируемый файл "input.txt"
        Сводная статистика
            Число предложений: 43.
            Число слов: 275.
            Число чисел: 40.
            Число сумм: 3.
            Число дат: 3.
        Статистика по предложениям
            Число предложений: 43 (43 различных).
            Минимальное предложение: "Аргументы командной строки: локаль текста, локаль вывода, файл с текстом, файл отчета.".
            Максимальное предложение: "Число чисел: 40.".
            Минимальная длина предложения: 13 ("Число дат: 3.").
            Максимальная длина предложения: 211 ("GK: если сюда поставить реальное предложение, то процесс не сойдётся").
            Средняя длина предложения: 55,465.
        Статистика по словам
            Число слов: 275 (157 различных).
            Минимальное слово: "GK".
            Максимальное слово: "языках".
            Минимальная длина слова: 1 ("с").
            Максимальная длина слова: 14 ("TextStatistics").
            Средняя длина слова: 6,72.
        Статистика по числам
            Число чисел: 40 (24 различных).
            Минимальное число: -12345,0.
            Максимальное число: 12345,67.
            Среднее число: 207,676.
        Статистика по суммам денег
            Число сумм: 3 (3 различных).
            Минимальная сумма: 100,00 ₽.
            Максимальная сумма: 345,67 ₽.
            Средняя сумма: 222,83 ₽.
        Статистика по датам
            Число дат: 3 (3 различных).
            Минимальная дата: 22 мая 2021 г..
            Максимальная дата: 8 июн. 2021 г..
            Средняя дата: 30 мая 2021 г..
        ```

2. Вы можете рассчитывать на то, что весь текст помещается в память.
3. При выполнении задания следует обратить внимание на:
    1. Декомпозицию сообщений для локализации
    2. Согласование сообщений по роду и числу
4. Напишите тесты, проверяющее вышеуказанное поведение приложения.
    - Для реализации тестов рекомендуется использовать JUnit (Tutorial). Множество примеров использования можно найти в тестах.
    - Если вы знакомы с другим тестовым фреймворком (например, TestNG), то можете использовать его.
    - Использовать самописные фреймворки и тесты запускаемые через main нельзя.