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 32-битную версию алгоритма FVN.
    5. Если при чтении файла возникают ошибки, укажите в качестве его хеш-суммы `00000000`.
    6. Кодировка входного и выходного файлов — UTF-8.
    7. Размеры файлов могут превышать размер оперативной памяти.
    8. Пример

        Входной файл

        ```
        java/info/kgeorgiy/java/advanced/walk/samples/1
        java/info/kgeorgiy/java/advanced/walk/samples/12
        java/info/kgeorgiy/java/advanced/walk/samples/123
        java/info/kgeorgiy/java/advanced/walk/samples/1234
        java/info/kgeorgiy/java/advanced/walk/samples/1
        java/info/kgeorgiy/java/advanced/walk/samples/binary
        ```

        Выходной файл

        ```
        050c5d2e java/info/kgeorgiy/java/advanced/walk/samples/1
        2076af58 java/info/kgeorgiy/java/advanced/walk/samples/12
        72d607bb java/info/kgeorgiy/java/advanced/walk/samples/123
        81ee2b55 java/info/kgeorgiy/java/advanced/walk/samples/1234
        050c5d2e java/info/kgeorgiy/java/advanced/walk/samples/1
        8e8881c5 java/info/kgeorgiy/java/advanced/walk/samples/binary
        ```

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

        Входной файл

        ```
        java/info/kgeorgiy/java/advanced/walk/samples/binary
        java/info/kgeorgiy/java/advanced/walk/samples
        ```

        Выходной файл

        ```
        8e8881c5 java/info/kgeorgiy/java/advanced/walk/samples/binary
        050c5d2e java/info/kgeorgiy/java/advanced/walk/samples/1
        2076af58 java/info/kgeorgiy/java/advanced/walk/samples/12
        72d607bb java/info/kgeorgiy/java/advanced/walk/samples/123
        81ee2b55 java/info/kgeorgiy/java/advanced/walk/samples/1234
        8e8881c5 java/info/kgeorgiy/java/advanced/walk/samples/binary
        ```

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 `ru.ifmo.ctddev.фамилия.walk`.

Домашнее задание 2. Множество на массиве

1. Разработайте класс `ArraySet`, реализующие неизменяемое упорядоченное множество.
    - Класс `ArraySet` должен реализовывать интерфейс `SortedSet` (упрощенная версия) или `NavigableSet` (усложненная версия).
    - Все операции над множествами должны производиться с максимально возможной асимптотической эффективностью.
2. При выполнении задания следует обратить внимание на:
    - Применение стандартных коллекций.
    - Избавления от boilerplate кода.

Домашнее задание 3. Implementor

1. Реализуйте класс `Implementor`, который будет генерировать реализации классов и интерфейсов.
    - Аргументы 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 уровня сложности:
   - *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 и Generic-классы и интерфейсы. Сгенерированный код должен иметь корректные параметры типов.

Домашнее задание 4. 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, что бы 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 предъявлены:
   - скрипт для создания запускаемого `.jar`-файла, в том числе, исходный код манифеста;
   - запускаемый `.jar`-файл.

Домашнее задание 5. Javadoc

1. Документируйте класс `Implementor` и сопутствующие классы с применением Javadoc.
   - Должны быть документированы все классы и все члены классов, в том числе закрытые (`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 предъявлены:
   - скрипт для генерации документации;
   - сгенерированная документация.

Домашнее задание 6. 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;
   - `concat(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 7. Параллельный запуск

1. Напишите класс `ParallelMapperImpl`, реализующий интерфейс `ParallelMapper`.

```
public interface ParallelMapper extends AutoCloseable {
    <T, R> List<R> run(
        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. Модифицируйте касс `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`.
   - Должна быть возможность одновременного запуска и работы нескольких клиентов, использующих один `ParallelMapper`.
   - При наличии `ParallelMapper` сам `IterativeParallelism` новые потоки создавать не должен.

Домашнее задание 8. Web Crawler

1. Напишите 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 извлекаются ссылки;
      - `perHost` — максимальное число страниц, одновременно загружаемых с одного хоста. Для опредения хоста следует использовать метод `getHost` класса `URLUtils` из тестов.
   2. Класс `WebCrawler` должен реализовывать интерфейс `Crawler`

```
public interface Crawler extends AutoCloseable {
```

```
    List<String> download(String url, int depth) throws IOException;

    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.
- 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 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 [downloads [extractors [perHost]]]
```

   - Для загрузки страниц требуется использовать реализацию `CachingDownloader` из тестов.

2. Версии задания
   1. *Простая* — можно не учитывать ограничения на число одновременных закачек с одного хоста `(perHost >= downloaders)`.
   2. *Полная* — требуется учитывать все ограничения.

Домашнее задание 9. 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 заного.
   - 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 10. 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. Удаленные физические лица (`RemotePerson`) должны передаваться при помощи удаленных 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.
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.

Домашнее задание 11. Копирование файлов

1. Создайте приложение `UIFileCopy`, которое будет осуществлять копирование файлов и директорий с демонстрацией прогресса.
   1. Аргументы командной строки:

      `UIFileCopy <что копировать> <куда копировать>`

   2. В процессе копирования в графическом интерфейсе должны отображаться:
      - текущий прогресс (progress bar);
      - прошедшее время;
      - ожидаемое время до окончания копирования;
      - средняя скорость копирования;

        - текущая скорость копирования;
        - кнопка отмены копирования.
    3. При нажатии кнопки отмены копирования, копирование должно быть мгновенно прекращено. После отмены на диске могут оставаться скопированные файлы.
2. При выполнении задания следует обратить внимание на:
    1. Дизайн интерфейса пользователя.
    2. Отзывчивость интерфейса пользователя.
    3. Поведение интерфейса пользователя при изменении размеров окна.
3. *Простая версия*: В процессе копирования больших файлов можно не обновлять статистику по времени и скорости копирования.

Домашнее задание 12. JExplorer

Создайте приложение `JExplorer`, аналог Windows Explorer.

Минимальная функциональность:

1. Отображение дерева каталогов (с ленивой загрузкой).
2. Отображение содержимого каталога, выбранного в дереве, в виде таблицы, содержащей столбцы:
    - имя файла
    - размер
    - дата и время создания
    - дата и время последней модификации
3. Сортировка таблице по столбцу при клике на его заголовок.
4. Переход в подкаталоги и открытие файлов по двойному щелчку.
5. В действия должны быть доступны посредством мыши, меню, контекстного меню, панели инструментов (требуется поддержка всех типов доступа).

Расширенная функциональность:

1. Поддержка копирования/перемещения файлов и каталогов при помощи `UIFileCopy`.

Бонусная функциональность (чем больше — тем лучше):

1. Поддержка Cut-and-paste.
2. Поддержка Drag-and-drop.
3. Режим отображения иконок.
4. Встроенный просмотрщик картинок.
5. Встроенный просмотрщик текстовых файлов (в том числе, больших).
6. Встроенный редактор текстовых файлов (в том числе, больших).