

# Домашние задания

## Домашнее задание 1. Обход файлов

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

```
java Walk <входной файл> <выходной файл>
```
  - Входной файл содержит список файлов, которые требуется обойти.
  - Выходной файл должен содержать по одной строке для каждого файла. Формат строки:

```
шестнадцатеричная хеш-сумма <путь к файлу>
```
  - Для подсчета хеш-суммы используйте 64-битную версию алгоритма [PJW](#).
  - Если при чтении файла возникнут ошибки, укажите в качестве его хеш-суммы `0000000000000000`.
  - Кодировка входного и выходного файлов — UTF-8.
  - Если роунд-ельская директория выходного файла не существует, то соответствующий путь надо создать.
  - Размеры файлов могут превышать размер оперативной памяти.
- Пример

Входной файл

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

Выходной файл

```
0000000000000031 samples/1
0000000000001132 samples/12
0000000000313233 samples/123
0000000031323334 samples/1234
000000000000031 samples/1
005501015554abff samples/binary
000000000000000 samples/no-such-file
```
- Сложный вариант:
  - Разработайте класс `RecursiveWalk`, осуществляющий подсчет хеш-сумм файлов в директориях
  - Входной файл содержит список файлов и директорий, которые требуется обойти. Обход директорий осуществляется рекурсивно.
- Пример

Входной файл

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

Выходной файл

```
005501015554abff samples/binary
0000000000000031 samples/1
0000000000001132 samples/12
0000000000313233 samples/123
0000000031323334 samples/1234
005501015554abff samples/binary
000000000000000 samples/no-such-file
```

- 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.
- Требования к оформлению задания.
  - Проверяется исходный код задания.
  - Весь код должен находиться в пакете `info.kgeorgiy.gj.фамилия.walk`.

[Тесты к домашним заданиям](#)

## Домашнее задание 2. Множество на массиве

- 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.
- При выполнении задания следует обратить внимание на:
  - Применение стандартных коллекций.
  - Избавление от повторяющегося кода.

## Домашнее задание 3. Студенты

- 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.
- При выполнении задания следует обратить внимание на:
  - применение лямбда-выражений и потоков;
  - избавление от повторяющегося кода.

## Домашнее задание 4. Implementor

- Реализуйте класс `Implementor`, который будет генерировать реализации классов и интерфейсов.
  - Аргумент командной строки: полное имя класса/интерфейса, для которого требуется генерировать реализацию.
  - В результате работы должен быть сгенерирован java-код класса с суффиксом `Imp1`, 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.
- 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

- 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`.
- Модифицируйте `Implementor` так, чтобы при запуске с аргументами `-jar имя-класса файл.jar` он генерировал `-jar`-файл с реализацией соответствующего класса (интерфейса).
- 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.
- Данное домашнее задание оцется только вместе с предыдущим. Предыдущее домашнее задание отдельно сдать будет нельзя.
- Сложный вариант.** Решение должно быть модуляризовано.

## Домашнее задание 6. Javadoc

- 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.
- Для проверки, кроме исходного кода так же должны быть представлены:
  - скрипт для генерации документации;
  - сгенерированная документация.
- Данное домашнее задание слается только вместе с предыдущим. Предыдущее домашнее задание отдельно сдать будет нельзя.

## Домашнее задание 7. Итеративный параллелизм

- Реализуйте класс `IterativeParallelism`, который будет обрабатывать списки в несколько потоков.
- 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.
- 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 строчковых представлений элементов списка.
- Во все функции передается параметр `threads` — сколько потоков надо использовать при вычислениях. Вы можете рассчитывать, что число потоков не велико.
- Не следует рассчитывать на то, что переданные компараторы, предикаты и функции работают быстро.
- При выполнении задания **нельзя** использовать *`Concurrency Utilities`*.
- Рекомендуется подумать, какое отношение к этому отношению имеют [миссии](#).

## Домашнее задание 8. Параллельный запуск

- Напишите класс `ParallelMapperImp1`, 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 `ParallelMapperImp1(int threads)` создает `threads` рабочих потоков, которые могут быть использованы для распараллеливания.
  - К одному `ParallelMapperImp1` могут одновременно обращаться несколько клиентов.
  - Задания на исполнение должны накапливаться в очереди и обрабатываться в порядке поступления.
  - В реализации не должно быть активных ожиданий.
- Доработайте класс `IterativeParallelism` так, чтобы он мог использовать `ParallelMapper`.
  - Добавьте конструктор `IterativeParallelism(ParallelMapper)`
  - Методы класса должны делегировать 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

- Напишите потокобезопасный класс `WebCrawler`, который будет рекурсивно обходить сайты.
    - 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.
```
  - 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.
  - 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 на загруженной странице:

```
public interface Document {
    List<String> extractLinks() throws IOException;
}
```
  - Ссылки, возвращаемые документом, являются абсолютными и имеют схему `http` или `https`.
  - Должен быть реализован метод `main`, позволяющий запустить обход из командной строки

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

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

## Домашнее задание 10. HelloUDP

- Реализуйте клиент и сервер, взаимодействующие по UDP.
  - Класс `HelloUDPClient` должен отправлять запросы на сервер, принимать результаты и выводить их на консоль.
    - Аргументы командной строки:
      - Имя или ip-адрес компьютера, на котором запущен сервер;
      - номер порта, на который отсылать запросы;
      - префикс запросов (строка);
      - число параллельных потоков запросов;
      - 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>.
  - Класс `HelloUDPServer` должен принимать задания, отсылаемые классом `HelloUDPClient` и отвечать на них.
    - Аргументы командной строки:
      - номер порта, по которому будут приниматься запросы;
      - число рабочих потоков, которые будут обрабатывать запросы.
    - Ответом на запрос должно быть `hello`, <текст запроса>.
    - Если сервер не успевает обрабатывать запросы, прием запросов должен быть временно приостановлен.

## Домашнее задание 11. Физические лица

- Добавьте к банковскому приложению возможность работы с физическими лицами.
  - У физического лица (`Person`) можно запросить имя, фамилию и номер паспорта.
  - Локальные физические лица (`LocalPerson`) должны передаваться при помощи механизма сериализации.
  - Удалённые физические лица (`RemotePerson`) должны передаваться при помощи удалённых объектов.
  - Должна быть возможность поиска физического лица по номеру паспорта, с выбором типа возвращаемого лица.
  - Должна быть возможность создания записи о физическом лице по его данным.
  - У физического лица может быть несколько счетов, в котором должен предоставляться доступ.
  - Счету физического лица с идентификатором *subId* должен соответствовать банковский счет с *id* вида *passport.subId*.
  - 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.
  - Изменения в счетах, производимые через `RemotePerson`, должны сразу применяться глобально, а производимые через `LocalPerson` — только локально для этого конкретного `LocalPerson`.
- Реализуйте приложение, демонстрирующее работу с физическим лицами.
  - Аргументы командной строки: имя, фамилия, номер паспорта физического лица, номер счета, изменение суммы счета.
  - Если информация об указанном физическом лице отсутствует, то оно должно быть добавлено. В противном случае – должны быть проверены его данные.
  - Если у физического лица отсутствует счет с указанным номером, то он создается с нулевым балансом.
  - После обновления суммы счета новый баланс должен выводиться на консоль.
- Напишите тесты, проверяющие 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.
- Сложный вариант**
  - Тесты не должны рассчитывать на наличие запущенного RMI Registry.
  - Создайте класс `BankTests`, запускающий тесты.
  - Создайте скрипт, запускающий `BankTests` и возвращающий код (status) 0 в случае успеха и 1 в случае неудачи.
  - Создайте скрипт, запускающий тесты с использованием стандартного подхода для вашего тестового фреймворка. Код возврата должен быть как в предыдущем пункте.






















## Домашнее задание 12. HelloNonblockingUDP

- Реализуйте клиент и сервер, взаимодействующие по UDP, используя только неблокирующий ввод-вывод.
- Класс `HelloNonblockingClient` должен иметь функциональность аналогичную `HelloUDPClient`, но без создания новых потоков.
- Класс `HelloNonblockingServer` должен иметь функциональность аналогичную `HelloUDPServer`, но все операции с сокетом должны производиться в одном потоке.
- В реализации не должно быть активных ожиданий, в том числе через `Selector`.
- Обратите внимание на выделение общего кода старой и новой реализации.
- 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

- Создайте приложение `TextStatistics`, анализирующее тексты на различных языках.
  - Аргументы командной строки:
    - локаль текста,
    - локаль вывода,
    - файл с текстом,
    - файл отчета.
  - Поддерживаемые локали текста: все локали, имеющиеся в системе.
  - Поддерживаемые локали вывода: русская и английская.
  - Файлы имеют кодировку UTF-8.
  - Подсчет статистики должен вестись по следующим категориям:
    - предложения,
      - слова,
      - числа,
      - деньги,
      - даты.
    - Для каждой категории должна собираться следующая статистика:
      - число вхождений,
      - число различных значений,
      - минимальное значение,
      - максимальное значение,
      - минимальная длина,
      - максимальная длина,
      - среднее значение/длина.
  - 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 сида поставишь 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 (".").  
Максимальная длина слова: 14 ("TextStatistics").  
Средняя длина слова: 6,72.  
Статистика по суммам  
Число чисел: 40 (24 различных).  
Минимальное число: -12345,6.  
Максимальное число: 12345,67.  
Среднее число: 207,676.  
Статистика по суммам денег  
Число сумм: 3 (3 различных).  
Минимальная сумма: 100,00 P.  
Максимальная сумма: 345,67 P.  
Средняя сумма: 222,83 P.  
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..
- Вы можете рассчитывать на то, что весь текст помещается в память.
- При выполнении задания следует обратить внимание на:
  - Декомпозицию сообщений для локализации
  - Согласование сообщений по роду и числу
- Напишите тесты, проверяющие 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.

|   |   |
|---|---|
|     | Домашнее задание 1. Обход файлов            |
|    | Домашнее задание 2. Множество на массиве    |
|    | Домашнее задание 3. Студенты                |
|    | Домашнее задание 4. Implementor             |
|    | Домашнее задание 5. Jar Implementor         |
|    | Домашнее задание 6. Javadoc                 |
|    | Домашнее задание 7. Итеративный параллелизм |
|    | Домашнее задание 8. Параллельный запуск     |
|    | Домашнее задание 9. Web Crawler             |
|    | Домашнее задание 10. HelloUDP               |
|    | Домашнее задание 11. Физические лица        |
|    | Домашнее задание 12. HelloNonblockingUDP    |
|    | Домашнее задание 13. Статистика текста      |
|    |   |
|    |   |
|    |   |
|    |   |
|    |   |
|    |   |
|    |   |
|    |   |
|   |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |
|  |   |