

Домашние задания

Домашнее задание 1. Обход файлов

- Разработайте класс `Walk`, осуществляющий подсчет хэш-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:

```
<сумма> <наименование хэш-суммы> <тип> < файл>
```
 - Для подсчета хэш-суммы используйте последние 64 бита [SHA-256](#) (поддержка есть в стандартной библиотеке).
 - Если при чтении файла возникают ошибки, укажите в качестве его хэш-суммы все нули.
 - Коировка входного и выходного файлов — UTF-8.
 - Размеры файлов могут превышать размер оперативной памяти.
 - Пример

Входной файл
<pre>samples/1 samples/12 samples/123 samples/1234 samples/1 samples/binary samples/no-such-file</pre>
Выходной файл
<pre>6b86b273ff34fcef samples/1 6b51d431df5d7f14 samples/12 a665a45920422f9d samples/123 03ac674216f3e15c samples/1234 6b86b273ff34fcef samples/1 40affe2e9d2b9922e samples/binary 0000000000000000 samples/no-such-file</pre>

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

Входной файл
<pre>samples/binary samples samples/no-such-file</pre>
Выходной файл
<pre>40affe2e9d2b9922e samples/binary 6b86b273ff34fcef samples/1 6b51d431df5d7f14 samples/12 a665a45920422f9d samples/123 03ac674216f3e15c samples/1234 40affe2e9d2b9922e samples/binary 0000000000000000 samples/no-such-file</pre>

- При выполнении задания следует обратить внимание на:
 - Дизайн и обработку исключений, диагностику ошибок.
 - Программа должна корректно завершаться даже в случае ошибки.
 - Корректная работа с входом-выходом.
 - Отсутствие утечки ресурсов.
 - Возможность повторного использования кода.
- 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

- 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 наилучшей асимптотической эффективностью.
- При выполнении задания следует обратить внимание на:
 - Применение стандартных коллекций.
 - Итабление от повторяющегося кода.
 - Отсутствие `unchecked warnings` при компиляции.
 - Отсутствие излишних подавленных `unchecked warnings`.

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

Домашнее задание 4. Сплитераторы и коллекторы

- Разработайте класс `Lambda`, реализующий сплитераторы для деревьев и дополнительные коллекторы.
 - Простой вариант* (интерфейс `EasyLambda`) — реализовать:
 - сплитераторы для различных деревьев, двучленных деревьев с известным размером, *k*-ичных деревьев;
 - коллекторы первого, последнего, среднего элементов;
 - коллекторы общего префикса и сурфакса строк.
 - Сложный вариант* (интерфейс `HardLambda`) — дополнительно реализовать:
 - сплитераторы всех видов деревьев над списками элементов;
 - коллектор *n*-ого элемента;
 - коллекторы первых и последних *n* элементов.

- При выполнении задания следует обратить внимание на:
 - характеристики создаваемых сплитераторов;
 - избавление от повторяющегося кода.

Домашнее задание 5. Implementor

- 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.
- 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.
 - Богусный* — `Implementor` должен уметь реализовывать `generic`-классы и интерфейсы. Сгенерированный код должен иметь корректные параметры типов и не порождать `unchecked warnings`.

Домашнее задание 6. Jar Implementor

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

- 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 компиляции можно использовать код из тестов.
- Вы можете создавать файлы и директории в текущем каталоге, но не за его пределами.
- Для проверки, кроме исходного кода, также должны быть представлены:
 - скрипт для создания запускаемого `.jar`-файла, в том числе, исходный код манифеста;
 - запускаемый `.jar`-файл.
- Сложный вариант.** Решение должно быть модуляризовано.

Домашнее задание 7. Javados

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

- Документируйте класс `Implementor` и сопутствующие классы с применением `Javados`.
 - 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 и модулей `info.kgeorgiy.java.advanced.*`.
- Для проверки, кроме исходного кода, также должны быть представлены:
 - скрипт для генерации документации (он может рассчитывать, что рядом с вашим репозиторием скопирован репозиторий курса);
 - сгенерированная документация.

В **последующих** домашних заданиях все `public` и `protected` сущности должны быть документированы.

Домашнее задание 8. Итеративный параллелизм

- Реализуйте класс `IterativeParallelism`, который будет обрабатывать списки в нескольких потоках.
- В простом варианте должны быть реализованы следующие методы:
 - `argMax(threads, list, comparator)` — индекс первого максимума;
 - `argMin(threads, list, comparator)` — индекс первого минимума;
 - `indexOf(threads, list, predicate)` — индекс первого элемента, удовлетворяющего предикат;
 - `lastIndexOf(threads, list, predicate)` — индекс последнего элемента, удовлетворяющего предикат;
 - `sumIndices(threads, list, predicate)` — сумма индексов элементов, удовлетворяющих предикат;
- В сложном варианте должны быть дополнительно реализованы следующие методы:
 - `indices(threads, list, predicate)` — индексы элементов, удовлетворяющих предикат;
 - `filter(threads, list, predicate)` — вернуть список, содержащий элементы удовлетворяющие предикату;
 - `map(threads, list, function)` — вернуть список, содержащий результаты применения функции;
- Во все функции передается параметр `threads` — сколько потоков надо использовать при вычислениях. Вы можете рассчитывать, что число потоков относительно мало.
- Не следует рассчитывать на то, что переданные компараторы, предикаты и функции работают быстро.
- Можно сделать `O(threads)`, но не `O(list.size())` действий без распараллеливания.
- При выполнении задания **нельзя** использовать *`Concurrency Utilities`* и *`Parallel Streams`*.

Домашнее задание 9. Параллельный запуск

- 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();
}
```

 - Метод `map` должен параллельно вычислять функцию `f` на каждом из указанных аргументов (`args`).
 - Конструктор `ParallelMapperImpl(int threads)` должен создавать `threads` рабочих потоков, которые используются для распараллеливания.
 - Метод `close` должен останавливать все рабочие потоки.
 - К одному `ParallelMapperImpl` могут одновременно обращаться несколько клиентов.
 - При недостатке потоков для распараллеливания, задания на исполнение должны накапливаться в очереди и обрабатываться в порядке поступления.
 - В реализации не должно быть активных соединений.
 - Код должен находиться в пакете `iterative`.

- Обратите внимание на обработку исключений, кидаемых функцией `f`.
 - Исключения не должны приводить к сокращению числа рабочих потоков.
- Сложный вариант.** Исключения должны выкидываться из метода `map`.
 - 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`.

- При выполнении задания всё ещё **нельзя** использовать *`Concurrency Utilities`* и *`Parallel Streams`*.

Домашнее задание 10. 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 `AutoCloseable`

```
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.
 - Метод `download` может выполняться 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 `downloaders + extractors` вспомогательных потоков.
 - Повторно 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 по 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
 - Командная строка

```
WebCrawler url [depth [downloaders [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 обход в ширину.
- Задание подразумевает активное использование *`Concurrency Utilities`*, в частности, в решении не должно быть «велосипедов», аналогичных/легко сводящихся к классам из *`Concurrency Utilities`*.

Домашнее задание 11. 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 запроса в потоке><номер потока>`.
- 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.
 - Ответа на запрос должно быть `Hello, <смет запросов>`.
 - Несмотря на то, что текущий способ получения ответа по запросу очень прост, сервер должен быть рассчитан на ситуацию, когда этот процесс может требовать много ресурсов и времени.
 - Если сервер не успевает обрабатывать запросы, прием запросов может быть временно приостановлен.

Домашнее задание 12. Физические лица

- Добавьте к банковскому приложению возможность работы с физическими лицами.
 - У физического лица (`Person`) можно запросить имя, фамилию и номер паспорта.
 - Удалённые физические лица (`RemotePerson`) должны передаваться при помощи удалённых объектов.
 - Локальные физические лица (`LocalPerson`) должны передаваться при помощи механизма сериализации, и при последующем использовании не требовать связи с сервером.
 - 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, к которым должен предоставляться доступ через `Person`.
 - Счёту физического лица с идентификатором `subId` должен соответствовать банковский счёт с `id` вида `personId.subId`.
 - Изменения, производимые со счётом в банке (создание и изменение баланса), должны быть видны всем соответствующим `RemotePerson`, и только тем `LocalPerson`, которые были созданы по базе этого клиента.
 - 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 лицом.
 - Аргументы командной строки: имя, фамилия, номер паспорта физического лица, номер счёта, изменение суммы счёта.
 - 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 счёт с указанным номером, то он создается с нулевым балансом.
 - После обновления суммы счёта новый баланс должен выводиться на консоль.

- Сложный вариант**
 - На каждом счёту всегда должно быть неотрицательное количество денег.
- Приложение должно находиться в пакете `info.kgeorgiy.ja.*.bank` и его подпакетах.

Домашнее задание 13. Физические лица (тесты)

- Напишите тесты, проверяющие поведение банка и приложения из домашнего задания [Физические лица](#).
 - 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.
 - Добавьте `jar`-файлы используемых библиотек в каталог `lib` вашего репозитория.
 - Нельзя** использовать самописные фреймворки и тесты, запускаемые через `main`.
- Сложный вариант**
 - Тесты не должны рассчитывать на наличие запущенного RMI Registry.
 - Создайте класс `BankTests`, запускающий тесты.
 - Создайте скрипт, запускающий `BankTests` и возвращающий код (статус) 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.
- Тесты должны находиться в пакете `info.kgeorgiy.ja.*.bank` и его подпакетах.
- При сдаче вам нужно будет продемонстрировать работу тестов на вашем компьютере.

Это домашнее задание **связано** с домашним заданием [Физические лица](#) и будет приниматься только с ним.

Домашнее задание 14. HelloNonblockingUDP

- Реализуйте клиент и сервер, взаимодействующие по UDP, используя только неблокирующий ввод-вывод.
- Класс `HelloUDPNonblockingClient` должен иметь функциональность аналогичную `HelloUDPClient`, но без создания новых потоков.
- Класс `HelloUDPNonblockingServer` должен иметь функциональность аналогичную `HelloUDPServer`, но все операции с сокетом производятся в одном потоке.
- В реализации не должно быть активных соединений, в том числе через `Selector`.
- Обратите внимание на выделение общего кода старой и новой реализации.
- Можно использовать неблокирующий или асинхронный ввод-вывод. Нельзя использовать виртуальные потоки.
- Бонусный вариант.** Клиент и сервер могут перед началом работы выделить *O(число рабочих потоков)* памяти. Выделять дополнительную память во время работы запрещено.

Домашнее задание 15. 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.

- Средний отчет:
 - Анализируемый файл "quine.ru-RU.in".
 - Сводная статистика:
 - Число предложений: 30.
 - Число слов: 117.
 - Число чисел: 35.
 - Среднее число: 221,381.
 - Число дат: 3.
 - Статистика по предложениям
 - Число предложений: 30 (30 различными).
 - Минимальное предложение: "Анализируемый файл "quine.ru-RU.in".".
 - Максимальное предложение: "Число чисел: 35,".
 - Минимальная длина предложения: 13 ("Число дат: 3,").
 - Максимальная длина предложения: 109 ("ГК: если сюда поставить реальное предложение, то процесс не сойдётся").
 - Средняя длина предложения: 37,7.
 - Статистика по словам
 - Число слов: 117 (48 различными).
 - Минимальное слово: "GK".
 - Максимальное слово: "камаха".
 - Минимальная длина слова: 1 ("с").
 - Максимальная длина слова: 18 ("TextStatisticsTest").
 - Средняя длина слова: 6,752.
 - Статистика по числам
 - Число чисел: 35 (21 различными).
 - Минимальное число: -12345,67.
 - Максимальное число: 12345,67.
 - Среднее число: 221,381.
 - Статистика по суммам денег
 - Число сумм: 3 (3 различными).
 - Минимальная сумма: 100,00 **Р**.
 - Максимальная сумма: 345,67 **Р**.
 - Средняя сумма: 222,83 **Р**.
 - Статистика по датам
 - Число дат: 3 (3 различными).
 - Минимальная дата: 23 мая 2025 г..
 - Максимальная дата: 30 мая 2025 г..
 - Средняя дата: 26 мая 2025 г..

- Вы можете рассчитывать на то, что весь текст поместится в память.

- При выполнении задания следует обратить внимание на:
 - Декларацию переменных для локализации.
 - Согласование сообщений по ролю и числу.

- Приложение должно находиться в пакете `info.kgeorgiy.ja.*.libn` и его подпакетах.

[D3-1. Обход файлов](#)
[D3-2. Множество на массиве](#)
[D3-3. Студенты](#)
[D3-4. Сплитераторы и коллекторы](#)
[D3-5. Implementor](#)
[D3-6. Jar Implementor](#)
[D3-7. Javados](#)
[D3-8. Итеративный параллелизм](#)
[D3-9. Параллельный запуск](#)
[D3-10. Web Crawler](#)
[D3-11. HelloUDP](#)
[D3-12. Физические лица](#)
[D3-13. Физические лица \(тесты\)](#)
[D3-14. HelloNonblockingUDP](#)
[D3-15. Статистика текста](#)

