# Домашнее задание #7

Simple torrent

# Содержание

# Simple torrent

Марья Васильевна решила просмотреть разработанные приложения телефонных книг, однако ей это не удалось, поскольку сервер не рассчитан на такой объем скачиваемых данных.

Поэтому в данном домашнем задании необходимо **разработать упрощенный вариант клиент-серверного приложения для скачивания файлов**.

## Сервер

1. Сервер располагает *директорией с файлами*, доступными для скачивания клиентам (без использования поддиректорий).
2. Она указывается при запуске сервера.

*Сервер может быть консольным приложением.*

## Клиент

1. При запуске клиента запрашиваются **хост** и **порт**, на котором работает сервер.
2. Установив соединение с сервером, клиент получает *список файлов из директории скачивания* и отображает его пользователю.
   a. Пользователь может выбрать нужный файл из списка.
   b. При выборе файла для скачивания пользователю отображается диалог *подтверждения* скачивания.
      i. В нём указывается размер загружаемого файла.
3. В случае *подтверждения* пользователь должен указать *путь* для сохранения файла.
4. После указания *пути*:
   a. начинается загрузка файла
   b. отображается индикатор с указанием прогресса в процентах (progress bar).
5. Все загруженные пользователем файлы должны отображаться клиентом до тех пор, пока программа не завершит работу.

*Клиент может быть консольным приложением.*

## Общее

1. Максимальный размер файла для скачивания — 128GiB.
2. Клиент и сервер для взаимодействия друг с другом используют TCP/IP `Socket`'ы.
3. Если Вы решили использовать графический пользовательский интерфейс, то он должен быть на Swing или JavaFX.
4. К серверу могут подключаться несколько клиентов одновременно.

# Бонусы

Для повышения оценки можно предусмотреть и **описать** в файле readme дополнительную функциональность

# Результат работы

## Структура проекта

Проект должен иметь как минимум 2 модуля:

1. для сервера
2. для клиента

Каждый из модулей в свою очередь содержит 2 дерева исходных файлов:

1. **основное**, где находится код приложения.
2. **для тестов**. В нём — JUnit 5 `@Test`'ы.

## Тесты

Тесты в данном проекте должны покрывать код *каждого из модулей* на 40%.

Тестировать графический пользовательский интерфейс необязательно.

## `jar`

В результате выполнения данного домашнего задания необходимо собрать 2 `jar`-файла:

1. для клиента - `client.jar`
2. для сервера - `server.jar`

Пожалуйста, **не собирайте fat / uber `jar`'ы** без особых на то оснований: у проверяющих будет установлен javafx, а в переменных пути Idea и *переменных окружения операционной системы* будет добавлена `PATH_TO_FX` указывающая на директорию `lib` дистрибутива (в ней находятся `javafx.base.jar`, `javafx.fxml.jar` и другие).

## Архив

### Содержимое

Архив **должен** содержать:

1. Проект, созданный в Intellij IDEA (он *может* использовать maven или gradle).
2. Исходные файлы приложения.
3. `jar`-файлы, включая файлы *конфигурации*, необходимые для его создания.
   - Для запуска может быть приложен файл `sh` / `bat`-*скрипт*.
4. Файл readme, *если реализована дополнительная функциональность*.

Архив **не должен** содержать *другие* файлы, в которых нет необходимости для *сборки, запуска* и *открытия проекта в Intellij IDEA* (например, `class` -файлы).

Под файлами **конфигурации** для сборки `jar` в общем случае имеется в виду то, что должно быть понятно, как собран `jar`. Если он собирается в **Intellij IDEA**, то ничего для этого делать специально **не нужно**: IDE создаёт эти файлы автоматически в директории `idea/artifacts`.

## Имя архива

Имя архива составляется по формуле `HW7_zzz_Фамилия_Имя.zip`, где:

- `zzz` — номер группы.
- `Фамилия` — Ваша фамилия латиницей.
- `Имя` — Ваше имя латиницей.

Единственный допустимый формат архива - `.zip`.

# Как мы будем оценивать работу?

## `jar`, проект, архив

Сперва мы посмотрим, верно ли выполнены формальные требования к сдаче проекта.

Если:

- `jar`ы собраны правильно, а именно:
  - У них — верные имена.
  - Их можно запустить командой `java -jar` *или* с помощью **приложенных** `sh`, `bat` -скриптов.
    - Для запуска мы будем использовать Java 11.
  - Внутри `jar`'ов нет shaded классов из javafx.
- Выполнены требования к архиву.
- Структура проекта соответствует указанной,

то мы поставим здесь **0.75 балла**.

## Реализация клиента и сервера

Далее, мы посмотрим на код проекта, проверим соответствие реализации указанным требованиям См. Сервер и Клиент.

Если:

1. все требования к клиенту и серверу выполнены корректно
2. мы не обнаружили ошибок в реализации
3. логика отображения GUI **не смешивается** с логикой работы по сети, обработкой запросов и ответов
4. код соответствует Java Code Conventions,

то мы поставим **8.5 баллов** по этому пункту.

## Документация

Если в проекте ко всем `public`-методам и типам написаны корректные Javadoc'и, то мы добавим **0.25 балла**

### Тесты

Если:

1. Тесты покрывают код каждого из модулей на 40%

2. Они не написаны исключительно ради того, чтобы получить 40% покрытия
3. В каждом их них есть хотя бы один assert (не ключевое слово языка Java, а из тестового фреймворка, например `assertEquals`),

то мы добавим **0.5 балла**.

# Бонусы

*Максимальное* количество бонусных баллов - **2**.

Их можно получить, только если выполнены все пункты, указанные в "Клиент" и "Сервер".

# Баллы по усмотрению проверяющего

Проверяющий может добавить или снять баллы по своему усмотрению в пределах **[-0.75; 0.75]**, *обосновав своё решение*.

# Итого

*Таблица 1. Баллы, выставляемые при проверке*

| **Диапазон баллов** | **Описание** |
|---|---|
| [0; 0.75] | `jar`, архив, проект |
| [0; 8.5] | Реализация клиента и сервера |
| [0; 0.25] | Javadoc |
| [0; 0.5] | Тесты |
| [0; 2] | Бонусы |
| [-0.75; 0.75] | По усмотрению проверяющего |