## Общее описание

Задание состоит из 3 программ: ProxyServer, EchoServer и InitiatorServer. Программы запакованы в один архив innova-task-x.x.x.jar и отличаются только стартовыми классами. Для запуска программ используются скрипты, в которых прописаны нужные имена классов.

## Сборка

Для сборки запустите команду:

```
mvn package
```

Для сборки документации по Java-классам запустите команду:

```
mvn javadoc:javadoc
```

## Запуск

Определите, на каком хосте будет работать ProxyServer и на каком порту. Далее, в примерах считается, что ProxyServer работает на одной машине с остальными программами и прослушивает порт 12345. Настройте переменную окружения PATH, так чтобы была доступна программа java версии 1.7 или выше. В примерах показаны команды для ОС Windows.

Запустите в консоли команду

```
proxy-server.cmd 12345
```

Дождитесь сообщения `Enter command:`, наберите `start` и нажмите `Enter`.

Запустите в другой консоли команду

```
echo-server.cmd loclalhost 12345 3 3
```

Дождитесь сообщения `Enter command:`, наберите `start` и нажмите `Enter`.

Запустите в 3-й консоли команду

```
initiator-server.cmd loclalhost 12345 3 3
```

Дождитесь сообщения `Enter command:`, наберите `start` и нажмите `Enter`. В консоли с программой ProxyServer появится сообщение «The Initiator Server has started».

## Особенности реализации

Характерной особенностью задания является то, что прием и передача данных осуществляется одновременно несколькими потоками, причем количество потоков, передающих данные, и потоков, принимающих данные может различаться. Для хранения информации принятой потоками-получателями, до того как ее обработают потоки отправители, используются очереди.

Для того чтобы удовлетворить требованию упорядоченного вывода данных в файлы, применяется специальная реализация очереди RingQueue. В эту очередь данные могут поступать в произвольном порядке, но выходят они строго последовательно. Для хранения данных, которые не могут быть отправлены из-за отсутствия более ранних значений, класс RingQueue использует массив. Если места в массиве не хватает, его размер увеличивается в 2 раза.

Для передачи данных между программами используются TCP-сокеты. Каждый поток-отправитель и каждый поток-получатель устанавливает отдельное соединение с ProxyServer-ом. Потоки сообщают серверу о том, откуда они, и что намерены делать, при помощи специальных управляющих сообщений. Управляющие сообщения передаются по тому же соединению, что и основные данные. Поскольку передаваемые данные представляют собой целые числа, начинающиеся с 0, управляющие сообщения кодируются отрицательными целыми числами.

## Узкие места

Можно составить такой сценарий, при котором система развалится.

1. proxy start
2. echo start
3. initiator start
4. initiator stop
5. initiator exit
6. echo exit
7. echo start
8. initiator start

После завершения 1-го экземпляра инициатора ProxyServer будет считать, что он все еще подключен. Когда к нему пойдут данные от 2-го экземпляра инициатора, часть из них ProxyServer попытается отправить в соединения, оставшиеся от 1-го экземпляра, и эти данные потеряются. 2-й экземпляр, не дождавшись одного из чисел, будет бесконечно копить новые данные в объекте RingQueue, пока у него не кончится память.

Для решения этой проблемы, я делал отправку из прокси в инициатор специальных управляющих сообщений. Если клиент отключился, прокси об этом сразу же узнавал. Но я убрал из системы эту логику, потому что по условию задачи реализовывать такой сценарий не нужно, а мое решение снижало производительность.

## Недостатки

Поскольку управляющие сообщения передаются по тому же каналу, что и основные данные, прокси узнает о том, что инициатор остановился или завершился только после того, как обработает все числа, отправленные инициатором до возникновения события. Решить эту проблему можно, если держать для передачи управляющих сообщений 1 дополнительное соединение между инициатором и прокси, или передавать эти сообщения по UDP.

Поскольку в каждой программе задания присутствуют похожие функции по копированию данных из сети в очередь и из очереди в сеть, то все программы используют общие классы. Один из таких общих классов — Receiver. Он должен работать одинаково во всех программах и не должен содержать логики, специфической для конкретного случая. Реализацию функций, которые делает только одна из программ, Receiver должен делегировать другим классам. Однако мне пришлось в этом классе реализовать вывод сообщений ProxyServer-а о событиях инициатора. В классе Receiver уже есть объект NetworkReceiver, реализующий логику, относящуюся к прокси, но он отвечает только за работу с сокетом, поэтому в нем делать обработку данных тоже неправильно. Поскольку на рефакторинг нет времени, я отметил неудачное место комментарием `TODO`.