1. Ответить на вопросы (письменно):

1. Как организовать клиент-серверное взаимодействие?

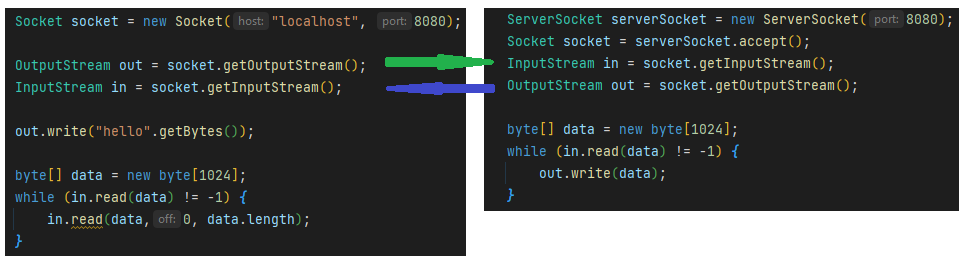
Возможны несколько вариантов клиент серверного взаимодействия в Java:

* 1. java.io.ServerSocket/Socket;

java.io является базовым пакетом встроенным в JDK, данная библиотека облегчает работу с потоками. Именно потоки в данном случае используются в взаимодействии клиент-сервер (**InputStream and OutputStream**).

Мы создаем сущность на стороне клиента java.net.Socket, в конструкторе которого возможно указать host и порт. Так же создаем сущность на стороне сервера java.net.ServerSocket в конструкторе достаточно указать только порт, так как host по умолчанию (localhost) будет применен в соответствии с устройством на котором запущен сервер, далее вызываем метод accept, который возвращает объект класса socket, который используется впоследствии для чтения\записи данных (bytes) при помощи .getOutputStream, getInputStream. Далее идет использование потоков для передачи информации между клиентом и сервером.

IO является типом blocked, то есть поток соединения блокируется пока не будут получены все запрошенные данные. Если stream на запись пуст, то читать из него нечего (запросить данные из канала не получится), а поток простаивает, потребляет ресурсы.



* 1. java.nio ServerSocketChannel/SocketChannel

Как понятно в данном подходе задействован пакет java.nio. NIO – new input output API, более продвинутый способ клиент серверного взаимодействия в сравнении с IO.

Необходимость появления NIO очевидна, так как в IO взаимодействие осуществляется по принципу одно соединение, один поток. Поток является достаточно ресурсоемкой сущностью. Следовательно, для IO все сводится к количеству потоков, которое может требовать весьма большой объём памяти. К примеру, сервер сетевого чата на 1000 пользователей сгенерирует 1000 потоков, но подавляющее количество времени эти потоки будут просто занимать память, так как каждый пользователь не может в режиме реального времени без остановки что-то печатать другому, то есть память используется нецелесообразно. Очевидно, будут простои потоков, в то время как память будет занята.

NIO использует другую абстракцию над соединениями. Имеется один поток, который работает с такой сущностью как selector. Selector в своем время работает с сущностью channel.

- Selector может мониторить несколько каналов сразу;

- Канал нужно регистрировать в селекторе;

- не нужен отдельный поток на каждый канал.

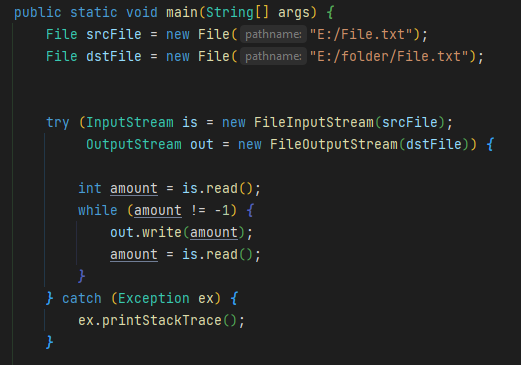
Главное преимущество NIO над IO это то, что не нужен отдельный поток на каждый канал. Потоки в NIO для чтения/записи используется буфер.

NIO является типом non-blocked в любой момент можно запросить данные из канала и получить то, что там есть.

1.3 NETTY фреймворк с сложной структурой и большим количеством подстроек, можно создать модель как IO так и NIO(BLOCKED/NON-BLOCKED)

1. Как и в каком виде передавать файлы?

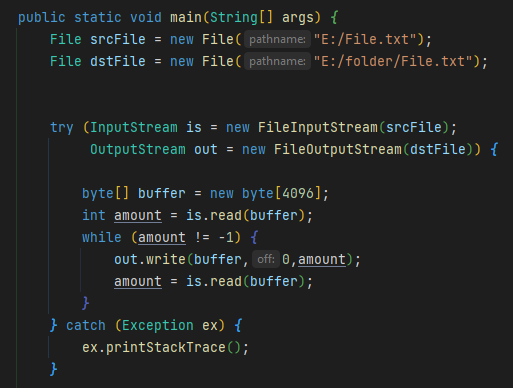
Для передачи или проще говоря копировании файла в java возможно использовать все тот же IO. В OutputStream пишутся байты, а через InputStream читаются.



Для передачи файла по сети можно задействовать клиент-серверное взаимодействие Client-ClientServer.

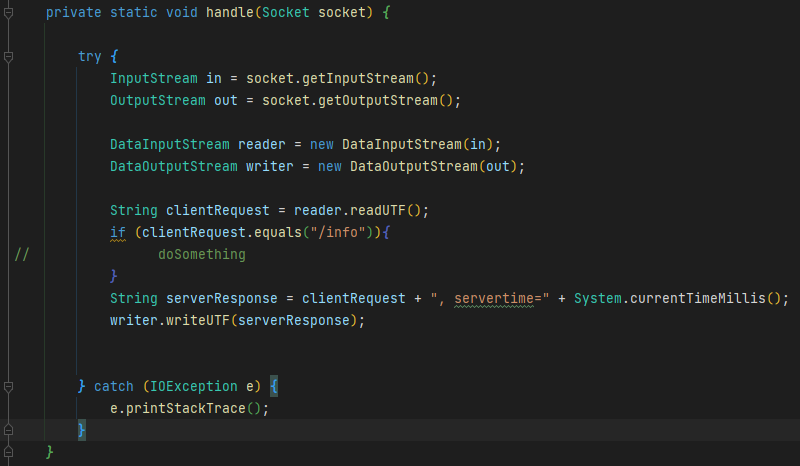
1. Как пересылать большие файлы?

Для передачи больших файлов следует указывать передачу информации байт блоками, к примеру 1024 или 4096.



1. Как пересылать служебные команды?

Аналогичными вверху способами, только команда представляет собой некий String к примеру “/info”. Чтение и запись в поток можно реализовать через DataStream.read(write)UTF:



1. Что хранить в базе данных?

Весьма непонятный вопрос, зависит от требований к данной базе. К примеру, это может быть база пользователей и паролей. Типы данных в большинстве случаев строковые String и целочисленные int.

1. Как передавать структуру каталогов/файлов?

Предполагаю что можно создать объект(с наименованием каталога) у которого есть массив с такими же объектами(подкаталоги). И для данной сущности реализовать интерфейс Serializable. После чего при помощи IO передавать на соответствующий клиент/сервер.

1. Какую библиотеку использовать для сетевого взаимодействия: **java.io**, **java.nio**, **Netty**?

В идеале освоить Netty и применять его, так как это самый универсальный и продвинутый инструмент.