**Теория по 6-й лабораторной**

Оглавление

[Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия. 2](#_Toc115428803)

[Протокол TCP. Классы Socket и ServerSocket. 2](#_Toc115428804)

[Протокол UDP. Классы DatagramSocket и DatagramPacket. 3](#_Toc115428805)

[Классы SocketChannel и DatagramChannel. 3](#_Toc115428806)

[Передача данных по сети. Сериализация объектов. 5](#_Toc115428807)

[Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов. 5](#_Toc115428808)

[Java Stream API. Создание конвейеров. Промежуточные и терминальные операции. 6](#_Toc115428809)

[Шаблоны проектирования: Decorator, Iterator, Factory method, Command, Flyweight, Interpreter, Singleton, Strategy, Adapter, Facade, Proxy. 8](#_Toc115428810)

# Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.

Архитектура состоит из 3 звеньев: Клиент – Сервер -База данных.

Основные протоколы: HTTP/HTTPS, FTP. HTTPS по сути тоже самое, что и HTTP, только он более защищённый. Каждый компьютер имеет свой IP-адрес. FTP и HTTP используются для скачивания и залива файлов в интернете и локальных сетях. Оба протокола работают поверх TCP/IP. **Скорость.** Для одиночного файла небольшого размера и медленного соединения FTP покажет себя лучше. Но при получении нескольких подряд файлов HTTP показывает обычно себя лучше. У HTTP есть автоматическое сжатие трафика, что уменьшает обхем передаваемых данных. Имеется еще конвейерная обработка, которая позволяет быстрее запрашивать несколько файлов с одного и того же сервера. У HTTP нет никаких данных по управлению заливаемых данных. FTP имеет представление о формате файла, поэтому может передавать данные как в ASCII, так и в двоичном виде. HTTP же всегда передает в двоичном виде. Таким образом, FTP умеет преобразовывать данные на лету, если они передаются между системами с разными архитектурами (Windows, Linux). FTP поддерживает скачивание и загрузку, а также восстановление разорванных соединений и продолжение передачи в обеих направлениях. HTTP же восстанавливает только при скачивании. Но в противовес FTP, HTTP поддерживает более продвинутые диапазоны для скачивания. HTTP клиент может держать одно постоянное соединения с сервером для любого количества передач файла. FTP должен создавать новое соединения для каждой новой передачи.

# Протокол TCP. Классы Socket и ServerSocket.

TCP – точно доставит информацию до адресата. Данный протокол работает с потоками. Разделяет на пакеты. Протокол позволяет фрагментировать пакеты данных про технологию передачи. У TCP гибкая система адресации узлов. Эффективно справляется с передачей трафика различного вида.

TCP – это протокол транспортного уровня, а HTTP – это протокол прикладного уровня.

**Socket и ServerSocket**

**ServerSocket –** серверное соединение. Чтобы создать сервер нужно создать ServerSocket и указываем порт.

**Socket** служит уде для соединения клиента с сервером. Один Socket создается на клиенте, где мы уже пишем host и port. Второй создается на сервере: Socket input = serverSocket.accept(). Для обмена информацией создаём stream поток. Со стороны сервера InputStream, а со стороны клиенты – OutputStream.

# Протокол UDP. Классы DatagramSocket и DatagramPacket.

Протоколу не обязательно устанавливать соединение между отправителем и получателем. Информация передается без предварительной проверки готовности принимающей стороны. Такой протокол менее надежен. При это порядок передачи данных не соблюдается. Возможен непоследовательный прием информации. Зато скорость передачи выше, чем в TCP. Для обмена данными отправитель и получатель создают сокеты датаграммного типа – объекты класса **DatagramSocket.** Класс имеет несколько конструкторов, разница которых в том, куда присоединится создаваемы сокет. **DatagramPacket –** класс, который представляет собой пакет датаграмм. Такие пакеты используются для реализации службы доставки пакетов без подключения. Хоть в UDP соединение не устанавливается, но с помощью метода connect(InetAddress addr, int port) у класса DatagramSocket мы можем установить соединение. Устанавливается одностороннее соединение по адресу и порту: либо на отправку, либо на прием. Такое соединение разрывается методом disconnect().

# Классы SocketChannel и DatagramChannel.

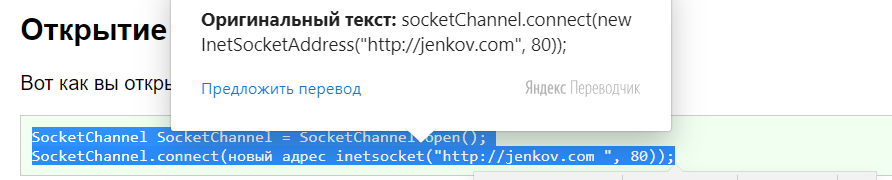
**SocketChannel** – это канал, который подключен к сетевому сокету TCP. Это эквивалент сетевых сокетов. **В чем разница между SocketCahnnel и Socket?** Socket – блокирующий ввод/вывод. Это позволяет блокировать чтение или запись, если базовый буфер заполнен. **SocketCahennel –** не блокирующий способ чтения из сокетов. Так что один поток может одновременно взаимодействовать с несколькими открытыми соединениями.

Есть два вида подключения:

1. Мы открываем SocketChannel и подключаемся к серверу в Интернете.
2. SocketChannel может быть создан, когда входящее соединение поступает на ServerSocketChannel (канал, который может прослушивать входящие TCP-соединения, как и ServerSocket.)

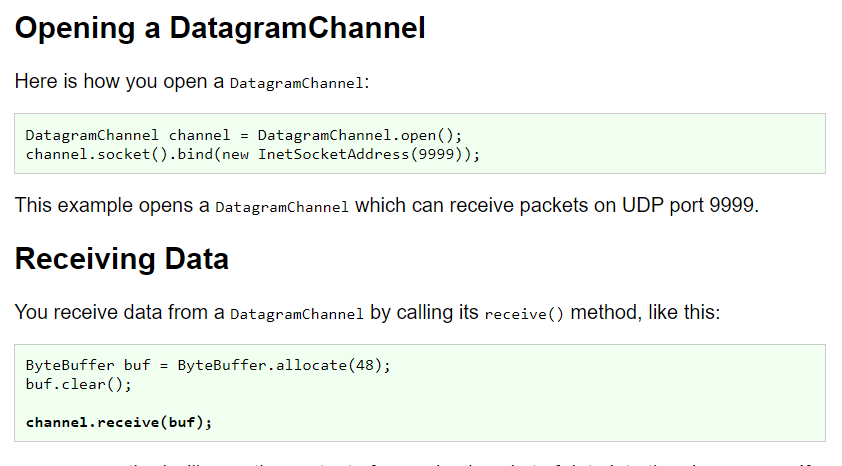
ServerSoacketCahnnel открывается с помощью метода open(), затем ServerSocketChannel.socket().bind(newInetSocketAddress( …)).

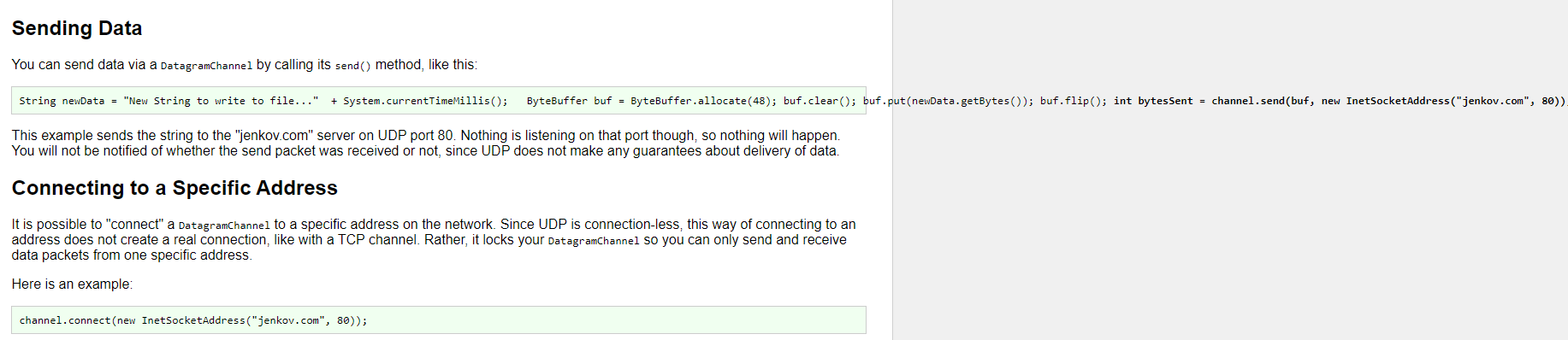
А открытие уже SocketCahnnel идте немного по-другому. Сначала вызывается метод open(). А соединение устанавливается с помощью connect(newInetSocketAddress( …))





**DatagrammChannel –** это канал, который может отправлять и получать UDP-пакеты.





# Передача данных по сети. Сериализация объектов.

**Сериализация –** это процесс сохранения объекта в последовательность байт. Основное назначение сериализации – сохранить состояние объекта для того, чтобы иметь возможность восстановить в случает необходимости.

## Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.

В java за процессы сериализации отвечает интерфейс **Serilizable.** Этот интерфейс крайне прост: чтобы им пользоваться, не требуется реализовывать ни одного метода. Класс сериализуемого объекта должен реализовать данный интерфейс**. Интерфейс Serilizable** говоритмеханизму, что класс может быть сериализован. Сериализация делается вызовом метода writeObject() класса ObjectOutputStream, а восстановление – readObject().

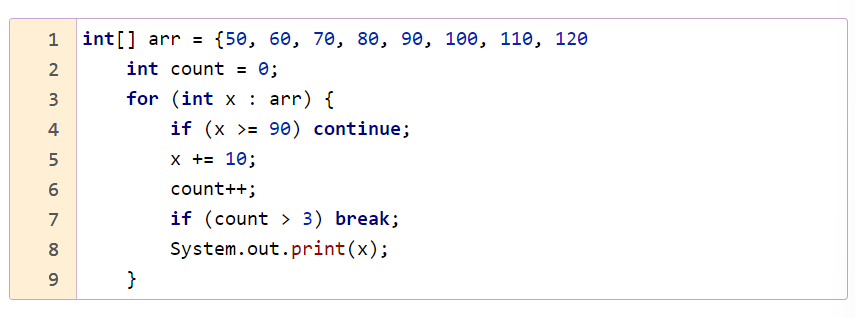
# Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.

**Stream API –** это новый способ работать со структурами данных в функциональном стиле. Stream API – это поток данных. С помощью этого потока программистам стало возножным написание больших частей кода, занимающих много строк, в более коротком.

**Пример с потоком**

IntStream.of(50, 60, 70, 80, 90, 100, 110, 120).filter(x -> x<90).map(x -> x + 10).limit(3).forEach(System.out::print);

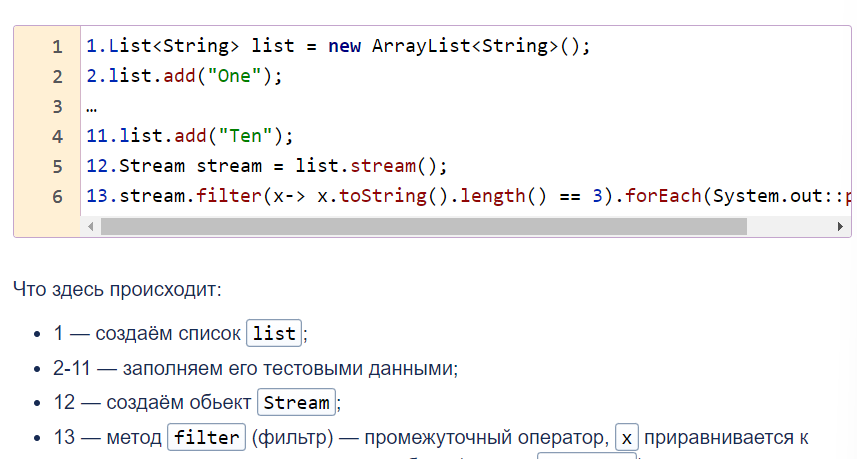
**Пример без потока**

****

Есть такое понятие как операторы (по сути методы класса Stream)

Они делятся на две группы:

1. Промежуточные – обрабатывают поступающие элемент и возвращают стрим. Таких операторов в цепочке обработки элементов может быть много
2. Терминальные – обрабатывают элементы и завершают работу стрима, так что терминальный оператор в цепочке может быть только один.





**Некоторые промежуточные операторы:**

* **Filter –** фильтрует стрим, пропускай только те элементы, что подходят по условию
* **Map –** даёт возможность создать функцию, с помощью которой мы будем изменять каждый элемент и пропускать дальше.
* **Flatmap –** как и в случае с map, служат для преобразования в примитивный стрим.

Различие между терминальными и промежуточными операциями в том, что промежуточная операция является ленивой, а терминальная - нет. Когда мы вызовем операцию в потоке, операция выполнится не сразу. Она выполняется тогда, когда в этом потоке вызываетмя терминальная операция. Все промежуточные операции возвращают поток, а терминальные – нет.

**Stream.filter().forEach(…) -> filter -** промежуточная операция, а **forEach –** терминальная.

**Примеры терминальных операций:**

* forEach
* toArray
* collect
* min
* max
* count
* findFirst

# Шаблоны проектирования: Decorator, Iterator, Factory method, Command, Flyweight, Interpreter, Singleton, Strategy, Adapter, Facade, Proxy.

**Паттерны или шаблоны проектирования –** часть работы разработчика, которую часто недооценивают. Паттерн переводится с английского как «шаблон».

**Паттерны есть трёх видов:**

* **Порождающие**
* **Структурные** (Эти паттерны решают проблемы эффективного построения связей между объектами)
* **Поведенческие** (Эти паттерны решают проблемы эффективного взаимодействия между объектами)

**Builder.** Данный паттерн позволяет создавать сложные объекты пошагово. В java самый известный пример – StringBuilder

Другим известным подходом к созданию объекта – вынос создания в отдельный метод. Называется «**Factory Method».** Который предоставляет интерфейс для создания самого себя. Например compale().

**Decorator.** Динамическинаделяет объект новыми возможностями и является альтернативой субклассирования в области расширения функциональности. Хорошим примером является следующее:

*InputStream inputStream = new BufferedInputStream(new FileInputStream(new File(“ “)))*

**Iterator. Цель этого шаблона –** получение последовательного доступа ко всем элементам составного объекта. Сначала создаем интрефейс Iterator с двумя методами: public boolean hashNext(), public Object next()

**Command.** Цель – создание структуры, в которой класс отправитель и класс получатель зависят друг от друга напрямую.

**Flyweight.**  Для поддержки большого количества мелких объектов. Основная идея – различие между внутренним и внешним состояниями объекта. Внешнее состояние передается клиентом, использующим приспособленца, в некотором контексте. Внутреннее состояние хранится непосредственно в приспособленце и позволяет разделять их. Под разделением понимается возможность одновременной работы нескольких клиентов с одним и тем же приспособленцем.

**Singleton.** Цель его проста - обеспечить единственный экземпляр объекта на всё приложение.

**Proxy.** Этот паттерн позволяет создавать классы, которые являются полной копией другого класса. Эти прокси перехватывают вызов методов основного класса и позволяют сделать что-то до и после вызова метода основного класса. Допустим у меня есть класс, и я хочу, чтобы перед каждым вызовом метода этого класса писалось в консоль название этого метода. Тогда я содаю прокси (просто новый класс, который наследуется от основного) и переопределяю там все методы, в которых я делаю вызов super.method(), а перед ним вывод в консоль.

**Strategy.** Мы можем менять стратегию прямо во время выполнения программы.

**Adapter.** С помощью него мы можем связать два объекта и заставлять их работать, хотя у них может даже и не быть одинакового интерфейса. Например, пишу я программу. В ней есть библиотека, которая скачивает картинки в png, и есть вторая библиотека, которая обрабатывает только jpeg. Значит мне стоит создать адаптер, который превращает png в jpeg.

**Faced.** Нужен для упрощения управления большими библиотеками. Например, я работаю с супер сложной библиотекой. И через неё мне надо получать данные. Но чтобы их получать, мне требуется делать огромное количество вызовов. Потому я создаю фасад, делаю там метод – getData() и прописываю там все это большое количество вывзовов.