Мо́дульное программи́рование — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам.[1] Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Мо́дуль — функционально законченный фрагмент программы.

**1) Сетевое взаимодействие - клиент-серверная архитектура, основные протоколы, их сходства и отличия.**

Клиент-серверная архитектура: клиенты сначала запрашивают ресурсы с сервера, затем сервер обратно отправляет соответствующие ответы на запросы клиентов. Чтобы взимодействие происходило, необходим механизм коммуникации посредством сокетов.

Протокол – правила обмена информацией, которые описывают, как образом должны обмениваться взаимодействующие стороны.

Пакет – определенно оформленный блок данных, передаваемый по сети.

Разметка пакета: управляющая информация и данные пользователя (полезная нагрузка).

Управляющая информация содержит адрес отправителя, адрес получателя, коды обнаруженных ошибок (контрольных сумм) и информацию об очередности.

Сравнение пакетов и дейтаграмм:

Пакет распространяется на любое сообщение, форматированное как пакет, тогда как дейтаграмма обычно используется для пакетов ненадежных служб (не уведомляет о доставке).

Мультиплексирование данных – транспортный уровень способен одновременно обрабатывать несколько потоков данных между двумя системами.

Сетевая модель OSI (The Open Systems Interconnection model):



Уровни:

\* Прикладной – обеспечивает взаимодействие сети и пользователя. Уровень разрешает приложениям пользователя иметь доступ к сетевым службам, таким, как обработчик запросов к базам данных, доступ к файлам, перессылке электронных сообщений, а также передачу служебной инфомации (информации об ошибках), формирует запросы к уровню представления. (HTTP, FTP, SMTP, SNMP)

\* Представления – отвечает за преобразование протоколов и кодирование/декодирование данных (полученные данные с прикладного уровня преобразует в формат, понятный адресату). Также на этом уровне может осуществляться сжатие/распаковка или кодирование/декодирование данных, а также перенаправление запросов другому сетевому ресурсу. (XDR)

\* Сеансовый – отвечает за поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Отвечает за создание/завершение сеанса, синхронизацию задач, определением права на передачу данных и поддержание самого сеанса в период неактивности приложений. (X.225 или ISO 8327)

\* Транспортный – предназначен для доставки данных (представляет собой механизм передачи). Длинные запросы разбиваются на несколько мелких, а короткие объединяются в один, восстаналивается порядок запросов. (TCP, UDP, SCTP)  
\* Сетевой уровень – предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имен в физические, определяет кратчайшие маршруты, коммуникацию и маршрутизацию (проход данных через перемычки до тех пор, пока они не достигнут адресата), отслеживание неполадок и заторов в сети. Сетевой уровень отвечает за запросы обслуживания от Транспортного уровня и направляет запросы обслуживания на Канальный уровень. (IPv4/IPv6, IPX, ICMP)

В IPv4 применяются числовые методы адресации, а в и IPv6 — буквенно-числовые. Длина адреса IPv4 составляет 32 бита, у IPv6 — 128 бит. ... В IPv4 используются уникальные публичные и «частные» адреса для трафика.

\* Канальный уровень (уровен передачи данных) – предназначен для передачи данных узлам, находящимся в том же сегменте локальной сети. Также используется для обнаружения и устранения ошибок, возникающих на физическом уровне. Отвечает за доставку кадров между устройствами, подключенными к одному сегменту. На таком уровне работают коммутаторы и мосты. Основная функция – обеспечение доступа.

Подуровень управления логической связью (LLC – logical link control – управление передачей данных, проверка и правильность передачи информации по соединению)

Кадр – заголовок содержит аппаратные адреса (MAC – уникальный идентификатор в сети Ethernet) отправителя и получателя, чтобы определить от какого устройства был отправлен, а куда получен.

\* Физический уровень – описывает способы передачи бит через физические среды линий связи, соединяющие сетевые устройства. На этом уровне описываются параметры сигнала, решаются воспросы, связанные с синхронизацией, избавлением от помех, скорость передачи.

Модель TCP/IP – сетевая модель передачи данных, представленных в цифровом виде, на нем базируется интернет. Название базируется на двух протоколах семейства (TCP – Transmission Contol Protocol, IP – Internet Protocol).

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объемом задействованных сетей.

Cлои абстракции:

\* Уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети;

\* Интернет-уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями;

\* Транспортный уровень, обрабатывающий связь между хостами;

\* Прикладной уровень, который обеспечивает обмен данными между процессами для приложений.

IP-пакеты состоят из заголовка и полезной нагрузки. Заголовок пакета IPv4 состоит из:

* 4 бита содержат версию пакета: IPv4 или IPv6.
* 4 бита содержат длину интернет-заголовка, которая измеряется отрезками по 4 байта (например, 5 означает 20 байт).
* 8 бит содержат тип обслуживания, известный также как качество обслуживания (QoS), описывающее приоритеты пакета.
* 16 бит содержат длину пакета в байтах.
* 16 бит содержат тег идентификации, помогающий восстановить пакет из нескольких фрагментов.
* 3 бита содержат нуль, флаг разрешения фрагментации пакета (DF: не фрагментировать), а также флаг разрешения дальнейшей фрагментации (MF: фрагментировать дальше).
* 13 бит содержат смещение фрагмента, поле для идентификации положение фрагмента в исходном пакете.
* 8 бит содержат время жизни (TTL), которое определяет количество переходов (через маршрутизаторы, компьютеры и сетевые устройства), разрешённых сделать пакету, прежде чем он исчезнет (например, пакету с TTL 16 разрешено пройти не более 16 маршрутизаторов, чтобы добраться до места назначения).
* 8 бит содержат протокол (TCP, UDP, ICMP и т. д.).
* 16 бит содержат контрольную сумму заголовка, используемую при обнаружении ошибок.
* 32 бит содержат IP-адрес источника.
* 32 бит содержат адрес назначения.

UDP дейтаграммы:

Заголовок пакета определяет тип данных, номер пакета, общее количество пакетов и [IP-адреса](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81) отправителя и получателя.

HTTP – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

TCP — ориентированный на соединение протокол, что означает необходимость «рукопожатия» для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

* Надёжность — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
* Упорядоченность — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
* Тяжеловесность — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
* Потоковость — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

UDP — более простой, основанный на сообщениях протокол без установления соединения. Протоколы такого типа не устанавливают выделенного соединения между двумя хостами. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя. В приложениях для голосовой связи через интернет-протокол (Voice over IP, TCP/IP) UDP имеет преимущество над TCP, в котором любое «рукопожатие» помешало бы хорошей голосовой связи. В VoIP считается, что конечные пользователи в реальном времени предоставят любое необходимое подтверждение о получении сообщения.

* Ненадёжный — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
* Неупорядоченность — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
* Легковесность — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.
* Датаграммы — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.
* Нет контроля перегрузок — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

**2) Протокол TCP. Классы Socket и ServerSocket.**

[Transmission Control Protocol — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/Transmission_Control_Protocol)

TCP (Transmission Control Protocol) – пакеты в TCP называются сегментами.

Механизм:

Поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий оджного пакета, гарантируя тем самым целостность передаваемых данных и уведомление отправителя о результатах передачи.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Структура заголовка** | | | | |
| [**Бит**](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) | **0 — 3** | **4 — 6** | **7 — 15** | **16 — 31** |
| **0** | Порт источника, **Source Port** | | | Порт назначения, **Destination Port** |
| **32** | Порядковый номер, **Sequence Number (SN)** | | | |
| **64** | Номер подтверждения, **Acknowledgment Number (ACK SN)** | | | |
| **96** | Длина заголовка, (**Data offset**) | Зарезервировано | Флаги | Размер Окна, **Window size** |
| **128** | Контрольная сумма, **Сhecksum** | | | Указатель важности, **Urgent Point** |
| **160** | Опции (необязательное, но используется практически всегда) | | | |
| **160/192+** | Данные | | | |

**Sequence number** -измеряется в байтах, и каждый переданный байт полезных данных увеличивает это значение на 1.

**Флаг SYN** (идет установление сессии), то поле содержит изначальный порядковый номер. В целях безопасности изначальный порядковый нормер генерируется случайным образом.

Номер подтвержения (**Acknowledgment Number (ACK SN)**) – если установлен соответствующий флаг, то это поле содержит порядковый номер байта, который отправитель данного сегмента желает получить (значит от ISN+1 до ACK-1) уже были получены.

Размер окна - размер данных, которые может адресат принять.

Указатель важности – если установлен соответствующий флаг URG, то поле указывает номер байта, которым заканчиваются важные данные.

|  |  |
| --- | --- |
| **SYN-RECEIVED** | Сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения |
| **ESTABLISHED** | Соединение установлено, идёт передача данных |
| **CLOSED** | Начальное состояние узла. Фактически фиктивное |

Установка соединения:

Процесс начала сеанса TCP (также называемый «рукопожатие» (англ. handshake)), состоит из трёх шагов. 1.

* Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN.
* Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.
* В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.
* В случае неудачи сервер посылает клиенту сегмент с флагом RST.

2. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.

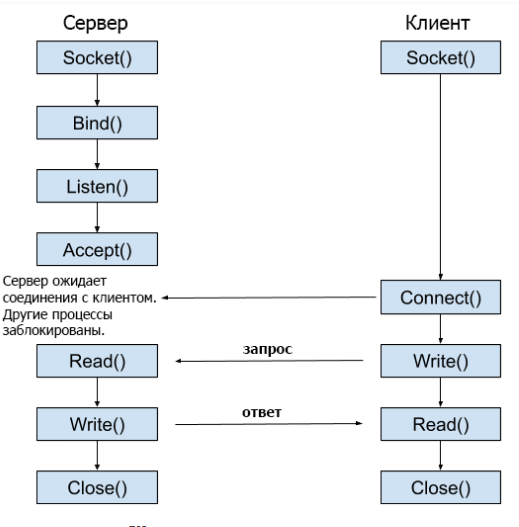
* Если клиент одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.
* Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.
* Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

3. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

* В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

Сокет – программная (логическая) конечная точка, устанавливающая двунаправленную коммуникацию

между сервером и одной или нескольними клиентскими программами.

\* Socket() — на сервере создается конечная точка для

коммуникации.

\* Bind() — сокету присваивается уникальный номер и

для него резервируется уникальная комбинации

IP-адреса и порта.

\* Listen() — после создания сокета сервер ожидает

подключения клиента.

\* Accept() — сервер получает запрос на подключение

от клиентского сокета.

\* Connect() — клиент и сервер соединены друг с

другом.

\* Send()/Recieve() — обмен данными между

клиентом и сервером.

\* Close() — после обмена данными сервер и

клиент разрывают соединение.

Адрес подразумевает под собой идентификатор машины в пространстве сети Internet. Он может быть доменным именем, например, «javarush.ru», или обычным IP.

Порт — уникальный номер, с которым связан определённый сокет проще говоря, его занимает определённая служба для того что бы по нему могли связаться с ней.

Класс Socket: реализует идею сокета для клиента.

InetAddress – работает как с доменными адресами, так и с IP.

AutoClosable – можно использовать в конструкции try-with-resources.

Методы:

* InetAddress getInetAddress() – возвращает объект содержащий данные о сокете. В случае если сокет не подключен – null
* int getPort() – возвращает порт по которому происходит соединение с сервером
* int getLocalPort() – возвращает порт к которому привязан сокет. Дело в том, что «общаться» клиент и сервер могут по одному порту, а порты, к которым они привязаны – могут быть совершенно другие
* boolean isConnected() – возвращает true, если соединение установлено
* void connect(SocketAddress адрес) – указывает новое соединение
* boolean isClosed() – возвращает true, если сокет закрыт
* boolean isBound() - возвращает true, если сокет действительно привязан к адресу
* InputStream getInputStream() – получаем поток, как бы связанный с сервером на получение. Если его обернуть в Reader, то мы можем прочитать то, что было отправлено с сервера.
* OutputStream getOutputStream() – аналогично только для отправки.

Класс ServerSocket: предназначен для сервера. Ожидает запроса, чтобы его обработать и отправить результат клиенту.

Можно указать максимум соединений, принимаемых сервером (по умолч. 50)

Методы:

* accept() – подключаемся сервером к клиенту.

When the communication between the Server and Client ends, you can call the close () method of the Socket class to close the Socket and tear down the connection.

ServerSocket is generally only used to set the port number and listen. The real communication is between the server socket and the client socket. After accepting the ServerSocket, the initiative is transferred.

Socket socket = ss.accept();

InetAddress для локального сервера:

* InetAddress.getByName(null) - null указывает на локальный хост
* InetAddress.getByName("localhost")
* InetAddress.getByName("127.0.0.1")

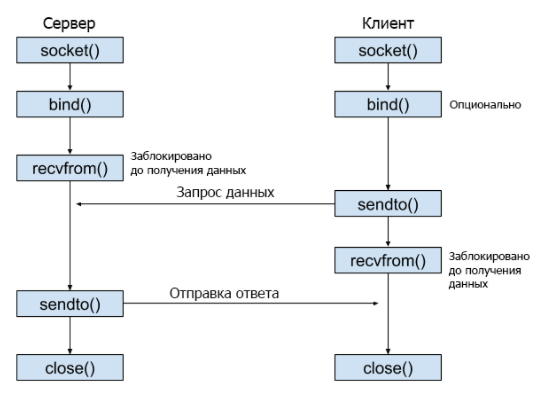
Класс SocketChannel: предназначен для того, чтобы от него создавать сокеты (который открыт, но не подключен) или через него можно самостоятельно отправлять или получать byteBuffer.

Дополнительные методы:

* int read(ByteBuffer []) – читает данные в соответствующий буфер.
* int write(ByteBuffer []) – пишет.

**3) Протокол UDP. Классы DatagramSocket и DatagramPacket.**

UDP (User Datagram Protocol) – с помощью него приложения могу посылать сообщения (датаграммы) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных. Кроме того, запрос, что данные пришли или отправились не посылаются на сокеты. Также нет контроля перегрузок (в случае высокой пропускной способности).

socket() — прежде всего сокет определяется как для сервера, так и для клиента. Это не обязательно должно происходить одновременно. Чтобы объяснение было исчерпывающим, я буду на каждом этапе обсуждать действия как сервера, так и клиента.

bind() — сокету, который определен, присваивается идентификатор и порт на работающей машине. Для клиентского сокета это необязательно, потому что даже если клиентский сокет не привязан, привязка осуществляется автоматически всякий раз, когда клиент инициирует подключение к серверу.

recvfrom() — после привязки к порту компьютера серверный сокет ожидает подключения от клиентского сокета. Тем временем дальнейшее выполнение текущего потока останавливается (блокируется) до тех пор, пока серверный сокет не получит соединение. То же самое происходит и с клиентским сокетом при ожидании ответа сервера.

sendto() — после соединения с клиентом серверный сокет отправляет данные клиенту. Этот же метод используется клиентским сокетом для выполнения запроса на подключение к серверу.

close() — после успешного обмена данными оба сокета закрываются, т.е. освобождаются ресурсы системы, выделенные для сокетов.

Класс DatagramSocket: является контрольной точкой (сокетом) для отправки и получения пакетов (датаграмм).

InetAddress

Методы:

* void bind(SocketAddress addr) связывает сокет с определенным портом и адресом.
* void close () – закрывает сокет.
* void connect() – связывает сокет с удаленным адресом для этого сокета.
* void disconnect() – обрывает связь.
* get,set ...
* void receive(DatagramPacket p) – получить пакет
* void send(DatagramPackat p) – отправить пакет.

Класс DatagramPacket: используются для отправки и получения между точками. Отправка пакетов не гарантирована, порядок тоже.

[DatagramPacket (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/net/DatagramPacket.html)

DatagramPacket(byte[] buf, int length, [InetAddress, offset, port])

Структура датаграммы:

|  |  |  |
| --- | --- | --- |
| **Биты** | 0 - 15 | 16 - 31 |
| **0-31** | Порт отправителя (Source port) | Порт получателя (Destination port) |
| **32-63** | Длина датаграммы (Length) | Контрольная сумма (Checksum) |
| **64-...** | Данные (Data) | |

IPv4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Биты** | **0 — 7** | **8 — 15** | **16 — 23** |  |
| **0** | Адрес источника | | | |
| **32** | Адрес получателя | | | |
| **64** | Нули | Протокол | Длина UDP | |
| **96** | Порт источника | | Порт получателя | |
| **128** | Длина | | Контрольная сумма | |
| **160+** | Данные | | | |

IPv6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Биты** | **0 — 7** | **8 — 15** | **16 — 23** | **24 — 31** |
| **0** | Адрес источника | | | |
| **32** |
| **64** |
| **96** |
| **128** | Адрес получателя | | | |
| **160** |
| **192** |
| **224** |
| **256** | Длина UDP | | | |
| **288** | Нули | | | Следующий заголовок |
| **320** | Порт источника | | Порт получателя | |
| **352** | Длина | | Контрольная сумма | |
| **384+** | Данные | | | |

**4) Отличия блокирующего и неблокирующего ввода-вывода, их преимущества и недостатки. Работа с сетевыми каналами.**

[Основные отличия Java IO и Java NIO / Хабр (habr.com)](https://habr.com/ru/post/235585/)

|  |  |
| --- | --- |
| **IO** | **NIO** |
| Потокоориентированный | Буфер-ориентированный |
| Блокирующий (синхронный) ввод/вывод | Неблокирующий (асинхронный) ввод/вывод |
|  | Селекторы |

При блокирующем вводе-выводе блокировка происходит до тех пор, пока данные не будут считаны или записаны.

При неблокирующем вводе-выводе есть возможность запрашивать считанные данные из канала и получать только то, что доступно на данный момент, или вообще ничего.

Каналы – это логические (не физические) порталы, через которые осуществляется ввод/вывод данных, а буферы являются источниками или приёмниками этих переданных данных.

Селекторы в Java NIO позволяют одному потоку выполнения мониторить несколько каналов ввода

IO:

ByteBuffer buffer = ByteBuffer.allocate(48);

int bytesRead = inChannel.read(buffer);

Наиболее важные каналы:

* FileChannel: Чтение и запись данных из файлов (только режим блокировки).
* DatagramChannel: Чтение и запись данных в сети через UDP (блокирование и неблокирование по желанию).
* SocketChannel: Чтение и запись данных в сети через TCP (блокирование и неблокирование по желанию).
* ServerSocketChannel: Он может отслеживать новое входящее TCP-соединение, как веб-сервер. Для каждого нового входящего соединения создается SocketChannel.

[Java IO: буферно-ориентированный, синхронный, неблокирующий ввод-вывод (NIO) - Русские Блоги (russianblogs.com)](https://russianblogs.com/article/50581572834/)

**5) Классы SocketChannel и DatagramChannel.**

[SocketChannel (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/nio/channels/SocketChannel.html)

[DatagramChannel (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/nio/channels/DatagramChannel.html)

Класс SocketChannel: предназначен для того, чтобы от него создавать сокеты (который открыт, но не подключен. Чтобы отправлять, получать данные, необъходимо связать connectом) или через него можно самостоятельно отправлять или получать byteBuffer.

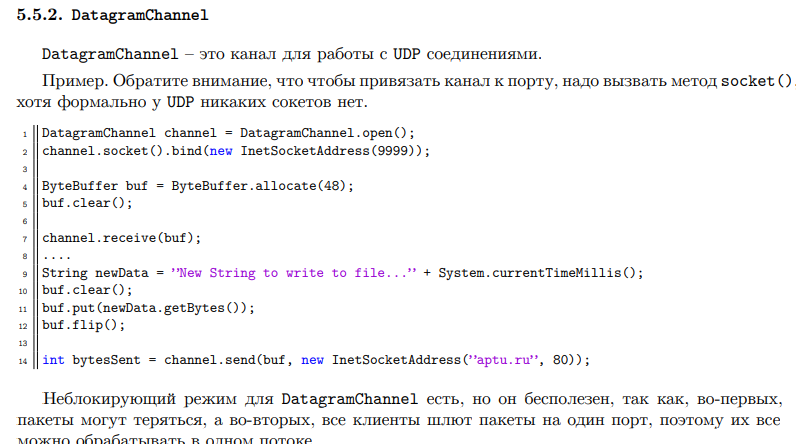
Дополнительные методы:

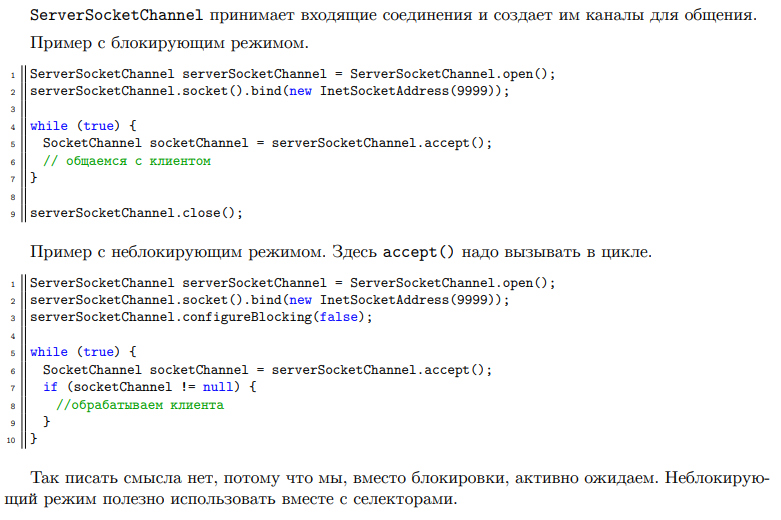
* int read(ByteBuffer []) – читает данные в соответствующий буфер.
* int write(ByteBuffer []) – пишет.

Та же идея и с DatagramChannel.

[java - Socket vs SocketChannel - Stack Overflow](https://stackoverflow.com/questions/14225957/socket-vs-socketchannel)

Главное отличие Socket от SocketChannel это то, что первый является блокирующим, в то время к SocketChannel поддерживает неблокирующее соединение и асинхронное завершение работы.





**6) Передача данных по сети. Сериализация объектов.**

Передача данных по сети уже готова.

Serialization (сериализация) – процесс перевода объектов в последовательность байтов, по которой затем его можно полностью восстановить. (дает жизнь объекту между запусками программы, а также сохранение кроссплатформенности)

Так как механизм сериализации связан с базовой системой ввода/вывода и переводит объект в поток байтов, для его выполнения необходимо создать выходной поток OutputStream, упаковать его в ObjectOutputStream и вызвать метод writeObject(). Для восстановления объекта нужно упаковать InputStream в ObjectInputStream и вызвать метод readObject().

Чтобы объект сериализовать, должны быть сериализуемы и все его зависимости.

//Сериализация в файл с помощью класса ObjectOutputStream

ObjectOutputStream objectOutputStream = new ObjectOutputStream(

new FileOutputStream("person.out"));

objectOutputStream.writeObject(igor);

objectOutputStream.writeObject(renat);

objectOutputStream.close();

// Востановление из файла с помощью класса ObjectInputStream

ObjectInputStream objectInputStream = new ObjectInputStream(

new FileInputStream("person.out"));

Person igorRestored = (Person) objectInputStream.readObject();

Person renatRestored = (Person) objectInputStream.readObject();

objectInputStream.close();

//Сериализация с помощью класса ByteArrayOutputStream

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

ObjectOutputStream objectOutputStream2 = new ObjectOutputStream(byteArrayOutputStream);

objectOutputStream2.writeObject(igor);

objectOutputStream2.writeObject(renat);

objectOutputStream2.flush();

//Восстановление с помощью класса ByteArrayInputStream

ObjectInputStream objectInputStream2 = new ObjectInputStream(

new ByteArrayInputStream(byteArrayOutputStream.toByteArray()));

Person igorRestoredFromByte = (Person) objectInputStream2.readObject();

Person renatRestoredFromByte = (Person) objectInputStream2.readObject();

objectInputStream2.close();

**7) Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.**

В процессе сериализации вместе с сериализуемым объектом сохраняется его граф объектов. Т.е. все связанные с этим объектом, объекты других классов так же будут сериализованы вместе с ним.  
При указании полей и методов ключевым словом transient, они не сериализуются.

private static final long serialVersionUID. Запоминает версии сериализованного класса, чтобы при загрузке версии сериализованного объекта и класса совпадали. (or InvalidClassException)

Помимо интерфейса Serializable существует интерфейс Externalizable, где можно явно задавать, как сериализуется объект. Однако если в данном классе не найдется публичного конструктора по умолчанию, то выбросится ошибка, и объект не будет десереализован.

[Сериализация в Java. Не все так просто / Хабр (habr.com)](https://habr.com/ru/post/431524/#:~:text=%D0%9D%D0%B5%20%D0%B2%D1%81%D0%B5%20%D1%82%D0%B0%D0%BA%20%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE,-%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%2C&text=%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20(Serialization)%20%E2%80%94%20%D1%8D%D1%82%D0%BE%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81,%D0%B7%D0%B0%D1%82%D0%B5%D0%BC%20%D0%B5%D0%B3%D0%BE%20%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE%20%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D1%81%D1%82%D1%8C%D1%8E%20%D0%B2%D0%BE%D1%81%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%82%D1%8C.&text=%D0%95%D1%81%D0%BB%D0%B8%20%D0%B2%D0%B0%D0%BC%20%D0%BD%D0%B5%D0%BE%D0%B1%D1%85%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%82%D1%8C%20%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82,%D0%B8%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%82%D1%8C%20%D0%BF%D0%BE%20%D1%81%D0%B5%D1%82%D0%B8%20%D0%BF%D0%BE%D0%BB%D1%83%D1%87%D0%B0%D1%82%D0%B5%D0%BB%D1%8E.)

**8) Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.**

[Шпаргалка Java программиста 4. Java Stream API / Блог компании Luxoft / Хабр (habr.com)](https://habr.com/ru/company/luxoft/blog/270383/)

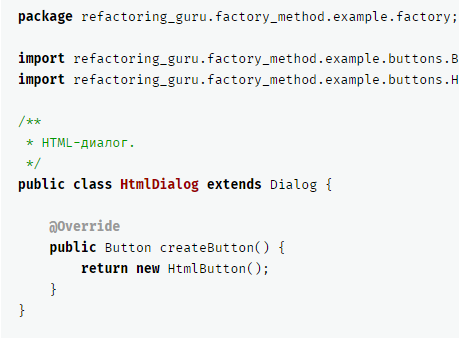
[Определения Java - Google Таблицы](https://docs.google.com/spreadsheets/d/1H8VuFExEf8oJY-12-8nE0YYpyMPLrkxnGffosa8wXmU/edit#gid=0)

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

Iterator

Просто пишем итератор. Можно указать методы hasNext, next, reset и т.п. В коде написать их реализацию.

Factory Method

Command

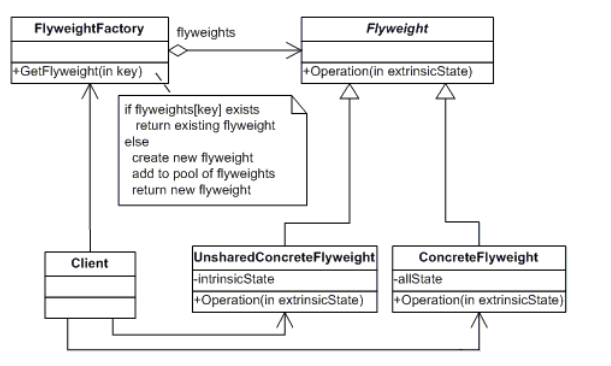
<https://javarush.ru/groups/posts/1684-pattern-command-svoimi-slovami>



Flyweight

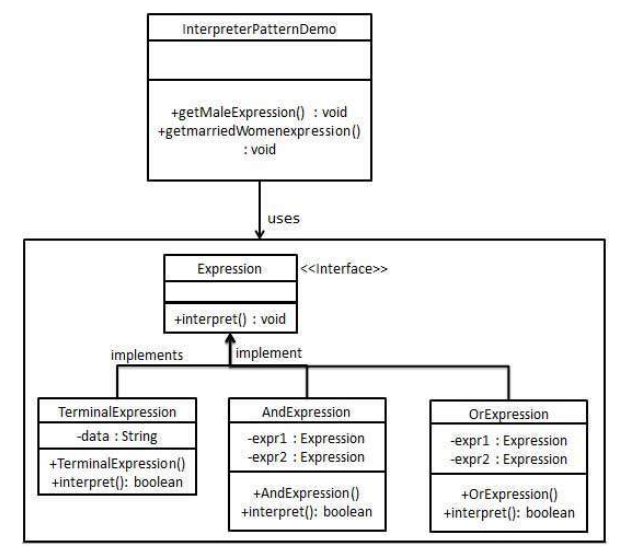
Легковес — это структурный паттерн, который экономит память, благодаря разделению общего состояния, вынесенного в один объект, между множеством объектов.

Легковес позволяет экономить память, кешируя одинаковые данные, используемые в разных объектах.



Interpreter

<https://www.tutorialspoint.com/design_pattern/interpreter_pattern.htm>



Singleton

<https://refactoring.guru/ru/design-patterns/singleton/java/example>

Strategy Java

<https://refactoring.guru/ru/design-patterns/strategy/java/example>

Пример: выбор оплаты картой или наличными через один интефейс, который выполняет функцию pay.

Adapter

Адаптер — это структурный паттерн, который позволяет подружить несовместимые объекты.

Адаптер выступает прослойкой между двумя объектами, превращая вызовы одного в вызовы понятные другому.

<https://refactoring.guru/ru/design-patterns/adapter/java/example>

Façade

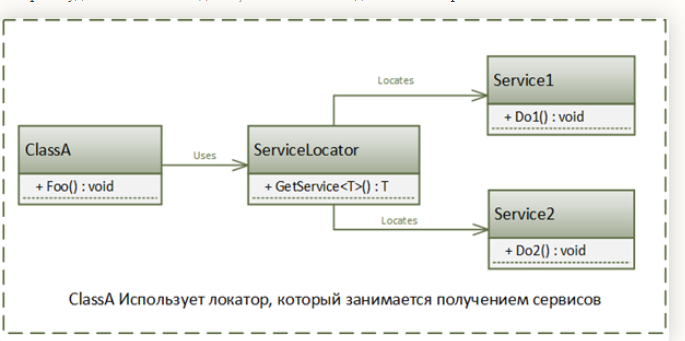
Фасад — это структурный паттерн, который предоставляет простой (но урезанный) интерфейс к сложной системе объектов, библиотеке или фреймворку.

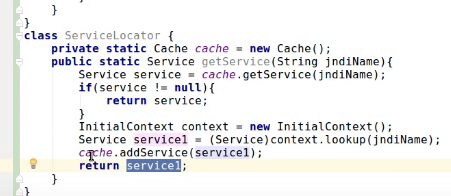
Кроме того, что Фасад позволяет снизить общую сложность программы, он также помогает вынести код, зависимый от внешней системы в единственное место.

Proxy

Заместитель — это объект, который выступает прослойкой между клиентом и реальным сервисным объектом. Заместитель получает вызовы от клиента, выполняет свою функцию (контроль доступа, кеширование, изменение запроса и прочее), а затем передаёт вызов сервисному объекту.

ServiceLocator





Как фабрика (сингтонская), можно организовать кеширование, чтобы не создавать кучу сервисов