ТИНЬКОФФ

Стандартная библиотека JDK

Работа с датами, файлами и сетью

Победённый Алексей

11.2021

Коммуникация

- Во время чтения отдельных секций не вижу реакции и комментарии.
- Будут мини перерывы на каждом слайде-отбивке и слайде с вопросами.
- Можете задавать вопросы в чате толка или поднимать руку.

Примеры слайдов с остановками

Кратко о себе

Образование

Специалист по защите информации, кандидат наук.

Карьера

Работаю Teamlead-ом в отделе Комплаенс с мая 2021.

Общий опыт разработки 20 лет.

Преподавание

Преподавал «Системное программирование на C», читал лекции на Fintech.middle и в Академии бэкенда.

Периодически выступаю на локальных митапах.

Что было на прошлой лекции?

Все о строках и регулярках

Детально разобрали класс String и регулярные выражения

java.util.Date

Посмотрели как работать с датами при помощи класса java.util.Date

Работа с файлами

Коротко разобрались с File и InputStream/OutputStream

Что ждет нас сегодня?

01 Работа с датами и временем

Рассмотрим пакет java.time для работы с датами. Разберемся с часовыми поясами

02 Работа с файлами

Разберемся с классом Path, узнаем что такое бинарная сериализация данных, рассмотрим nio и MemoryMapped-файлы

03 Работа с сетью

Поговорим про сокеты, узнаем как с помощью nio реализовать неблокирующее сетевое взаимодействие, рассмотрим HttpClient

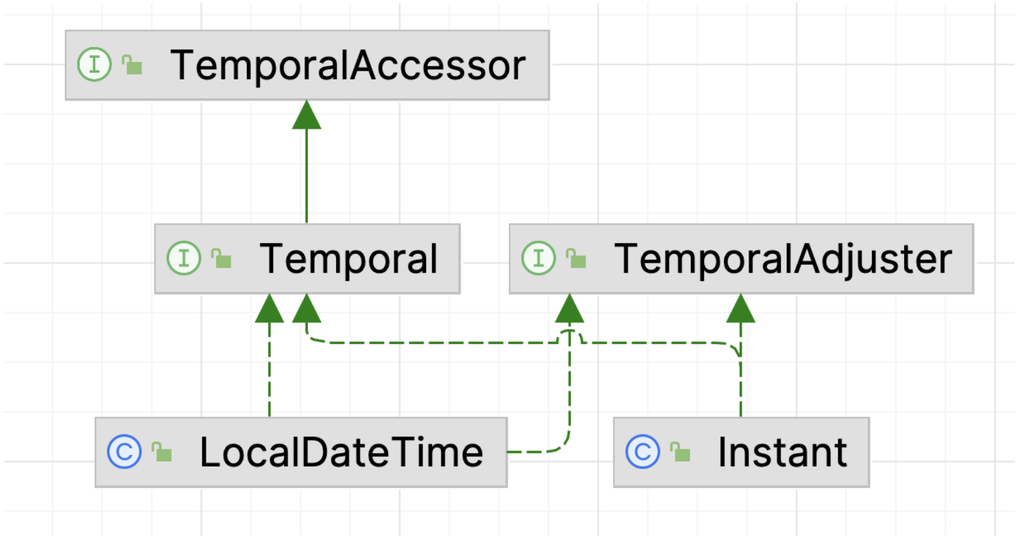
ТИНЬКОФФ

Работа с датами. Часть 2

Две версии API

Date-time classes in Java	Modern class	Legacy class
Moment in UTC	java.time. **Instant**	java.util. ~~**Date**~~ java.sql. ~~**Timestamp**~~
Moment with offset-from-UTC (hours-minutes-seconds)	java.time. **OffsetDateTime**	(lacking)
Moment with time zone (`Continent/Region`)	java.time. **ZonedDateTime**	java.util. ~~**GregorianCalendar**~~
Date & Time-of-day (no offset, no zone) Not a moment	java.time. **LocalDateTime**	(lacking)

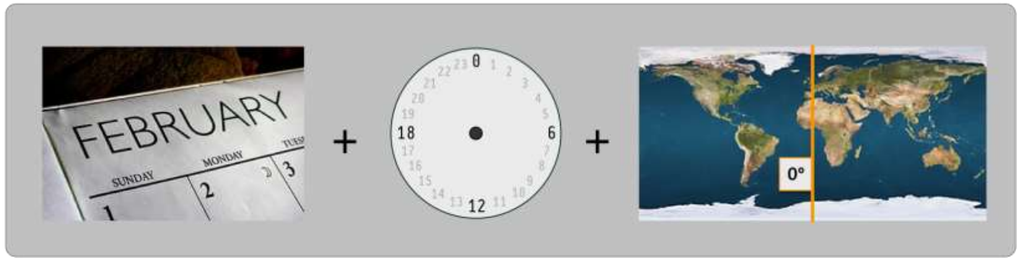
Instant/ LocalDateTime



Дерево наследования у обоих классов одинаковое

Instant/ LocalDateTime

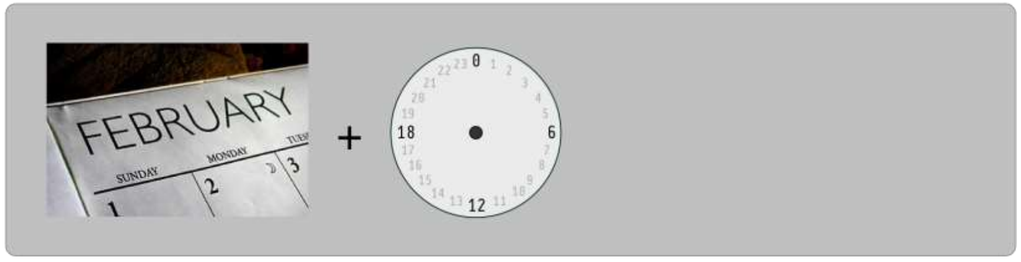
Instant



Instant - момент времени на временной шкале

- хранит количество секунд, прошедших с 01.01.1970 + количество наносекунд прошедших в текущей секунде
- фактически это момент времени по Гринвичу, без учета часового пояса

LocalDateTime



LocalDateTime - текущая дата+время без учета часовых поясов

- хранит отдельно дату как день, месяц, год; отдельно время (часы, минуты, секунды)
- для чего: напоминание «сделать зарядку завтра в 12:00»

TimeZone

Почему важны

- Один и тот же момент UTC будет соответствовать разным часам и даже дням в разных таймзонах. Пример. 00:00 по Гринвичу будет соответствовать 03:00 того же дня в Мск и 20:00 предыдущего дня в Монреале (Канада)
- В некоторых регионах есть зимнее/летнее время

ZoneId

ZoneId - timezone

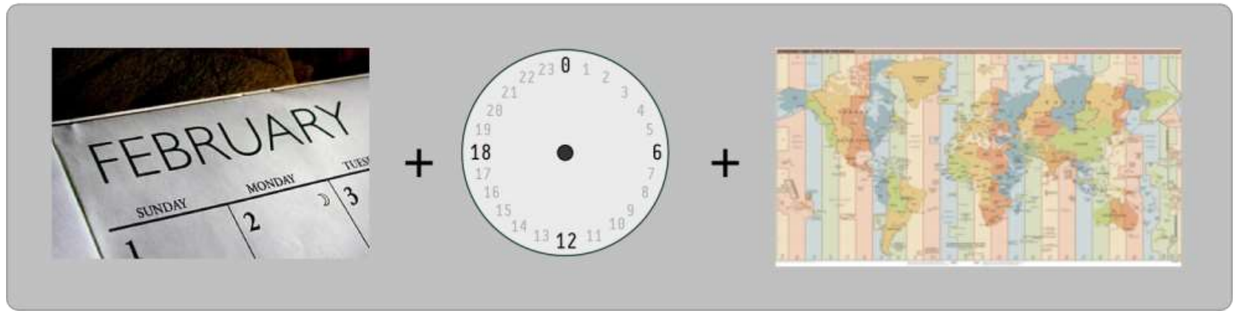
- Определяет правила преобразования Instant к LocalDateTime для конкретного часового пояса.
- Может быть фиксированным смещением (например EST – Eastern Standart Time), либо георграфическим регионом, например Europe/Moscow

ZonedDateTime/ OffsetDateTime

ZonedDateTime

- Instant + ZoneId
- Позволяет получить день, час, минуты в конкретном часовом поясе

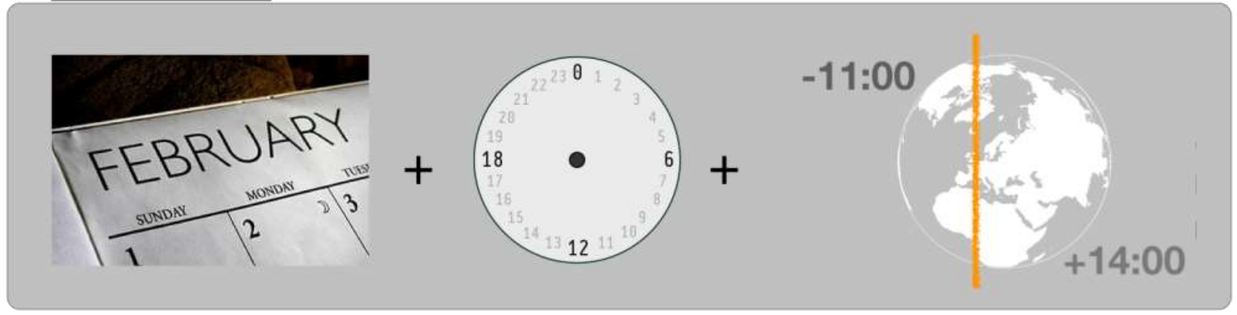
ZonedDateTime



OffsetDateTime

- Instant + ZoneOffset
- Позволяет получить день, час, минуты для конкретного смещения

OffsetDateTime



Методы Instant

(коротко)

Получение текущего момента

- Instant.now();
- Clock.systemUTC().instant();

Получение части момента

- Instant.now().getLong(ChronoField.INSTANT_SECONDS);
- Instant.now().getLong(ChronoField. NANO_OF_SECOND);
- Instant.now().atZone(ZoneId.systemDefault()).getDayOfWeek();

Преобразовать к Zoned/OffsetDateTime

- atZone(ZoneId);
- atOffset(ZoneOffset);

Изменить

- plus/minus
- truncatedTo – обнуляет поля момент после переданного

java.time.Clock

Clock – поставщик instant

- используется во всех методах получения текущей даты/времени
- возвращает Instant

Какие бывают

- Clock.tick() – отсчитывает время интервалами Duration
- Clock.fixed() – всего возвращает один и тот же Instant (для тестирования)
- Clock.offset() – возвращает время со смещением

Методы LocalDateTime (коротко)

Создать конкретную дату/время

```
LocalDate ld = LocalDate.of( 2023 , Month.NOVEMBER , 7 ) ;
LocalTime lt = LocalTime.MIN ; // 00:00:00
LocalDateTime ldt = LocalDateTime.of( ld , lt ) ;
LocalDateTime.now();
```

Преобразовать к Zoned/OffsetDateTime

- atZone(ZoneId);
- atOffset(ZoneOffset);

Изменить

- plus/minus
- truncatedTo – обнуляет поля момент после переданного

Duration

Duration – время между двумя событиями

```
Instant first = Instant.now();
Thread.sleep(1_000);
Instant second = Instant.now();
Duration.between(first, second);
```

Можно создать из строки

- Duration.parse(“PT3H”);
- Duration.parse(“P3D”);
- Duration.parse(“-PT10M”); //может быть отрицательным

Изменить

- plus/minus
- truncatedTo – обнуляет поля момент после переданного
- multipliedBy – увеличивает продолжительность в указанное число раз

Period

Period – Дни/месяца/года между событиями

```
LocalDate first = LocalDate.now();
LocalDate second = first.plusDays(3);
Period.between(first, second);
```

Можно создать из строки

- Period.parse(“P5D”);
- Period.parse(“-P1Y”);

Изменить

- plus/minus
- multipliedBy – увеличивает период в указанное число раз

DateTimeFormatter

Форматирует или парсит дату

```
LocalDate date = LocalDate.now();
var formatter = DateTimeFormatter.ofPattern("yyyy MM dd");
String text = date.format(formatter);
LocalDate parsedDate = LocalDate.parse(text, formatter);
```

Есть несколько уже готовых форматов

- Например DateTimeFormatter.ISO_LOCAL_DATE_TIME

currentTimeMillis vs nanoTime

В чем отличие, что использовать

- System.nanoTime возвращает значение системного, монотонно-возрастающего счетчика наносекунд
- System.currentTimeMillis. Возвращает wall clock и не гарантирует монотонно-возрастающее значение из-за clock-drift.

ClockDrift

- Для вычисления локального времени используются кварцевые генераторы, которые имеют среднюю ошибку ~1 секунды в 10 дней
- Для синхронизации времени на многих вычислительных узлах используется протокол NTP. При корректировке время может измениться как в болшую, так и в меньшую сторону

Лекция Клеппманна о ClockDrift

ТИНЬКОФФ

Работа с файлами. Часть 2

IO vs NIO

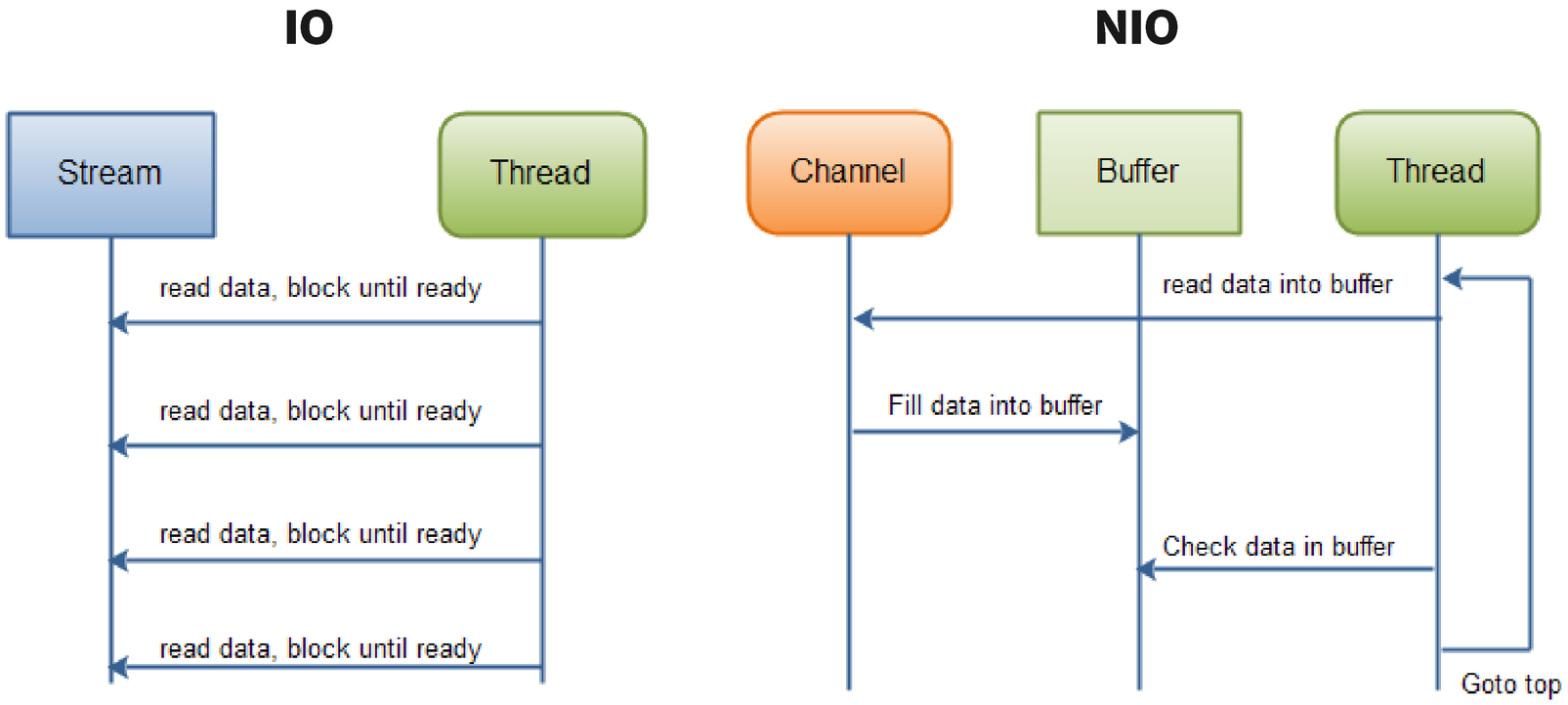
IO

- потоко-ориентированный. Есть поток на чтение, есть поток на запись. Поток читается последовательно - нельзя перемещаться вперед/назад по потоку. Кеширование прочитанных данных нужно реализовывать в коде
- блокирующий. Вызов методов read/write приостанавливает текущий поток до тех пор пока данные не будут считаны/записаны

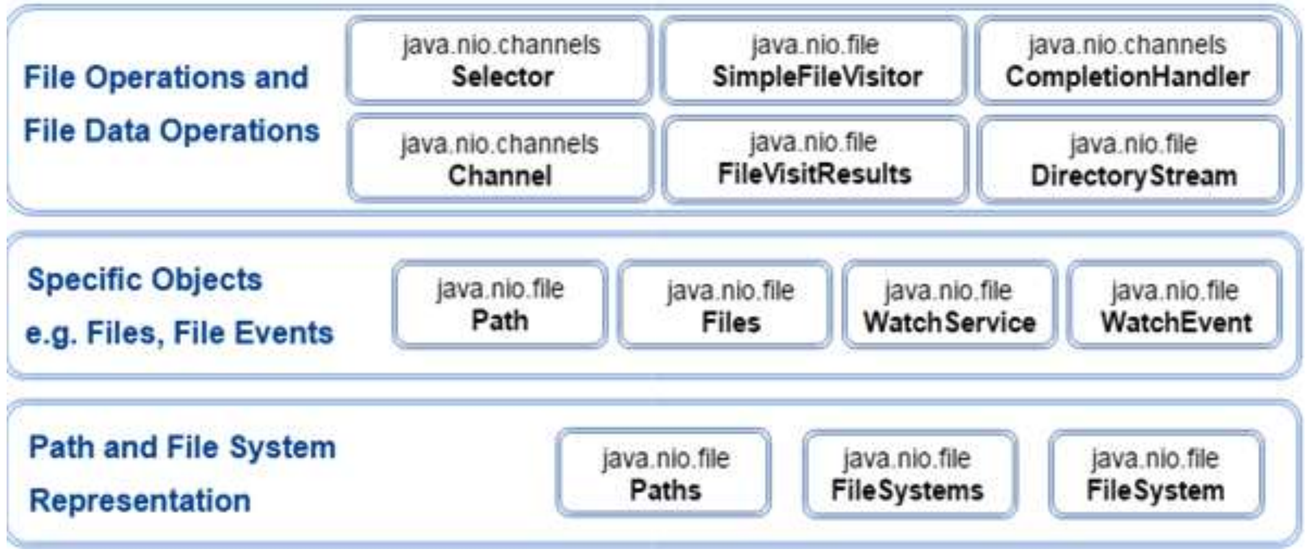
NIO

- Ориентирован на работу с буфером. Данные вычитываются в буфер по которому программа свободно может перемещаться вперед/назад (при условии что нужные данные поместились в буфер)
- неблокирующий. При чтении поток получает в буфере только то, что доступно на текущий момент. Если ничего нет, то нужно вернуться позже. При записи достаточно положить данные в буфер, не блокируясь.
- один thread может обрабатывать несколько источников входных данных, переключаясь между ними по мере появления данных (selectors)

IO vs NIO


IO
Stream
Thread
read data, block until ready
read data, block until ready
read data, block until ready
read data, block until ready
NIO
Channel
Buffer
Thread
read data into buffer
Fill data into buffer
Check data in buffer
Goto top

NIO Основные классы



- **Paths, Path – представление пути к ресурсу (файлу)**
- **Channel – канал поступления и/или отправки данных**
- **Selector – мультиплексор channel-ов. В каждый момент времени выбирает для работы один из каналов, который имеет готовые для обработки данные**
- **WatchService – позволяет получать события от ФС, например, появление нового файла в директории**

Path как замена строк

Жизнь до Path

File f = new File(ROOT_DIR + config.getConfDir() + “\\” + “settings.yml”);

- нужно следить за разделителями
- неудобно конструировать пути
- риск ошибок

Жизнь с Path

Path p = Paths.get(ROOT_DIR, config.getConfDir()).resolve(“settings.yml”);

- Может указать на файл рядом с текущим, например

Paths.get(“c:\\1.txt”).resolveSibling(“2.txt»); // c:\\2.txt

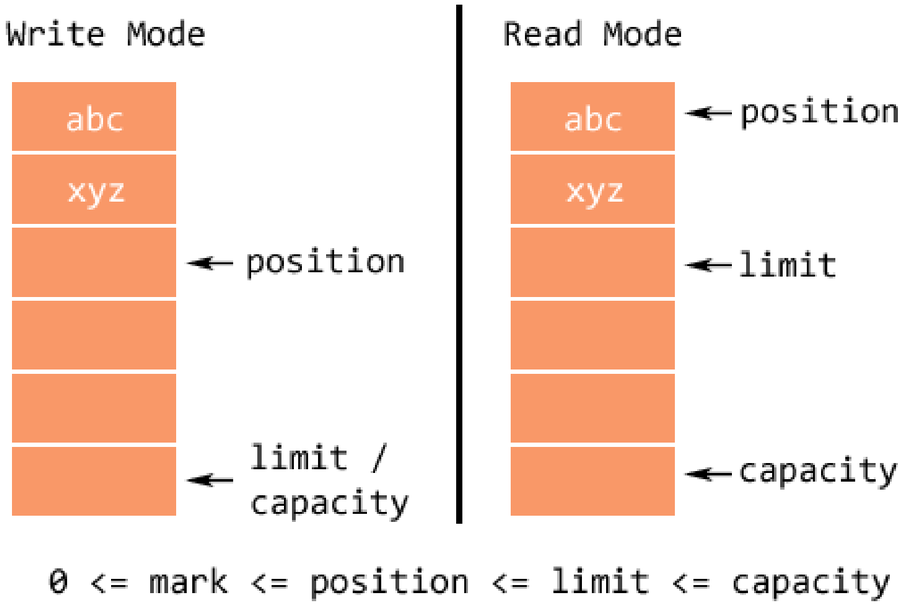
- Умеет работать с URI
- java.nio.file. Files содержит кучу удобных методов работы с файлами, которые принимают именно Path, например

Files.createDirectories(path) – создаст все несуществующие директории в пути

Files.createFile(path) – создаст файл по пути

var out = Files.newOutputStream(path, CREATE, APPEND) - открывает OutputStream в файл по пути

ByteBuffer



- Capacity – максимальный размер буфера, указанный при создании
- limit – при записи = capacity, при чтении указывает сколько данных максимально можно считать
- position – где сейчас читаем/пишем

- ByteBuffer#flip – переключает в режим чтения. Limit = position; position = 0

FileChannel

FileChannel – channel для записи, чтения файлов

```
public static void main(String[] args) throws IOException {
    //в файле 3 байта
    FileChannel inChannel = FileChannel.open(Path.of( first: "nio.txt"));

    ByteBuffer buf = ByteBuffer.allocate( capacity: 48);

    int bytesRead = inChannel.read(buf); //position = 3, bytesRead = 3
    while (bytesRead != -1) {
        buf.flip(); // limit = 3, position = 0

        while(buf.hasRemaining()) { // position < limit
            System.out.print((char) buf.get()); // position++
        }

        buf.clear(); //position = 0
        bytesRead = inChannel.read(buf);
    }
    inChannel.close();
}
```

ТИНЬКОФФ

Вопросы?

MemoryMappedFile

MemoryMappedFile – область памяти, которая байт-в-байт отражает содержимое части или всего файла

- ОСь берет на себя заботу по синхронизации области памяти и данных на диске
- С одной областью памяти могут работать одновременно несколько процессов
- Высокая скорость работы с такими файлам
- Нужные части файла автоматически загружаются в память, ненужные выгружаются.

Это позволяет работать с очень большими файлами

```
public static void main(String[] args) throws Exception {
    try (RandomAccessFile file = new RandomAccessFile( name: "memory_mapped.dat",  mode: "rw")) {
        MappedByteBuffer out = file.getChannel()
                .map(FileChannel.MapMode.READ_WRITE,  position: 0, length);

        for (int i = 0; i < length; i++) {
            out.put((byte) 'x');
        }
    }
}
```

ТИНЬКОФФ

Сериализация объектов

ObjectOutputStream/ ObjectInputStream (LEGACY)

Позволяет превратить объект в массив байт

```
var demoObject = new DemoClass( number: 1,  name: "test");
ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
try (var oos = new ObjectOutputStream(byteArrayOutputStream)) {
    oos.writeObject(demoObject);
}
byteArrayOutputStream.close();
System.out.println(Arrays.toString(byteArrayOutputStream.toByteArray()));
```

Требования

- Класс должен имплементить интерфейс Serializable
- Пишутся non-transient поля
- Желательно определить поле serialVersionUID – версия класса.

```
try(var ois = new ObjectInputStream(new ByteArrayInputStream(toByteArray))) {
    var obj = (DemoClass)ois.readObject();
    System.out.println(obj);
}
```

Externalizable

- Класс может имплементить инетерфейс Externalizable, в этом случае логика чтения/записи реализуется в классе в методах readExternal/writeExternal

ТИНЬКОФФ

Работа с сетью

Sockets

Socket

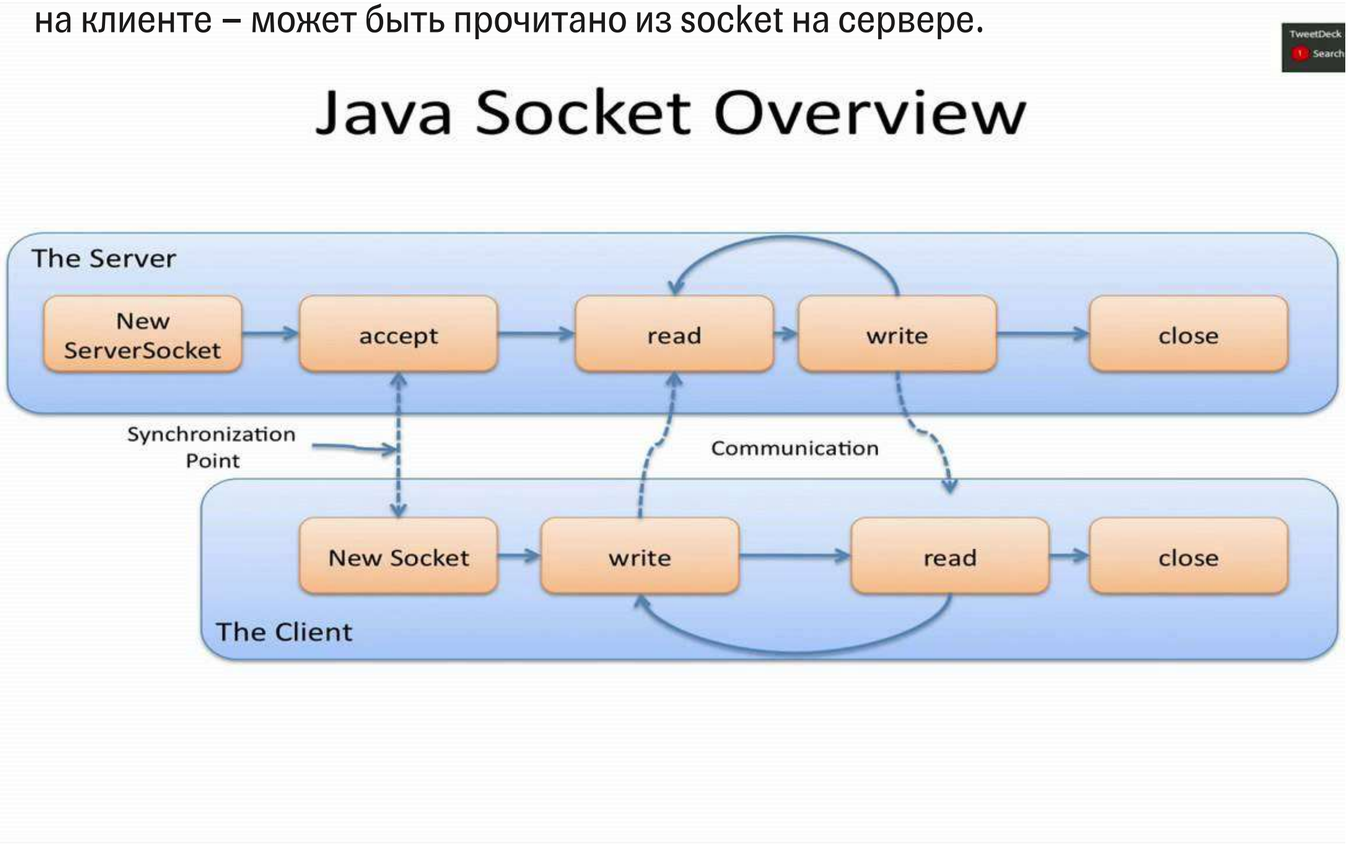
Представляет собой одно клиентское сетевое подключение к серверу

ServerSocket

Представляет собой входящую точку для приема клиентских подключений

Accept

Прием (accept) подключения создает «зеркальный» socket. То, что записывается в socket на клиенте – может быть прочитано из socket на сервере.



Server

Простейшая реализация сервера

```
ServerSocket server = new ServerSocket( port: 18080);
var socket = server.accept();
try (var writer = new PrintWriter(socket.getOutputStream(), autoFlush: true)) {
    writer.println("Sending test data");
}
socket.close();
server.close();
```

accept блокирует поток, пока не подключится клиент

Чтобы обработать более одного подключения необходимо:

- выполнять accept в цикле

- каждый полученный от accept сокет отдавать на обработку в отдельный поток

Модель Thread-per-request. Невозможно обработать больше одновременных запросов, чем есть обслуживающих потоков

Client

Простейшая реализация клиент

```
Socket client = new Socket(InetAddress.getByName( host: "localhost"),  port: 18080);
var response = new BufferedReader(new InputStreamReader(client.getInputStream())).readLine();
System.out.println(response);
```

InputStream возвращает то, что отправляет сервер

InputStream можно читать до тех пор, пока сервер не выполнит на своей стороне socket.close(), либо пока есть необходимость

NIO Selectors

Selector

Единая точка обработки событий от одного или нескольких channel

SelectableChannel#register

Метод SelectableChannel, который регистрирует его в селекторе с указанием того, какие события будут попадать в селектор

```
Selector selector = Selector.open();
ServerSocketChannel serverSocket = ServerSocketChannel.open();
serverSocket.bind(new InetSocketAddress( hostname: "localhost",  port: 18080));
serverSocket.configureBlocking(false);
serverSocket.register(selector, SelectionKey.OP_ACCEPT);
```

Selector#select

Блокирует поток и возвращает список ключей (SelectionKey) соответвующих channel-ам, имеющим интересующие события. Каждый ключ связан с конкретным channel-ом.

```
while (true) {
    selector.select();
    Set<SelectionKey> selectedKeys = selector.selectedKeys();
    Iterator<SelectionKey> iter = selectedKeys.iterator();
    while (iter.hasNext()) {

        SelectionKey key = iter.next();

        if (key.isAcceptable()) {
```

ТИНЬКОФФ

Вопросы?

ТИНЬКОФФ

HttpClient

URI/URL

URI – Universal Resource Identifier

это строка символов, которая используется для идентификации какого-либо ресурса по его адресу или по его имени, либо по тому и тому вместе

URL – Universal Resource Locator

это строка символов, которая используется для идентификации какого-либо ресурса, но только по его адресу, по его местоположению

HttpRequest

HttpRequest – builder для запроса HTTP

это строка символов, которая используется для идентификации какого-либо ресурса по его адресу или по его имени, либо по тому и тому вместе

```
var request = HttpRequest.newBuilder()
        .uri(new URI( str: "https://postman-echo.com/get"))
        .GET()
        .header( name: "AcceptEncoding",  value: "gzip")
        .timeout(Duration.of( amount: 10, SECONDS))
        .build();
```

- HttpRequest.Builder содержит удобные методы для создания GET/POST/PUT/DELETE запросов
- Метод header задает заголовки, передаваемые с запросом
- Метод timeout задает время в течение которого ожидается ответ. В случае превышения времени ожидания бросается HttpTImeoutException

HttpClient

HttpClient сам клиент

Задача клиента – отправить на сервер сформированный HttpRequest, распарсить запрос и вернуть HttpResponse

```
var response = newHttpClient()
        .send(request, HttpResponse.BodyHandlers.ofString());
```

Метод **send** – блокирующий. Есть его неблокирующий аналог – **sendAsync,** который возвращает CompletableFuture<HttpResponse<T>>

BodyHandler определят как обрабатывать тело ответа, например:
- **BodyHandlers.ofString()** – возвращает тело как строку
- **BodyHandlers.discarding()** – игнорирует тело ответа
- **BodyHandlers.ofByteArray()** – возвращает тело ответа как массив байт
- **BodyHandlers.ofInputStream()** – возвращает тело как InputStream

ТИНЬКОФФ

Остальное

Utils

Properties

Позволяет читать конфигурационные файлы в формате ключ=значение.

SecureRandom

В отличие от Random, который в качестве начального значения использует текущее время, SecureRandom использует системных источник случайных значений. Вероятность предсказать следующее значение – низкая.

Math vs StrictMath

StrictMath предоставляет методы для расчета тригонометрические, логарифмические операции и квадратного корня.

Math делает все то же самое, но при этом оставляет возможность вернуть приблизительный результат для увеличения перформанса.

JEP-442

Начиная с Java 21 предоставляет возможность вызова функций, написанных не на Java, без JNI. Кроме того улучшенное управление внешней памятью (off-heap)

ТИНЬКОФФ

Спасибо!