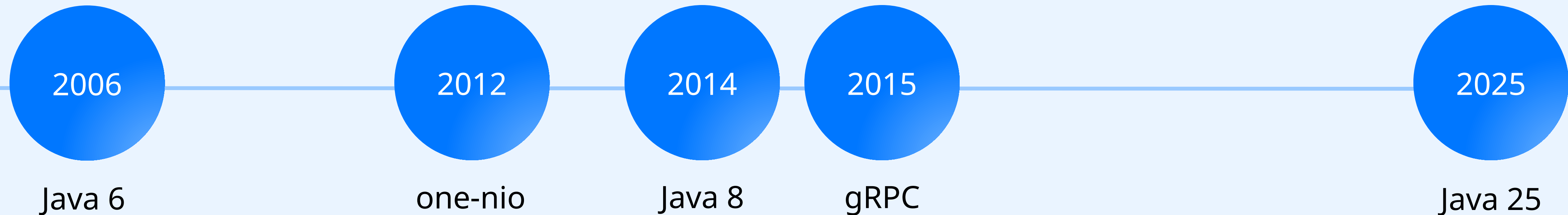
# Сериализация one-nio. От истоков к поддержке JDK 25

## MagicAccessorImpl, Unsafe, VarHandles

Михаил Богданов
Ведущий разработчик

# Кратко о библиотеке




| Одноклассники 2026 | |
|---|---|
| DAU | ~18 млн |
| RPS | 750 000 |
| Количество сервисов | 1 200 |
| Количество реплик | 12 000+ |
| Используемые виртуальные ядра в облаке | 150 000+ |

# О чём будет рассказ

1. Зачем была нужна ещё одна библиотека сериализации?
2. Немного о внутренней архитектуре one-nio и используемых подходах
3. Unsafe, MagicAccessorImpl, VarHandles
4. Почему мы вдруг начали что-то менять!?
5. Побенчмаркаем
6. Посмотрим в возможное будущее стандартной сериализации

# Сериализация вокруг нас

Клиент

Сервер

Method m, Object[] args

Десериализация

byte[] request

Сериализация

Method m, Object[] args

Исполнение

Object result

Десериализация

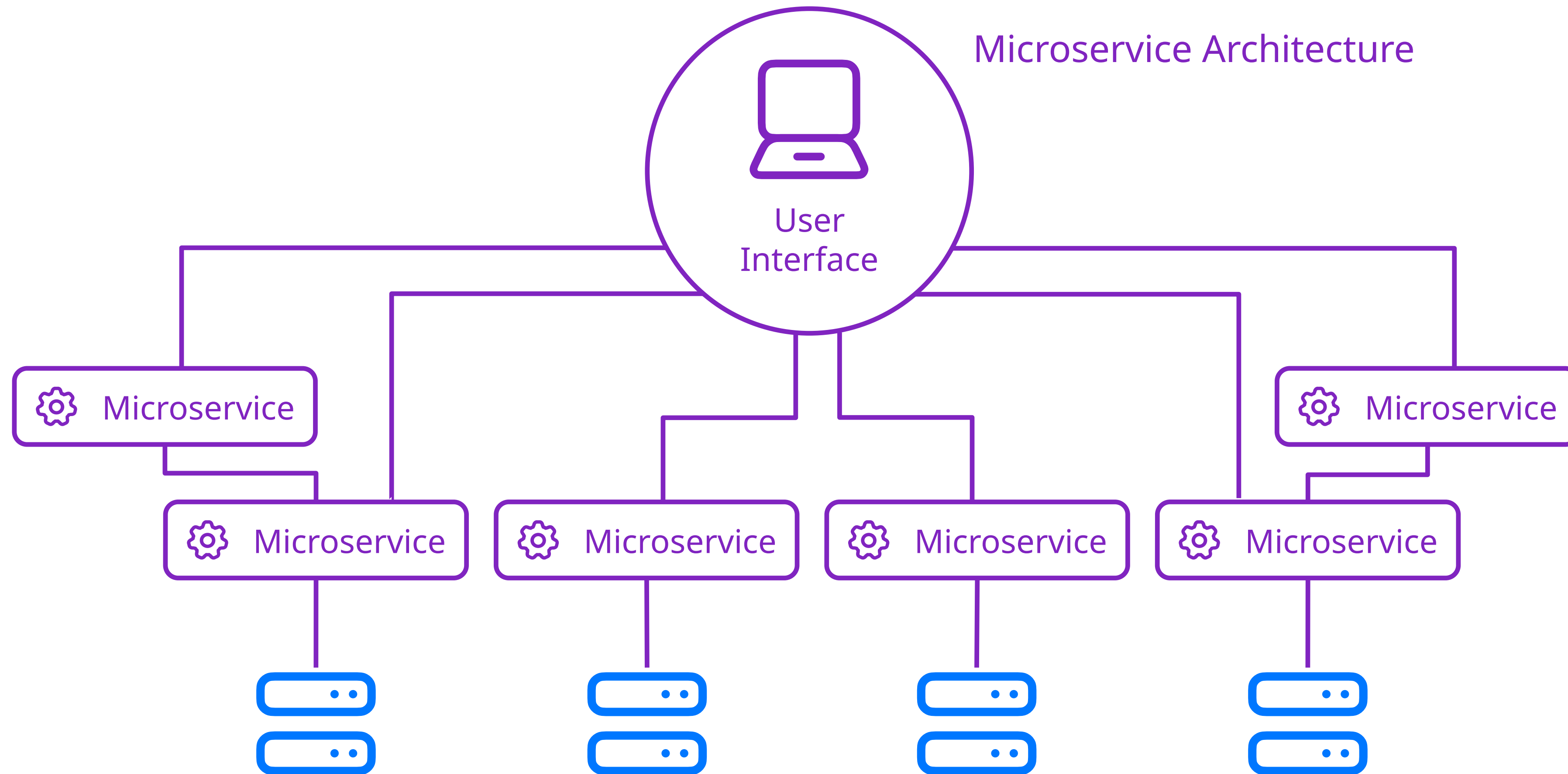
byte[] response

Object result

Сериализация

А.Паньгин "Особенности разработки высоконагруженного сервера на Java», Июль 2012
https://www.youtube.com/watch?v=gIh0X-RkftY

# Рост сетевого трафика и нагрузки на оборудование



# Рост сетевого трафика и нагрузки на оборудование

Помимо роста коммуникаций, появилось системы перерабатывающие террабайты данных:

Hadoop Kafka Cassandra

Им требовались **компактные** форматы хранения и передачи данных

# Рост сетевого трафика и нагрузки на оборудование

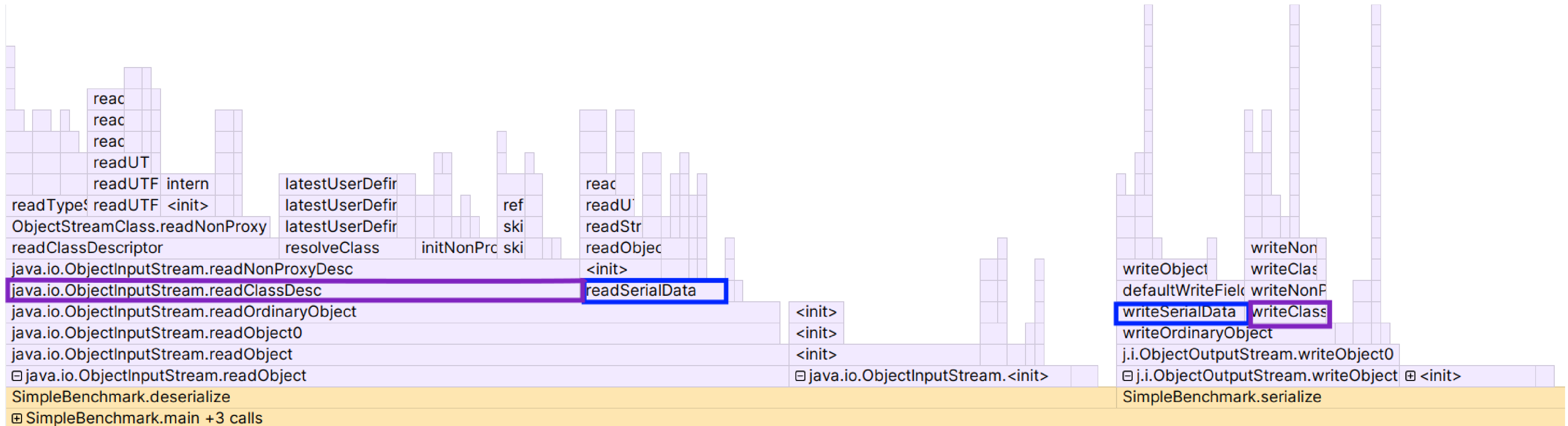
Стандартная сериализация создавалась в 90-х годах с прицелом на **максимальную универсальность**, и это стало её ограничением в новом мире распределённых систем:

тяжёлые метаданные классов (заголовки, дескрипторы, информация о базовых классах)

# Flame Graph стандартной сериализации

Data

- final String finalField
- String nonFinalField



# Ограничения JSON Rest API

Оверхед на текстовый формат (JSON): Передаем лишние байты (ключи, кавычки, скобки). Число, время передаем как строки

Дорогой парсинг (CPU tax): дорогое обратное преобразование строк в числа.....

Тяжеловесность HTTP/1.1: Передача текстовых заголовков в каждом запросе (Host, User-Agent, Accept, Connection, Cookie, Authorization, Content-Type, Content-Length...)

Плоский формат JSON: не умеет работать с множественными ссылками и циклами из коробки

# Ответ на новые требования

Ответом в конце 2000 – первую половину 2010 стало появление множества новых библиотек сериализации

| Год | Библиотека | Разработчик / Экосистема |
|---|---|---|
| 2007 | **Thrift** | Facebook/Apache |
| 2008 | **Protobuf (v2)** | Google |
| 2008 | **MessagePack** | Sadayuki Furuhashi |
| 2009 | **BSON** | MongoDB |
| 2009 | **Avro** | Hadoop |
| 2009 | **Kryo** | Esoteric Software |
| 2010 | **Jackson Smile** | FasterXML |
| 2012 | **One-nio** | Одноклассники (А. Паньгин) |
| 2012 | **FST** | Ruediger Moeller |
| 2013 | **Cap'n Proto** | Kenton Varda (ex-Google) |
| 2014 | **FlatBuffers** | Google |

# Что хотелось получить на выходе

Быстрая

Компактная

Минимум ручной работы

Поддержка эволюции схемы (прямая и обратная совместимость)

# Результат

- **Быстрый** фреймворк с **компактной** сериализацией
- **Автоматический вывод схем** по классам и их **преобразование**
- **2 режима работы**: с запись схемы и без записи в выходной поток
- **Фундамент** Java-сервисов в компании

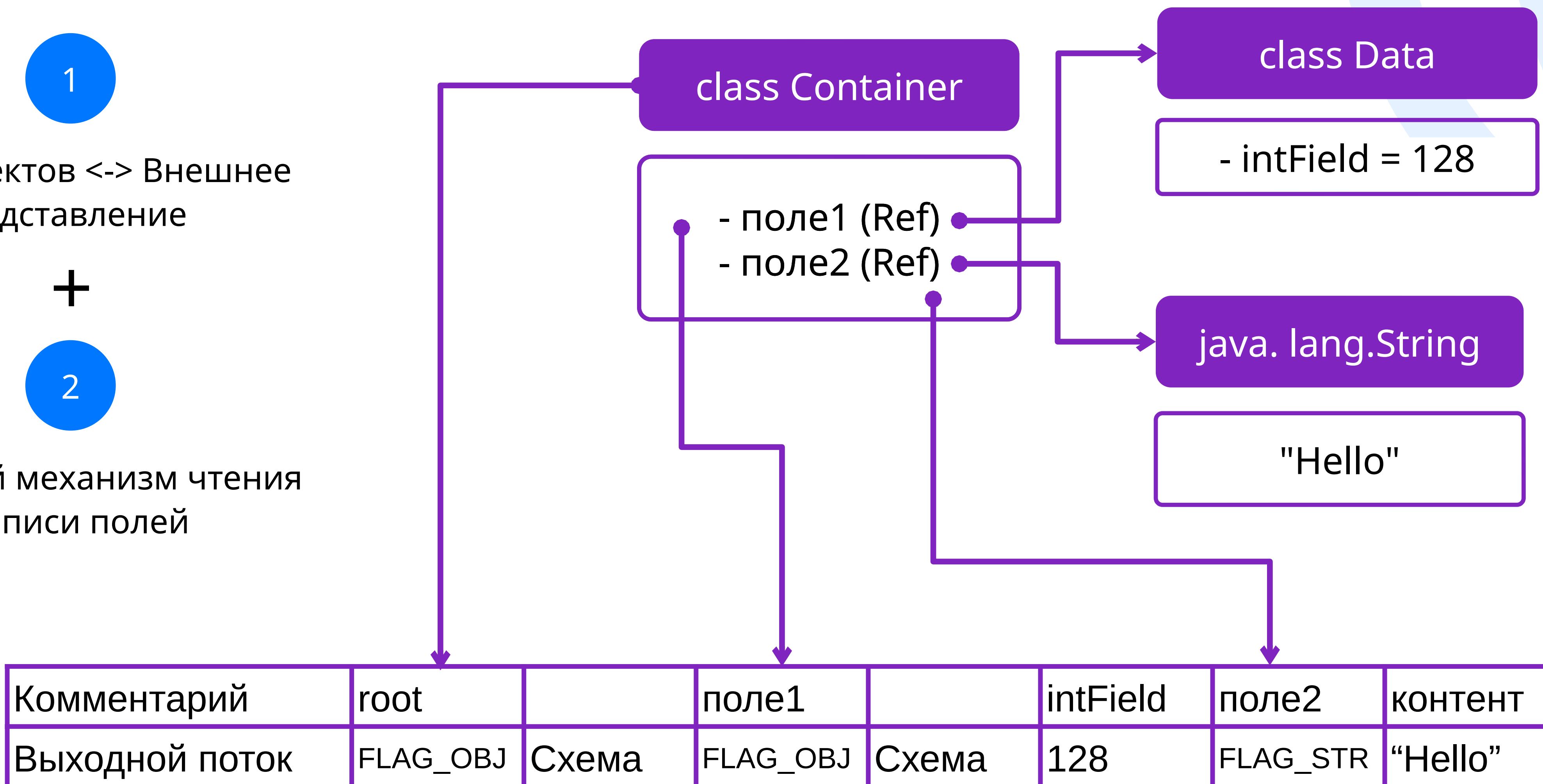
# О сериализации в общем и архитектуре фреймворка

# Сериализация

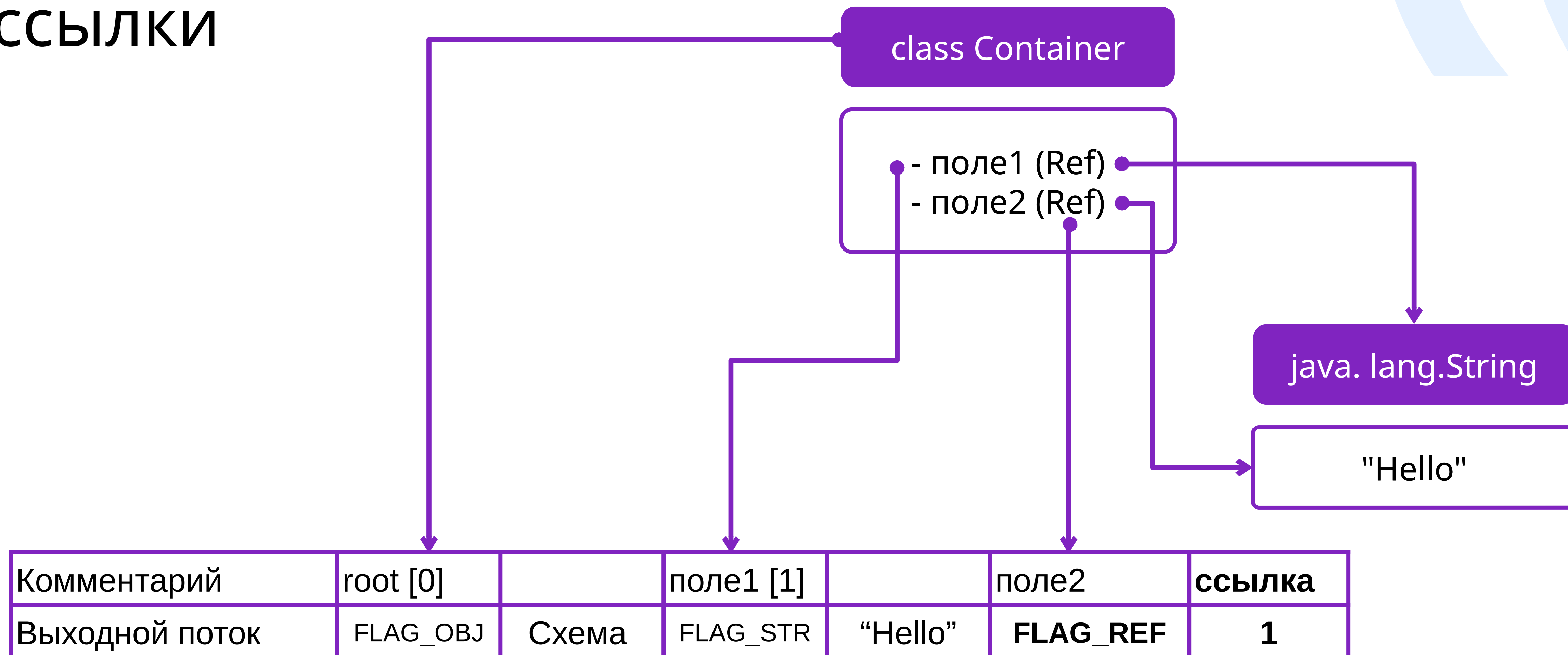
1. Граф объектов <-> Внешнее представление

\+

2. Некоторый механизм чтения и записи полей

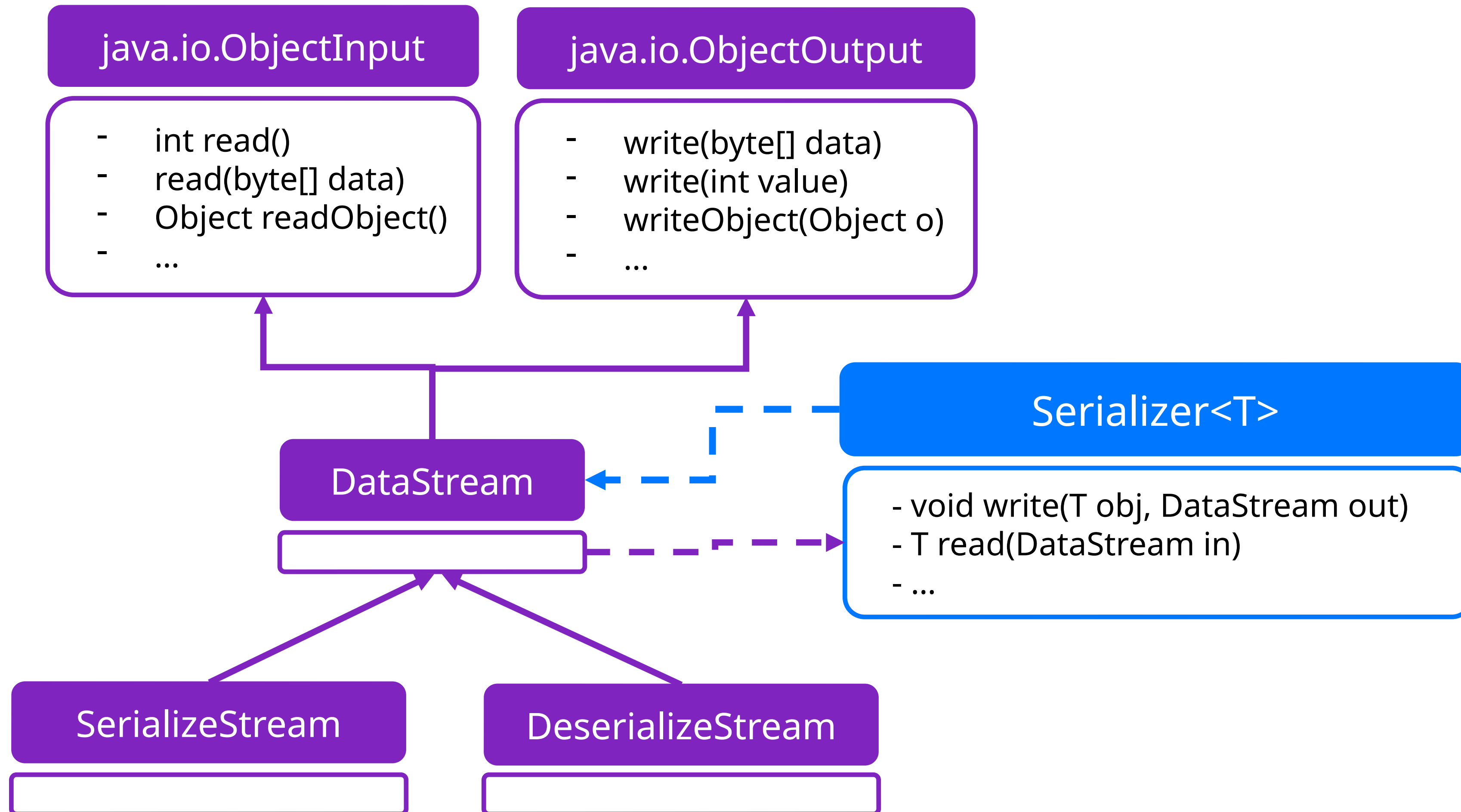


# Сериализация, циклы и множественные ссылки

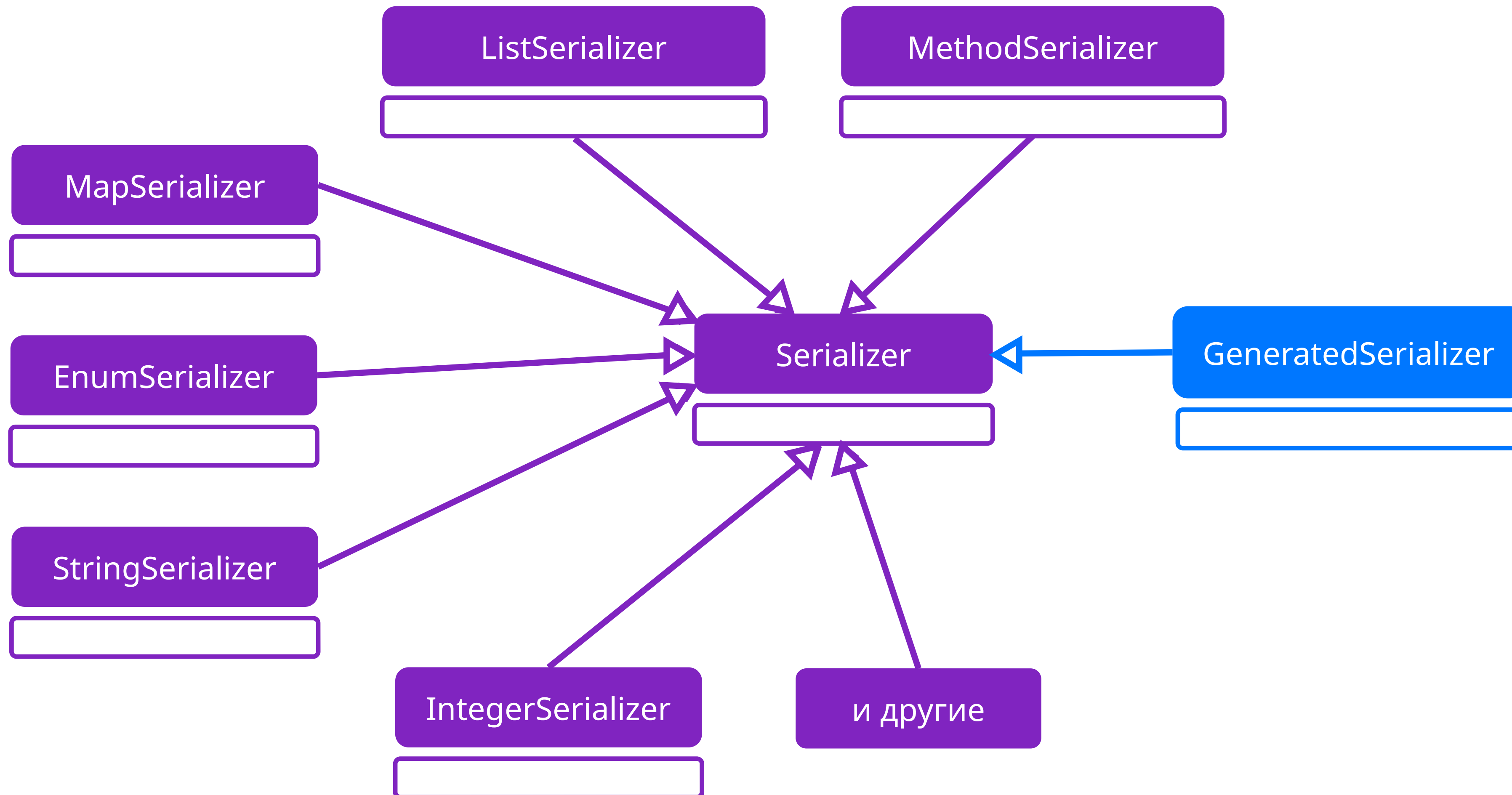



| Комментарий | root [0] | | поле1 [1] | | поле2 | **ссылка** |
|---|---|---|---|---|---|---|
| Выходной поток | FLAG_OBJ | Схема | FLAG_STR | “Hello” | **FLAG_REF** | **1** |

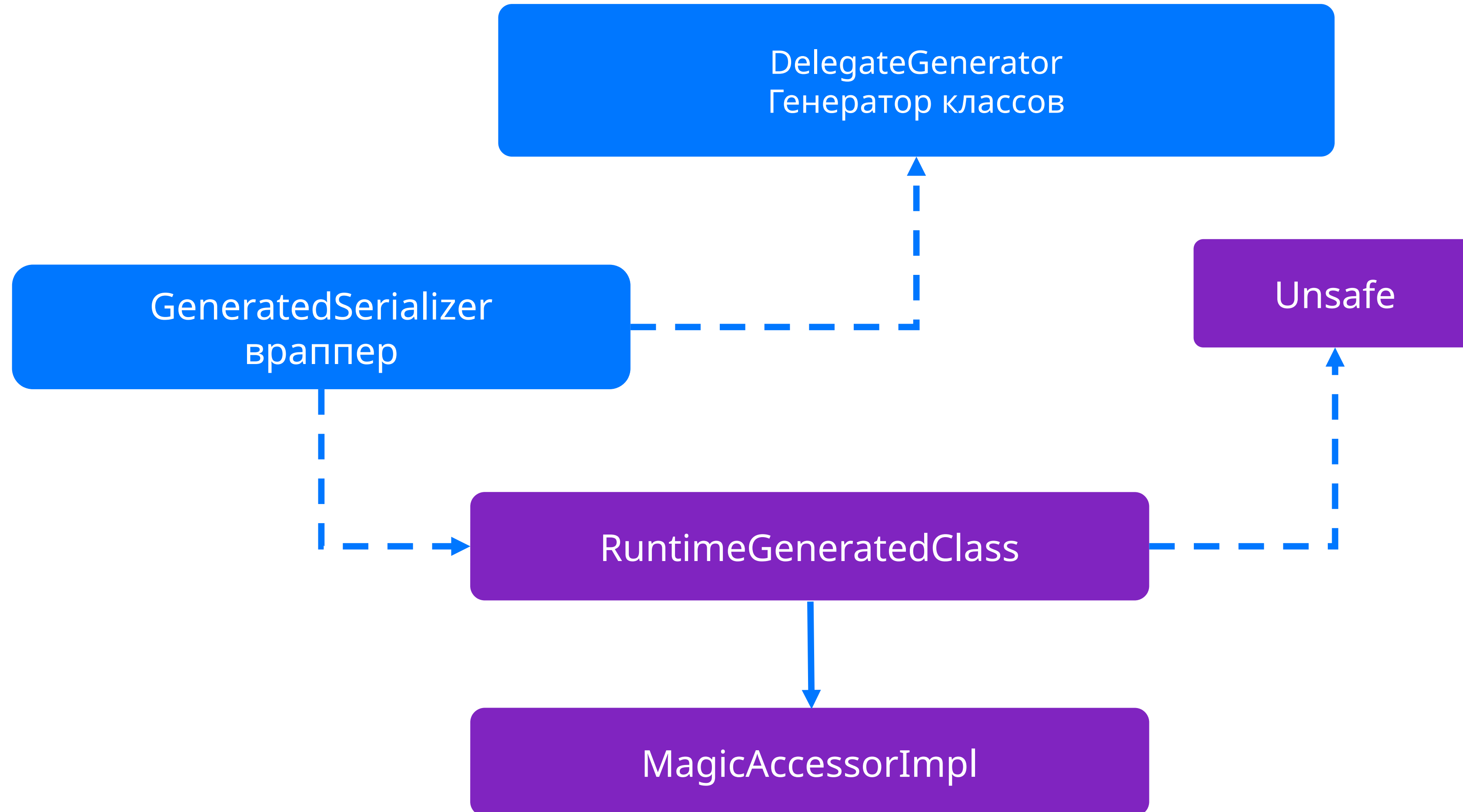
# Немного об архитектуре



# Немного об архитектуре



# Немного об архитектуре



# MagicAccessorImpl

## JavaDoc for MagicAccessorImpl:

All subclasses of this class are "magically" granted access by the VM
to otherwise inaccessible fields and methods of other classes.

+ другие возможности

# sun.reflect.MagicAccessorImpl

```
package myPackage;

public class MyClass {

   private String data;

   private void sayHello() {
      println(data);
   }
}
```

# sun.reflect.MagicAccessorImpl

```java
package myPackage;

public class MyClass {

   private String data;

   private void sayHello() {
      println(data);
   }
}
```

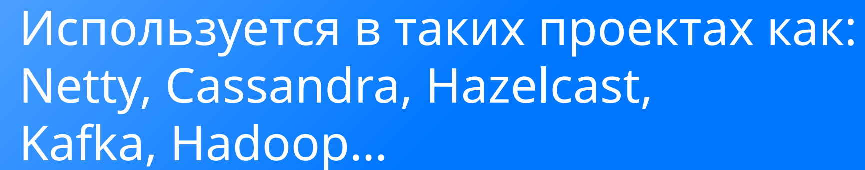
```java
package sun.reflect;
import mypackage.MyClass;

public class ConfMagic extends MagicAccessorImpl {

   public void doSomeWork(MyClass myClass) {
       myClass.data = "Hello Conf"; //PUT_FIELD в байткоде
       myClass.sayHello();
   }
}
```

# Final поля

Всё хорошо, но загвоздка с final-ными полями, которые не можем инициализировать с помощью MagicAccessorImpl

# sun.misc.Unsafe

Unsafe в JVM — это внутренний API, предоставляющий низкоуровневые операции: прямое управление памятью (off-heap), атомарные инструкции, манипуляции с полями классов и создание объектов в обход конструкторов.




Небезопасен, может привести к падению JVM и не рекомендуется для обычного использования.

# Unsafe

```
public class MyClass {
    private final String data;

    public MyClass(String data) {
      this.data = data;
    }

    public void sayHello() {
      println(data);
    }
}
```

# Unsafe

```
public class MyClass {
    private final String data;

    public MyClass(String data) {
      this.data = data;
    }

    public void sayHello() {
      println(data);
    }
}
```

```
public Unsafe getUnsafe() throws ... {
  Field field = Unsafe.class.getDeclaredField("theUnsafe");
  field.setAccessible(true);
  return (sun.misc.Unsafe) field.get(null);
}
```

# Unsafe

```
public class MyClass {
    private final String data;

    public MyClass(String data) {
      this.data = data;
    }

    public void sayHello() {
      println(data);
    }
}
```

```
public Unsafe getUnsafe() throws ... {
  Field field = Unsafe.class.getDeclaredField("theUnsafe");
  field.setAccessible(true);
  return (sun.misc.Unsafe) field.get(null);
}
```

```
public MyClass doSomeWork(Class<MyClass> cls) throws … {
  MyClass data = (MyClass) unsafe.allocateInstance(cls);
  ...
}
```

# Unsafe

```java
public class MyClass {
    private final String data;

    public MyClass(String data) {
      this.data = data;
    }

    public void sayHello() {
      println(data);
    }
}
```

```java
public Unsafe getUnsafe() throws ... {
  Field field = Unsafe.class.getDeclaredField("theUnsafe");
  field.setAccessible(true);
  return (sun.misc.Unsafe) field.get(null);
}
```

```java
public MyClass doSomeWork(Class<MyClass> cls) throws … {
  MyClass data = (MyClass) unsafe.allocateInstance(cls);
  long fieldOffset =
        unsafe.objectFieldOffset(
          cls.getDeclaredField(“data”)
        );
  ...
}
```

# Unsafe

```
public class MyClass {
    private final String data;

    public MyClass(String data) {
      this.data = data;
    }

    public void sayHello() {
      println(data);
    }
}
```

```
public Unsafe getUnsafe() throws ... {
  Field field = Unsafe.class.getDeclaredField("theUnsafe");
  field.setAccessible(true);
  return (sun.misc.Unsafe) field.get(null);
}


public MyClass doSomeWork(Class<MyClass> cls) throws … {
  MyClass data = (MyClass) getUnsafe().allocateInstance(cls);
  long fieldOffset =
       unsafe.objectFieldOffset(
         cls.getDeclaredField(“data”)
       );
  unsafe.putObject(data, fieldOffset, “New Value”);
  return data;
}
```

# Что принёс JDK 24+

MagicAccessorImpl и его аналоги
были убраны в JDK 23-24

Также Unsafe постепенно выпиливается из JDK.
На замену идут VarHandles (JDK 9+) и Foreign Function
& Memory API (Java 22+)

# VarHandles

Низкоуровневое API для работы с переменными (полями классов, элементами массивов...).

Из документации:

A VarHandle is a dynamically strongly typed reference to a variable, or to a parametrically-defined family of variables, including static fields, non-static fields, array elements, or components of an off-heap data structure. Access to such variables is supported under various access modes, including plain read/write access, volatile read/write access, and compare-and-set.

# VarHandles

```
public class MyClass {
   private String data;
   ...
}
```

# VarHandles

```
public class MyClass {

   private String data;

   ...

}
```

```
VarHandle getVarHandle() throws Exception {
  MethodHandles.Lookup lookup =
      MethodHandles.privateLookupIn(
         MyClass.class,
         MethodHandles.lookup()
      );

      return ...;
    }
```

# VarHandles

```
public class MyClass {

   private String data;

   …

}
```

```
VarHandle getVarHandle() throws Exception {
  MethodHandles.Lookup lookup =
      MethodHandles.privateLookupIn(
         MyClass.class,
         MethodHandles.lookup()
      );

      return lookup.findVarHandle(MyClass.class, “data”,
                         String.class);
    }
```

# VarHandles

```
public static void main(String[] args) throws ... {
    MyClass instance = new MyClass("Hello");

    VarHandle varHandle = getVarHandle();
    String value = (String) varHandle.get(instance);
    System.out.println(value);

    varHandle.set(instance, value + " Updated");
    System.out.println((String) varHandle.get(instance));
}
```

# VarHandles, производительность

```
final static VarHandle staticFinalHandle;
static VarHandle staticHandle;
final VarHandle instanceFinal;
VarHandle instance;
```

JDK 21
Linux x86_64
CPU i7-1185G7

| Benchmark | Mode | Cnt | Score | | Error | Units |
|---|---|---|---|---|---|---|
| VarHandleBenchmark.**directFieldAccess** | thrpt | 5 | **3 617 924** | ± | 2 448 | ops/ms |
| VarHandleBenchmark.**staticFinalHandle** | thrpt | 5 | **3 443 090** | ± | 11 581 | ops/ms |
| VarHandleBenchmark.staticHandle | thrpt | 5 | 417 151 | ± | 49 809 | ops/ms |
| VarHandleBenchmark.instanceFinalHandle | thrpt | 5 | 409 420 | ± | 1 967 | ops/ms |
| VarHandleBenchmark.instanceHandle | thrpt | 5 | 409 084 | ± | 971 | ops/ms |

Делаем статическим final полем, чтобы сработали оптимизации JVM

https://github.com/max-kammerer/jvm-benchmarks

# JEP 416, Reimplement Core Reflection with Method Handles

JDK 21
Linux x86_64
CPU i7-1185G7

**final static** Field …;
**static** Field …;
**final** Field …;
Field …;

| ops/ms | Reflection, JDK17 |
|---|---|
| directFieldAccess | 3 471 983 |
| static final | 594 156 |
| static | 531 546 |
| (instance) final | 479 901 |
| (instance) | 468 899 |

https://github.com/max-kammerer/jvm-benchmarks

# JEP 416, Reimplement Core Reflection with Method Handles

JDK 21
Linux x86_64
CPU i7-1185G7

**final static** Field ...;
**static** Field ...;
**final** Field ...;
Field ...;

| ops/ms | Reflection, JDK17 | Reflection, JDK 21 |
|---|---|---|
| directFieldAccess | 3 471 983 | 3 455 030 |
| static final | 594 156 | **1 652 825** |
| static | 531 546 | 346 108 |
| (instance) final | 479 901 | 338 279 |
| (instance) | 468 899 | 339 136 |

Неожиданный поворот

https://github.com/max-kammerer/jvm-benchmarks

# JEP 416, Reimplement Core Reflection with Method Handles

JDK 21
Linux x86_64
CPU i7-1185G7

**final static** Field ...;
**static** Field ...;
**final** Field ...;
Field ...;

| ops/ms | Reflection, JDK17 | Reflection, JDK 21 | VarHandles, JDK 21 |
|---|---|---|---|
| directFieldAccess | 3 471 983 | 3 455 030 | 3 528 086 |
| static final | 594 156 | **1 652 825** | **3 514 334** |
| static | 531 546 | 346 108 | 410 862 |
| (instance) final | 479 901 | 338 279 | 422 750 |
| (instance) | 468 899 | 339 136 | 406 005 |

Неожиданный поворот

https://github.com/max-kammerer/jvm-benchmarks

# Изменения в генерируемых классах

```
class TestData implements Serializable {
    private String nonFinalField;
    private final String finalField;
    …
}
```

```
public final read(Lone/nio/serial/DataStream;)Ljava/lang/Object;
    ...
    ALOAD 1
    INVOKEVIRTUAL one/nio/serial/DataStream.readObject ()Ljava/lang/Object;
    CHECKCAST java/lang/String
```

# Изменения в генерируемых классах

```
class TestData implements Serializable {
    private String nonFinalField;
    private final String finalField;
    …
}
```

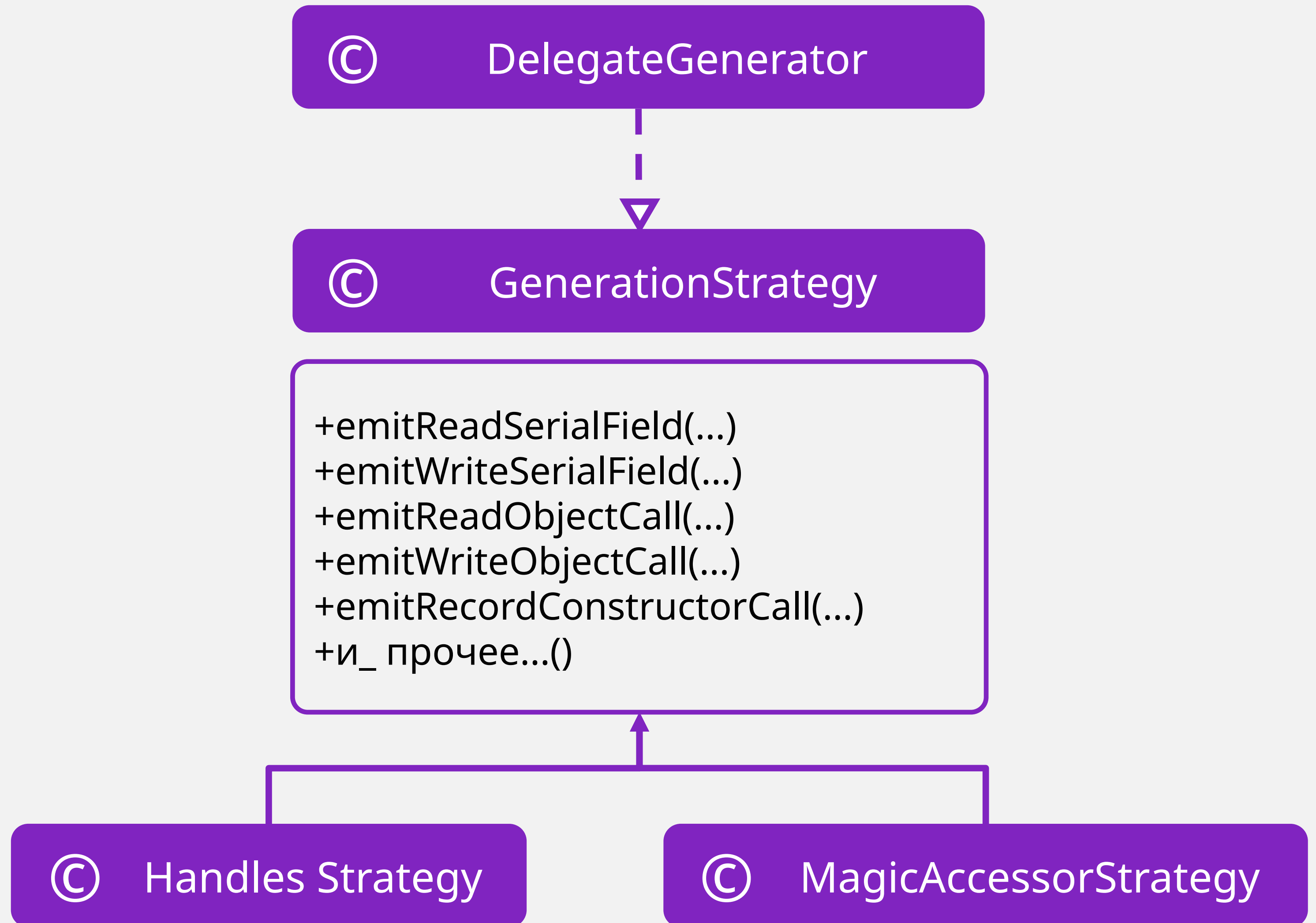
```
public final read(Lone/nio/serial/DataStream;)Ljava/lang/Object;
    ...
    ALOAD 1
    INVOKEVIRTUAL one/nio/serial/DataStream.readObject ()Ljava/lang/Object;
    CHECKCAST java/lang/String

    //старый вариант с MagicAccessorImpl
    PUTFIELD TestData.nonFinalField : Ljava/lang/String;

    // новый с VarHandle
    GETSTATIC sun/reflect/Delegate0_TestData.$nonFinalField : Ljava/lang/invoke/VarHandle;
    … //переупорядочиваем аргументы на стеке
    INVOKEVIRTUAL java/lang/invoke/VarHandle.set (Ljava/lang/Object;Ljava/lang/String;)V
```

# Изменения в генерируемых классах

```
class TestData implements Serializable {
    private String nonFinalField;
    private final String finalField;
    …
}
```

```
public final read(Lone/nio/serial/DataStream;)Ljava/lang/Object;
    ...
    // инициализация final по прежнему работает с Unsafe
    ALOAD 1
    INVOKEVIRTUAL one/nio/serial/DataStream.readObject ()Ljava/lang/Object;
    CHECKCAST java/lang/String

    LDC 16L //смещение final поля
    INVOKESTATIC one/nio/util/JavaInternals.putObject (Ljava/lang/Object;Ljava/lang/Object;J)V
    ARETURN
```

# Что поменялось

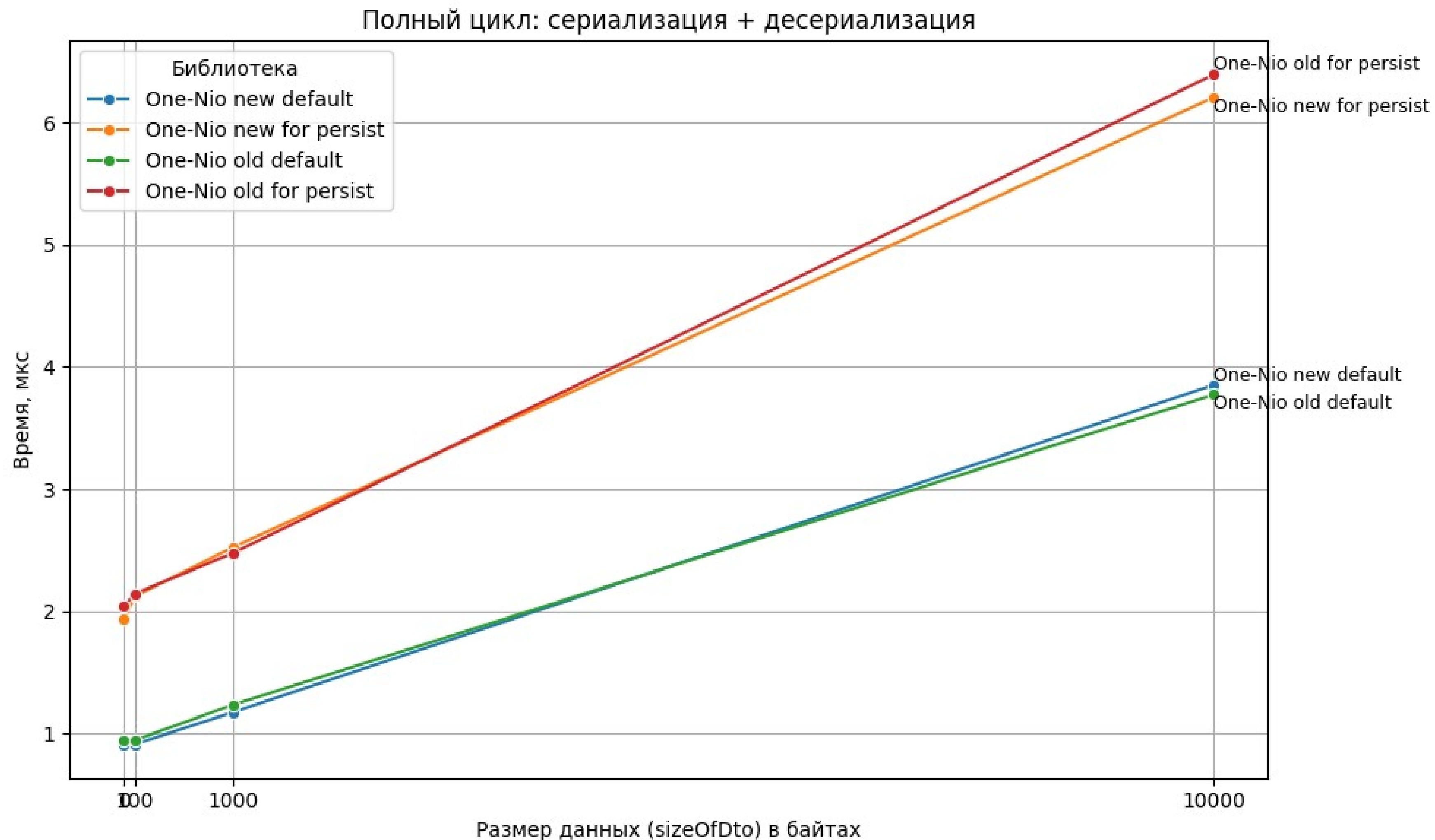


Бенчмарки библиотеки

# Паритет старого и нового режима

One-nio 2.2
old vs new

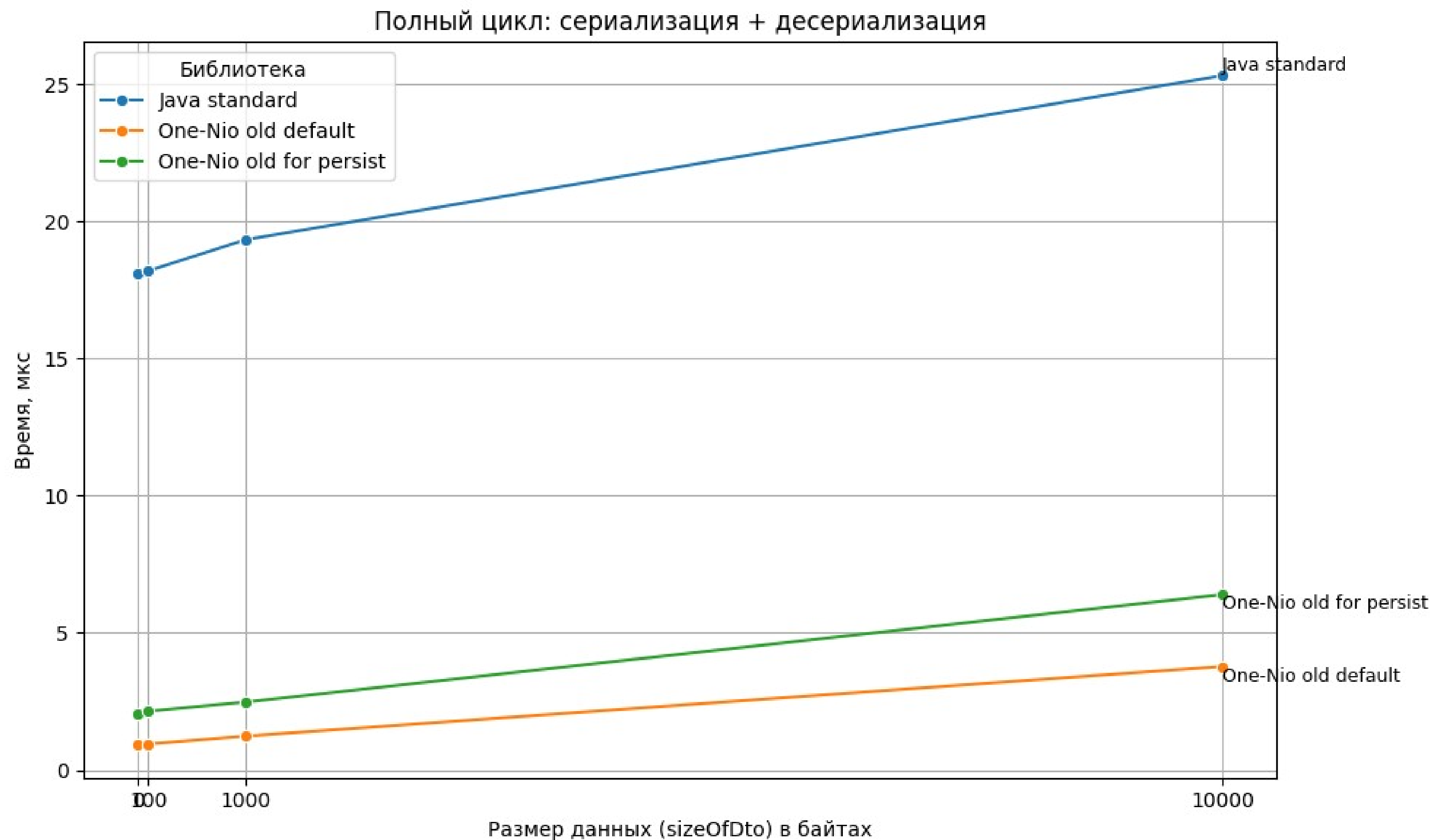
OpenJDK 21
Linux x86_64
Intel Ultra 5 235H




https://github.com/max-kammerer/serializers-benchmark

# One-nio vs стандарная Java

OpenJDK 21
Linux x86_64
Intel Ultra 5 235H



# One-nio и другие по бенчмарку А. Чернова

«Как мы упростили жизнь высоконагруженным сервисам с Platform V SessionsData», Апрель 2023

Время, мс

0,050
0,040
0,030
0,020
0,010
0,000

Jackson JSON
Bson4Jackson
Java standard
Jackson Smile
Kryo
BSON MongoDb
One Nio
FST

100
1 000
10 000

byte[].length в DTO, байты

Цикл сериализация + десериализация

# One-nio vs ProtoBuf

Сравниваем следующие сценарии:

One-nio: Domain object -> byte []
vs
Protobuf: Domain object -> proto -> byte []

| Benchmark | Mode | Cnt | Score | Error | Units |
|---|---|---|---|---|---|
| UserSerDeserJMH.oneNio_serialize | thrpt | 5 | 9020 | ± 238 | ops/ms |
| UserSerDeserJMH.proto_serialize | thrpt | 5 | 12822 | ± 200 | ops/ms |
| UserSerDeserJMH.oneNio_deserialize | thrpt | 5 | 20838 | ± 395 | ops/ms |
| UserSerDeserJMH.proto_deserialize | thrpt | 5 | 17441 | ± 381 | ops/ms |

# Особенности сериализации Recordов в Java

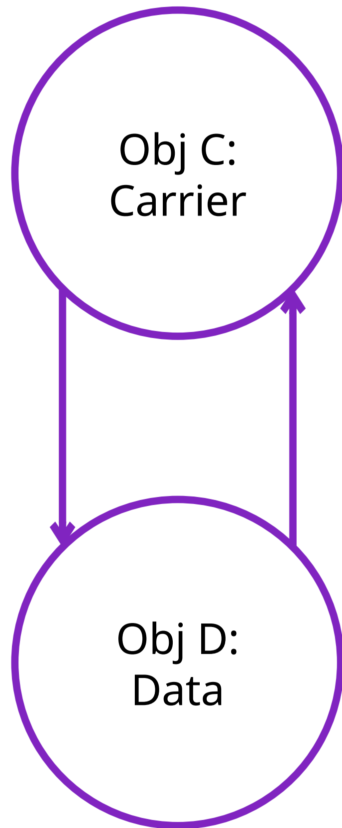
В recordах разработчики JDK переосмыслили подход к сериализации, закрыв заодно дыры, которые могли приводить к уязвимостям (CVE).

Всегда вызывается канонический конструктор при десериализации

Unsafe падает при попытке посчитать офсет поля в record, где все поля final:

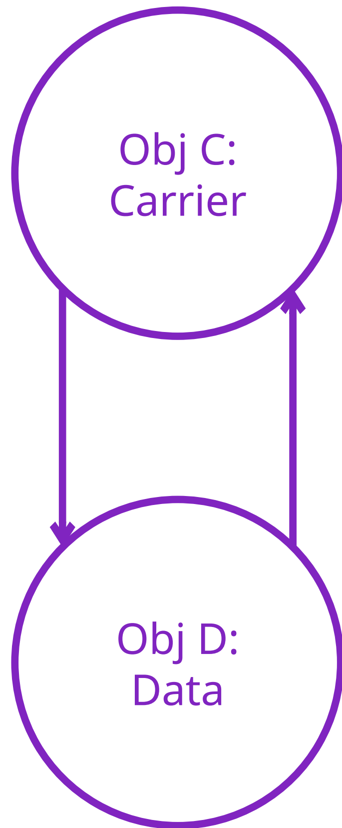
UnsupportedOperationException: can't get field offset on a record class

# Особенности сериализации Recordов в Java



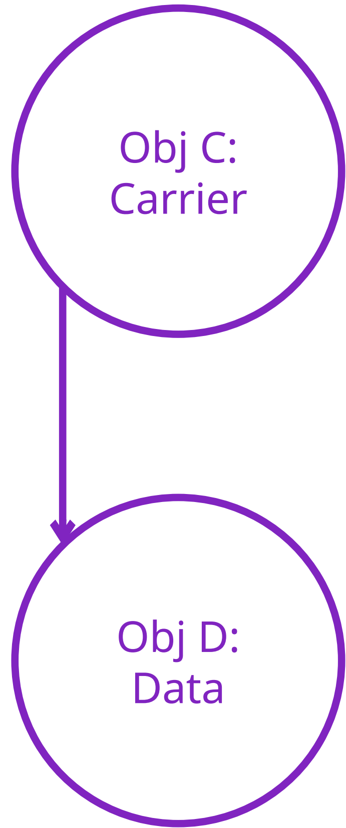

```
class Carrier(Data d) implements Serializable { }

class Data implements Serializable {
   Object obj;
}

// create an instance of both Data and Carrier, and a cycle
between them

Data d1 = new Data();
Carrier c1 = new Carrier(d1);
d1.obj = c1;
```

# Особенности сериализации Recordов в Java




```
record Carrier(Data d) implements Serializable { }

class Data implements Serializable {
   Object obj;
}

// create an instance of both Data and Carrier, and a cycle
between them

Data d1 = new Data();
Carrier c1 = new Carrier(d1);
d1.obj = c1;
```

# Потеря обратной ссылки при десериализации




```
record Carrier(Data d) implements Serializable { }
```

```
class Data implements Serializable {
   Object obj;
}
```

# Records in Java serialization Spec

1.13 Serialization of Records

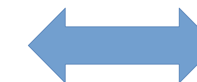
...Like other serializable or externalizable objects, record objects can function as the target of back references appearing subsequently in the serialization stream. However, a cycle in the graph where the record object is referred to, either directly or transitively, by one of its components, is not preserved.
The record components are deserialized prior to the invocation of the record constructor, hence this limitation (see Section 1.14, "Circular References" for additional information)...

# Постойте, но раньше же работало с MagicAccessorImpl. Как?

**NEW** MyClass //создание пустого объекта
<загрузка аргументов>
//и вызов конструктора
**INVOKESPECIAL** MyClass.<init>(список типов аргументов) V

new MyClass (arg1, arg2...)

В простейшем случае:
NEW java/lang/Object
INVOKESPECIAL java/lang/Object.<init> () V



new Object()

# Основное отличия режимов работы

### One-nio

- Десериализация рекордов
- Инициализация объектов
- Доступ к полям и методам
- Дополнительные расходы на инициализацию

### Старый режим

- Восстанавливаем полный граф зависимостей
- NEW + <init>
- Unsafe + MagicAccessorImpl

### Новый режим

- Потеря обратных ссылок в циклах на рекорды
- Unsafe.allocateInstance()
- Unsafe + Var/MethodHandle
- Создание Var/MethodHandle при загрузке классов (<clinit>)

Что готовят грядущие
релизы JDK?

# Сделаем final действительно final

**JEP 471(JDK 23):**
**Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal**

**JEP 498 (JDK 24):**
**Warn upon Use of Memory-Access Methods in sun.misc.Unsafe**

**JEP 500 (JDK 26):**
**Prepare to Make Final Mean Final**

# Что готовят грядущие релизы?
# не Unsafe единым ломаем семантику

```
class C {

  final int field;

  C() {
    field = 100;
  }
}
```

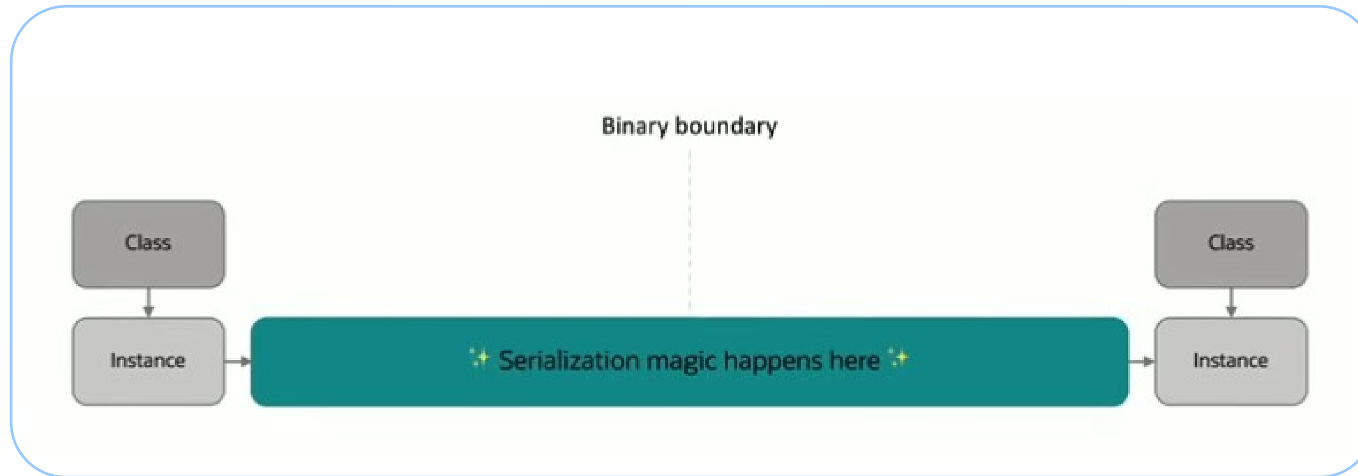
```
Field f = C.class.getDeclaredField("field");
f.setAccessible(true); // Делаем mutable

C obj = new C();
System.out.println(obj.field);  // 100

f.set(obj, 200);
System.out.println(obj.field);  // 200
```

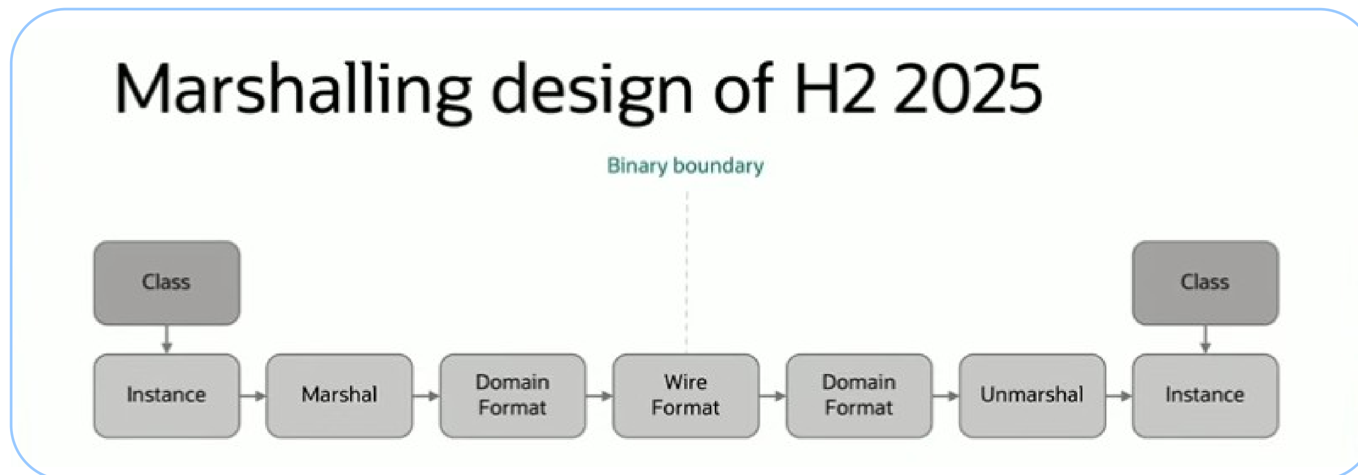
# JEP 500: новый флаг --illegal-final-field-mutation

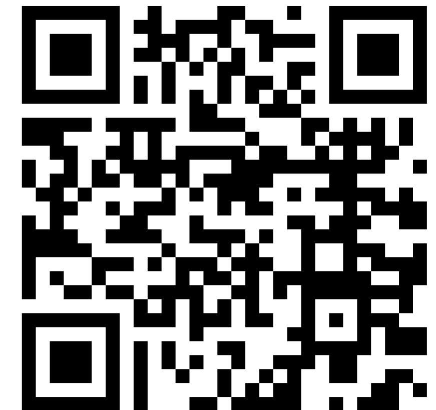
| Режим (mode) | Результат выполнения | Реакция JVM |
|---|---|---|
| allow | ✅ Успешно | Мутации развешены (старое поведение) |
| warn | ✅ Успешно | **Дефолт в JDK 26**. Одно предупреждение на каждый модуль |
| debug | ✅ Успешно | Сообщение + Stack Trace на каждую попытку записи. |
| deny | ❌ Ошибка | Выбрасывает IllegalAccessException. **Дефолт в будущем** |

# Как же будет работать сериализация при наличии final-полей в данных?

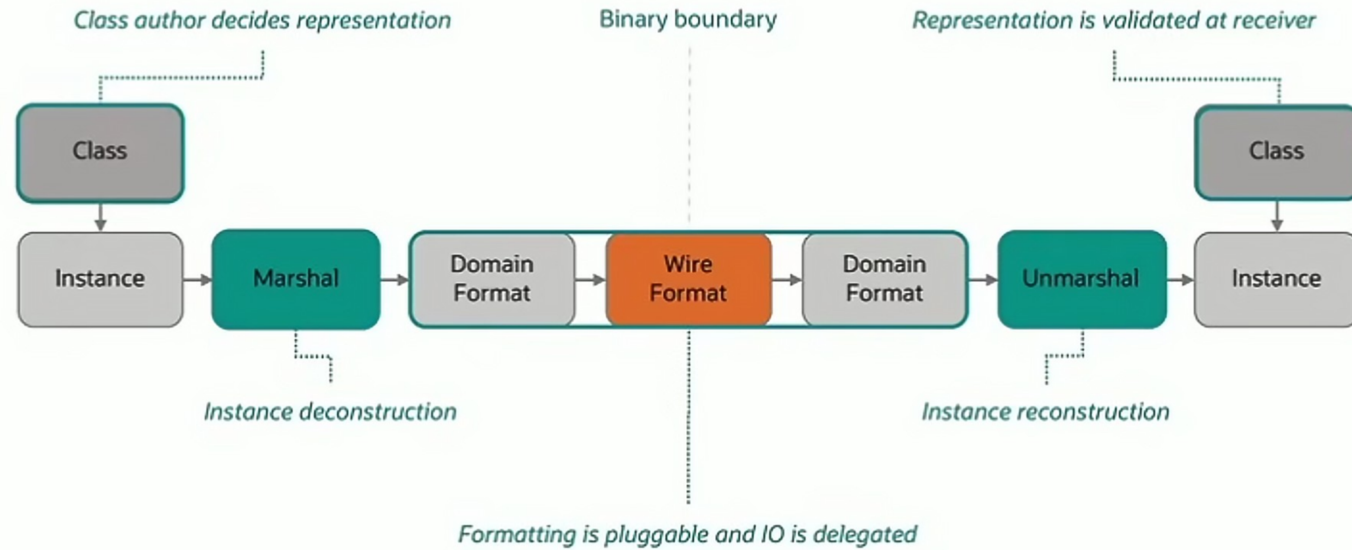
# Разговоры о Сериализации 2.0



Viktor Klang, Serialization 2.0: A Marshalling Update!
https://www.youtube.com/watch?v=F89sNgG9dRY




Viktor Klang, Serialization 2.0: A Marshalling Update!
https://www.youtube.com/watch?v=F89sNgG9dRY

# Маршаллинг:
# интеграция внешней структуры в модель программирования

Marshalling:
bringing *external structure* into the **programming model**


Copyright © 2025, Oracle and/or its affiliates

```java
public class Point {
    final int x;
    final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }


    //record SchemaV1 описывает внешнюю структуру данных объекта Point
    public record SchemaV1(int x, int y) implements SchemaRecord<Point> {

         ...
    }


}
```

```java
public class Point {
    ...

    //Record V1 описывает внешнюю структуру данных объекта Point
    public record SchemaV1(int x, int y) implements SchemaRecord<Point> {

        public V1(Point p) {
            this(p.x, p.y);
        }

        @Override
        public Point unmarshal() {
            return new Point(x, y);
        }
    }
}

// Схема дополнительно может себя регистрировать во фреймворке и быть не публичной
//    в таком случае
```

# The *essence* of Marshalling

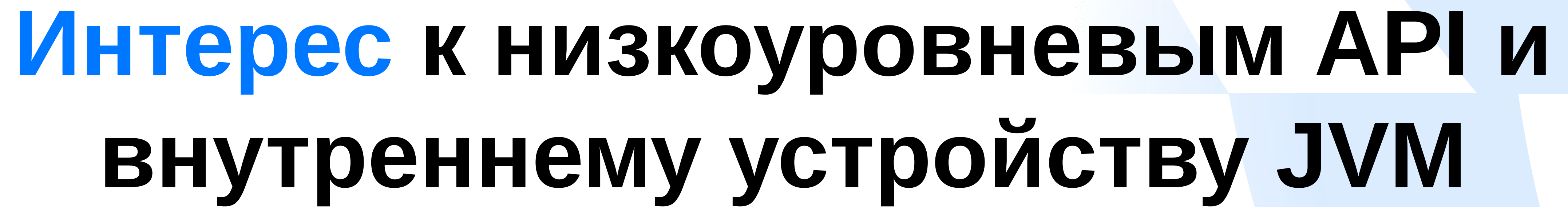
**Design choices**

- Centered around data
- Low-to-no magic
- Focus on external structure and mapping to/from that
- Use constructors and/or factories for instance creation
- No circularity or identity preservation
- No "access busting"
- Users choose the wire formats

**Benefits**

- ~~transient~~
- ~~serialVersionUID~~
- ~~serialPersistentFields~~
- ~~readObject/writeObject~~
- ~~readObjectNoData~~
- ~~readResolve/writeReplace~~
- ~~constructorless instance creation~~
- ~~field scraping & poking~~
- ~~broken reference cycles~~
- ~~feature virality~~
- ~~implicit contracts~~


Copyright © 2025, Oracle and/or its affiliates


Интерес к низкоуровневым API и
внутреннему устройству JVM

# Как попробовать новый экспериментальный режим

- Информация о релизе 2.2.0 и режимах работы библиотеки:

- Поддержка работы нового режима на JDK 9-25+:
  -Done.nio.serial.gen.mode=method_handles

## Другие полезные флаги

- Сдампить генерируемые классы в папку:
  -Done.nio.gen.dump=/path/to/dump/folder

- Посмотреть на текстовое представление классов в логах/консоли:
  -Done.nio.gen.debug.dump_generated_classes_as_text=true

# Заключение

- Большие системы — это набор **условностей-компромиссов**: **возможностей** и **ограничений**
- Системы меняются, сталкиваясь с новыми запросами-вызовами, становясь при этом и **более сложными**
- Рост требует **новых подходов** и **перебалансировки системы, ограничения становятся опорой для следующих шагов**

# В какое ограничение уперлась ваша система? О чем это ограничение для вас?

# Спасибо за внимание!

Михаил Богданов
Ведущий разработчик