# Разработка многопоточного сериализатора и десериализатора для JSON на Kotlin

*Мельникова Маргарита KFD 2024/2025*

В данном отчёте рассмотрена разработка и оценка производительности многопоточного сериализатора и десериализатора JSON на языке Kotlin. Сравнительно была проведена оценка с использованием существующих библиотек Gson и Jackson.

Цели и задачи

1. Разработка многопоточного сериализатора и десериализатора для JSON.
2. Проведение бенчмарка по сравнению с библиотеками Gson и Jackson.
3. Анализ результатов.

## 1.Разработка многопоточного сериализатора и десериализатора для JSON.

Выбор библиотек для работы с JSON и их последующего сравнения:

1. org.json
2. Jackson
3. Gson

В данном примере мы будем использовать org.json.

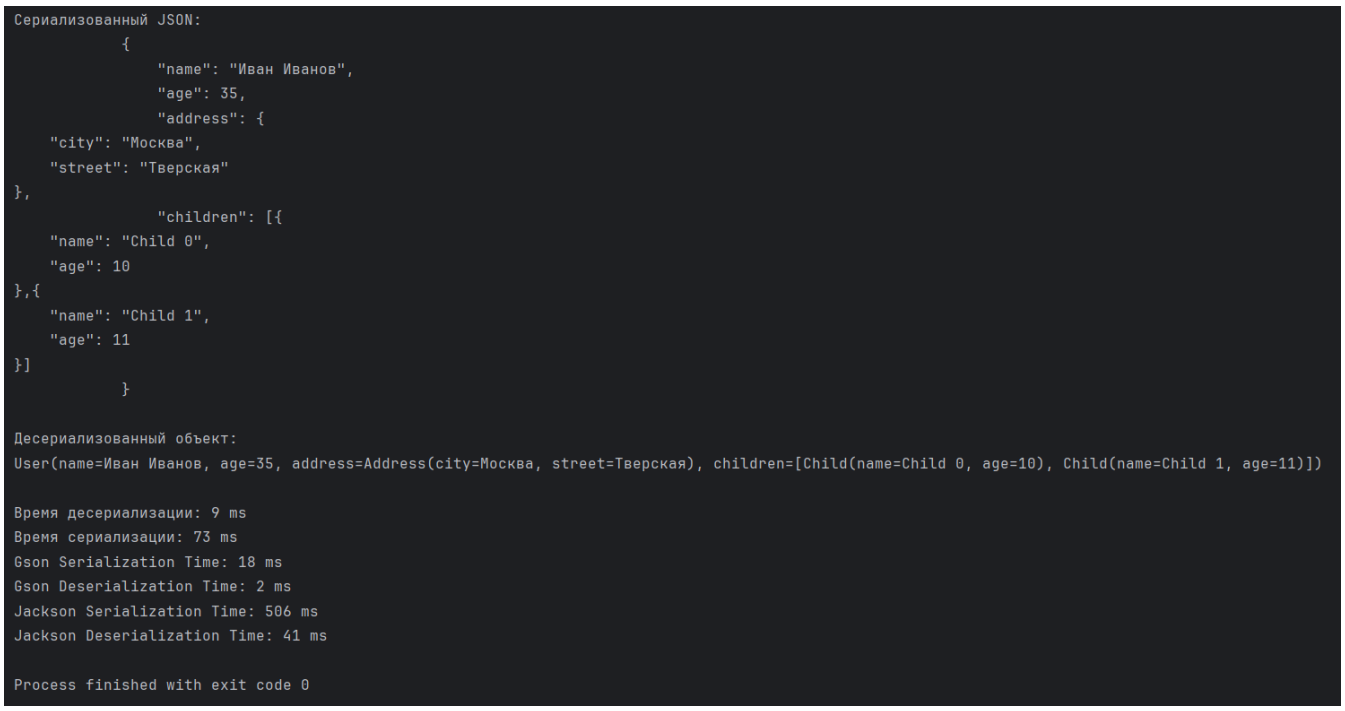
Классы, которые будут сериализироваться и десериализироваться:

```
data class Address(
    val city: String,
    val street: String
)

data class Child(
    val name: String,
    val age: Int
)

data class User(
    val name: String,
    val age: Int,
    val address: Address,
    val children: List<Child>
)
```

Написанная программа должна вывести количество сериализованных и десериализованных пользователей, и результате сериализации и десериализации. Пример вывода представлен на рис.1.




(Рисунок №1 «Пример вывода программы»)

**2. Проведение бенчмарка по сравнению с библиотеками Gson и Jackson.**

Для оценки производительности нашего многопоточного сериализатора и десериализатора, необходимо провести бенчмаркинг и сравнить его с существующими решениями, такими как Gson и Jackson. С помощью файла Benchmark.kt, можно измерить время выполнения сериализации и десериализации для разных библиотек.

Бенчмаркинг:

- Реализован бенчмарк с использованием стандартной функции measureTimeMillis для измерения времени выполнения операций.
- Сравнены три сериализатора: наш многопоточный сериализатор, Gson и Jackson.
- Тест проводился на выборке из объекта класса User со списком Children длиной 10000.

Результаты

| Сериализатор | Время сериализации (мс) | Время десериализации (мс) | Общее время (мс) |
|---|---|---|---|
| Многопоточный | 194 | 31 | 225 |
| Gson | 50 | 29 | 79 |
| Jackson | 357 | 74 | 431 |

**3. Анализ результатов.**

- Многопоточный сериализатор: 225. Несмотря на использование параллельной обработки, оказался на втором месте, большую часть заняла сериализация, но результат десериализации был намного ближе к Gson. На меньшей выборке
- Gson сериализатор: 79 мс. Наиболее быстрый из трёх, что ожидаемо, учитывая информацию о данном сериализаторе.
- Jackson сериализатор: 431 мс. Наименее производителен среди всех трёх.

**Заключение**

Проведённый бенчмарк показывает, что разработанный многопоточный сериализатор и десериализатор на основе org.json превосходит по скорости существующее решение Jackson, но недотягивает до решения Gson, особенно в плане сериализации.