

# Aleksey Shipilëv: One Stop Page

## Quick Bio

Or, "the text I keep copyasting around conferences", therefore written in third person.

If you want to see me bragging about myself in more detail, [look at my CV](#).



🇺🇸 (ENG) Aleksey is working on Java performance for 10+ years. Today he is employed by Red Hat, where he does OpenJDK development and performance work. Aleksey develops and maintains a number of OpenJDK subprojects, including JMH, JOL, and JCSstress. He is also an active participant in expert groups and communities dealing with performance and concurrency. Prior joining Red Hat, Aleksey was working on Apache Harmony at Intel, then moved to Sun Microsystems, which was later consumed by Oracle.

🇷🇺 (RUS) Алексей работает над производительностью Java больше 10 лет. Сегодня он работает в Red Hat, где разрабатывает OpenJDK и работает над его производительностью. Алексей разрабатывает и поддерживает несколько проектов в OpenJDK, включая JMH, JOL, и JCSstress. Алексей также активно участвует в экспертных группах и сообществах, работающих над вопросами производительности и многопоточности. Перед тем как перейти в Red Hat, Алексей работал над Apache Harmony в Intel, а затем перешёл в Sun Microsystems, которая была поглощена Oracle.

(This picture can be used as conference avatar, click for larger version)

## Contact

I usually respond to interesting questions over [personal email](#). "Interesting" means something that is not already discussed on Internet, that could not be found within an hour of searching the Internet, while being very relevant to realistic use cases, etc. Note that my personal email is **not a support channel** for either JDK or any related work; use proper project-specific channels for support requests.

I usually post the updates to projects and this page on Twitter: [@shipilev](#). Please do not abuse the direct messages as the replacement for email (see above).

## Binary Builds

Some of my personal CI servers publish the binaries, workspace tarballs, and patches for the projects I am interested in. You can find those things at [builds.shipilev.net](#). Please note those builds are not well-tested, not virus-checked, may contain horrible bugs that could lead to data corruption, engulfing machines in flames, selling your firstborns at eBay, etc. etc. everything that applies for binaries^W code^W anything downloaded from the Internet. Be cautious. If in doubt, build from source yourself, and/or run on staging environment that is not painful to restore.

Additionally, [Docker Hub](#) contains some of the images built from those binaries. For example, this quickly bootstraps Shenandoah GC enabled image:

```
$ docker run -it --rm shipilev/openjdk-shenandoah:8 java -XX:+UseShenandoahGC --version
openjdk version "1.8.0-force"
OpenJDK Runtime Environment (build 1.8.0-force-sobornost-builds.shipilev.net-shenandoah-jdk8-b291-aarch64-jdk8u181-b15)
OpenJDK 64-Bit Server VM (build 25.71-b291-aarch64-jdk8u181-15, mixed mode)
```

## Recent Posts

🇺🇸 (ENG) [JVM Anatomy Quarks](#) (listing)  
Recurrent short stories on JVM, performance, benchmarking, concurrency. Updated frequently.

🇺🇸 (ENG) [Home Network Overview](#)  
Lab notes: home network overview.

🇺🇸 (ENG) [Java Objects Inside Out](#)  
Large-ish treatise on how Hotspot deals with internal object layouts, and what it can mean for low-level library developers.

🇺🇸 (ENG) [Threadripper Efficiency for OpenJDK builds](#)  
Lab notes: build server efficiency tuning.

🇺🇸 (ENG) [Do It Yourself \(OpenJDK\) Garbage Collector](#)  
Because why not.

🇺🇸 (ENG) [Close Encounters of The Java Memory Model Kind](#)  
Horror details about Java Memory Model abuses.

🇺🇸 (ENG) [Arrays of Wisdom of the Ancients](#)  
Collection.toArray(new T[0]) or Collection.toArray(new T[size]), that's the question.

🇺🇸 (ENG) [Faster Atomic\\*FieldUpdaters for Everyone](#)  
Improve A\*FU performance with these weird tricks.

🇺🇸 (ENG) [The Black Magic of \(Java\) Method Dispatch](#)  
Explores how HotSpot deals with Java method calls.

🇺🇸 (ENG) [Safe Publication and Safe Initialization in Java](#)  
Explores the "Safe Publication" and "Safe Initialization" idioms.  
This post is a translation and an update for a very outdated [Russian post](#).

🇺🇸 (ENG) [On The Fence With Dependencies](#)  
Explores the better instruction selection from StoreLoad barriers. Recaps the [JDK-8050147](#) work.

🇺🇸 (ENG) [Java Memory Model Pragmatics](#)  
Describes what is actually written in JMM Spec. This is the transcript for "[Java Memory Model](#)" talks.

- 🇺🇸 (ENG) [Java vs. Scala: Divided We Fail](#)  
Highlights how to analyze bottlenecks in benchmarks; and how to deal with multi-language benchmarks.
- 🇺🇸 (ENG) [Nanotrusting the Nanotime](#)  
Unfolds why `System.nanoTime()` is bad for your health.
- 🇺🇸 (ENG) [All Accesses Are Atomic](#)  
Quantifies the access atomicity costs.  
(Work in progress)
- 🇺🇸 (ENG) [All Fields Are Final](#)  
Quantifies the final fields memory semantics, in the wake of JMM overhaul.  
(Work in progress)
- 🇺🇸 (ENG) [The Exceptional Performance of Lil' Exception](#)  
Explores the performance magic behind Java exceptions.  
This is an update for two quite outdated Russian posts: [1](#), [2](#)  
Caution: a long article.
- 🇺🇸 (ENG) [What Heap Dumps Are Lying to You About](#)  
Rant about HPROF; and how it confuses the offline heapdump tools.  
Shameless promotion of Java Object Layout @ OpenJDK
- 🇺🇸 (ENG) [Plans for 2014](#)  
Semi-reflection on 2013, and what's going to happen in 2014.
- 🇷🇺 (RUS) [SettableFuture<V>: How To Shoot Oneself in the Foot with Spherical Bicycle in the Vacuum](#)  
(saved from rapidly deteriorating Habrahabr)
- 🇷🇺 (RUS) [PGP Web of Trust: graphs and basic estimations](#)  
(saved from rapidly deteriorating Habrahabr)
- 🇷🇺 (RUS) [Exploring Reflection on HotSpot](#)  
(saved from rapidly deteriorating Habrahabr)
- 🇷🇺 (RUS) [Exploring multiple dispatch on HotSpot](#)  
(saved from rapidly deteriorating Habrahabr)
- 🇷🇺 (RUS) [Estimating compression and deduplication ratios on real data](#)  
(saved from rapidly deteriorating Habrahabr)
- 🇷🇺 (RUS) [VisualVM: monitoring, profiling, and diagnostics for Java applications](#)  
(saved from rapidly deteriorating Habrahabr)

---

## Public Talks

The talks are arranged in themes. Many talks were given over multiple years, choose the most recent one. All other versions are kept for reference.

### # Shenandoah GC

#### 🇺🇸 Bleeding-edge in English

🇺🇸 (ENG) [Slides](#)

Delivered at JUG Berlin-Brandenburg in September 2019. Work in progress.

#### 🇺🇸 Latest in English

🇺🇸 (ENG) [Slides](#)

🇺🇸 (ENG) [Video \(cached here\)](#)

🇺🇸 (ENG) [Video \(Vimeo\)](#)

The major problem for large Java applications is G... (wait for it...) C pauses. Large heaps storing lots of live data, the failure to adhere to generational hypothesis, fragmentation due to old objects coming and going, exacerbate the issues even more. OpenJDK GCs managed to solve the first large part of the puzzle, concurrent marking — the ability to estimate the object reachability graph without stopping the application for a long time. Shenandoah is the new low-pause collector that tries to solve the second large part of the puzzle — the ability to move the objects without stopping the application, cutting the GC pauses even more. This talk is the basic introduction in Shenandoah's design choices, important internal details, performance benefits and trade-offs.

#### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Слайды](#)

🇷🇺 (RUS) [Видео \(YouTube\) — быстрее и кратче \(конференция\)](#)

🇷🇺 (RUS) [Видео \(YouTube\) — длиннее и подробнее \(JUG\)](#)

Одна из главных проблем больших Java-приложений — это сбо... рка мусо... ра. Хранение больших куч данных, активно фрагментирующие приложения и прочие выпадающие из гипотезы о поколениях нагрузки приносят ещё больше проблем. Промышленные GC давно решили первую большую часть проблемы сборки, concurrent marking — выяснение графа объектов без долгой остановки приложения. Shenandoah — новый сборщик мусора, который пытается решить вторую большую часть головоломки, а именно перемещение объектов без остановки приложения, тем самым сбивая паузы ещё больше. Этот доклад об особенностях дизайна и реализации Shenandoah, достоинствах, которыми можно гордиться, и недостатках, с которыми приходится мириться.

---

### # Shenandoah GC (part II)

#### 🇷🇺 Latest in Russian

🇺🇸 (ENG) [Slides](#)

🇷🇺 (RUS) [Видео \(копия\)](#)

После того, как мы разобрались с главными фазами и превратили их в конкурентные, паузы в основном стали определяться более короткими, но всё равно зачастую stop-the-world активностями между большими конкурентными эпохами. В них придётся заниматься всяким: сканировать GC roots, взаимодействовать с языковыми фичами, которые

в курсе про существование GC (например, weak references), разбираться с проблемами в реализации safepoint-ов, менеджить память и как-то делиться ей с ОС и т.п. Этот доклад ныряет в кроличью нору проблем, с которыми вынужден столкнуться низкопаузный GC вроде Shenandoah, размышляет, что можно сделать с этими проблемами на уровне JVM, а также над тем, что могут предпринять предусмотрительные разработчики низкопаузных Java-систем, зная об этих граблях.

---

## # Performance (Keynote)

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Слайды](#)

🇷🇺 (RUS) [Видео \(YouTube\)](#).

🇷🇺 (RUS) [Транскрипт](#)

Оптимизация производительности берedit умы опытных разработчиков с начала компьютерных времён. В коллективном бессознательном оптимизация — это то, что делает программирование интересным, конференции раскупаемыми, личный послужной лист — золотым.

В этом обзорном докладе мы поговорим об оптимизации больших/инфраструктурных проектов (к примеру, OpenJDK): общих принципах, тенденциях и соотношениях; жизненном цикле проекта и экономике оптимизаций; роли и жизненном цикле тестов производительности; типичных ловушках, разногласиях и противоречиях, в которых оказываются оптимизационные задачи в крупных проектах.

---

## # VarHandles

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео \(YouTube\)](#).

🇷🇺 (RUS) [Слайды](#)

sun.misc.Unsafe уйдёт и Мы ВСЕ УМРЁМ! В этом докладе мы посмотрим на работы вокруг VarHandles (JEP 193): что там за API, как в крупную клетку устроена референсная реализация, какие новые режимы доступа (acquire/release, oraque/relaxed, compareAndSet/compareAndExchange) она даёт, и как мы умудряемся её скомпилировать в практически голые доступы.

Кроме того, мы посмотрим на то, какой Unsafe плохой, какие грабли нам подкладывают текущие JDK/JVM, какие хардварные проблемы подтачивают красивый гранит реализации. С позитивной стороны мы увидим побочные улучшения в JDK/JVM: оптимизации в ByteBuffers, Atomic\*FieldUpdaters, и прочие общие кодогенерационные улучшения.

---

## # Java Memory Model, Unlearning Experience

### 🇬🇧 Latest in English

🇬🇧 (ENG) [Video \(cached\)](#).

🇬🇧 (ENG) [Video \(YouTube\)](#).

🇬🇧 (ENG) [Slides](#)

This talk is another attempt at explaining the Java Memory Model (JMM). This time we would assume people come to concurrency world with their preconceptions how the world works, and conjecture that is the major reason learning the low-level concurrency is hard for them. Therefore, rather than explaining what JMM **is**, we would try to see what JMM **is not**, and this should eliminate a few misconceptions about the Java concurrency at large. This would be the **unlearning** experience! The talk would try to build simple intuitive rules that we can use every day, and would try to outline the proofs one could employ to verify those rules. We shall also see surprising behaviors that are allowed by JMM, but not by naive (mis)interpretations of it. Prerequisites: the talk assumes the audience understands basic JMM, at least on the level of "[JMM Pragmatics](#)". Understanding more advanced topics, like the ones discussed in "[Close Encounters of The Java Memory Model Kind](#)" would be a plus.

---

## # Java Memory Model, Close Encounters

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео \(YouTube\)](#).

🇷🇺 (RUS) [Слайды](#)

Со времён «Прагматики Java Memory Model» прошло больше двух лет. Но даже у изучавших прошлый доклад специалистов остались странные предубеждения, не подкреплённые спецификацией. В этом докладе мы попытаемся разобрать и развенчать часть этих предубеждений: про всемогущие барьеры, про реордеринги, про недосинхронизацию и другое недоделосипедостроение.


Доклад основан на уже опубликованной статье, и будет включать себя наиболее вкусные примеры. Доклад не будет останавливаться на базовых принципах модели и поэтому требует понимания JMM как минимум на уровне «Прагматики JMM».

---

## # Java Memory Model Pragmatics


### 🇷🇺 Latest in Russian

 (RUS) [Видео, часть 1 \(копия\)](#).

 (RUS) [Видео, часть 2 \(копия\)](#).

 (RUS) [Видео, часть 1 \(YouTube\)](#).

 (RUS) [Видео, часть 2 \(YouTube\)](#).

 (RUS) [Слайды](#)


Спецификация Java Memory Model в JLS пытается быть предельно сжатой и полной. Поскольку JMM пытается объять очень большой пласт явлений, её формализм весьма громоздок, что обернулось потерей понимаемости модели простыми смертными.

Понять хотя бы наполовину, что написано в спецификации Java Memory Model (далее — JMM), получается раза с третьего. Понять, почему записано именно так, по спецификации вообще невозможно, и приходится обращаться к дополнительным источникам, которые пытаются как-то формализм JMM переосмыслить, дополнить, и привести примеры.

В этом докладе мы попытаемся проследить за логикой построения модели; поговорим о том, каких прагматических результатов модель пыталась добиться; посмотрим, с какими ограничениями злого внешнего мира при этом пришлось столкнуться; увидим, как JMM пытается балансировать между требованиями девелоперов и требованиями разработчиков рантаймов и железа.

## Latest in English

 (ENG) [Video \(YouTube\)](#).

 (ENG) [Slides](#)

 (ENG) [Talk transcript](#)

The Java Memory Model is the most complicated part of Java spec that must be understood by at least library and runtime developers. Unfortunately, it is worded in such a way that it takes a few senior guys to decipher it for each other.

Most developers, of course, are not using JMM rules as stated, and instead make a few constructions out of its rules, or worse, blindly copy the constructions from senior developers without understanding the limits of their applicability. If you are an ordinary guy who is not into hardcore concurrency, you can pass this talk, and read high-level books, like “Java Concurrency in Practice”. If you are one of the senior folks interested in how all this works, join us!

In this talk, we will follow the logic of the model; review what pragmatic results the model was trying to achieve; look closely at the real world limitations the model had to endure; see how JMM tries to balance between developers’ needs and runtime/hardware maintainers requests.

## Historical (e.g. outdated) materials

 (ENG) [JVMLS 2014: Java Memory Model Pragmatics](#)  
Workshop collaterals.

 (RUS) [JEEConf 2014: Java Memory Model Pragmatics](#)  
Video: [JEEConf, Direct: part 1, 364 MB](#), [Direct: part 2, 365 MB](#)

 (RUS) [CodeFest 2014: Java Memory Model Pragmatics](#)

 (RUS) [JUG.Ru 2014: Java Memory Model Pragmatics](#)  
Video: [Youtube](#), [Direct: 1090 MB](#)

 (RUS) [JavaOne Russia 2013: Java Memory Model](#)  
(hosted for Sergey Kuksenko)

 (RUS) [JavaOne Moscow 2012: Java Memory Model](#)  
(hosted for Sergey Kuksenko)  
Video: [Youtube](#)


 (RUS) [JavaDay Kiev 2011: Java Memory Model](#)  
(hosted for Sergey Kuksenko)

---

## # java.lang.String Catechism

### Latest in English

 (ENG) [Video \(YouTube\)](#).


 (ENG) [Slides](#)

Hardcore enterprise solutions, as well as other products, normally deal with large amount of text data. Those applications spend considerable time and memory to mess with Strings. It had been repeatedly shown that optimizing String usages will almost always give the immediate performance boosts. This is not to mention dodging OutOfMemoryErrors and the like.

In this talk, we will revisit the basic sins of working with Strings: gluttony of concatenation, wrath of substrings, greed of interning, pride of deduplication et cetera. We will also see the costs of sloth, believing JVM Almighty will do all the work for us.

### Latest in Russian

 (RUS) [Video \(YouTube\)](#).

 (RUS) [Слайды](#)

В приложениях кровавого энтерпрайза и прочих продуктах, что так или иначе связаны с обработкой текстовых данных, порядочное количество памяти и времени тратится на возню со строками. Оптимизации работы со строками часто дают немедленный выигрыш, а то и уворачивание от OutOfMemoryError.

В этом докладе будут рассмотрены основные пороки работы со строками: чревоугодие конкатенации, блуд подстрок, корыстолюбие интернирования, гордыню дедупликации и прочее. Кроме того, речь пойдет о том, чего стоит излишняя надежда на JDK и JVM.

## Historical (e.g. outdated) materials


None yet!

---

## # The Lord Of The Strings

### Latest in English

 (ENG) [Video \(YouTube\)](#)

 (ENG) [Slides](#)


java.lang.String is pervasively and perversely used in most Java applications. Not surprisingly, we are looking into optimizing it both on small and large scale.

In this talk, we will take a deeper look into two interesting String-related features coming in JDK 9: a) [Compact Strings](#), that saves memory for Strings representable with single-byte chars, with little or none performance regressions, and in many cases, significant performance improvements; b) [Indify String Concat](#), that uses the magic of invokedynamic to concatenate Strings, to free runtime implementors for optimizing string concatenation without pushing users to recompile their programs.

We will talk about the rationale, pitfalls and caveats of implementing the intrusive core library/runtime changes.

### Latest in Russian

 (RUS) [Видео \(YouTube\)](#)

 (RUS) [Слайды](#)

java.lang.String — один из наиболее часто используемых классов в Java приложениях. Не удивительно, что мы пытаемся его улучшать и микро- и макро-оптимизациями. В докладе будут освещены вопросы рациональности, подходов к реализации, практических граблей, с которыми сталкиваются разработчики JDK, пытающиеся ничего не сломать в огромной экосистеме, а также чем эта подковёрная деятельность грозит простым пользователям.

В этом докладе мы посмотрим на две грядущие фишки в JDK 9, направленные на оптимизацию строк: [Compact Strings](#), сжимающие строки с однобайтовыми символами, что улучшает футпринт и даже общую производительность; и [Indify String Concat](#), использующий магию invokedynamic для конкатенации строк, позволяющий подкручивать реализацию конкатенации без recompilation программ.

### Historical (e.g. outdated) materials

 (RUS) [JPoint 2016: The Lord of the Strings](#)


 (RUS) [Joker 2015: The Lord of the Strings](#)

---

## # Compress Me Tightly


### Latest in Russian

 (RUS) [Видео \(YouTube\)](#)

 (RUS) [Слайды](#)

Контроль над использованием памяти — это ключ к написанию высокопроизводительного софта. В этом докладе мы покопаемся в кишках JVM и JDK, в поисках того, как сама платформа пытается сэкономить на памяти. Посмотрим на упаковку заголовков и полей объектов, сжатие указателей, учёт ссылок между поколениями в куче, трюки в сгенерированном коде, кэш автобоксинга и т. п. Всё это обильно сдобрим описанием возможных граблей и измерениями производительности.

### Historical (e.g. outdated) materials


 (RUS) [JPoint 2015: Compress Me Tightly](#)

---

## # Performance Optimization 101

### Latest in English

 (ENG) [Video \(cached copy\)](#)


 (ENG) [Slides](#)

[Talk Mindmap](#)

Performance optimization has always thought to be a fine art as it could not be easily formalized, or constrained into one solid workflow. However, there are common patterns all performance engineers could follow in their investigations. This session describes some approaches and tools to analyse modern application performance problems in J2SE and hardware.

### Latest in Russian

 (RUS) [Видео \(YouTube\)](#)


 (RUS) [Слайды](#)

Хотите сделать ваше приложение быстрее, и для этого оптимизируете Java-код? Мы не будем рассказывать, как оптимизировать Java-программы. Мы не будем рассказывать, как использовать ваш любимый профайлер. В первой части мы расскажем, как делать «правильные вещи» (TM), а не тратить две недели на бесполезные приседания; эти вещи не всегда лежат в самом приложении. Во второй части мы рассмотрим то же самое, применительно к случаю многоядерных машин, разберём типичные казусы с производительностью программ, работающих на больших машинах; опишем их симптомы, а также обсудим типичные подходы к диагностике проблем и их классические решения.

### Historical (e.g. outdated) materials

 (RUS) [JUG.Ru 2012 Video, 451 MB](#)

 (RUS) [JavaOne Moscow 2012: How To Train Your Dragon: Attack The Scaling on Multicore Machines](#)  
[Youtube video](#)

 (RUS) [JavaOne Moscow 2012: Performance Methodology Intro](#)  
[Youtube video](#)


 (RUS) [JavaDay SPB 2012: Performance Methodology Intro](#)

---

## # Benchmarking ("Two Timestamps" Story)

### Latest in English

 (ENG) [Video \(cached copy\)](#).

 (ENG) [Slides](#)


This talk includes short versions of "[Nanotrusting the Nanotime](#)" and "[Java vs. Scala: Divided We Fail](#)" posts.

Performance measurement is easy! Get `System.nanoTime()` once, get it twice, subtract, multiply, divide, stare at the numbers and go for optimizations. Nope! In this talk, we will see how does one uses benchmarks to measure the application performance, what mistakes are usually done in the course of that work, how to avoid those mistakes, and all-in-all how to get any sensible data from the world where everything depends on everything.

We will use Java Microbenchmark Harness (JMH), the standard harness in OpenJDK, in our tutorials.

### Latest in Russian

 (RUS) [Видео \(YouTube\)](#).

 (RUS) [Слайды](#)

Хотите сделать ваше приложение быстрее, и для этого оптимизируете Java-код? Мы не будем рассказывать, как оптимизировать Java-программы. Мы не будем рассказывать, как использовать ваш любимый профайлер. В первой части мы расскажем, как делать «правильные вещи» (ТМ), а не тратить две недели на бесполезные приседания; эти вещи не всегда лежат в самом приложении. Во второй части мы рассмотрим то же самое, применительно к случаю многоядерных машин, разберём типичные казусы с производительностью программ, работающих на больших машинах; опишем их симптомы, а также обсудим типичные подходы к диагностике проблем и их классические решения.

### Historical (e.g. outdated) materials

 (RUS) [JEEConf 2014: Java Benchmarking, Timestamping Failures](#)  
Video: [JEEConf, Direct: 351 MB](#)

 (RUS) [JPoint 2014: Java Benchmarking, Timestamping Failures](#)


 (RUS) [CodeFest 2014: Java Benchmarking](#)

---

## # Benchmarking ("The Lesser of Two Evils" Story)

### Latest in English

 (ENG) [Video \(YouTube\)](#).


 (ENG) [Slides](#)

Measuring the performance is the fine art, and measuring the performance on microbenchmarks is double so. There are multiple caveats one should take a great care of while designing the experiment involving microbenchmarks. The advanced experience with VM technology is required, the exposure with particular nits on exact VM is also a plus.

Some say the benchmarking is inherently evil. We agree with that assertion, but also realize the benchmarking is nevertheless essential, and we need to learn how to benchmark without shooting ourselves in the foot all the time. In this session, we take the crash course in fine microbenchmarking, and introducing OpenJDK's Java Microbenchmark Harness (JMH) along the way.

### Latest in Russian


 (RUS) [Видео \(YouTube\)](#).

 (RUS) [Слайды](#)

Доклад про построение корректных (микро)бенчмарков для JVM, учитывая всевозможные грабли, которые нас поджидают от автоматических оптимизаций, особенностей работы рантайма, случайной игры в кости богоподобных существ и прочих жизненных неприятностей. Доклад расширяет и углубляет первый доклад про микробенчмарки, читавшийся в 2009 году. Примеры микробенчмарков показаны на примере JMH — фреймворка, использующегося для тестирования OpenJDK.


### Historical (e.g. outdated) materials

 (ENG) [Oredev 2013: \(The Art of\) \(Java\) Benchmarking](#)  
Video: [Vimeo, Direct: 188 MB](#)

 (ENG) [JVMLS 2013: JVM Benchmarking](#)  
Video: [QTJN, Direct: 968 MB](#)

 (ENG) [JavaOne SF 2011: \(The Art of\) \(Java\) Benchmarking](#)

 (RUS) [JavaDay Kiev 2011: \(The Art of\) \(Java\) Benchmarking](#)

 (RUS) [JavaOne Moscow 2011: \(The Art of\) \(Java\) Benchmarking](#)

---

## # Java Concurrency Stress Tests

## 🇬🇧 Latest in English

No video.

🇺🇸 (ENG) [JVMLS 2013: Breaking Concurrency Bad, or jcstress](#)

A very short lightning talk.

## 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео \(копия\)](#)

🇷🇺 (RUS) [Слайды](#)

Доклад про опыт тестирования concurrency в JDK/JVM, разного рода загадки и примеры как оптимизации и дефекты в реализации виртуальной машины и библиотек ломают модель памяти, и что нам с этим делать. Доклад \*требуется\* знания JVM, понимания внутренней работы JRE, и устройства железа.

## Historical (e.g. outdated) materials

🇷🇺 (RUS) [JEEConf 2013: Java Concurrency, Battle for Correctness](#)  
Video: [JEEConf, Direct: 414 MB](#)

🇷🇺 (RUS) [JavaOne Russia 2013: Bullet-Proof Java Concurrency](#)  
Video (courtesy of JPoint): [YouTube, Direct: 431 MB](#)

---

## # Fork/Join

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео \(копия\)](#)

🇷🇺 (RUS) [Видео \(YouTube\)](#)

🇷🇺 (RUS) [Слайды](#)

В JDK 7 появилась поддержка параллельных операций, лежащих в модель Fork/Join. Этот доклад рассматривает особенности реализации ForkJoinPool, явные ограничения и подводные камни, а также следующие из них правильные и неправильные модели использования новой функциональности.

## Historical (e.g. outdated) materials

🇷🇺 (RUS) [JEEConf 2013: Fork/Join](#)

🇷🇺 (RUS) [JUG.Ru \(Spb\): Fork/Join](#)  
Video: [YouTube, Direct: 889 MB](#)

🇺🇸 (RUS) [JEEConf 2012: Fork/Join](#)  
Video: [Yandex](#)

🇷🇺 (RUS) [JavaOne Moscow 2012: Fork/Join](#)  
Video: [Youtube](#)

---

## # Java Object Layout

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео \(копия\)](#)

🇷🇺 (RUS) [Слайды](#)

[JOL](#)

[JOL Samples](#)

Программисты — как дети, всегда пытаются разломать свои любимые игрушки, чтобы понять, как же те работают. Некоторым детям, правда, действительно нужно знать, сколько лишних деталек напихали туда глупые конструкторы. Для возмужавших детей — этот доклад.

Возмужавшие дети (tm) обычно берут в руки heap dump и начинают ковыряться в нём отвёрткой. Но heap dump — это так же надёжно, как информация, выбитая из пленного партизана. Он «сдаст» вам координаты аэродрома с кукурузниками, а найдёте вы там пустое кукурузное поле.

Нам нужно взять объекты тёплыми прямо в рабочей JVM, где они живут своей подпольной жизнью. Только так мы сможем в деталях рассмотреть, как разложены поля, как разложены объекты в памяти, как их связи влияют на укладку, как и когда они меняют места дислокации, пуская пыль в глаза честным разработчикам.

Очная ставка с захваченными за линией фронта объектами гарантируется!

## Historical (e.g. outdated) materials

🇷🇺 (RUS) [Joker 2013: What Heap Dumps Are Lying To You About](#)  
Video: [Youtube, Direct: 497 MB](#)  
Collaterals: [JOL](#), [JOL Samples](#)


---

## # JDK 8, Lambdas

### 🇷🇺 Latest in Russian

🇷🇺 (RUS) [Видео, часть 1 \(копия\)](#)

🇷🇺 (RUS) [Видео, часть 2 \(копия\)](#)

 (RUS) [JUG.Ru \(SPb\): Per Aspera Ad Lambdas](#)


Самым существенным со времен Java 5 изменением языка станет поддержка лямбда-выражений в Java 8. Эта встреча посвящена техническим аспектам Project Lambda, рассказу о ситуации «изнутри».

Речь пойдёт:

1. Про лямбды: что такое лямбды в контексте Java; как они соотносятся с существующими примитивами языка; лямбды — сахар или нет; как jsr292 победил javac; сколько, где, и кому стоят лямбды; плюшки, грабли, и прочие особенности;
2. Про stream (bulk) операции: что это такое, и с чем их едят; при чём тут Fork/Join; сколько и где стоит их использование; пышки, шишки, и прочие характерные свойства;
3. Про defender (default) методы: что это такое, зачем нужны; чем всё-таки отличается абстрактный класс от интерфейса; прочие весёлые неурядицы


## Historical (e.g. *may be outdated*) materials

 (ENG) [JavaDay Riga 2013: JDK 8: Stream Style](#)  
(hosted for Sergey Kuksenko)

 (RUS) [JEEConf 2013: JDK 8: Lambda Malleus](#)  
(modified by Sergey Kuksenko)


 (RUS) [JavaOne Russia 2013: JDK 8: Lambda Malleus](#)


 (RUS) [JEEConf 2013: JDK 8: J. Lambda](#)  
(hosted for Sergey Kuksenko)

 (RUS) [JavaOne Russia 2013: JDK 8: J. Lambda](#)  
(hosted for Sergey Kuksenko)

---

## # Quantum Performance Effects


 (ENG) [JavaDay Riga 2013: "Quantum" Performance Effects](#)  
(hosted for Sergey Kuksenko)


 (RUS) [JavaOne Russia 2013: "Quantum" Performance Effects](#)  
(hosted for Sergey Kuksenko)  
Video (courtesy of JPoint): [YouTube, high quality](#) (514 MB)

---

## # Java Platform Performance BOF

 (RUS) [JEEConf 2011: Java Platform Performance BoF](#)


 (RUS) [JavaOne Moscow 2011: Java Platform Performance BoF](#)

 (RUS) [JavaTechDay SPB, 2011: Java Platform Performance BoF](#)

---

## # Assorted

 (ENG) [JVMLS 2013: False Sharing, and @Contented](#)

 (ENG) [JavaOne SF 2011: Java or C++ Practical Advice You Can Use](#)  
(co-presenter with Sergey Kuksenko and Charlie Hunt)  
Video: [Parleys](#)

---

## Papers

 (ENG) [SPECjbb2012: Updated Metrics for a Business Benchmark](#) @ ACM/ICPE, Boston, April 22-25, USA

## Projects

### [Apache Harmony](#)

Open-source implementation of Java 5 under APLv2.  
Parts of Harmony's class libraries are now parts of Android.  
Retired.

### [OpenJDK](#)

Open-source, Oracle-endorsed implementation of Java 8 under GPLv2 with CP exception.  
The most used Java runtime in the world.

### [Java Microbenchmark Harness \(jmh\)](#)

The microbenchmark harness used to drive most of our performance experiments.

### [Java Concurrency Stress Tests \(jctstress\)](#)

The harness and the set of functional tests to break Java implementations on concurrency front.

### [Java Object Layout \(jol\)](#)

The minimalistic tool box for field/object layout studies.

### [Shenandoah GC](#)

Shenandoah is an ultra-low pause time garbage collector that reduces



GC pause times by performing more garbage collection work concurrently with the running Java program.

#### [Epsilon GC](#)

Develop a GC that handles memory allocation but does not implement any actual memory reclamation mechanism. Once the available Java heap is exhausted, the JVM will shut down.

[Other projects on GitHub](#)

## Non-technical articles and toy projects

 (ENG) [XKCD-1110 full-view](#)

