Kotlin Coroutines

Asynchronous Programming Made Simple

Ruslan Ibragimov

- Belarus Kotlin User Group Leader
  - bkug.by
  - June 13: BKUG #4
    - DSL in Kotlin
    - Kotlin in Action
- Java Professionals BY Leader
  - jprof.by
- FullStack Developer at ObjectStyle
  - Kotlin, Java
  - TypeScript
- Kotliner

Kotlin

Kotlin 1.0 - February 2016

Kotlin 1.1 - March 2017

Kotlin Coroutines

- the key new feature in Kotlin 1.1
- brings the support of:
  - async/await
  - yield
  - and more

Kotlin Coroutines

Experimental status

```
// build.gradle
kotlin {
   experimental {
       coroutines 'enable'
   }
}
```

kotlin.coroutines.experimental -> kotlin.coroutines

Hello, World!

```
compile("org.jetbrains.kotlinx:kotlinx-coroutines-core:0.15")
```

```
fun main(args: Array<String>) {
   delay(1000)
   println("Hello, World!")
}
```

Error:(56, 5) Kotlin: Suspend function **'delay'** should be called only from a **coroutine** or another **suspend function**

Hello, World!

```
fun main(args: Array<String>)  {
    launch(CommonPool) {
        delay(1000)
        println("Hello, World!")
    }
}

// Nothing
```

Hello, World!

```
fun main(args: Array<String>)  {
    launch(CommonPool) {
        delay(1000)
        println("Hello, World!")
    }

    Thread.sleep(2000)
}

// Hello, World!
```

Hello, World!

```
fun main(args: Array<String>)  {
    runBlocking {
        delay(1000)
        print("Hello, ")
    }

    print("World!")
}

// Hello, World!
```

Hello, World!

```
fun main(args: Array<String>) = runBlocking {
    val result = http.get(args[0]).await()
    println(result)
}
```

Coroutines & Kotlin

- suspend – language
- low-level core API – kotlin.coroutines (kotlin-stdlib)
- libraries – example: kotlinx.coroutines (kotlinx-coroutines-core)

Suspend

Suspending Functions

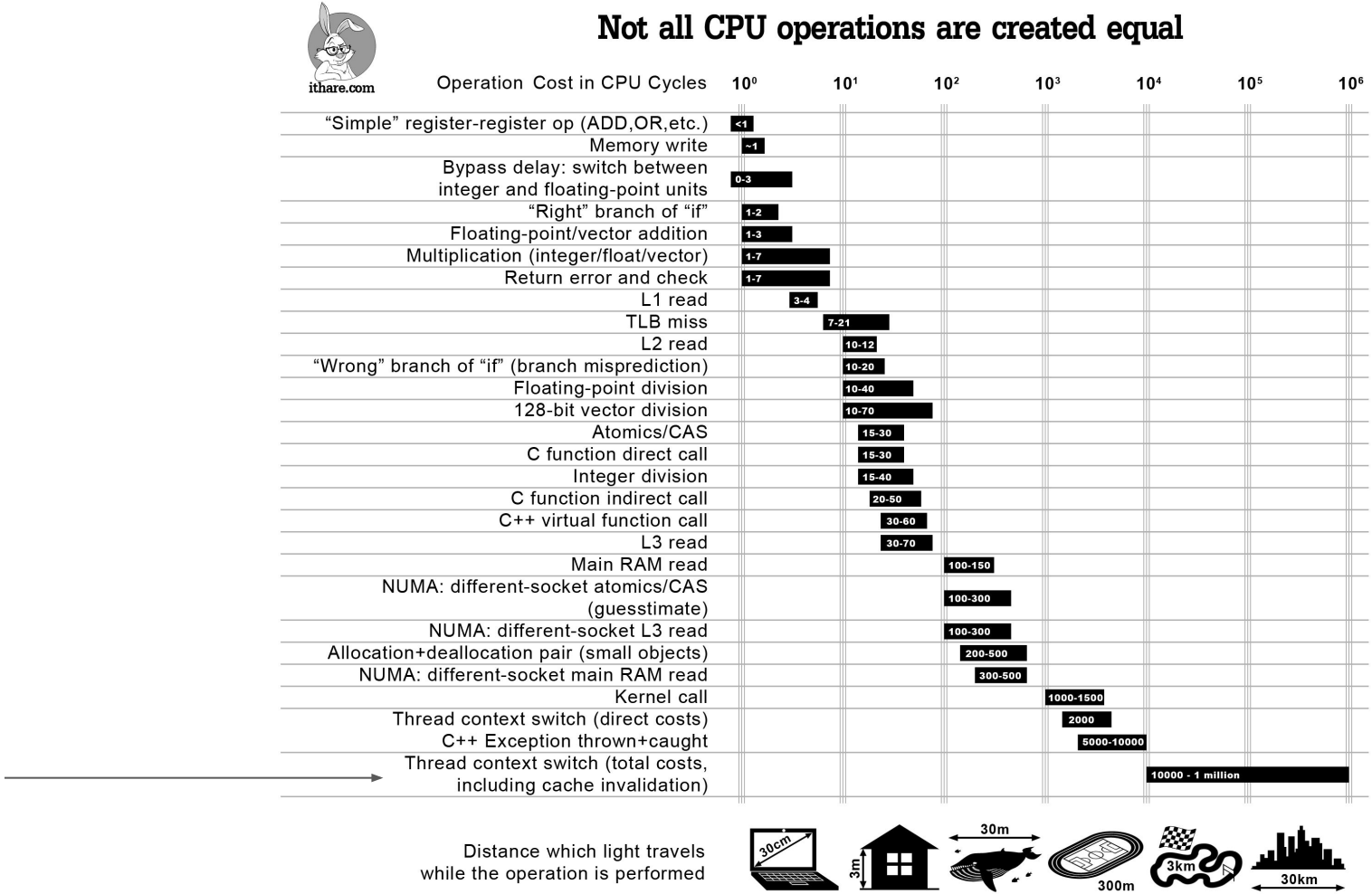
```
suspend fun delay(
    time: Long,
    unit: TimeUnit = TimeUnit.MILLISECONDS
) {
    // ...
}
```

Threads & Coroutines

```
fun getContacts(url: String): String =
    http.get(url)
```

1. Thread Blocked
2. Context Switch
3. Wait
4. Context Switch
5. Continue executing code

Not all CPU operations are created equal



Threads & Coroutines

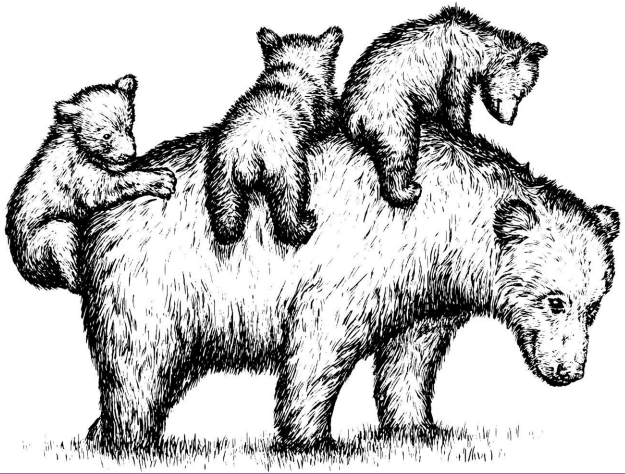
```
fun main(args: Array<String>) {
    (1..100_000).forEach {
        thread(start = true) {
            sleep(1000)
        }
    }
}
```

```
Exception in thread "main" java.lang.OutOfMemoryError: unable to create new native thread
    at java.lang.Thread.start0(Native Method)
    at java.lang.Thread.start(Thread.java:717)
    at kotlin.concurrent.ThreadsKt.thread(Thread.kt:30)
    at kotlin.concurrent.ThreadsKt.thread$default(Thread.kt:15)
    at by.heap.komodo.samples.coroutines.SuspendKt.main(Suspend.kt:40)
```

Threads & Coroutines

```
fun main(args: Array<String>) = runBlocking {
    (1..100_000).forEach {
        launch(CommonPool) {
            delay(1000)
        }
    }
}
```

Getting the wrong idea from that conference talk you attended

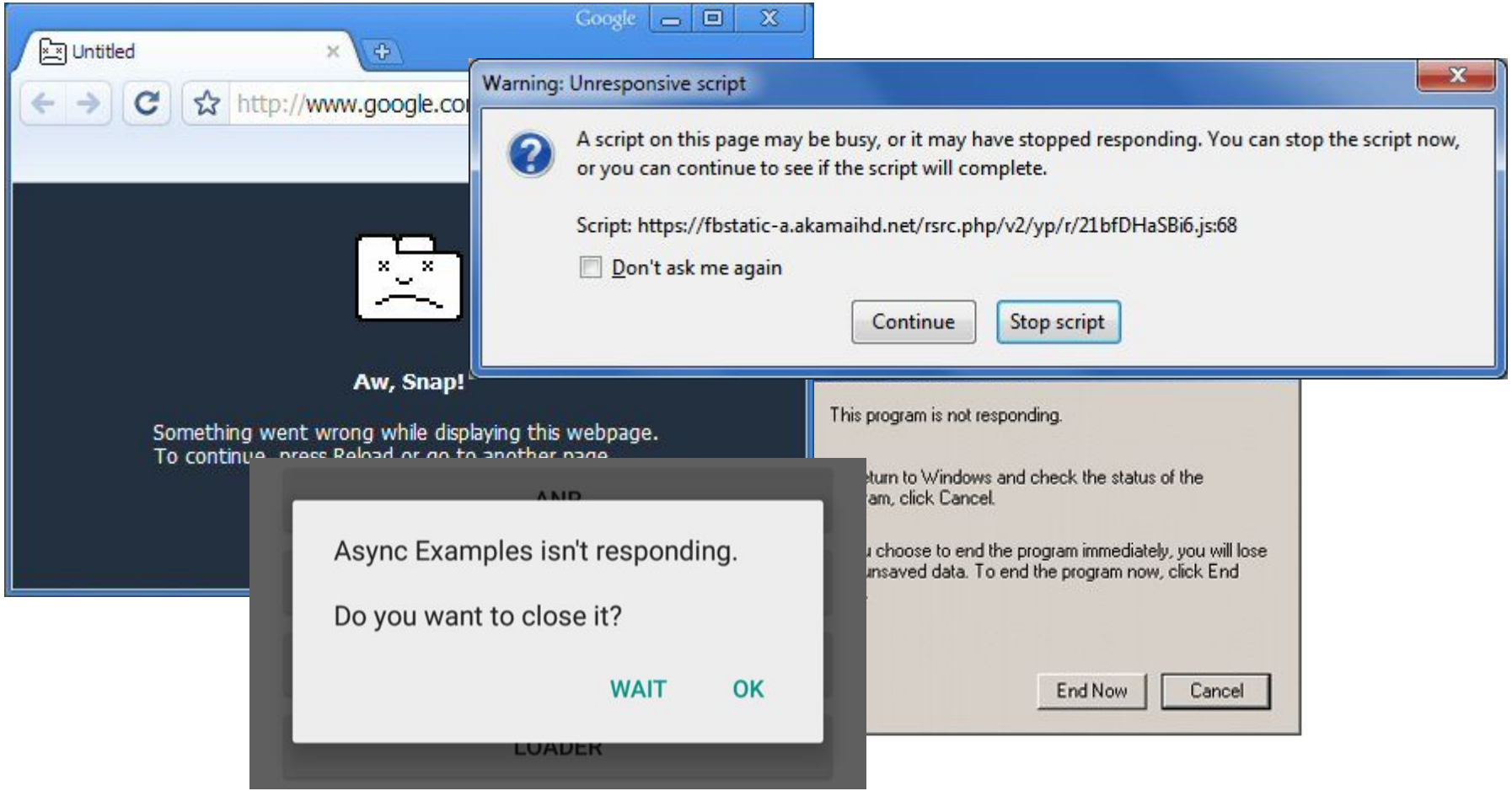


Solving Imaginary Scaling Issues

At Scale

O RLY?

@ThePracticalDev


Google
Untitled
http://www.google.co
Aw, Snap!
Something went wrong while displaying this webpage.
To continue, press Reload or go to another page.
Warning: Unresponsive script
A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete.
Script: https://fbstatic-a.akamaihd.net/rsrc.php/v2/yp/r/21bfDHaSBi6.js:68
Don't ask me again
Continue
Stop script
This program is not responding.
turn to Windows and check the status of the am, click Cancel.
choose to end the program immediately, you will lose unsaved data. To end the program now, click End
End Now
Cancel
Async Examples isn't responding.
Do you want to close it?
WAIT
OK

Suspend

Suspending Functions

```
suspend fun foo() {
    delay(1000)
}
```

```
16
17  suspend fun foo() {
18      delay( time: 1000)
19  }
20
```

Suspending Lambda

```
public fun launch(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    …
}
```

Core API

createCoroutine

kotlin.coroutines.experimental.**createCoroutine**

```
public fun <R, T> (suspend R.() -> T).createCoroutine(
        receiver: R,
        completion: Continuation<T>
): Continuation<Unit> = SafeContinuation(
        createCoroutineUnchecked(receiver, completion),
        COROUTINE_SUSPENDED
)
```

```
block.createCoroutine(receiver, completion)
```

```
launch(CommonPool) {
    delay(1000)
    println("Hello, World!")
}
```

Coroutine

- the term from 1960s
- was used in “The Art of Computer Programming” by Donald Knuth

  a main routine and a subroutine

  vs

  coroutines, which call on each other

Bytecode

```
package by.heap.komodo.samples.coroutines.bytecode

import kotlinx.coroutines.experimental.delay

suspend fun fetch() {
    delay(1000)
}
```

Bytecode

```
-rw-r--r-- 1 yoda yoda 1342 Jun  1 08:03 ExampleKt.class
-rw-r--r-- 1 yoda yoda 1833 Jun  1 08:03 ExampleKt$fetch$1.class
```

Bytecode

```
public final class ExampleKt {
  public static final Object fetch(
    Continuation<? super Unit>
  );
}
```

Bytecode

```
public final class ExampleKt {
    @Nullable
    public static final Object fetch(@NotNull final Continuation<?
super Unit> $continuation) {
        Intrinsics.checkParameterIsNotNull((Object)$continuation,
"$continuation");
        return new
ExampleKt$fetch.ExampleKt$fetch$1((Continuation)$continuation).doR
esume((Object)Unit.INSTANCE, (Throwable)null);
    }
}
```

Bytecode

```
final class ExampleKt$fetch$1 extends CoroutineImpl {
  public final Object doResume(Object, Throwable);
  ExampleKt$fetch$1(Continuation);
}
```

Bytecode

```
static final class ExampleKt$fetch$1 extends CoroutineImpl {
      @Nullable
      public final Object doResume(@Nullable final Object data, @Nullable final Throwable throwable) {
      final Object coroutine_SUSPENDED = IntrinsicsKt.getCOROUTINE_SUSPENDED();
      switch (super.label) {
            case 0: {
            ...
            break;
            }
            case 1: {
                  ...
                  break;
            }
            default: {
                  throw new IllegalStateException("call to 'resume' before 'invoke' with coroutine");
            }
      }
      return Unit.INSTANCE;
      }
}
```

startCoroutine

kotlin.coroutines.experimental.**startCoroutine**

```
public fun <R, T> (suspend R.() -> T).startCoroutine(
        receiver: R,
        completion: Continuation<T>
) {
    createCoroutineUnchecked(receiver, completion).resume(Unit)
}
```

```
block.startCoroutine(receiver, completion)
```

suspendCoroutine

kotlin.coroutines.experimental.**suspendCoroutine**

```
public inline suspend fun <T> suspendCoroutine(
    crossinline block: (Continuation<T>) -> Unit
): T = suspendCoroutineOrReturn { c: Continuation<T> ->
    val safe = SafeContinuation(c)
    block(safe)
    safe.getResult()
}
```

kotlin.coroutines.experimental.**suspendCoroutine**

```
suspend fun <T> CompletableFuture<T>.await(): T =
    suspendCoroutine { cont: Continuation<T> ->
        whenComplete { result, exception ->
            if (exception == null)
                cont.resume(result)
            else
                cont.resumeWithException(exception)
        }
    }
```

suspendCancellableCoroutine

kotlin.coroutines.experimental.

suspendCancellableCoroutine

```
public inline suspend fun <T> suspendCancellableCoroutine(
    holdCancellability: Boolean = false,
    crossinline block: (CancellableContinuation<T>) -> Unit
): T = suspendCoroutineOrReturn { cont ->
    val cancellable = CancellableContinuationImpl(cont, active = true)
    if (!holdCancellability) cancellable.initCancellability()
    block(cancellable)
    cancellable.getResult()
}
```

kotlin.coroutines.experimental.

suspendCancellableCoroutine

```
suspend fun <T> CompletableFuture<T>.await(): T =
    suspendCancellableCoroutine { cont: CancellableContinuation<T> ->
        whenComplete { result, exception ->
            if (exception == null)
                cont.resume(result)
            else
                cont.resumeWithException(exception)
        }
        cont.invokeOnCompletion { this.cancel(false) }
    }
```

kotlinx.coroutines.experimental.(withTimeout/withTimeoutOrNull)

```
withTimeout(100) {
    request.await()
}

withTimeoutOrNull(100) {
    request.await()
}
```

kotlinx.coroutines.experimental.**launch**

```
val job = launch(CommonPool) {
    while (isActive) {
        delay(100)
        println(42)
    }
}
job.cancel()
```

kotlinx.coroutines.experimental.NonCancellable

```
val job = launch(CommonPool) {
    try {
        // ...
    } finally {
        run(NonCancellable) {
            // this code isn't cancelled
        }
    }
}
job.cancel()
```

suspend
createCoroutine
startCoroutine
suspendCoroutine
suspendCancellableCoroutine

Generators

buildSequence

kotlin.coroutines.experimental.

buildSequence

```
public fun <T> buildSequence(
    builderAction: suspend SequenceBuilder<T>.() -> Unit
): Sequence<T> = Sequence { buildIterator(builderAction) }
```

kotlin.coroutines.experimental.buildSequence

```
val lazySeq: Sequence<Int> = buildSequence {
    for (i in 1..100) {
        yield(i)
    }
}

lazySeq.take(3).forEach { print(it) }
// 123
```

kotlin.coroutines.experimental.

buildSequence

```
val lazySeq: Sequence<Int> = buildSequence {
    for (i in 1..100) {
        delay(1000)
        yield(i)
    }
}
```

Error:(22, 9) Kotlin: Restricted suspending functions can only invoke member or extension suspending functions on their restricted coroutine scope

```
public fun <T> buildSequence(
    builderAction: suspend SequenceBuilder<T>.() -> Unit
): Sequence<T> = Sequence { buildIterator(builderAction) }
```

kotlin.coroutines.experimental.

SequenceBuilder

```
@RestrictsSuspension
public abstract class SequenceBuilder<in T> internal constructor() {
    public abstract suspend fun yield(value: T)
    public abstract suspend fun yieldAll(iterator: Iterator<T>)
    ...
}
```

kotlin.coroutines.experimental.

SequenceBuilder

```
suspend fun SequenceBuilder<Int>.answer() {
    this.yield(42)
}

val ultimateAnswerSeq: Sequence<Int> = buildSequence {
    while (true) {
        answer()
    }
}
```

buildIterator

Iterator<T>

buildSequence
buildIterator
@RestrictsSuspension

kotlinx.coroutines

kotlin.coroutines.experimental.

launch

```
public fun launch(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> Unit
): Job {
    ...
}
```

CoroutineContext

- Unconfined
- CommonPool
- newSingleThreadContext, newFixedThreadPoolContext
- Executor.asCoroutineDispatcher

CoroutineStart

```
CoroutineStart.DEFAULT ->
            block.startCoroutine(receiver, completion)
CoroutineStart.UNDISPATCHED ->
            block.startCoroutineUndispatched(receiver, completion)
CoroutineStart.LAZY -> Unit // will start lazily
```

CoroutineScope

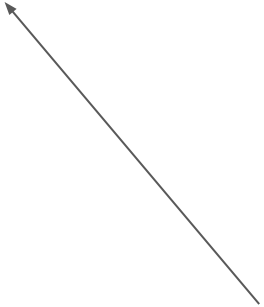
```kotlin
public interface CoroutineScope {
    public val isActive: Boolean
    public val context: CoroutineContext
}
```

kotlin.coroutines.experimental.**async**

```
public fun <T> async(
    context: CoroutineContext,
    start: CoroutineStart = CoroutineStart.DEFAULT,
    block: suspend CoroutineScope.() -> T
): Deferred<T> {
    ...
}
```

kotlin.coroutines.experimental.**yield**

```
suspend fun foo() {
    list.forEach {
        // compute relatively heavy
        yield()
    }
}
```



Shared mutable state and concurrency

- Thread-safe data structures (Atomics)
- Thread confinement fine-grained
- Thread confinement coarse-grained
- Mutual exclusion (suspending)
- Actors
- Read more

Recursive Coroutines

```
suspend fun test() {
    println(Instant.now())
    test()
}

tailrec suspend fun test() {
    println(Instant.now())
    test()
}
```

Debug

-Dkotlinx.coroutines.debug

Thread.currentThread().name

[main @coroutine#2]
[main @coroutine#3]
[main @coroutine#1]

```
public fun newCoroutineContext(context: CoroutineContext):
CoroutineContext = if (DEBUG) context +
CoroutineId(COROUTINE_ID.incrementAndGet()) else context
```

Call Coroutines from Java

```
suspend fun foo(): Int {
    //...
}

fun fooJava(): CompletableFuture<Int> =
    future { foo() }
```

Not Covered

- Channels
- Select
- ...

Learn Kotlin Coroutines

- Guide to kotlinx.coroutines by example
- Coroutines for Kotlin
- #coroutines Kotlin Slack
- Андрей Бреслав — Асинхронно, но понятно. Сопрограммы в Kotlin
- Andrey Breslav — Kotlin Coroutines (JVMLS 2016, old coroutines!)

Q&A

Ruslan Ibragimov @HeapyHop

Belarus Kotlin User Group: https://bkug.by/

Java Professionals BY: http://jprof.by/

Awesome Kotlin: https://kotlin.link/

Slides: https://goo.gl/mAoBXd