KotlinConf

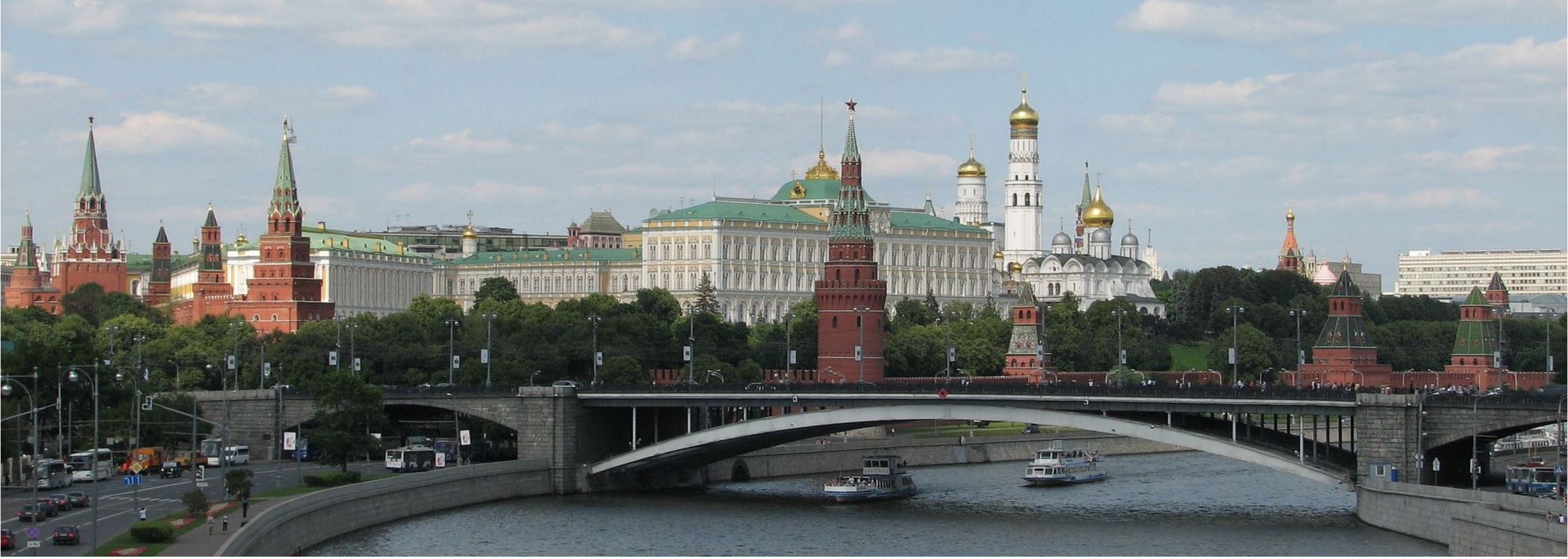
You can, but should you?

Mike Gouline
@mgouline

I live here…

Sydney, Australia

Image courtesy of Mike Gouline



I’m originally from here…

Moscow, Russia

Image courtesy of Artur Janas (Pixabay)

I work here…

Cochlear

Image courtesy of Mike Gouline

I organise this…

Sydney Kotlin User Group

INTRODUCTION TIME

IS OVER

Image courtesy of Fox Broadcasting

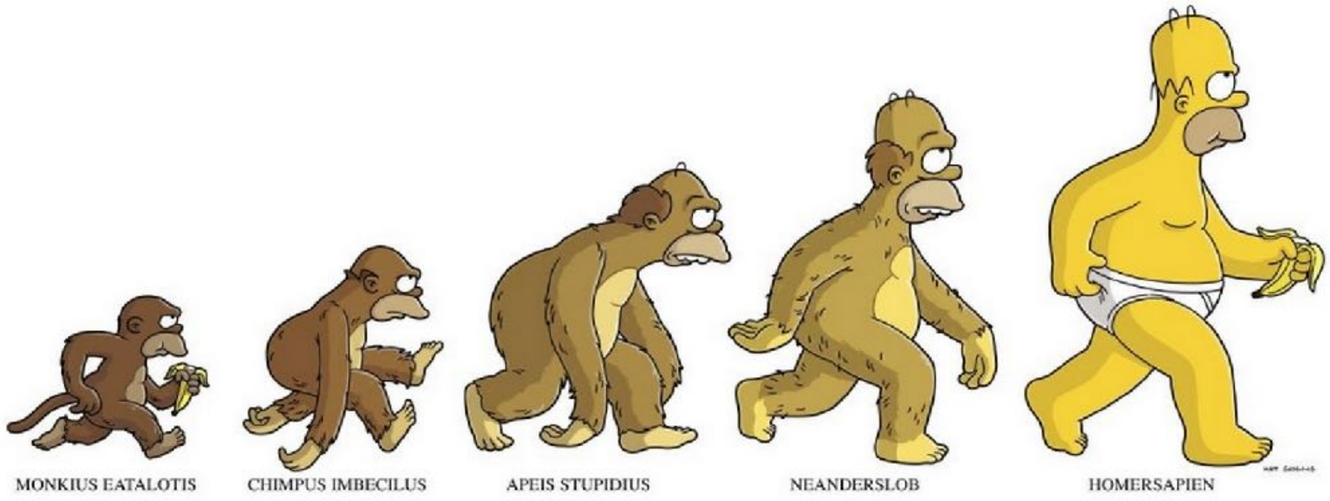
Agenda

- Background
- Potential for problems
- Real-world issues
- Perspective
- Conclusion

KotlinConf

Background

5 stages* of learning a new language



* - Your actual number of stages may vary

Image courtesy of Fox Broadcasting

Stage 1

Reading the ‘Get Started’ section stage

“Hmm, this looks pretty cool…”

Stage 2

Installing tools and running code samples stage

“Err… which version do I need?”

Stage 3

Setting a superficial goal and trying to solve it stage

“Why can’t I just use [insert another language feature]?”

Stage 4

Know enough to write a basic application stage*

"Good enough, I don't need to maintain this code."

* - Also known as the **"JavaScript stage"** or **"expert beginner stage"**

Stage 5

Losing sleep over proper practices stage

*"This works, but is this how I'm **meant** to do it?"*

YOU ARE HERE

New car smell

You want everything to be perfect…

Solve **all** the problems!

KotlinConf

Potential for problems

‘Kid in a candy store’ syndrome

- Various features to choose from
- Java developers may feel overwhelmed

Kotlin is not opinionated

- Other languages give you fewer options
- Kotlin welcomes many audiences/styles/tastes
- Audiences bring their own habits

Time pressure

- Production != pet projects
- Stack Overflow driven development
- “If it ain’t broke…”

3
votes

2
answers

124 views

Not enough mature documentation

- There **are** tutorials/books about what you **can** do
- Not so much what you **shouldn’t** do
  - Static analysis
  - Coding guidelines

KotlinConf

Real-world issues

Warning!

1. My examples are basic*
2. Use your imagination to make them relevant

* - Correction: ~~basic~~ *terrible*

Image courtesy of Nickelodeon

Shadowed variables

- What makes this new? Lambdas!
- Never write nested `it`

```kotlin
// Example #1

getCarsObservable().map {
    it.filter {
        "BMWwmb" == it.make.let {
            it.toUpperCase() + it.toLowerCase()
        }
    }
}
```

```kotlin
// Example #1

getCarsObservable().map { cars ->
    cars.filter { car ->
        "BMWwmb" == car.make.let { carMake ->
            carMake.toUpperCase() + carMake.toLowerCase()
        }
    }
}
```

```kotlin
// Example #2

fun updateAdapter(adapter: Adapter) {
    this.adapter?.clear()

    adapter.setListener(stateListener)
    this.adapter = adapter
}
```

```kotlin
// Example #2

fun updateAdapter(newAdapter: Adapter) {
    adapter?.clear()

    newAdapter.setListener(stateListener)
    adapter = newAdapter
}
```

Opportunistic extension functions

- Extension functions == good
- For extending functionality of an object
- **Not** for creating *any* function with that type

```kotlin
// Example #1

fun Int.toHexString() = String.format("%02X", this)
```

```kotlin
// Example #2

fun Context.getLayoutInflater() =
        getSystemService(Context.LAYOUT_INFLATER_SERVICE)
                as LayoutInflater
```

```kotlin
// Example #2

fun Context.getLayoutInflater() =
        getSystemService(Context.LAYOUT_INFLATER_SERVICE)
                as LayoutInflater

// Alternative
object ContextUtils {
    fun getLayoutInflater(context: Context) =
            context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
                    as LayoutInflater
}
```

```kotlin
// Example #3

fun String.toGitHubApiUrl() = “https://api.github.com/$this"
```

```kotlin
// Example #3

fun String.toGitHubApiUrl() = “https://api.github.com/$this"

// Alternative
object GitHubApiUtils {
    private val BASE_API = "https://api.github.com/"

    fun buildUrl(path: String) = BASE_API + path
}
```

Opportunistic top-level functions

- Same as extension functions
- Autocomplete pollution

Image courtesy of National Geographic

```kotlin
// Carelessly dumping all the movie-related utilities…

const val STAGING_API_CLIENT_KEY = "32nor91fhn23n0fh18h48f7h43f"
const val PRODUCTION_API_CLIENT_KEY = "3901823u94m823xr0h1f30293f8"

fun getItem(adapter: MovieCoverAdapter, position: Int) =
        adapter.items[position]

fun copy(adapter: MovieCoverAdapter) = MovieCoverAdapter(adapter)

fun debugMovieDetails(movie: Movie) {
    println(movie.title)
}
```

```kotlin
/**
 * Some unrelated class working with TV shows.
 */
class TvShowsAdapter : Adapter() {
    init {

    }
}
```

```kotlin
/**
 * Some unrelated class working with TV shows.
 */
class TvShowsAdapter : Adapter() {
    init {
        |
}
```

debugMovieDetails(movie: Movie) (net.gouline.app)	Uni
copy(adapter: MovieCoverAdapter) (net.goulin…	MovieCoverAdapter
STAGING_API_CLIENT_KEY (net.gouline.app)	String
getItem(adapter: MovieCoverAdapter, position: Int) (net.…	Movie
PRODUCTION_API_CLIENT_KEY (net.gouline.app)	String
DEFAULT_BUFFER_SIZE (kotlin.io)	Int
finish() (net.gouline.app)	Unit
main(args: Array<String>) (net.gouline.app)	Unit
Runnable {...} (function: () -> Unit) (java.lang)	Runnable
AutoCloseable {...} (function: () -> Unit) (java…	AutoCloseable
Readable {...} (function: (CharBuffer!) -> Int) (java…	Readable

Use ⇧⌘⏎ to syntactically correct your code after completing (balance parentheses etc.) >> π

Inferred types

- Explicit types optional in many situations
- They can solve typing bugs in others

```kotlin
// Example #1
val allowed = true

// Example #2
val count = 7

// Example #3
val payload = factory.createWithParam(TYPE, "default")

// Example #4
fun checksum(list: List<String>) =
        list.map { it.hashCode() }
                .filter { it != 0 }
                .fold(0) { acc, i -> acc + i * 2 }
                .let { checksumInternal(it) }
```

```kotlin
// Example #1
val allowed = true

// Example #2
val count = 7

// Example #3
val payload: DefaultPayload = factory.createWithParam(TYPE, "default")

// Example #4
fun checksum(list: List<String>): Long? =
        list.map { it.hashCode() }
                .filter { it != 0 }
                .fold(0) { acc, i -> acc + i * 2 }
                .let { checksumInternal(it) }
```

Borrowing from other languages

- Not necessarily a ‘faux pas’
- Just don’t break intentional design

```kotlin
// Go-style defer statement

applyDefers {
    // 1. Open file
    val file = openFile("test.txt")

    // 3. Close file
    defer { closeFile(file) }

    // 2. Write bytes
    file.writeBytes(bytes)
}
```

```kotlin
// Based on Andrey Breslav's sample implementation

class Deferrer {
    private val actions = arrayListOf<() -> Unit>()

    fun defer(f: () -> Unit) {
        actions.add(f)
    }

    fun done() {
        actions.reversed().forEach { it() }
    }
}

inline fun <T> applyDefers(body: Deferrer.(T) -> Unit) {
    val deferrer = Deferrer()
    val result = deferrer.body(this)
    deferrer.done()
    return result
}
```

```kotlin
// Java-style ternary operator

val visibility = visible yes 1 no 0
```

```kotlin
// Java-style ternary operator
val visibility = visible yes 1 no 0
// Easy, but please don't!
class YesNo<out T>(val condition: Boolean, val y: T)
infix fun <T> Boolean.yes(y: T) = YesNo(this, y)
infix fun <T> YesNo<T>.no(n: T) = if (condition) y else n
```

One-liner functions

- Encouraged by Kotlin plugin (since 1.2)
- Return type danger

```kotlin
/**
 * Removes listener for a [pos] in the list.
 */
fun removeListener(pos: Int) {
    listeners.removeAt(pos)
}
```

```kotlin
/**
 * Removes listener for a [pos] in the list.
 */
fun removeListener(pos: Int) = listeners.removeAt(pos)
```

```kotlin
/**
 * Removes listener for a [pos] in the list.
 */
fun removeListener(pos: Int): Listener = listeners.removeAt(pos)
```

```kotlin
// Solution #1

/**
 * Removes listener for a [pos] in the list.
 */
fun removeListener(pos: Int) {
    listeners.removeAt(pos)
}
```

```kotlin
// Solution #2

/**
 * Removes listener for a [position] in the list.
 */
fun removeListener(pos: Int) = listeners.removeAt(pos).ignore()

/**
 * F#-style return type ignore.
 */
fun Any?.ignore() = Unit
```

Seemingly identical solutions

- What would compiler do?
- Performance vs readability

Let’s play…

THE SAME OR

NOT THE SAME

```kotlin
// Example #1: For-loop

// Classic
for (i in 0..10) { print(i) }

// Functional
(0..10).forEach { i -> print(i) }
```

```java
// Example #1: For-loop

// Classic
int i = 0;
for(byte var1 = 11; i < var1; ++i) {
    System.out.print(i);
}

// Functional
byte var0 = 0;
Iterable $receiver$iv = (Iterable)(new IntRange(var0, 10));
Iterator var1 = $receiver$iv.iterator();
while(var1.hasNext()) {
    int element$iv = ((IntIterator)var1).nextInt();
    System.out.print(element$iv);
}
```

```kotlin
// Example #2: Foreach-loop

// Classic
for (i in list) { print(i) }

// Functional
list.forEach { i -> print(i) }
```

```java
// Example #2: Foreach-loop

// Classic
Iterator var2 = list.iterator();
while(var2.hasNext()) {
    String i = (String)var2.next();
    System.out.print(i);
}

// Functional
Iterable $receiver$iv = (Iterable)list;
Iterator var2 = $receiver$iv.iterator();
while(var2.hasNext()) {
    Object element$iv = var2.next();
    String i = (String)element$iv;
    System.out.print(i);
}
```

```kotlin
// Example #3: Argument vs receiver

// Argument
with(list) { print(size) }

// Receiver
list.apply { print(size) }
```

```java
// Example #3: Argument vs receiver

// Argument
int var2 = list.size();
System.out.print(var2);

// Receiver
int var3 = list.size();
System.out.print(var3);
```

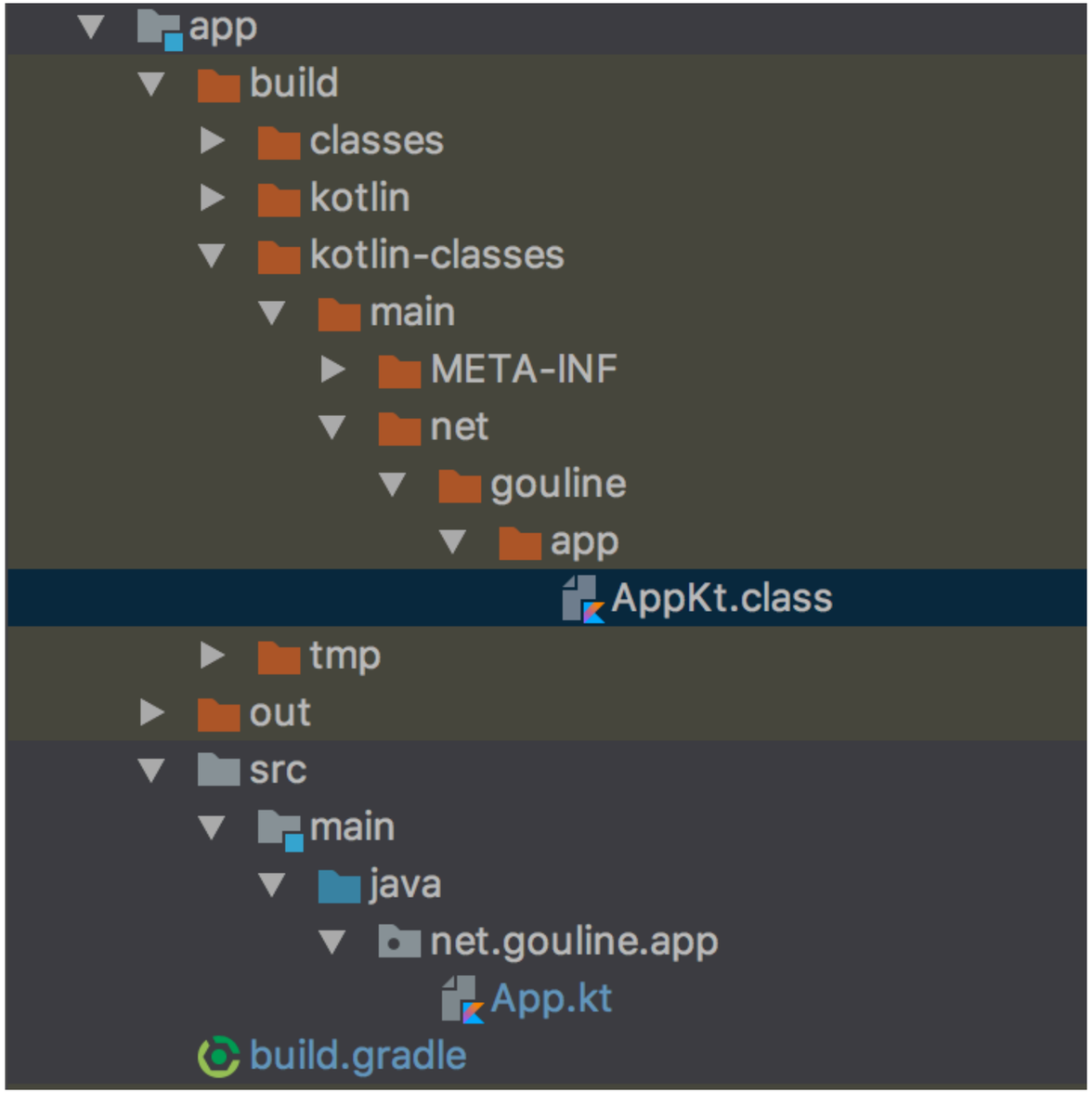
```kotlin
// Example #4: Iterator vs functional

// Iterator
val iterator = list.iterator()
while (iterator.hasNext()) {
    val current = iterator.next()
    if (current % 2 == 0) {
        print(current.toString())
    }
}

// Functional
list.filter { it % 2 == 0 }.forEach { print(it) }
```

```java
// Example #4: Iterator vs functional

// Iterator
Iterator iterator = list.iterator();
while(iterator.hasNext()) {
   int current = ((Number)iterator.next()).intValue();
   if (current % 2 == 0) {
      System.out.print(String.valueOf(current));
   }
}

// Functional
Collection destination$iv$iv = (Collection)(new ArrayList());
Iterator var4 = (Iterable)list.iterator();
while(var4.hasNext()) {
   Object element$iv$iv = var4.next();
   int it = ((Number)element$iv$iv).intValue();
   if (it % 2 == 0) {
      destination$iv$iv.add(element$iv$iv);
   }
}
Iterator var2 = (Iterable)((List)destination$iv$iv).iterator();
while(var2.hasNext()) {
   Object element$iv = var2.next();
   System.out.print(((Number)element$iv).intValue());
}
```

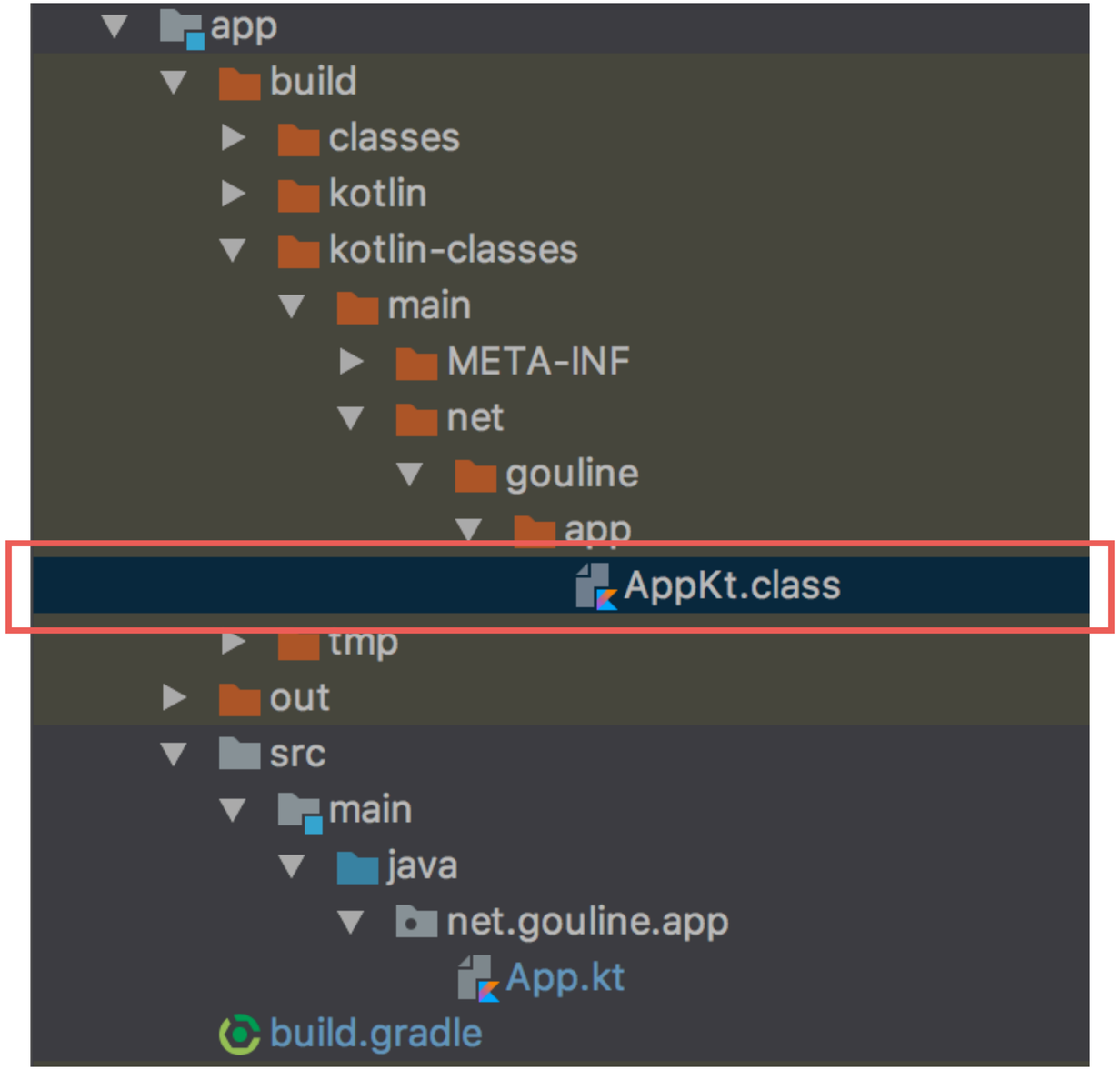
How to decompile?

Let me show you…

Step 1


app
build
classes
kotlin
kotlin-classes
main
META-INF
net
gouline
app
AppKt.class
tmp
out
src
main
java
net.gouline.app
App.kt
build.gradle

Step 1


app
build
classes
kotlin
kotlin-classes
main
META-INF
net
gouline
app
AppKt.class
tmp
out
src
main
java
net.gouline.app
App.kt
build.gradle

Step 2

App.kt × AppKt.class ×

```
1  // IntelliJ API Decompiler stub source generated from a class file
2  // Implementation of methods is not available
3
4  package net.gouline.app
5
6  public fun main(args: kotlin.Array<kotlin.String>): kotlin.Unit { /* compiled code */ }
7
8
```

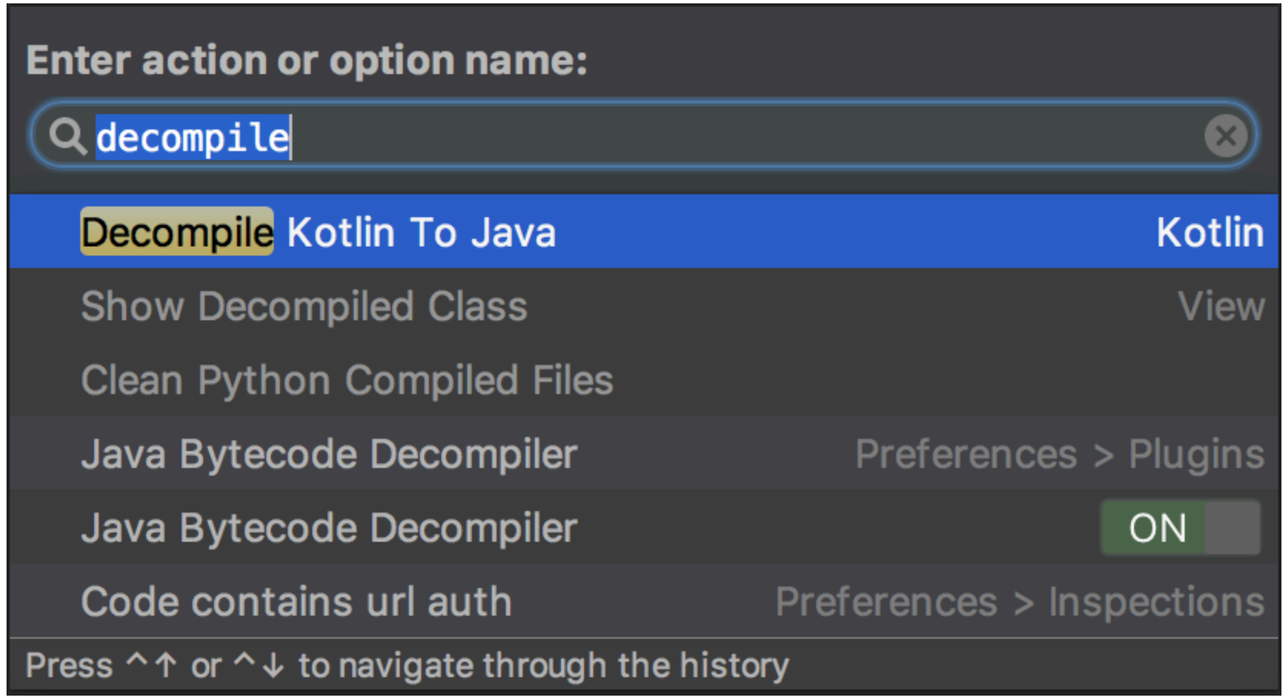
Step 2

App.kt × AppKt.class ×

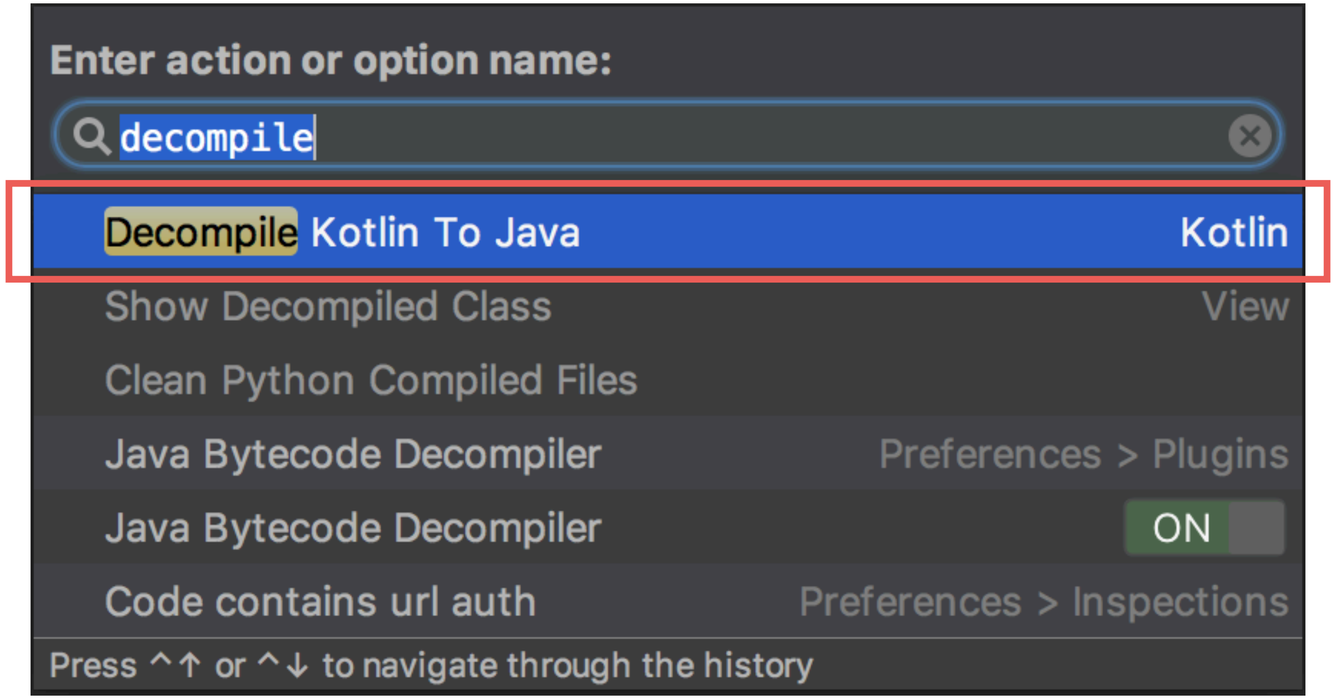
```
1  // IntelliJ API Decompiler stub source generated from a class file
2  // Implementation of methods is not available
3
4  package net.gouline.app
5
6  public fun main(args: kotlin.Array<kotlin.String>): kotlin.Unit { /* compiled code */ }
7
8
```

Cmd + Shift + A

Step 3


Enter action or option name:
decompile
Decompile Kotlin To Java
Kotlin
Show Decompiled Class
View
Clean Python Compiled Files
Java Bytecode Decompiler
Preferences > Plugins
Java Bytecode Decompiler
ON
Code contains url auth
Preferences > Inspections
Press ^↑ or ^↓ to navigate through the history

Step 3


Enter action or option name:
decompile
Decompile Kotlin To Java
Kotlin
Show Decompiled Class
View
Clean Python Compiled Files
Java Bytecode Decompiler
Preferences > Plugins
Java Bytecode Decompiler
ON
Code contains url auth
Preferences > Inspections
Press ^↑ or ^↓ to navigate through the history

Step 4

App.kt | AppKt.class | AppKt.decompiled.java

```
package net.gouline.app;

import ...

@Metadata(
    mv = {1, 1, 9},
    bv = {1, 0, 2},
    k = 2,
    xi = 2,
    d1 = {"\u0000\u0014\n\u0000\n\u0002\u0010\u0002\n\u0000\n\u0002\u0010\u0011\n\u0002\u0010\u000e\n\u0
    d2 = {"main", "", "args", "", "", "([Ljava/lang/String;)V", "app"}
)
public final class AppKt {
    public static final void main(@NotNull String[] args) {
        Intrinsics.checkParameterIsNotNull(args, paramName: "args");
        String var1 = "Hello world!";
        System.out.print(var1);
    }
}
```

KotlinConf

Perspective

Agree or disagree

- Think of reasons why
- Wonder about that for new features

Image courtesy of New Line Cinema

Read your code

- Straight after writing
- Many weeks later
- Many sprints/releases later
- Any changes?

Coding guidelines are crucial

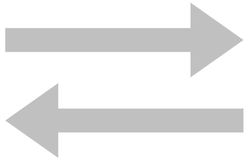
- Formalise (dis)agreements
- Minimise code review bickering
- Refine guidelines regularly

Refine coding guidelines

- Others may be having the same problems
- Many people don’t mind change

What about future features?

- Same process
- If something limits you, speak up



Kool-Aid-free community

- Fanboyism breeds unexplained decisions
- Kotlin developers go against the grain

91

This answer is outdated but remains for historical value. As of Xcode 7, Connor's answer from Jun 8 '15 is more accurate.

No, there are no generics in Objective-C unless you want to use C++ templates in your own custom collection classes (which I strongly discourage).

Objective-C has dynamic typing as a feature, which means that the runtime doesn't care about the type of an object since all objects can receive messages. When you add an object to a built-in collection, they are just treated as if they were type `id`. But don't worry, just send messages to those objects like normal; it will work fine *(unless of course one or more of the objects in the collection don't respond to the message you are sending)*.

Generics are needed in languages such as Java and C# because they are strong, statically typed languages. Totally different ballgame than Objective-C's dynamic typing feature.

share edit flag

edited Sep 8 '16 at 4:09

Aaron Brager
47.1k • 13 • 101 • 193

answered May 11 '09 at 15:32

Marc W
16.8k • 4 • 51 • 69

Performance arguments

- No empty statements
- Decompile your code!

KotlinConf

Conclusion

Conclusion

- Don’t stop refining your code
- Looking for a better way == good
- Balance performance with readability
- Don’t be afraid to disagree
- Back up your disagreement

Thank you!

Mike Gouline
@mgouline

#kotlinconf17