Kotlin DEV DAY

Lies Told By The Compiler

Looking under the hood of Kotlin JVM

POWERED BY INSTIL.

@GarthGilmour

@ryangadams

INSTIL

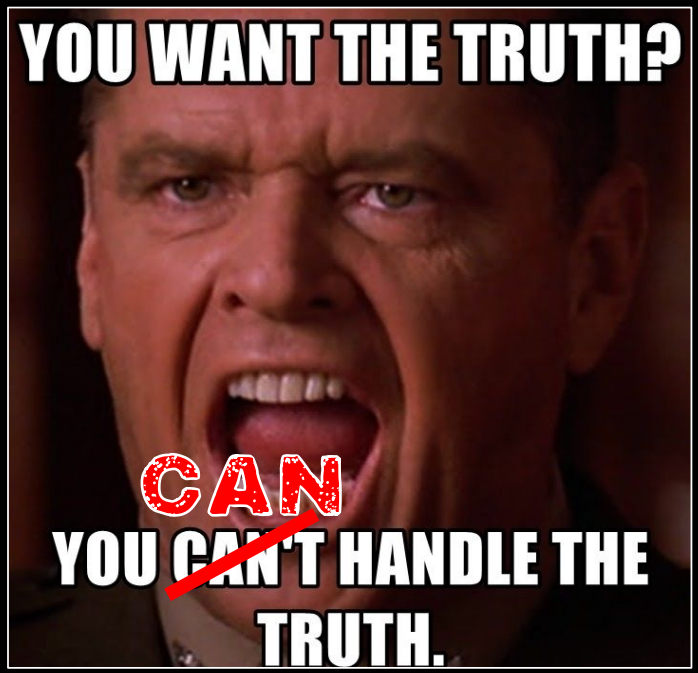
https://instil.co

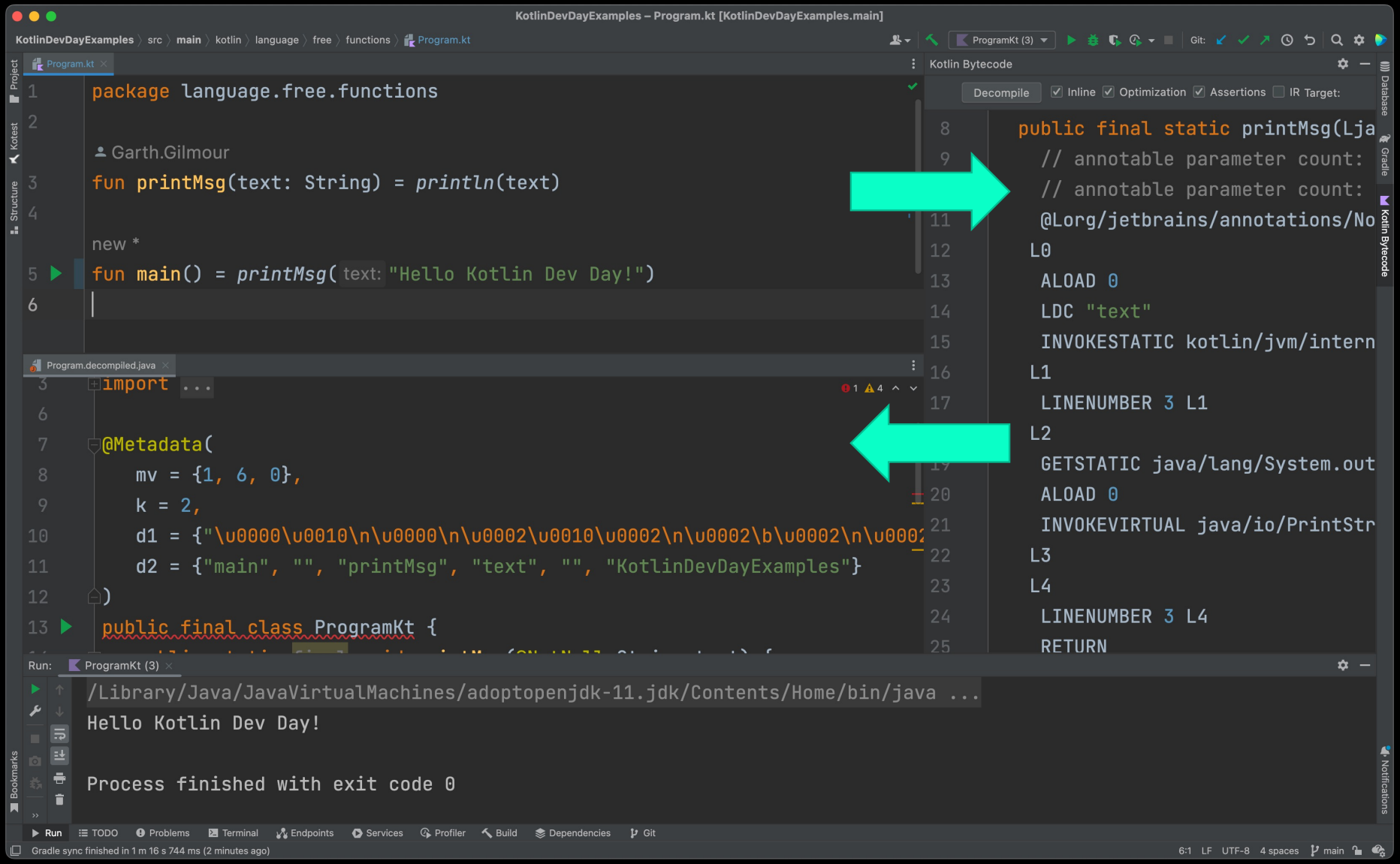
introducing this talk
why do we need lies?

- Kotlin is awesome!
- It provides unique features
- Which don’t exist in reality
- So the compiler must fake them
- Initially we don’t care how this is done

how do we find the truth?
by using a decompiler

- Nothing is ever hidden in software
- We can decompile JVM bytecode produced via the Kotlin compiler
- Similar options exist for other platforms (but more complex)





introducing javap

the java bytecode decompiler

```
% javap
Usage: javap <options> <classes>
where possible options include:
  ...
  -p  -private              Show all classes and members
  -c                        Disassemble the code
  -s                        Print internal type signatures
  ...
  --module-path <path>      Specify where to find application modules
  --system <jdk>            Specify where to find system modules
  ...
  -cp <path>                Specify where to find user class files
  ...
```

Three different levels of deception
lies, damned lies and jetpack compose ☺

- **The Kotlin Language**
- **Suspending Functions & Coroutines**
- **Jetpack Compose (Desktop and Web)**

Lies from the core language

1

Free Functions

free functions

```
//Program.kt

fun printMsg(text: String) = println(text)

fun main() = printMsg("Bonjour Kotlin Dev Day!")
```

```
Bonjour Kotlin Dev Day!
```

free functions

```
public final class ProgramKt {
  public static final void printMsg(String);
    Code:
      15: return


  public static final void main();
    Code:
      5: return


  public static void main(String[]);
    Code:
       3: return
}
```

Nested Functions

nested functions

```
fun main() {
    fun printMsg(text: String) = println(text)

    printMsg("Bonjour Kotlin Dev Day!")
}
```

```
Bonjour Kotlin Dev Day!
```

nested functions

```
public final class ProgramKt {
  public static final void main();
    Code:
       0: ldc           #8      // String Bonjour Kotlin Dev Day!
       2: invokestatic  #12     // Method main$printMsg:(String;)V
       5: return


  public static void main(java.lang.String[]);
    Code:
       0: invokestatic  #15     // Method main:()V
       3: return
```

nested functions

```
  private static final void main$printMsg(java.lang.String);
    Code:
       0: iconst_0
       1: istore_1
       2: getstatic     #23     // Field System.out
       5: aload_0
       6: invokevirtual #29     // Method PrintStream.println:(Object)V
       9: return
}
```

Nested Functions (extended)

nested functions – extended

```
fun main() {
    demo(123)
    demo(45.6)
}

fun demo(input: Int) {
    fun printMsg(text: String) = println(text)
    printMsg("Hello Kotlin Dev Day!")
}

fun demo(input: Double) {
    fun printMsg(text: String) = println(text)
    printMsg("Bonjour Kotlin Dev Day!")
}
```

```
Hello Kotlin Dev Day!
Bonjour Kotlin Dev Day!
```

nested functions – extended

```
public static final void demo(int);
  Code:
     0: ldc           #17   // String Hello Kotlin Dev Day!
     2: invokestatic  #21   // Method demo$printMsg:(String;)V
     5: return


public static final void demo(double);
  Code:
     0: ldc           #25   // String Bonjour Kotlin Dev Day!
     2: invokestatic  #28   // Method "demo$printMsg-0":(String;)V
     5: return
```

nested functions - extended

```
  private static final void demo$printMsg(String);
    Code:
       0: iconst_0
       1: istore_1
       2: getstatic     #40     // Field System.out
       5: aload_0
       6: invokevirtual #46     // Method PrintStream.println:(Object;)V
       9: return


  private static final void demo$printMsg-0(String);
    Code:
       0: iconst_0
       1: istore_1
       2: getstatic     #40     // Field System.out
       5: aload_0
       6: invokevirtual #46     // Method PrintStream.println:(Object;)V
       9: return
}
```

Primary Constructors

constructors

```
class Person(val name: String, var age: Int) {
    private val fullName: String

    init {
        fullName = "$name Jones"
    }

    override fun toString() = "$fullName aged $age"
}
 fun main() {
     val person = Person("Bridget", 30)
     println(person)
 }
```

```
Bridget Jones aged 30
```

constructors

```
public final class Person {
  private final String name;
  private int age;
  private final String fullName;

  public Person(String, int);
    Code:
      26: ldc           #27     // String  Jones
      28: invokestatic  #31     // Intrinsics.stringPlus:(String; Object;)String;
      31: putfield      #34     // Field fullName:String;
      34: nop
      35: return

  public final String getName();
  public final int getAge();
  public final void setAge(int);
  public String toString();
}
```

Extension Methods

extension methods

```
class Person(val name: String)

fun Person.sayHello() = println("Hello from $name")

fun String.times(num: Int) = (1..num).joinToString { "$this" }

fun main() {
    val person = Person("Jane")
    person.sayHello()

    println("Dave".times(3))
}
```

```
Hello from Jane
Dave, Dave, Dave
```

extension methods

```
public static final void sayHello(Person);
  Code:
    ...
    25: return


public static final String times(String, int);
  Code:
    ...
    40: return
```

Destructuring

destructuring

```
data class Person(val name: String, val age: Int)

fun main() {
    val person = Person("Lucy", 36)
    val (x, y) = person

    println(x)
    println(y)
}
```

```
Lucy
36
```

destructuring

```
public final Person {
  private final String name;
  private final int age;

  public Person(String, int);
  public final String getName();
  public final int getAge();

  public final String component1();
  public final int component2();

  public final Person copy(String, int);
  public static Person copy$default(Person, String, int, int, Object);

  public java.lang.String toString();
  public int hashCode();
  public boolean equals(java.lang.Object);
}
```

destructuring

```
public final class ProgramKt {
  public static final void main();
    Code:
      ...
      15: invokevirtual #18  // Person.component1:()String;
      18: astore_2
      19: aload_1
      20: invokevirtual #22  // Person.component2:()I
      23: istore_3
      ...
      44: return

  public static void main(java.lang.String[]);
    Code:
       0: invokestatic  #46   // Method main:()V
       3: return
}
```

Object Declarations

object declarations

```
object Math {
    fun add(no1: Int, no2: Int) = no1 + no2
}

fun main() {
    println(Math.add(12,34))
}
```

46

object declarations

```
public final class Math {
  public static final Math INSTANCE;

  private Math();

  public final int add(int, int);

  static {};
    Code:
       0: new           #2    // class Math
       3: dup
       4: invokespecial #17   // Method "<init>":()V
       7: putstatic     #20   // Field INSTANCE:Math;
      10: return
}
```

object declarations

```
public final class ProgramKt {
  public static final void main();
    Code:
       0: getstatic     #12       // Field Math.INSTANCE:Math;
       3: bipush        12
       5: bipush        34
       7: invokevirtual #16       // Method Math.add:(II)I
      10: istore_0
      11: iconst_0
      12: istore_1
      13: getstatic     #22       // Field System.out;
      16: iload_0
      17: invokevirtual #28       // Method PrintStream.println:(I)V
      20: return

  public static void main(String[]);
}
```

Companion Objects

companion objects

```
class Employee(val name: String, val dept: String) {
    companion object {
        fun buildForHR(name: String) = Employee(name, "HR")
        fun buildForIT(name: String) = Employee(name, "IT")
    }

    override fun toString() = "$name working in $dept"
}

fun main() {
    val emp1 = Employee.buildForHR("Dave")
    val emp2 = Employee.buildForIT("Jane")

    println(emp1)
    println(emp2)
}
```

```
Dave working in HR
Jane working in IT
```

companion objects

```
public final class Employee {
  public static final Employee$Companion Companion;
  private final String name;
  private final String dept;

  public Employee(String, String);
  public final String getName();
  public final String getDept();
  public String toString();

  static {};
    Code:
       0: new           #45  // class Employee$Companion
       3: dup
       4: aconst_null
       5: invokespecial #48  // Employee$Companion."<init>"
       8: putstatic     #52  // Field Companion:Employee$Companion;
      11: return
}
```

Companion Objects (renamed)

companion objects (renamed)

```
class Employee(val name: String, val dept: String) {
    companion object EmployeeFactory {
        fun buildForHR(name: String) = Employee(name, "HR")
        fun buildForIT(name: String) = Employee(name, "IT")
    }

    override fun toString() = "$name working in $dept"
}

fun main() {
    val emp1 = Employee.buildForHR("Dave")
    val emp2 = Employee.buildForIT("Jane")

    println(emp1)
    println(emp2)
}
```

```
Dave working in HR
Jane working in IT
```

companion objects (renamed)

```
public final class Employee {
  private final Ljava/lang/String; name
  private final Ljava/lang/String; dept
  public final static Employee$EmployeeFactory; EmployeeFactory

  public toString()Ljava/lang/String;
  public final getName()Ljava/lang/String;
  public final getDept()Ljava/lang/String;

  public <init>(Ljava/lang/String;Ljava/lang/String;)V

  // access flags 0x8
  static <clinit>()V
    NEW Employee$EmployeeFactory
    DUP
    ACONST_NULL
    INVOKESPECIAL Employee$EmployeeFactory.<init> (...)V
    PUTSTATIC Employee.EmployeeFactory : Employee$EmployeeFactory;
    RETURN
    MAXSTACK = 3
    MAXLOCALS = 0
}
```

Enumerations

Enumerations

```
enum class Colour {
    Red,
    Green,
    Blue
}

fun main() {
    Colour.values().forEach(::println)
}
```

```
Red
Green
Blue
```

Enumerations

```
public final class Colour extends java.lang.Enum<Colour> {
  public static final Colour Red;
  public static final Colour Green;
  public static final Colour Blue;

  private static final Colour[] $VALUES;

  private Colour();

  public static Colour[] values();
  public static Colour valueOf(String);

  private static final Colour[] $values();
```

Enumerations

```
static {};
  Code:
     0: new           #2      // class Colour
     3: dup
     4: ldc           #47     // String Red
     6: iconst_0
     7: invokespecial #48     // Method "<init>":(String;I)V
    10: putstatic     #39     // Field Red:Colour;
    13: new           #2      // class Colour
    16: dup
    17: ldc           #49     // String Green
    19: iconst_1
    20: invokespecial #48     // Method "<init>":(String;I)V
    23: putstatic     #42     // Field Green:Colour;
```

Enumerations

```
      26: new           #2      // class Colour
      30: ldc           #50     // String Blue
      32: iconst_2
      33: invokespecial #48     // Method "<init>":(String;I)V
      36: putstatic     #45     // Field Blue:Colour;
      39: invokestatic  #52     // Method $values:()[Colour;
      42: putstatic     #22     // Field $VALUES:[Colour;
      45: return
}
```

Delegated Properties

Delegated Properties

```
class Person(val name: String, job: String) {
    var job: String by LoggingDelegate(job)
}

fun main() {
    val person = Person("Jane", "Junior Developer")

    person.job = "Senior Developer"
    println(person.job)
}
```

```
Creating logger for property 'job'
Writing to 'job'
Reading from 'job'
Senior Developer
```

Delegated Properties

```
class LoggingDelegate<T>(var value: T) : ReadWriteProperty<Any?, T> {
    operator fun provideDelegate(
        thisRef: Any?,
        prop: KProperty<*>
    ): ReadWriteProperty<Any?, T> {
        println("Creating logger for property '${prop.name}'")
        return this
    }

    override fun setValue(thisRef: Any?, property: KProperty<*>, value: T) {
        println("Writing to '${property.name}'")
        this.value = value
    }

    override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        println("Reading from '${property.name}'")
        return value
    }
}
```

Delegated Properties

```
public final class Person {

  ...

  private final Lkotlin/properties/ReadWriteProperty; job$delegate

  public <init>(Ljava/lang/String;Ljava/lang/String;)V
    ...
    NEW language/delegated/properties/LoggingDelegate
    ...
```

Delegated Properties

```
public final getJob()Ljava/lang/String;
  ...
  ALOAD 0
  GETFIELD language/delegated/properties/Person.job$delegate
  ALOAD 0
  GETSTATIC language/delegated/properties/Person.$$delegatedProperties
  ICONST_0
  AALOAD
  INVOKEINTERFACE kotlin/properties/ReadWriteProperty.getValue (...)Object;
  CHECKCAST java/lang/String
  ARETURN
  ...
```

Delegated Properties

```
  public final setJob(Ljava/lang/String;)V
    ...
    GETFIELD language/delegated/properties/Person.job$delegate
    ALOAD 0
    GETSTATIC language/delegated/properties/Person.$$delegatedProperties
    ICONST_0
    AALOAD
    ALOAD 1
    INVOKEINTERFACE kotlin/properties/ReadWriteProperty.setValue (...)V
    RETURN
    ...

}
```

Lambdas With Receivers

lambdas with receivers

```
data class Person(val name: String, var age: Int)

fun demo(person: Person, action: Person.() -> Unit) = person.apply(action)

fun main() {
    val person = Person("Jane", 25)
    demo(person) {
        age += 10
        println(this)
    }
}
```

Person(name=Jane, age=35)

lambdas with receivers

```
public final class ProgramKt {
  public static final Person demo(
    Person,
    kotlin.jvm.functions.Function1<? super Person, kotlin.Unit>
  );
    Code:
      ...
      21: invokeinterface #24,2 //Function1.invoke:(Object;)Object;
      ...

 public static final void main();
    Code:
      ...
      13: getstatic     #42  // Field ProgramKt$main$1.INSTANCE:ProgramKt$main$1;
      16: checkcast     #20  // class kotlin/jvm/functions/Function1
      19: invokestatic  #44  // Method demo:(Person;Function1;)Person;
      ...
```

lambdas with receivers

```
javap -cp kotlin-stdlib-1.5.10.jar -c -p kotlin.jvm.functions.Function1

public interface Function1<P1, R> extends Function<R> {
  public abstract R invoke(P1);
}
```

```
javap -cp kotlin-stdlib-1.5.10.jar -c -p kotlin.jvm.functions.Function

public interface kotlin.Function<R> {
}
```

lambdas with receivers

```
javap -cp kotlin-stdlib-1.5.10.jar -c -p kotlin.jvm.functions.Function2

public interface Function2<P1, P2, R> extends Function<R> {
  public abstract R invoke(P1, P2);
}
```

```
javap -cp kotlin-stdlib-1.5.10.jar -c -p kotlin.jvm.functions.Function3

public interface Function3<P1, P2, P3, R> extends Function<R> {
  public abstract R invoke(P1, P2, P3);
}
```

Lambdas With Receivers (21+1 params)

lambdas with receivers (21+1 params)

```
typealias EvilLambda = Person.(
    p1: Int,
    p2: Int,
    p3: Int,
    p4: Int,
    ...
    p14: Int,
    p15: Int,
    p16: Int,
    p17: Int,
    p18: Int,
    p19: Int,
    p20: Int,
    p21: Int
) -> Unit
```

lambdas with receivers (21+1 params)

```
fun demo(
    person: Person,
    action: EvilLambda
) = person.action(1, 2, 3, 4 ... 14, 15, 16, 17, 18, 19, 20, 21)

fun main() {
    val person = Person("Jane", 25)
    demo(person) { a, b, c, d ... n, o, p, q, r, s, t, u ->
        age += 10
        println(this)
    }
}
```

lambdas with receivers (21+1 params)

```
public final static demo(Person;Lkotlin/jvm/functions/Function22;)V
   ...
   ALOAD 1
   ALOAD 0
   ICONST_1
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   ICONST_2
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   ...
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   BIPUSH 20
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   BIPUSH 21
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   INVOKEINTERFACE kotlin/jvm/functions/Function22.invoke (...)Object;
   POP
   RETURN
```

Lambdas With Receivers (22+1 params)

lambdas with receivers (22+1 params)

```
typealias EvilLambda = Person.(
    p1: Int,
    p2: Int,
    p3: Int,
    p4: Int,
    ...
    p14: Int,
    p15: Int,
    p16: Int,
    p17: Int,
    p18: Int,
    p19: Int,
    p20: Int,
    p21: Int,
    p22: Int,
) -> Unit
```

lambdas with receivers (22+1 params)

```
fun demo(
    person: Person,
    action: EvilLambda
) = person.action(1, 2, 3, 4 ... 14, 15, 16, 17, 18, 19, 20, 21, 22)

fun main() {
    val person = Person("Jane", 25)
    demo(person) { a, b, c, d ... n, o, p, q, r, s, t, u, v ->
        age += 10
        println(this)
    }
}
```

lambdas with receivers (22+1 params)

```
public final static demo(Person;Lkotlin/jvm/functions/FunctionN;)V
   ALOAD 1
   ALOAD 0
   ICONST_1
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   ICONST_2
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   ...
   BIPUSH 20
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   BIPUSH 21
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   BIPUSH 22
   INVOKESTATIC java/lang/Integer.valueOf (I)Ljava/lang/Integer;
   INVOKEINTERFACE kotlin/jvm/functions/FunctionN.invoke (...)Object;
   POP
   RETURN
```

Lies told when suspending

2

consider suspending functions
is it all a lie?

- **Do suspending functions actually suspend?**
- **Are they really paused and then resumed?**
- **Let's use Ktor & Coroutines to investigate**
  - Because that's the main use case
  - Note other examples exist (e.g. Arrow)

let’s order breakfast
via a REST endpoint

- **First, we write some entity classes**
  - To represent food and beverages
- **Second, we write a server**
  - Which provides options for breakfast
- **Finally, we write a client that suspends**
  - This is the code we will dissasemble

our data model

```
enum class BeverageSize { ... }

enum class BeverageType { ... }

@Serializable
class Beverage( ... ) { ... }

@Serializable
class Food( ... ) { ... }

class Breakfast( ... ) { ... }
```

our server-side

```
fun Application.configureBreakfast() {
    routing {
        get("/breakfast/beverage") {
            call.respond(selectBeverage())
        }
        post("/breakfast/food") {
            val beverage = call.receive<Beverage>()
            call.respond(selectFood(beverage))
        }
    }
}
```

our client

```
fun main() = runBlocking {
    printInDetail("Program starts")
    val client = buildHttpClient()
    client.use {
        val breakfast = orderBreakfast(client)
        printInDetail("Breakfast ordered")
        breakfast.consume()
    }
    printInDetail("Program ends")
}
```

```
[Program starts at 12:21:16 on 1]
[Ordering breakfast at 12:21:16 on 24]
[Breakfast ordered at 12:21:17 on 1]
Drinking a Medium Americano
Eating 2 plain Pancakes
[Program ends at 12:21:17 on 1]
```

our client

```
suspend fun orderBreakfast(
    client: HttpClient
): Breakfast = withContext(Dispatchers.IO) {

    printInDetail("Ordering breakfast")

    val beverage: Beverage = orderBeverage(client)
    val food: Food = orderFood(client, beverage)
    Breakfast(beverage, food)
}
```

our client

```
private suspend fun orderFood(it: HttpClient, beverage: Beverage): Food =
    it.post("/breakfast/food") {
        body = beverage
        contentType(ContentType.Application.Json)
    }

private suspend fun orderBeverage(it: HttpClient): Beverage =
    it.get("/breakfast/beverage") {
        accept(ContentType.Application.Json)
    }
```

let's disassemble `orderBreakfast'
to see what lies are found within...

- **Note this is a suspending function**
- **Which calls two other suspending functions**

```
suspend fun orderBreakfast(client: HttpClient): Breakfast = withContext(Dispatchers.IO) {
    printInDetail( item: "Ordering breakfast")

    val beverage: Beverage = orderBeverage(client)
    val food: Food = orderFood(client, beverage)
    Breakfast(beverage, food)  ^withContext
}
```

let's disassemble `orderBreakfast`

```
public static final Object orderBreakfast(...) {
   return BuildersKt.withContext(... , ... {
      ...
      public final Object invokeSuspend(...) {
         ...
         label17: {
            ...
            switch(this.label) {
            case 0:
               ...
            case 1:
               ...
            case 2:
               ...
            default:
               ...
            }
            ...
         }
         ...
      }
      ...
   }), ...);
}
```

why it's just a big switch statement!

...nested inside a labelled block

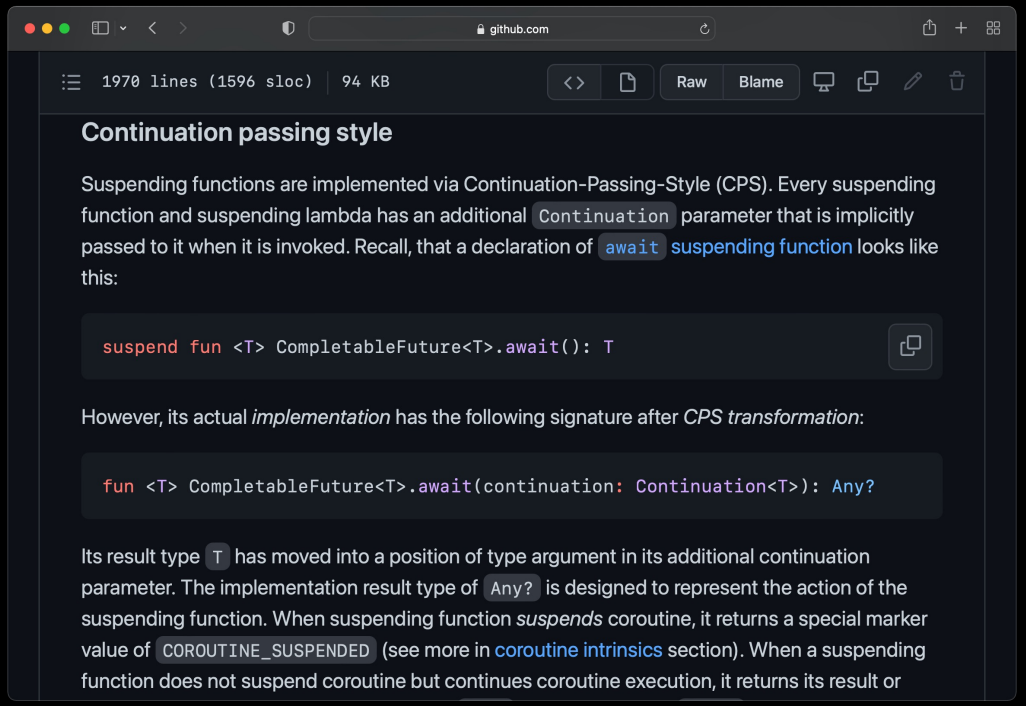
understanding suspending functions

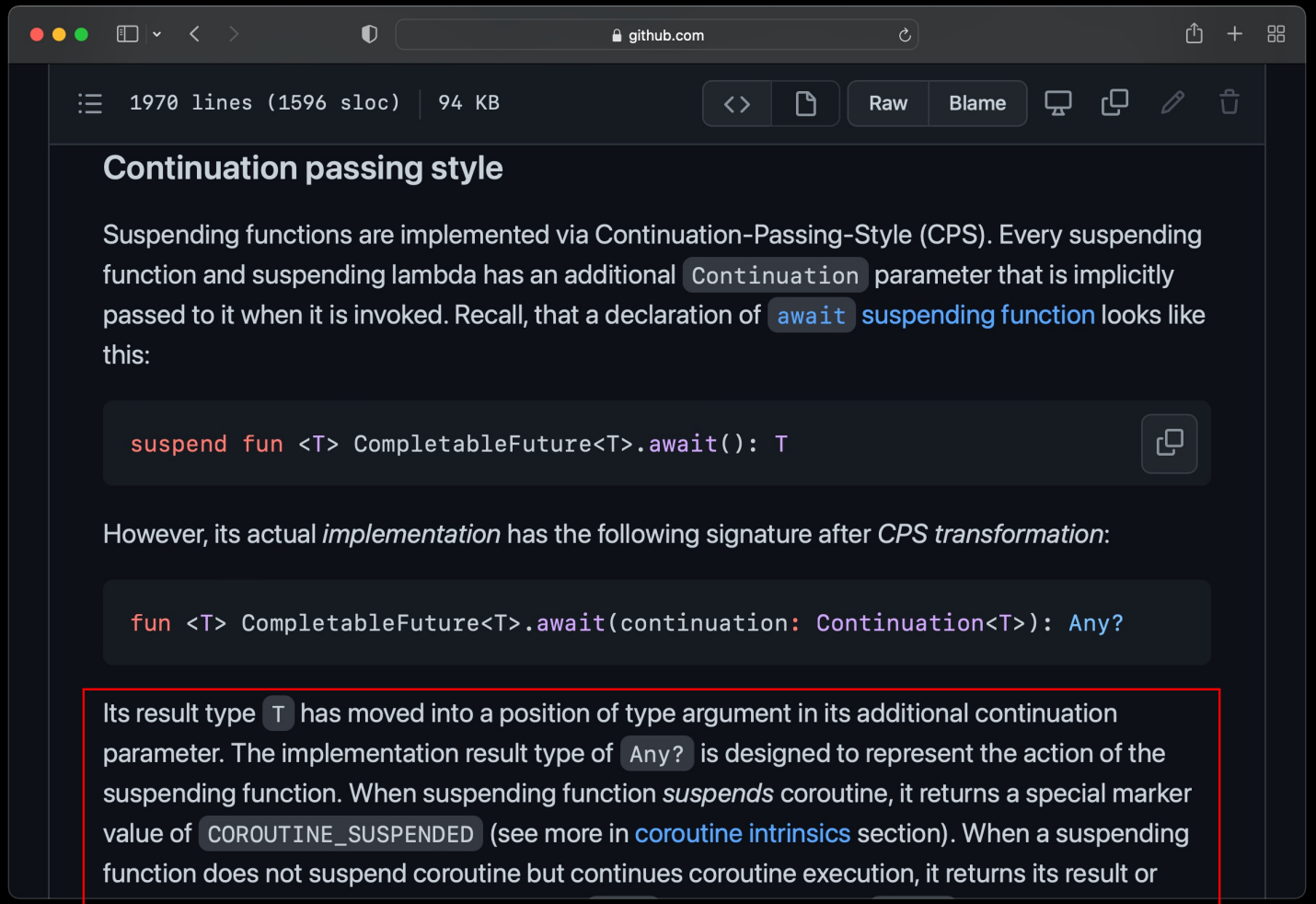
```
label17: {
    Object var4 = IntrinsicsKt.getCOROUTINE_SUSPENDED();
    HttpClient var5;
    switch(this.label) {
    case 0:
        ResultKt.throwOnFailure($result);
        UtilsKt.printInDetail("Ordering breakfast");
        var5 = client;
        this.label = 1;
        var10000 = ProgramKt.orderBeverage(var5, this);
        if (var10000 == var4) {
            return var4;
        }
        break;
```

var4 will hold the constant value for signaling suspension

we suspend by returning if that's what the *orderBeverage* function wants

how do suspending functions work?
via continuations...





understanding suspending functions

```
beverage = (Beverage)var10000;
var5 = client;
this.L$0 = beverage;
this.label = 2;

var10000 = ProgramKt.orderFood(var5, beverage, this);
if (var10000 == var4) {
    return var4;
}
```

when we have the return value from *orderBeverage* we store it and set the label to 2

we suspend by returning if that's what the *orderFood* function wants

understanding suspending functions

```
case 2:
    beverage = (Beverage)this.L$0;
    ResultKt.throwOnFailure($result);
    var10000 = $result;
    break label17;
```

when we have the return value from *orderFood* we retrieve the beverage from storage, put the food in *var10000* and use a labelled block to exit the switch statement

understanding suspending functions

```
public final Object invokeSuspend(@NotNull Object $result) {
    Object var10000;
    Beverage beverage;
    label17: {
        //switch statement lives here
    }

    Food food = (Food)var10000;
    return new Breakfast(beverage, food);
}
```

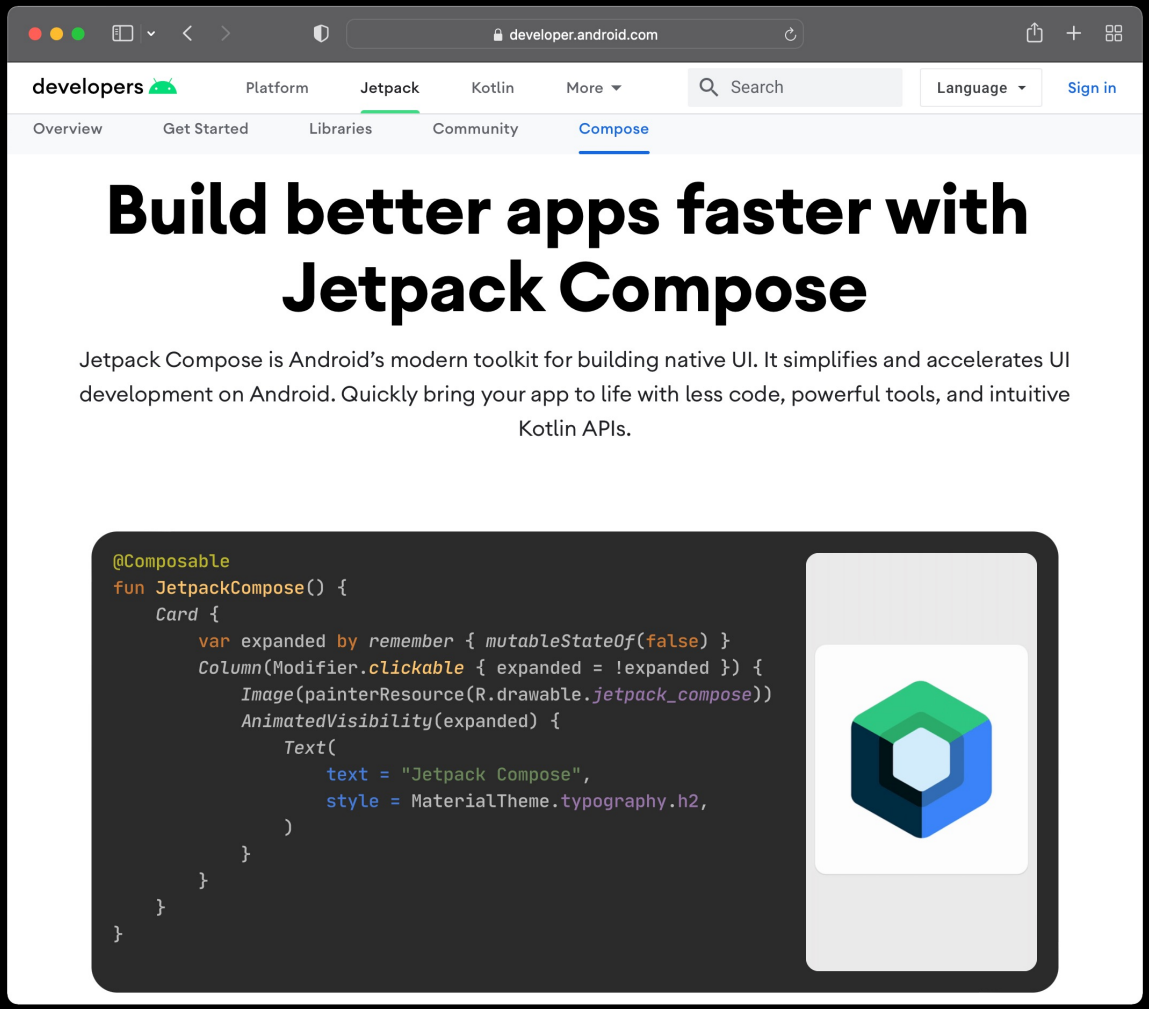
Once the mechanics of calling the other suspending functions are complete, we can build the final return value from the incremental results

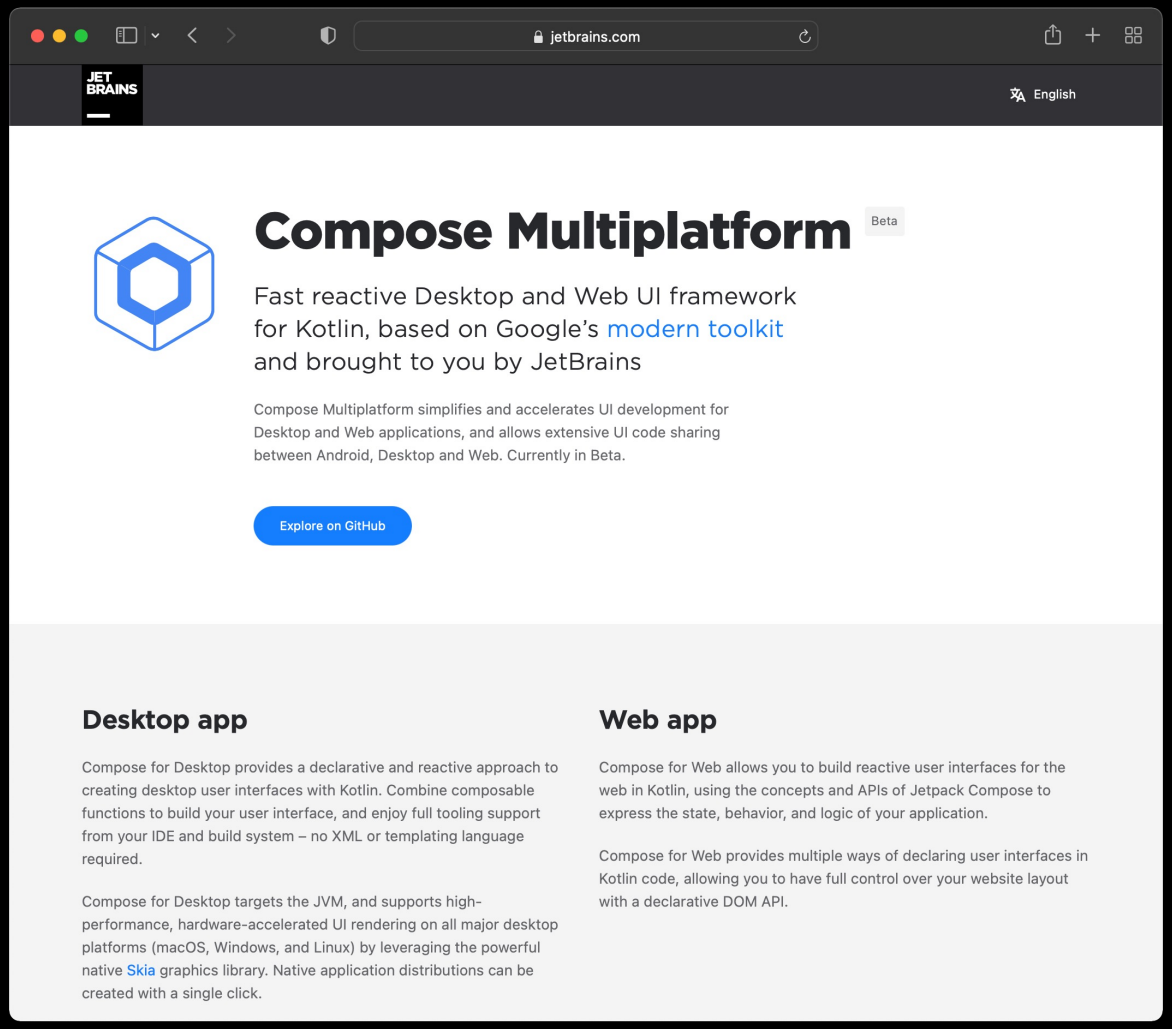
suspending functions do not suspend!!!
they return and are re-invoked

- Continuation objects hold labels and intermediate results
- A switch statement resumes processing at the correct point
- A labelled block short-circuits the switch when we are done

Lies told by compose

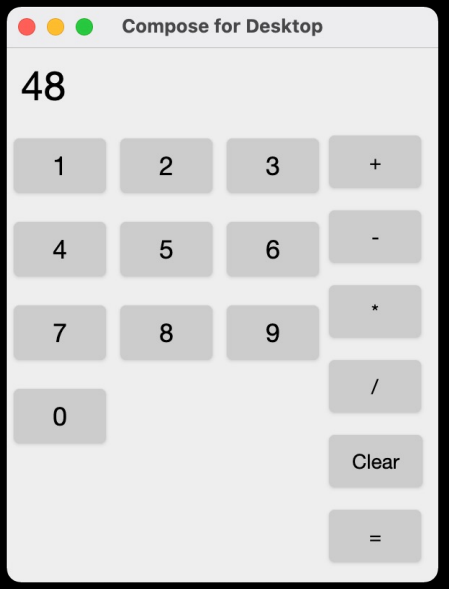
3





working with desktop compose
creating a simple calculator

- **Compose is implemented as a DSL**
- **But goes beyond a purely Internal DSL**
- **A compiler plugin is required**
  - To generate supporting infrastructure
- **This implies we are being deceived**
- **Let's look at a sample application...**



an example widget

```
@Composable
fun OperationButton(onClick: () -> Unit, label: String) =
    Button(
        onClick = onClick,
        modifier = Modifier.padding(all = 2.dp)
    ) {
        Text(
            label,
            style = TextStyle(color = Color.Black, fontSize = 14.sp)
        )
    }
```

decompiling the widgets

```
@Composable
fun DisplayText(text: String) =
    Text(
        text = text,
        style = TextStyle(color = Color.Black, fontSize = 28.sp),
        modifier = Modifier.padding(all = 10.dp)
    )
```

```
public static final void DisplayText(String, Composer, int);
    Code:
       0: aload_0
       // Lots and lots of bytecode...
       791: return
```

decompiling the calculator
the additional parameters

- **Every compose functions takes:**
  - An object of type composer
  - A unique integer value
- **The composer builds the logical tree of widgets**
- **The integer uniquely identifies an instance of a widget**

decompiling the calculator
groups and the slot table

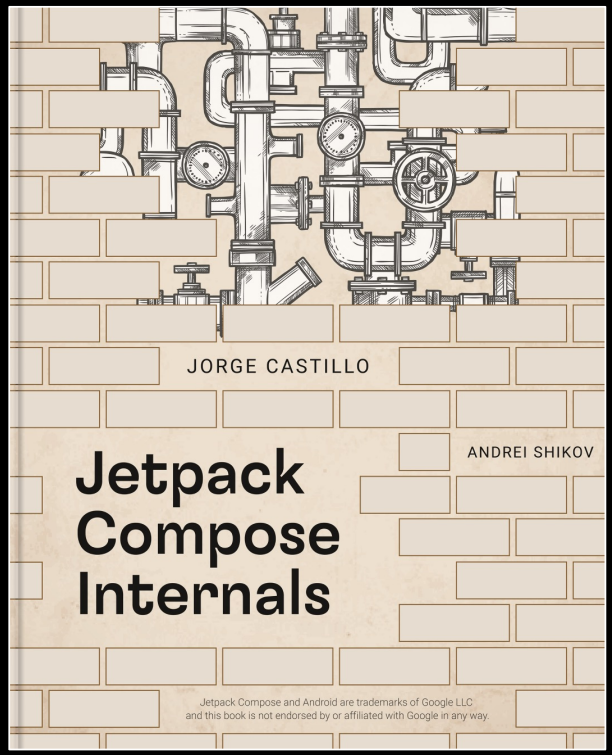
- **Composable functions support Positional Memoization**
  - Memory of how they were called at each point on the graph
- **They can also be re-invoked at any time (Recomposition)**
- **This is accomplished via a Gap Buffer / Slot Table**

decompiling the calculator
groups and the slot table

- **On every call we insert a new group into the table**
  - The group is identified by the integer parameter
- **Nested components add groups for child components**
  - We walk the graph depth first to build a linear structure

decompiling the calculator

```
public static final void DisplayText(String, Composer, int);
   Code:
      7: ldc           #110      // int 1244737340
      9: invokeinterface #25,  2 // Composer.startRestartGroup:(I)
         ...
     33: invokeinterface #37,  2 // Composer.changed:(Object)
         ...
     58: invokeinterface #41,  1 // Composer.getSkipping:()
         ...
    167: invokeinterface #79,  1 // Composer.skipToGroupEnd:()V
         ...
    173: invokeinterface #83,  1 // Composer.endRestartGroup:()
         ...
    202: invokeinterface #97,  2 // ScopeUpdateScope.updateScope:(Function2)
    207: return
```



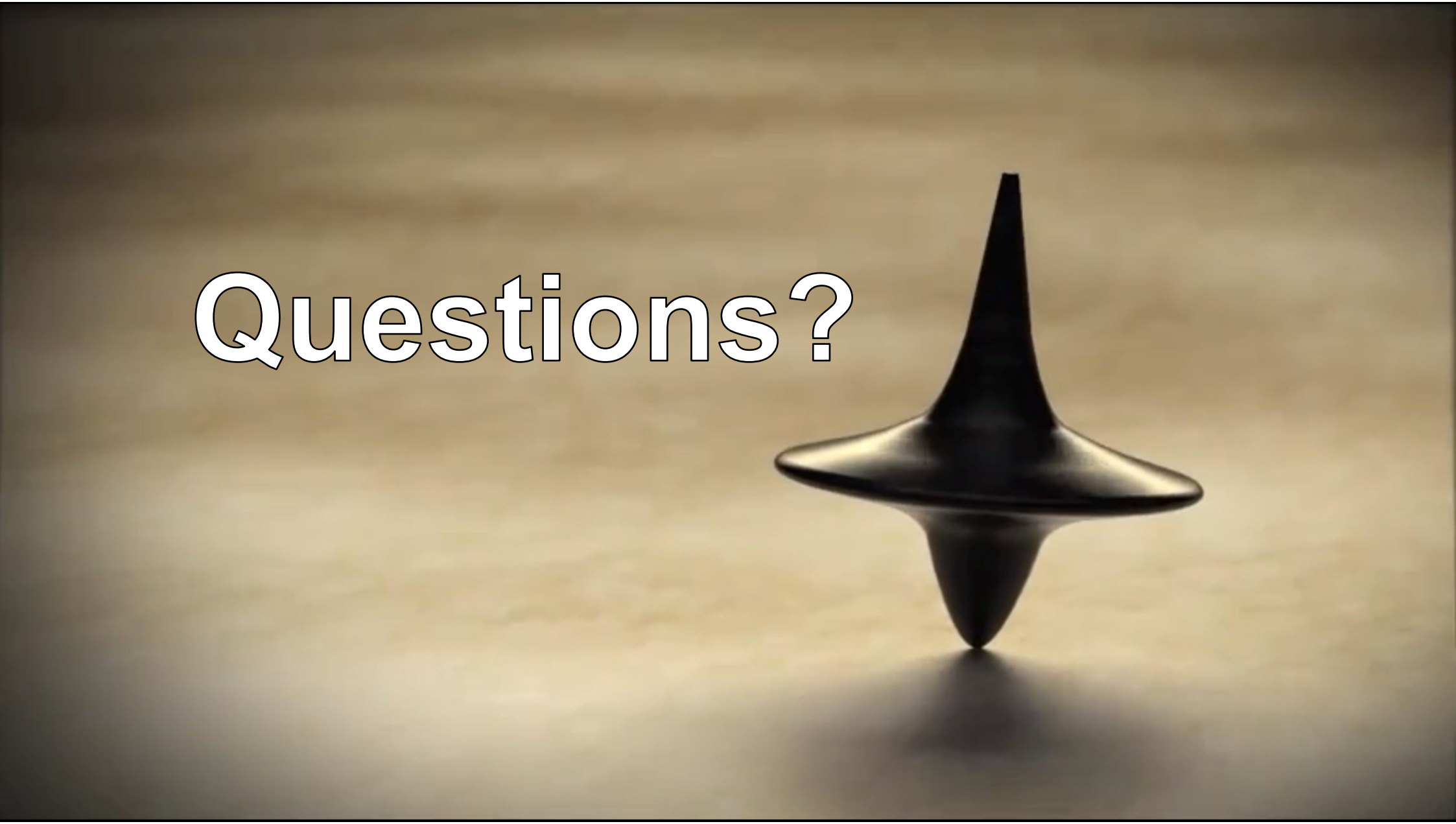
Conclusions

4

the lies are everywhere!

...and that's a good thing

- They provide advanced features
- They provide consistency across all the platforms the language supports
- They let us work at higher levels of abstraction (with certainty)
- They get us home by 5pm ☺


Questions?