Kotlin Puzzlers

●

Ruslan Ibragimov / ruslan@ibragimov.by

Challenge 1

```
fun main(args: Array<String>) {
  println(null.toString())
}
```

1) NullPointerException
2) NoSuchMethodException
3) “null”
4) “”

Challenge 1

```
fun main(args: Array<String>) {
  println(null.toString())
}
```

1) NullPointerException
2) NoSuchMethodException
3) [✓ “null”]
4) “”

Challenge 1

```
fun main(args: Array<String>) {
  println(null.toString())
}
```

1) NullPointerException
2) NoSuchMethodException
3) [✓ “null”]
4) “”

```
public fun kotlin.Any?.toString(): kotlin.String { /* compiled code */ }
```

Challenge 2

```
fun main(args: Array<String>) {
   (1..5).forEach {
      if (it == 3)
         return
      print(it)
   }

   print("Done")
}
```

1) 1245Done
2) 12Done
3) 12
4) ConcurentModificationException

Challenge 2

```kotlin
fun main(args: Array<String>) {
  (1..5).forEach {
     if (it == 3)
        return
     print(it)
  }

  print("Done")
}
```

1) 1245Done
2) 12Done
3) [✓ 12]
4) ConcurentModificationException

Challenge 2

```kotlin
fun main(args: Array<String>) {
  (1..5).forEach {
     if (it == 3)
        return@forEach
     print(it)
  }

  print("Done")
}
```

```kotlin
fun main(args: Array<String>) {
  for (i in (1..5)) {
     if (i == 3)
        continue
     print(i)
  }

  print("Done")
}
```

```kotlin
fun main(args: Array<String>) {
  (1..5).forEach(fun (it) {
     if (it == 3)
        return
     print(it)
  })

  print("Done")
}
```

Challenge 2

```java
class Java {
   private final Object object = new Object();

   public void doWork() {
      synchronized (object) {
         boolean condition = ...;
         if (condition)
            return;
      }
   }
}
```

```kotlin
class Kotlin {
   private val any = Any()

   fun doWork() {
      synchronized(any) {
         val condition = ...

         if (condition)
            return
      }
   }
}
```

```kotlin
public inline fun <R> synchronized(lock: Any, block: () -> R): R {
    // ...
}
```

Challenge 3

```
fun main(args: Array<String>) {
   func1()
   func2()
}

fun func1() = println("Hello1")

fun func2() = {
   println("Hello2")
}
```

1) Hello1, Hello2
2) Hello1
3) Don’t compile #1
4) Don’t compile #2
5) Don’t compile both wrong

Challenge 3

```
fun main(args: Array<String>) {
  func1()
  func2()
}

fun func1() = println("Hello1")

fun func2() = {
  println("Hello2")
}
```

1) Hello1, Hello2
2) [✓ Hello1]
3) Don’t compile #1
4) Don’t compile #2
5) Don’t compile both wrong

Challenge 4

```
open class Base {
   open var name: String? = null
      get() = field ?: "<unnamed>"
}

class Person : Base() {
   override var name: String? = null
      get() = super.name
      set(value) {
         field = "Mr $value"
      }
}

fun main(args: Array<String>) {
   val person = Person()
   person.name = "Anton"
   println(person.name)
}
```

1) Mr Anton
2) <unnamed>
3) Mr unnamed
4) Don’t compile

Challenge 4

```
open class Base {
  open var name: String? = null
     get() = field ?: "<unnamed>"
}

class Person : Base() {
  override var name: String? = null
     get() = super.name
     set(value) {
        field = "Mr $value"
     }
}

fun main(args: Array<String>) {
  val person = Person()
  person.name = "Anton"
  println(person.name)
}
```

1) Mr Anton
2) [✓ <unnamed>]
3) Mr unnamed
4) Don’t compile

Challenge 4

```kotlin
open class Base {
  open var name: String? = null
    get() = field ?: "<unnamed>"
}

class Person : Base() {
  override var name: String? = null
    get() = super.name
    set(value) {
      field = "Mr $value"
    }
}

fun main(args: Array<String>) {
  val person = Person()
  person.name = "Anton"
  println(person.name)
}
```

```kotlin
open class Base {
  open var name: String? = null
    get() = field ?: "<unnamed>"
}

class Person : Base() {
  override var name: String?
    get() = super.name
    set(value) {
      super.name = "Mr $value"
    }
}

fun main(args: Array<String>) {
  val person = Person()
  person.name = "Anton"
  println(person.name)
}
```

Challenge 5

```kotlin
class BooleanProvider {
  val bool = true
}

fun main(args: Array<String>) {
  val provider : BooleanProvider? = null
  if (provider?.bool) {
     print("True")
  } else {
     print("False")
  }
}
```

1) True
2) False
3) NullPointerException
4) Don’t compile

Challenge 5

```
class BooleanProvider {
  val bool = true
}

fun main(args: Array<String>) {
  val provider : BooleanProvider? = null
  if (provider?.bool) {
     print("True")
  } else {
     print("False")
  }
}
```

1) True
2) False
3) NullPointerException
4) [✓ Don’t compile]

Challenge 5

```
val Boolean?.isTrue: Boolean
  get() = this ?: false

class BooleanProvider {
  val bool = true
}

fun main(args: Array<String>) {
  val provider: BooleanProvider? = null
  if (provider?.bool.isTrue) {
    print("True")
  } else {
    print("False")
  }
}
```

Challenge 6

```
fun main(args: Array<String>) {
   val x = null
   println("${x}")
}
```

1) NullPointerException
2) NoSuchMethodException
3) “null”
4) “”

Challenge 6

```kotlin
fun main(args: Array<String>) {
   val x = null
   println("${x}")
}
```

1) NullPointerException
2) NoSuchMethodException
3) [✓ “null”]
4) “”

Challenge 7

```
class Smart {
  var prop: String? = null

  fun run() {
     var local: String? = "def"
     if (local != null) println(local.substring(1, 2))
     println(prop?.substring(1, 2))
     prop = "abc"
     if (prop != null) println(prop.substring(1, 2))
  }
}

fun main(args: Array<String>) {
  Smart().run()
}
```

1) NullPointerException
2) StringIndexOutOfBoundsException
3) e, null, b
4) Don’t compile

Challenge 7

```
class Smart {
  var prop: String? = "abc"

  fun run() {
     var local: String? = "def"
     if (local != null) println(local.substring(1, 2))
     println(prop?.substring(1, 2))
     prop = "abc"
     if (prop != null) println(prop.substring(1, 2))
  }
}

fun main(args: Array<String>) {
  Smart().run()
}
```

1) NullPointerException
2) StringIndexOutOfBoundsException
3) e, null, b
4) [✓ Don’t compile]

Challenge 7

```kotlin
class Smart {
   var prop: String? = "abc"

   fun run() {
      var local: String? = "def"
      if (local != null) println(local.substring(1, 2))
      println(prop?.substring(1, 2))
      prop = "abc"
      if (prop != null) println(prop.substring(1, 2))
   }
}

fun main(args: Array<String>) {
   Smart().run()
}
```

```kotlin
class Smart {
   var prop: String? = "abc"

   fun run() {
      var local: String? = "def"
      if (local != null) println(local.substring(1, 2))
      println(prop?.substring(1, 2))
      prop = "abc"
      prop?.let { println(it.substring(1, 2)) }
   }
}

fun main(args: Array<String>) {
   Smart().run()
}
```

Challenge 8

```
fun int2int(x : Int) : Int {
   return x + 1
}

fun main(args: Array<String>) {
   var x : ((Int) -> Int)? = null

   x = {it + 1}
   print(x(2))

   x = ::int2int
   print(x(2))

   x = fun(x) = x + 1
   print(x(2))
}
```

1) 333
2) Compile error #1
3) Compile error #2
4) Compile error #3

Challenge 8

```
fun int2int(x : Int) : Int {
  return x + 1
}

fun main(args: Array<String>) {
  var x : ((Int) -> Int)? = null

  x = {it + 1}
  print(x(2))

  x = ::int2int
  print(x(2))

  x = fun(x) = x + 1
  print(x(2))
}
```

1) 333
2) Compile error #1
3) Compile error #2
4) Compile error #3

Challenge 9

```
fun hello(block: () -> Unit) = block()

inline fun helloInline(block: () -> Unit) = block()

fun main(args: Array<String>) {
   hello {
       print("Hello")
       return
   }

   hello(fun() {
       print("Hello")
       return
   })

   helloInline {
       print("Hello")
       return
   }

   helloInline (fun() {
       print("Hello")
       return
   })
}
```

1) Hello
2) HelloHello
3) HelloHelloHello
4) HelloHelloHelloHello
5) Don’t compile

Challenge 9

```kotlin
fun hello(block: () -> Unit) = block()

inline fun helloInline(block: () -> Unit) = block()

fun main(args: Array<String>) {
   hello {
       print("Hello")
       return
   }

   hello(fun() {
       print("Hello")
       return
   })

   helloInline {
       print("Hello")
       return
   }

   helloInline (fun() {
       print("Hello")
       return
   })
}
```

1) Hello
2) HelloHello
3) HelloHelloHello
4) HelloHelloHelloHello
[✓ 5) Don't compile]

Challenge 10

```kotlin
fun hello(block: () -> Unit) = block()

inline fun helloInline(block: () -> Unit) = block()

fun main(args: Array<String>) {
   hello {
       print("Hello")
   }

   hello(fun() {
       print("Hello")
       return
   })

   helloInline {
       print("Hello")
       return
   }

   helloInline (fun() {
       print("Hello")
       return
   })
}
```

1) Hello
2) HelloHello
3) HelloHelloHello
4) HelloHelloHelloHello
5) Don’t compile

Challenge 10

```kotlin
fun hello(block: () -> Unit) = block()

inline fun helloInline(block: () -> Unit) = block()

fun main(args: Array<String>) {
    hello {
        print("Hello")
    }

    hello(fun() {
        print("Hello")
        return
    })

    helloInline {
        print("Hello")
        return
    }

    helloInline (fun() {
        print("Hello")
        return
    })
}
```

1) Hello
2) HelloHello
[✓ 3) HelloHelloHello]
4) HelloHelloHelloHello
5) Don’t compile

Challenge 11

```
val bigFunction = fun(arg1 : Int,  arg2 : Int,  arg3 : Int, arg4 : Int, arg5 : Int, arg6 : Int, arg7 : Int, arg8 : Int,
        arg9 : Int, arg10 : Int, arg11 : Int, arg12 : Int, arg13 : Int, arg14 : Int, arg15 : Int, arg16 : Int, arg17 : Int,
        arg18 : Int, arg19 : Int, arg20 : Int, arg21 : Int, arg22 : Int, arg23 : Int)
= {
    arg1 + arg2 + arg3 + arg4 + arg5 + arg6 + arg7 + arg8 + arg9 + arg10 + arg11 + arg12 + arg13 + arg14 + arg15 + arg16
    + arg17 + arg18 + arg19 + arg20 + arg21 + arg22 + arg23
}

fun main(args: Array<String>) {
   val value = bigFunction(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)
   print(value)
}
```

Challenge 11

```
val bigFunction = fun(arg1 : Int,  arg2 : Int,  arg3 : Int, arg4 : Int, arg5 : Int, arg6 : Int, arg7 : Int, arg8 : Int,
                     arg9 : Int, arg10 : Int, arg11 : Int, arg12 : Int, arg13 : Int, arg14 : Int, arg15 : Int, arg16 :
Int, arg17 : Int, arg18 : Int, arg19 : Int, arg20 : Int, arg21 : Int, arg22 : Int, arg23 : Int)
= {
   arg1 + arg2 + arg3 + arg4 + arg5 + arg6 + arg7 + arg8 + arg9 + arg10 + arg11 + arg12 + arg13 + arg14 + arg15 + arg16
   + arg17 + arg18 + arg19 + arg20 + arg21 + arg22 + arg23
}

fun main(args: Array<String>) {
   val value = bigFunction(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)
   print(value)
}
```

1) **140**
2) **() -> kotlin.Int**
3) **java.lang.NoClassDefFoundError:**
4) **Don’t compile**

Challenge 11

```kotlin
val bigFunction = fun(arg1 : Int,  arg2 : Int,  arg3 : Int, arg4 : Int, arg5 : Int, arg6 : Int, arg7 : Int, arg8 : Int,
                     arg9 : Int, arg10 : Int, arg11 : Int, arg12 : Int, arg13 : Int, arg14 : Int, arg15 : Int, arg16 :
Int, arg17 : Int, arg18 : Int, arg19 : Int, arg20 : Int, arg21 : Int, arg22 : Int, arg23 : Int)
= {
   arg1 + arg2 + arg3 + arg4 + arg5 + arg6 + arg7 + arg8 + arg9 + arg10 + arg11 + arg12 + arg13 + arg14 + arg15 + arg16
   + arg17 + arg18 + arg19 + arg20 + arg21 + arg22 + arg23
}

fun main(args: Array<String>) {
   val value = bigFunction(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23)
   print(value)
}
```

1) **140**
2) **() -> kotlin.Int**
[✓ 3) **java.lang.NoClassDefFoundError:**]
4) **Don't compile**

Challenge 12

```
val bigFunction = fun(arg1 : Int,  arg2 : Int,  arg3 : Int, arg4 : Int, arg5 : Int, arg6 : Int, arg7 : Int, arg8 : Int,
                     arg9 : Int, arg10 : Int, arg11 : Int, arg12 : Int, arg13 : Int, arg14 : Int, arg15 : Int, arg16 :
Int, arg17 : Int, arg18 : Int, arg19 : Int, arg20 : Int, arg21 : Int, arg22 : Int)
= {
   arg1 + arg2 + arg3 + arg4 + arg5 + arg6 + arg7 + arg8 + arg9 + arg10 + arg11 + arg12 + arg13 + arg14 + arg15 + arg16
   + arg17 + arg18 + arg19 + arg20 + arg21 + arg22
}

fun main(args: Array<String>) {
   val value = bigFunction(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15, 16, 17, 18, 19, 20, 21, 22)
   print(value)
}
```

1) **117**
2) **() -> kotlin.Int**
3) **java.lang.NoClassDefFoundError:**
4) **Don’t compile**

Challenge 12

```kotlin
val bigFunction = fun(arg1 : Int,  arg2 : Int,  arg3 : Int, arg4 : Int, arg5 : Int, arg6 : Int, arg7 : Int, arg8 : Int,
                     arg9 : Int, arg10 : Int, arg11 : Int, arg12 : Int, arg13 : Int, arg14 : Int, arg15 : Int, arg16 :
Int, arg17 : Int, arg18 : Int, arg19 : Int, arg20 : Int, arg21 : Int, arg22 : Int) = {
   arg1 + arg2 + arg3 + arg4 + arg5 + arg6 + arg7 + arg8 + arg9 + arg10 + arg11 + arg12 + arg13 + arg14 + arg15 + arg16
   + arg17 + arg18 + arg19 + arg20 + arg21 + arg22
}

fun main(args: Array<String>) {
   val value = bigFunction(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ,13, 14, 15, 16, 17, 18, 19, 20, 21, 22)
   print(value)
}
```

1) **117**
[✓ 2) **() -> kotlin.Int**]
3) **java.lang.NoClassDefFoundError:**
4) **Don't compile**

```
data class Contract(
      @NotEmpty val dealNo: String,
      @NotZero val amount: BigDecimal,
      @NotFuture val buyer: LocalDate
)
```

1. На параметрах конструктора class Contract(@NotEmpty String dealNo)
2. На геттерах
3. На сеттерах
4. На филде
5. Звонок другу

```
@Target(AnnotationTarget.FIELD)
annotation class NotEmpty
@Target(AnnotationTarget.PROPERTY)
annotation class NotFuture
annotation class NotZero

data class Contract(
        @NotEmpty val dealNo: String,
        @NotZero val amount: BigDecimal,
        @NotFuture val buyer: LocalDate
)
```

```
(10..1).forEach { println(it) }
```

```
(1..9999).map { it -> { -> println(it) } }
```

```kotlin
val listOf = listOf<String>("abc", "")
when (listOf) {
    is MutableList -> println("mutableList")
    is List -> println("list")
}

when (listOf) {
    is List -> println("list")
    is MutableList -> println("mutableList")
}
```

```kotlin
// different baking classes
val emptyList: List<String> = listOf()
val singletonList: List<String> = listOf("Rick")
val list: List<String> = listOf("Rick", "Morty")

try {
    val l = emptyList as MutableList
    l.add("Rick")
} catch (e: Exception) {
    e.printStackTrace()
}
```

```kotlin
try {
    val l = singletonList as MutableList
    l.add("Morty")
} catch (e: Exception) {
    e.printStackTrace()
}

try {
    val l = list as MutableList
    l.add("Mr. Poopybutthole")
} catch (e: Exception) {
    e.printStackTrace()
}
```

Last

```
private fun foo(one: (String) -> Unit
= {}, two: (String) -> Unit = {}) {
        one("one")
        two("two")
}

fun main(args: Array<String>) {
        foo({ print(it) })
        foo { print(it) }
}
```