Examples from the standard library

Kotlin standard library

=

Java standard library

+

extensions

Standard collections

```
val set = hashSetOf(1, 7, 53)
val list = arrayListOf(1, 7, 53)
val map = hashMapOf(1 to "one",
        7 to "seven", 53 to "fifty-three")
```

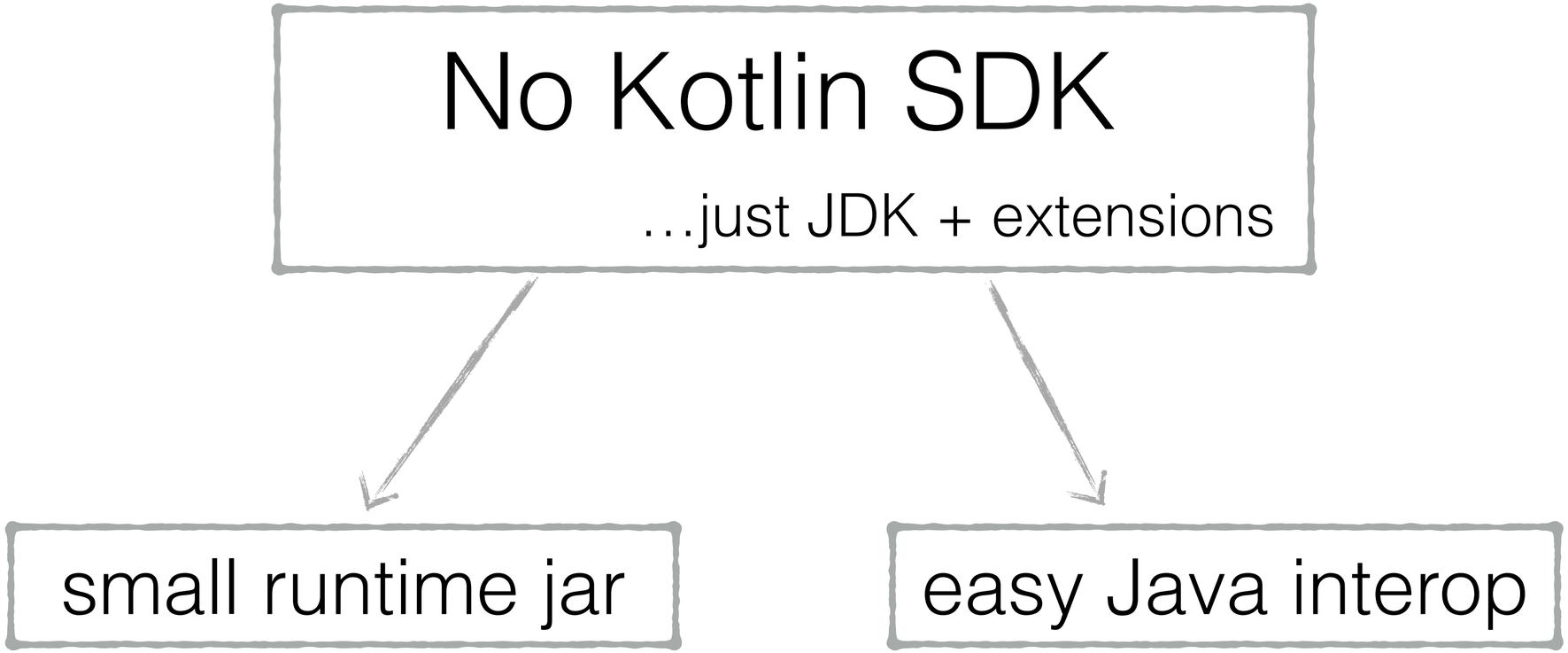
analogous to Java's `set.getClass()`

```
println(set.javaClass)        class java.util.HashSet
println(list.javaClass)       class java.util.ArrayList
println(map.javaClass)        class java.util.HashMap
```

Kotlin library: extensions on collections

```
list.max
λ max() for Iterable<T> in kotlin.collections                    Int
λ maxBy {...} (selector: (Int) -> R) for Iterable<T> in kot…     Int?
λ maxWith(comparator: Comparator<in Int>) for Iterable<T> i…     Int?
^↓ and ^↑ will move caret down and up in the editor >>
```

- filter
- map
- reduce
- count
- find
- any
- flatMap
- groupBy
- …


No Kotlin SDK
…just JDK + extensions
small runtime jar
easy Java interop

Examples

Extension function: `joinToString`

```
println(listOf('a', 'b', 'c').joinToString(
        separator = "", prefix = "(", postfix = ")"))
```

```
(abc)
```

```
fun <T> Iterable<T>.joinToString(
        separator: CharSequence = ", ",
        prefix: CharSequence = "",
        postfix: CharSequence = ""
): String
```

Extension function: `getOrNull()`

```
fun main(args: Array<String>) {
    println("Hello, ${args.getOrNull(0)}!")
}
```

```
fun <T> Array<T>.getOrNull(index: Int) =
    if (index in 0 until size) this[index] else null
```

Extension function: `getOrNull()`

```
val list = listOf("abc")
println(list.getOrNull(0))  // abc
println(list.getOrNull(1))  // null
```

```
fun <T> List<T>.getOrNull(index: Int) =
    if (index in 0 until size) this[index] else null
```

Extension function: `withIndex()`

```
val list = listOf("a", "b", "c")
for ((index, element) in list.withIndex()) {
    println("$index $element")
}
```

```
fun <T> Iterable<T>.withIndex(): List<IndexedValue<T>> { … }
```

Extension function: `until`

```
infix fun Int.until(to: Int): IntRange
```

```
1.until(10)
```

```
1 until 10
```

Extension function: `to`

```
infix fun <A, B> A.to(that: B) = Pair(this, that)
```

```
"ANSWER".to(42)
```

```
"hot" to RED
```

```
mapOf(0 to "zero", 1 to "one")
```

Extension functions on `Char`

```
fun Char.isLetter() = this in 'a'..'z' || this in 'A'..'Z'

fun Char.isLetterOrDigit() = isLetter() || this in '0'..'9'
```

```
'a'.isLetter()              // true
'%'.isLetterOrDigit()       // false
```

Extensions on `String`

Formatting multiline strings

```
val q = """To code,
    |or not to code?..""".trimMargin()
```

```
println(q)        To code,
                  or not to code?..
```

Formatting multiline strings

```
val q = """To code,
    #or not to code?..""".trimMargin(marginPrefix = "#")
```

```
println(q)        To code,
                  or not to code?..
```

Formatting multiline strings

```
val q = """To code,
    |or not to code?..""".trimMargin()
val a = """
    Keep calm
    and learn Kotlin""".trimIndent()

println(q)        To code,
                  or not to code?..

println(a)        Keep calm
                  and learn Kotlin
```

Using regular expressions

```
val regex = "\\d{2}\\.\\d{2}\\.\\d{4}".toRegex()
regex.matches("15.02.2016")          // true
regex.matches("15.02.16")            // false
```

Using regular expressions

```
val regex = """\d{2}\.\d{2}\.\d{4}""".toRegex()
regex.matches("15.02.2016")          // true
regex.matches("15.02.16")            // false
```

Conversion to numbers

```
"123".toInt()            // 123
"1e-10".toDouble()       // 1.0E-10

"xx".toInt()             // NumberFormatException

"123".toIntOrNull()      // 123
"xx".toIntOrNull()       // null
```

Extension used for tasks in this course

Extension function: eq

```
infix fun <T> T.eq(other: T) {
    if (this == other) println("OK")
    else println("Error: $this != $other")
}


fun getAnswer() = 42

getAnswer() eq 42   // OK
getAnswer() eq 43   // Error: 42 != 43
```

What is the type of `'a' to 1.0`?

1. `Char to Double`
2. `Pair<Char, Double>`
3. `List<Any>`

?

What is the type of `'a' to 1.0`?

1. `Char to Double`
2. `Pair<Char, Double>`
3. `List<Any>`

Extension function: `to`

```
infix fun <A, B> A.to(that: B): Pair<A, B> = Pair(this, that)
```

```
data class Pair<A, B>(val first: A, val second: B) {
    override fun toString(): String = "($first, $second)"
}
```