Kotlin 1.4 Online Event

News from the Kotlin Standard Library

Svetlana Isakova

@sveta_isakova

October 13, 2020

News from the Kotlin standard library

- New functionality in Kotlin 1.4
- Using stdlib in multiplatform projects
- Experimental functionality

New functionality in Kotlin 1.4

New functionality in stdlib

- Making stdlib consistent and meet your expectations
- New functions: runningFold, runningReduce
- New types: ArrayDeque
- New operations for bit manipulation

Naming conventions: *orNull

throw exception on error:

```
"string".
    toInt
    toBigDecimal

collection.
    first
    last
    elementAt
    random
    single
```

return null on error:

```
"string".
    toIntOrNull
    toBigDecimalOrNull

collection.
    firstOrNull
    lastOrNull
    elementAtOrNull
    randomOrNull
    singleOrNull
```

NEW IN 1.4

max, min

```
val list = listOf(1, 2, 3)
val max: Int? = list.max()
```

max, min

```
val list = listOf(1, 2, 3)
val max: Int? = list.max()
```

Replace with `maxOrNull`

max, min

```
val list = listOf(1, 2, 3)
val max: Int? = list.maxOrNull()
```

NEW IN  1.4

max, min

- `max/min` are deprecated in 1.4
- `maxOrNull/minOrNull` should be used instead
- `max/min` returning non-null value
  will be reintroduced in future versions

maxBy, minBy

- `maxBy/minBy` are deprecated in 1.4
- `maxByOrNull/minByOrNull` should be used instead
- `maxBy/minBy` returning non-null value will be reintroduced in future versions

maxBy vs maxOf

maxBy returns the element with the largest value

```
val oldest = people.maxByOrNull { it.age }
val maxAge = oldest?.age
```

maxOf returns the largest value itself

```
val maxAge = people.maxOf { it.age }
```

minOf, minOfOrNull

```
val minAge = people.minOf { it.age }
val minAgeOrNull = people.minOfOrNull { it.age }
```

NEW IN 1.4

Naming conventions: `*Indexed`

perform an operation on each element:

```
collection.
    forEach
    onEach
    filter
    map
    flatMap
    fold
    reduce
```

take an additional parameter, an index, to perform an operation:

```
collection.
    forEachIndexed
    onEachIndexed
    filterIndexed
    mapIndexed
    flatMapIndexed
    foldIndexed
    reduceIndexed
```

NEW IN 1.4

NEW IN 1.4

Naming conventions: `*Indexed`

perform an operation on each element:

```
collection.
    forEach
    onEach
    filter
    map
    flatMap
    fold
    reduce
```

take an additional parameter, an index, to perform an operation:

```
collection.
    forEachIndexed
    onEachIndexed
    filterIndexed
    mapIndexed
    flatMapIndexed
    foldIndexed
    reduceIndexed
```

forEach vs onEach

Performs the given action on each element

```
orders.forEach(::println)
```

Performs the given action on each element,
and returns the collection itself afterwards

```
orders
    .filter { order → order.client == clientId }
    .onEachIndexed { index, order →
        log("Order by $clientId [$index]: ${order.id}")
    }
    .flatMap(Order::items)
```

Naming conventions: `*IndexedOrNull`

```
collection.
    reduce
    reduceIndexed
    reduceOrNull                NEW IN 1.4
    reduceIndexedOrNull         NEW IN 1.4
```

reduceIndexedOrNull

```
val list = listOf(1, 2, 3)
println(list.reduce { a, b → a + b }) // 6
```

```
val emptyList = emptyList<Int>()
println(emptyList.reduceOrNull { a, b → a + b }) // null
```

Returns `null` if the collection is empty

reduceIndexedOrNull

```
val list = listOf(1, 2, 3)
println(list.reduceOrNull { a, b → a + b }) // 6
```

```
val emptyList = emptyList<Int>()
println(emptyList.reduceOrNull { a, b → a + b }) // null
```

Returns `null` if the collection is empty

```
emptyList.reduceIndexedOrNull { index, acc, e → ... }
```

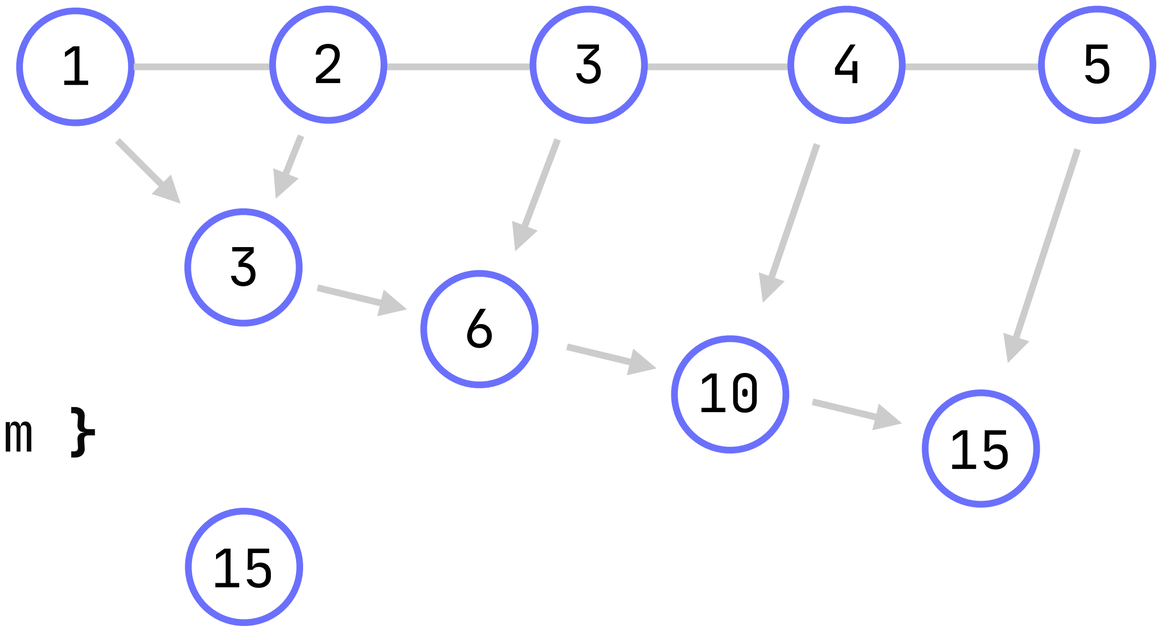
Returns `null` if the collection is empty
Takes an additional parameter,
an index, to perform an operation

New functions: runningReduce

```
(1..5).reduce { sum, elem → sum + elem }          // 15

(1..5).runningReduce { sum, elem → sum + elem }   // [1, 3, 6, 10, 15]
```

New functions: runningReduce




```
(1..5).reduce { sum, elem → sum + elem }
```

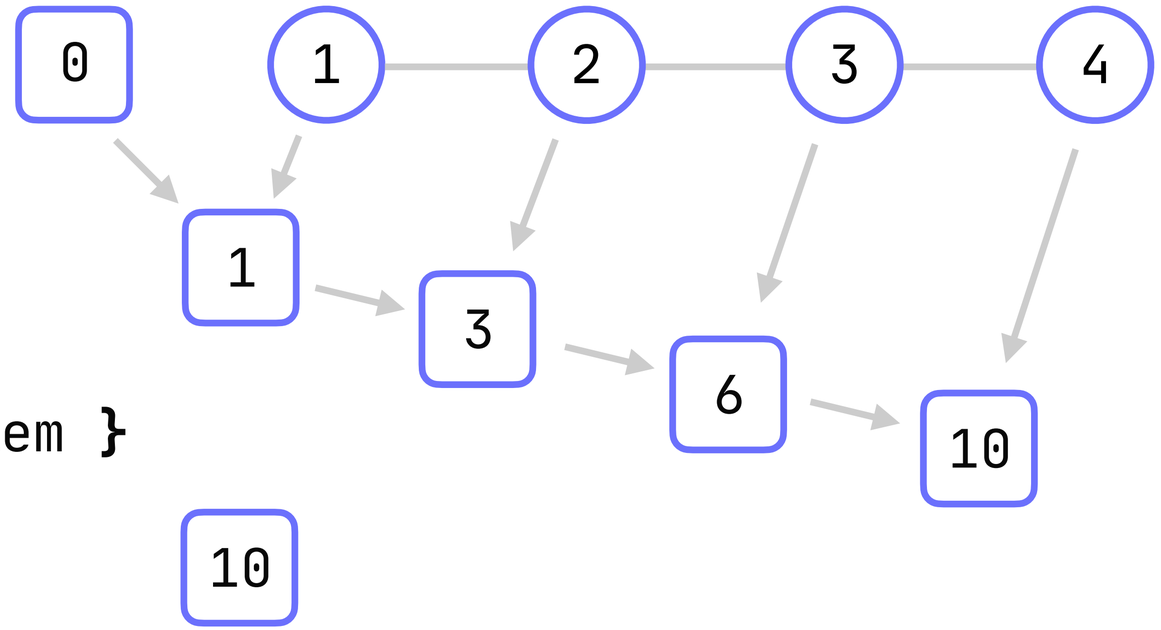
```
(1..5).runningReduce { sum, elem → sum + elem }
```

New functions: runningFold (scan)

```
(1..4).fold(0) { acc, elem → acc + elem }          // 10

(1..4).runningFold(0) { acc, elem → acc + elem }  // [0, 1, 3, 6, 10]
```

New functions: runningFold (scan)




```
(1..4).fold(0) { acc, elem → acc + elem }
```

```
(1..4).runningFold(0) { acc, elem → acc + elem }
```

New type: `ArrayDeque`

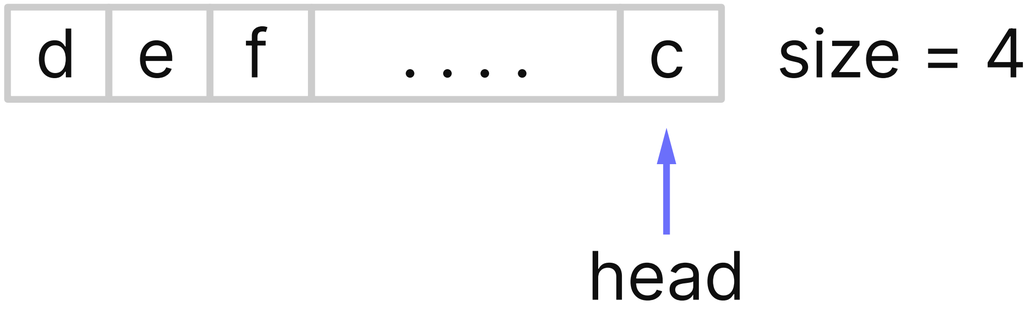
- “double-ended queue”
- provides methods for convenient both-ends access
- is part of the common library
- implements MutableList

Using ArrayDeque

```
val deque = ArrayDeque(listOf("d", "e", “f"))
```

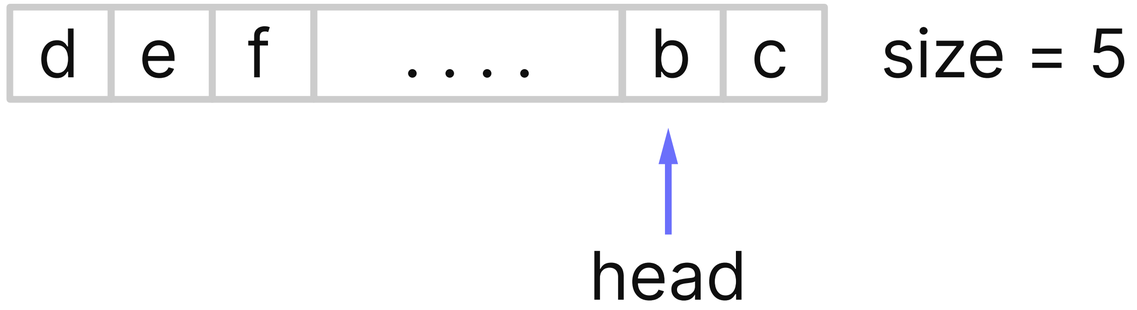
Using ArrayDeque

```
val deque = ArrayDeque(listOf("d", "e", “f"))
deque.addFirst("c")
```



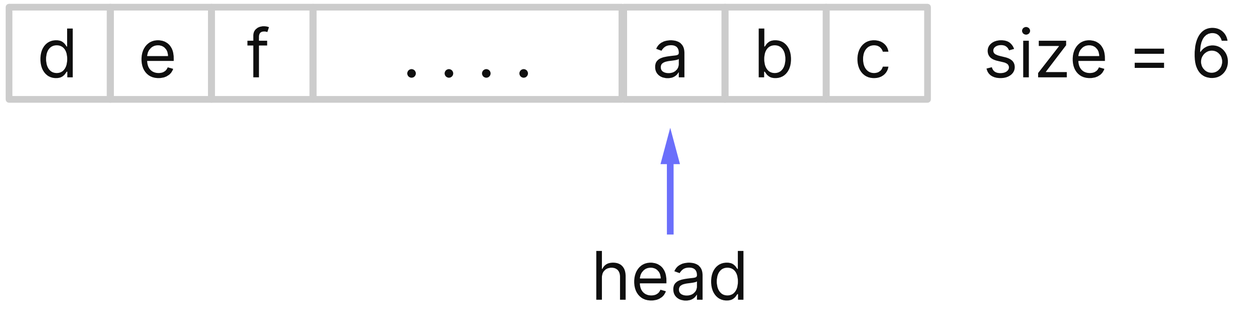
Using ArrayDeque

```
val deque = ArrayDeque(listOf("d", "e", “f"))
deque.addFirst("c")
deque.addFirst("b")
```



Using ArrayDeque

```
val deque = ArrayDeque(listOf("d", "e", “f"))
deque.addFirst("c")
deque.addFirst("b")
deque.addFirst("a")
```



New type: ArrayDeque

- use it whenever you need a Queue

```
val queue = ArrayDeque<Char>()
queue.addLast('a')
queue.addLast('b')
queue.removeFirst() // 'a'
```

Functions for bit manipulation

```
val number = "01010000".toInt(radix = 2)                    80
```

$2^6+2^4=64+16$

Functions for bit manipulation

```
val number = "1010000".toInt(radix = 2)
```

```
number.countOneBits()
```

Functions for bit manipulation

```
val number = "1010000".toInt(radix = 2)
```

```
number.countOneBits()
```

1010000 2

```
number.countTrailingZeroBits()
```

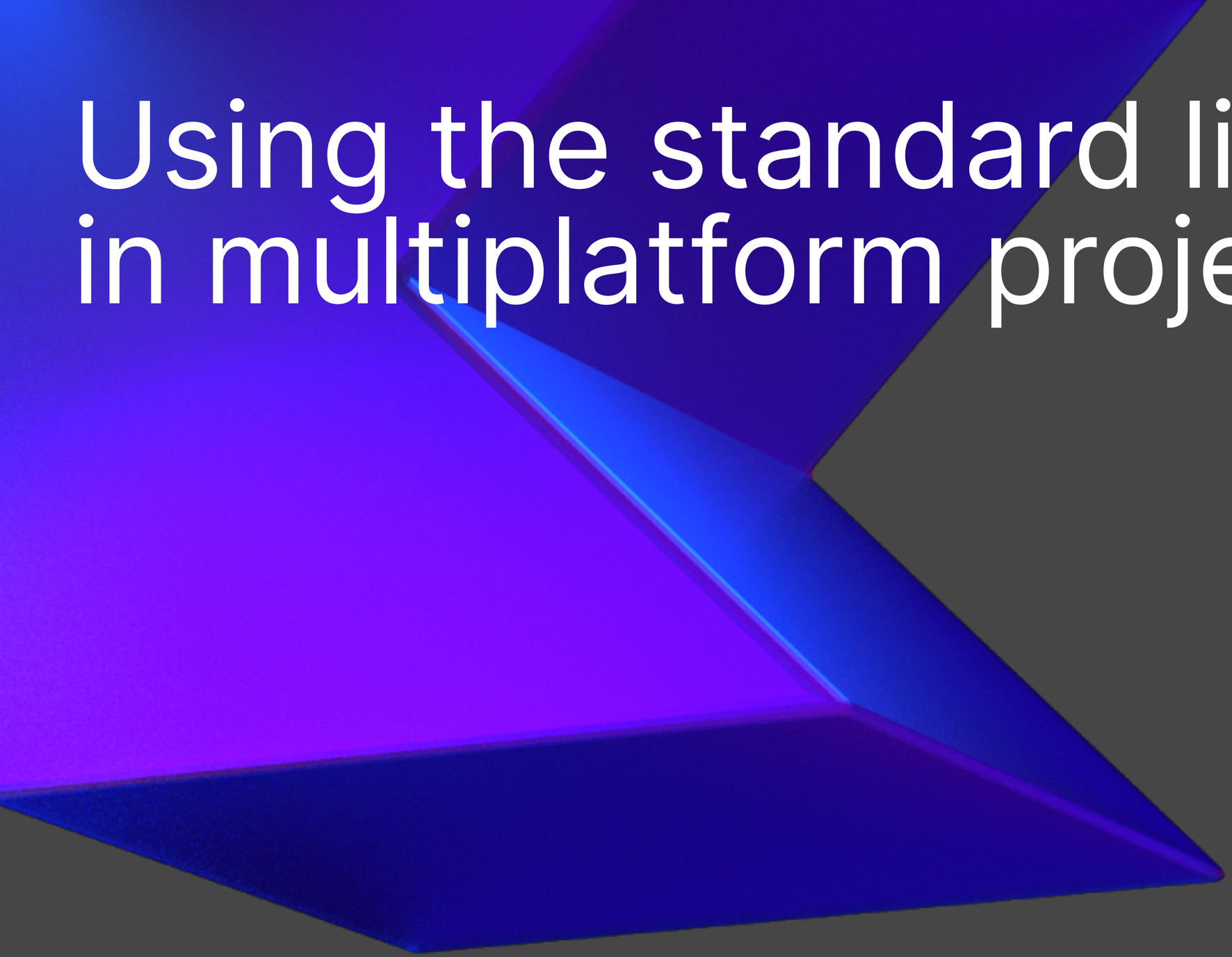
1010000 4

Functions for bit manipulation

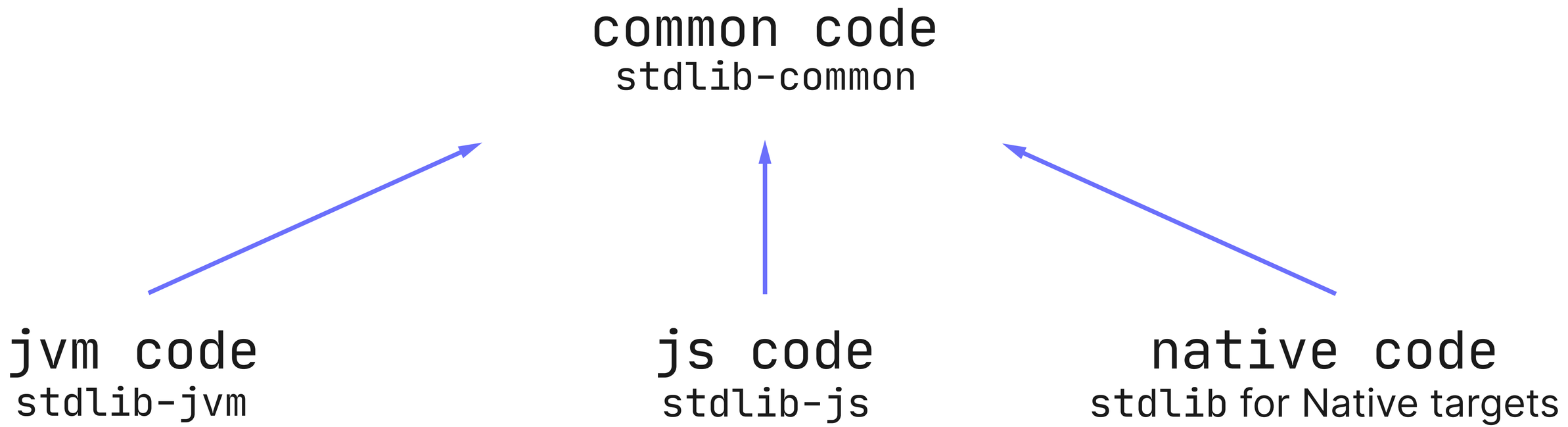
```
val number = "1010000".toInt(radix = 2)
```

`number.countOneBits()`	1010000	2
`number.countTrailingZeroBits()`	1010000	4
`number.takeHighestOneBit().toString(2)`	1010000	1000000

Using the standard library in multiplatform projects



Different versions for different platforms



Dependency on stdlib by default

- A dependency on the standard library is added automatically in each source set
- With the same version as the Kotlin Gradle plugin
- For platform-specific source sets, the platform-specific variant of the library is used

Extending the common library

- Trying to make it on par with the JVM stdlib
  (which the most used)

- Adding or moving missing functionality
  to the common library

Extending the common library

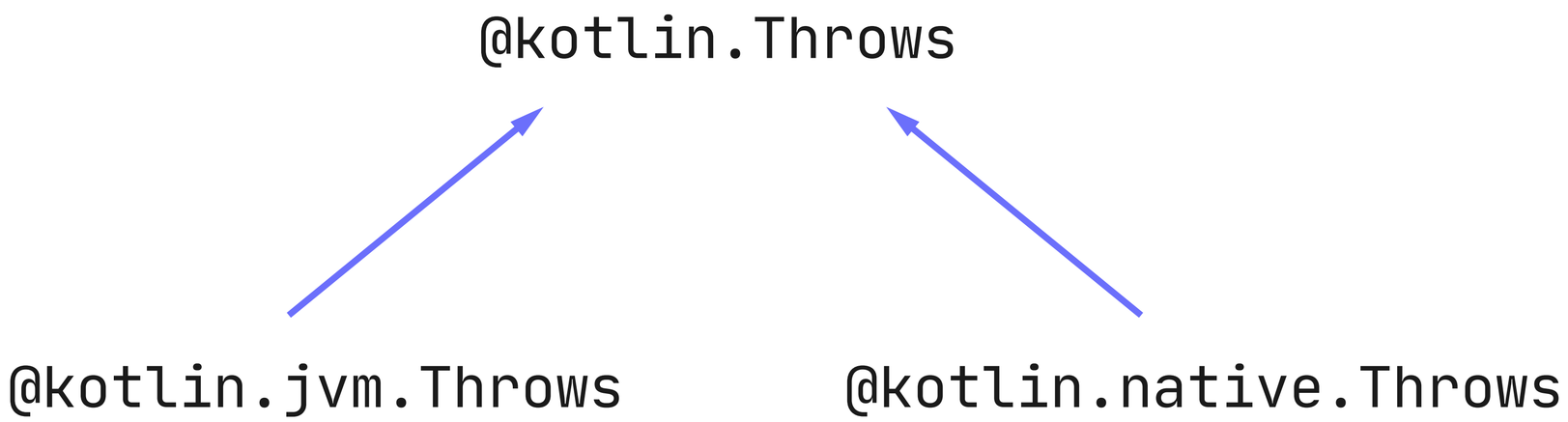
- Extending `StringBuilder` in the common library
- Common reflection API is revised
- Common exception processing API

Common exception processing API

- `Throwable.stackTraceToString()`
- `Throwable.printStackTrace()`
- `Throwable.addSuppressed()`

Common `@Throws` annotation

- For interoperability with languages that have checked exceptions or errors, like Java or Swift
- Can be used from common code



Experimental functionality

Experimental stdlib API

```
@OptIn(ExperimentalStdlibApi::class)
fun main() {
    // using experimental API
}
```

Opt-In requirements

- Allows requiring and giving explicit consent for using certain elements of APIs
- Was renamed from @UseExperimental to cover more use-cases
- The goal: let early adopters try the new features as soon as possible

New API in your library

```
@RequiresOptIn(
    level = Level.WARNING,
    message = "This API can change ⚠"
)
annotation class BleedingEdgeAPI
```

```
@BleedingEdgeAPI
class Foo {
    ...
}
```

```
@BleedingEdgeAPI
fun fetchFoo(): Foo {
    ...
}
```

New API in your library

```
@RequiresOptIn(
    level = Level.ERROR,
    message = "This API can change ⚠"
)
annotation class BleedingEdgeAPI
```

```
@BleedingEdgeAPI
class Foo {
    ...
}
```

```
@BleedingEdgeAPI
fun doSomething(): Foo {
    ...
}
```

Clients using your new API

```
fun doSomething() {
    val foo = Foo()

}
```

This API can change ⚠

Clients using your new API

```
@OptIn(BleedingEdgeAPI::class)
fun doSomething() {
    val foo = Foo()

}
```

New experimental stdlib API

- Collection builders
- Time measurement API

Collection builders

- `buildList`
- `buildSet`
- `buildMap`

Similar to `buildString`

```
val str: String = buildString {
    this.appendLine("Chars:")
    for (ch in 'a'..'i') {
        append(ch)
    }
}
println(str)
// Chars:
// abcdefghi
```

Collection builders

```
val needsZero = true
val initial = listOf(2, 6, 41)


val ints = buildList {
    if (needsZero) {
        add(0)
    }
    initial.mapTo(this) { it + 1 }
}
println(ints) // [0, 3, 7, 42]
```

Collection builders

```
val needsZero = true
val initial = listOf(2, 6, 41)


val ints = buildList {
    if (needsZero) {
        add(0)
    }
    initial.mapTo(this) { it + 1 }
}
println(ints) // [0, 3, 7, 42]
```

this: MutableList

Collection builders

```
val needsZero = true
val initial = listOf(2, 6, 41)

val ints = buildList {
    if (needsZero) {
        add(0)
    }
    initial.mapTo(this) { it + 1 }
}
println(ints) // [0, 3, 7, 42]
```

List

this: MutableList

Measuring time

```
import kotlin.time.*

val duration: Duration = measureTime {
    Thread.sleep(1050)
    // action
}
duration.inSeconds // 1.056486657
duration.inMilliseconds // 1056.486657
```

Measuring time

```
import kotlin.time.*

val clock = TimeSource.Monotonic
val mark = clock.markNow()
Thread.sleep(200)
println(mark.elapsedNow())
// e.g. 206ms
```

action

Measuring time

```
import kotlin.time.*

val clock = TimeSource.Monotonic
val mark = clock.markNow()
Thread.sleep(200)
println(mark.elapsedNow())
// e.g. 206ms
```

might be inside the first function

might be inside the second function

Measuring time
& getting the resulting value

```
import kotlin.time.*

val (value, duration) = measureTimedValue {
    Thread.sleep(100)
    42
}
println(value)      // 42
println(duration)   // e.g. 103ms
```

Dates and Time

- github.com/Kotlin/kotlinx-datetime
- ‘Introducing DateTime’ talk

Summary

What we’ve discussed

New stable functionality

consistent stdlib

`runningReduce`
`runningFold`

`ArrayDeque`

bit manipulation

Experimental functionality

@OptIn

collection builders

time measurement

stdlib in multiplatform projects

stdlib dependency by default

extending the common library

What’s New in 1.4 documentation page

What's New in Kotlin 1.4.0

Edit Page

In Kotlin 1.4.0, we ship a number of improvements in all of its components, with the focus on quality and performance. Below you will find the list of the most important changes in Kotlin 1.4.0.

Language features and improvements

- SAM conversions for Kotlin interfaces
- Explicit API mode for library authors
- Mixing named and positional arguments
- Trailing comma
- Callable reference improvements
- `break` and `continue` inside `when` included in loops

New tools in the IDE

- New flexible Project Wizard
- Coroutine Debugger

What’s New in 1.4 documentation page

Standard library

- Common exception processing API
- New functions for arrays and collections
- Functions for string manipulations
- Bit operations
- Delegated properties improvements
- Converting from KType to Java Type
- Proguard configurations for Kotlin reflection
- Improving the existing API
- module-info descriptors for stdlib artifacts
- Deprecations
- Exclusion of the deprecated experimental coroutines

Thanks!
Have a nice Kotlin!

@sveta_isakova