Kotlin 1.4 Online Event

Introducing kotlinx-datetime

Ilya Gorbunov

ilya.gorbunov@jetbrains.com

October 13, 2020

Platform API

- **JVM 8+**: java.time.*, Time4J v5.x
- **JVM 6**: legacy java.util.Date (I wouldn’t recommend it), JodaTime, ThreeTenBP/ThreeTenABP
- **JS**: JS Date (wouldn’t recommend it either), JSJoda, Luxon.js, …
- **Native**: platform.foundation.NSDate (on Apple platforms) platform.posix.*

Multiplatform alternatives

- **Klock,** 2017
  https://github.com/korlibs/klock
  https://korlibs.soywiz.com/klock

- **fluid-time,** 2019
  https://github.com/fluidsonic/fluid-time

- **Island Time,** 2019
  https://islandtime.io/
  https://github.com/erikc5000/island-time

kotlinx.datetime platforms

- **K/JVM** for JVM 8+
- **K/JS**: jar for classic backend, klib for IR backend
- **K/Native**
  Linux (x64), Windows (mingwX64), macOS (x64),
  iOS (x64, arm32, arm64) watchOS (x86, arm32, arm64)
  tvOS (arm64, x64)

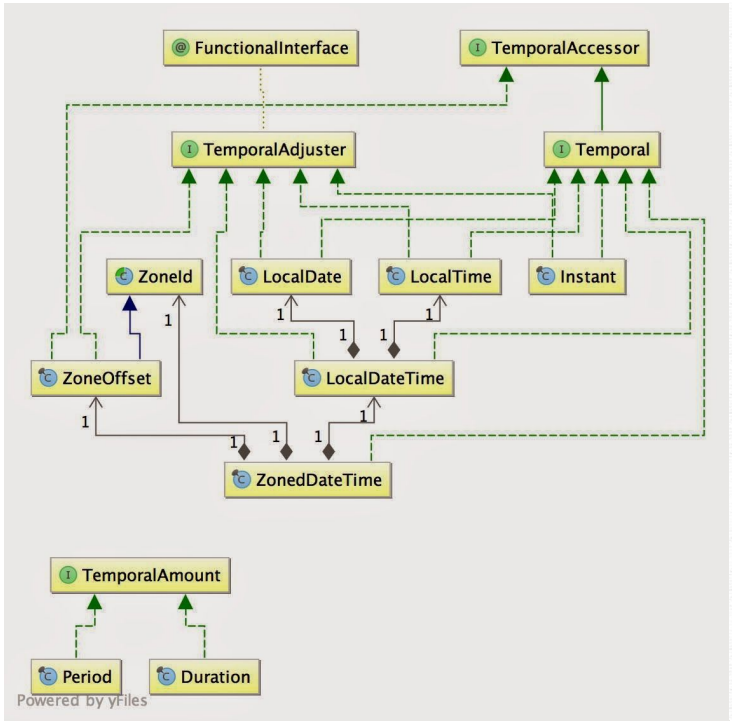
Design principles

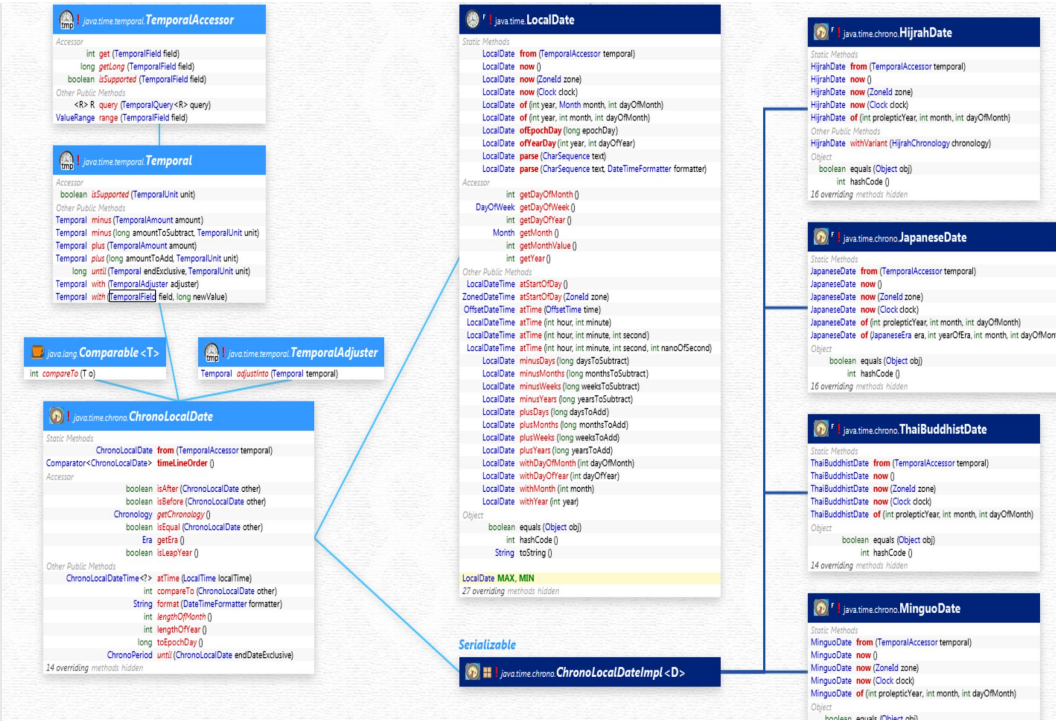
- Immutable types

Design principles

- Immutable types
- A minimum number of types that solve the practical use cases

java.time API





Design principles

- Immutable types
- A minimum number of types that solve the practical use cases
- Statically typed to prevent incorrect operations

java.time API

```
val date = LocalDate.now()
val tomorrow = date + Duration.ofDays(1)
```

java.time API

```
val date = LocalDate.now()
val tomorrow = date + Duration.ofDays(1)

Exception in thread "main"
java.time.temporal.UnsupportedTemporalTypeException:
Unsupported unit: Seconds
    at java.time.LocalDate.plus(LocalDate.java:1247)
```

java.time API

```
val date = LocalDate.now()
val tomorrow = date + Duration.ofDays(1)

Exception in thread "main"
java.time.temporal.UnsupportedTemporalTypeException:
Unsupported unit: Seconds
    at java.time.LocalDate.plus(LocalDate.java:1247)
```

- A reasonably looking operation should not fail unexpectedly if it’s allowed by the API

Design principles

- Immutable types
- A minimum number of types that solve the practical use cases
- Statically typed to prevent incorrect operations
- Foundational

kotlinx.datetime Types to represent temporals

- Flavors of time

Flavors of time

Physical

Just an integer number of nanoseconds since Epoch

Flavors of time

Physical

Just an integer number of
nanoseconds since Epoch

```
val timestamp: Instant = Clock.System.now()
```

Flavors of time

Physical

Just an integer number of nanoseconds since Epoch

```
val timestamp: Instant = Clock.System.now()
val particular =
    Instant.fromEpochMilliseconds(1597663227207)
```

Flavors of time

Physical

Just an integer number of nanoseconds since Epoch

```
val timestamp: Instant = Clock.System.now()
val particular =
    Instant.fromEpochMilliseconds(1597663227207)
val parsed =
    Instant.parse("2020-08-17T11:20:27.207Z")
```

Flavors of time

Physical

Civil


```
val timestamp: Instant = Clock.System.now()
val particular =
    Instant.fromEpochMilliseconds(1597663227207)
val parsed =
    Instant.parse("2020-08-17T11:20:27.207Z")
```

Flavors of time

Physical


```
val timestamp: Instant = Clock.System.now()
val particular =
    Instant.fromEpochMilliseconds(1597663227207)
val parsed =
    Instant.parse("2020-08-17T11:20:27.207Z")
```

Civil


```
LocalDate(
    2020, Month.OCTOBER, 12
)

LocalDateTime(
    2020, Month.OCTOBER, 12,
    hour: 15, minute: 20, second: 0
)
```

kotlinx.datetime Types to represent temporals

- Flavors of time
- Instant ↔ LocalDateTime conversions

TimeZone conversions: Instant → LocalDateTime

```
val timestamp: Instant = Clock.System.now()

val localTimestamp: LocalDateTime = ?
```

TimeZone conversions: Instant → LocalDateTime

```
val timestamp: Instant = Clock.System.now()

val localTimestamp: LocalDateTime = ?
```

- Gregorian calendar rules: days per year, days per month, hours per day, etc.

TimeZone conversions: Instant → LocalDateTime

```
val timestamp: Instant = Clock.System.now()

val localTimestamp: LocalDateTime = ?
```

- Gregorian calendar rules: days per year, days per month, hours per day, etc.
- Time zone offset for a particular location and time

TimeZone conversions: Instant → LocalDateTime

```
val timestamp: Instant = Clock.System.now()
val tz = TimeZone.currentSystemDefault()
val localTimestamp = timestamp.toLocalDateTime(tz)

println(localTimestamp)   // e.g. 2020-10-13T14:34:29.140
```

TimeZone conversions
LocalDateTime → Instant

```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.JULY, 25)
              .atTime(15, 45)

val instant = local.toInstant(tz) // 2020-07-25T13:45:00Z
```

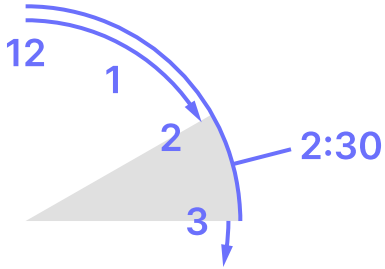
TimeZone conversions
LocalDateTime → Instant

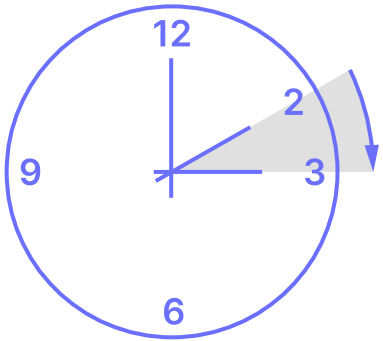
```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.JULY, 25)
                .atTime(15, 45)

val instant = local.toInstant(tz) // 2020-07-25T13:45:00Z

instant.toLocalDateTime(tz)       // 2020-07-25T15:45
```

TimeZone conversions
LocalDateTime → Instant

```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.MARCH, 29)
                .atTime(2, 30)
```

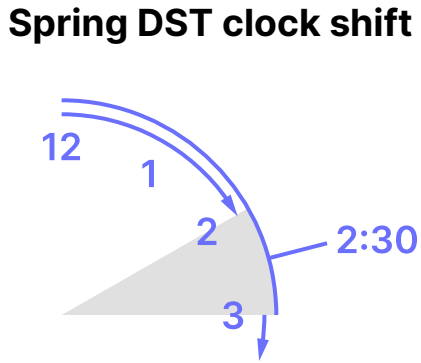
Spring DST clock shift



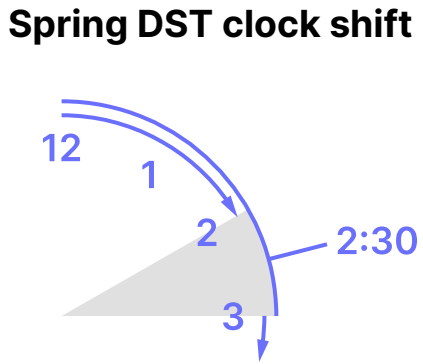


TimeZone conversions
LocalDateTime → Instant

```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.MARCH, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz)
```



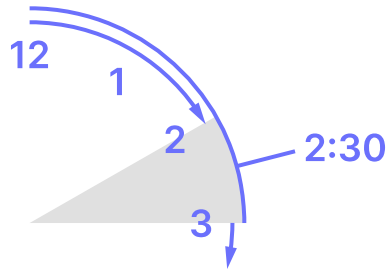
TimeZone conversions
LocalDateTime → Instant




```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.MARCH, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz) // 2020-03-29T01:30:00Z

instant.toLocalDateTime(tz)       // 2020-07-25T03:30
```

TimeZone conversions
LocalDateTime → Instant

Spring DST clock shift

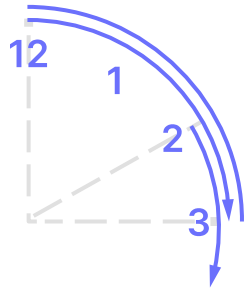



```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.MARCH, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz) // 2020-03-29T01:30:00Z

instant.toLocalDateTime(tz)       // 2020-07-25T03:30
```

Forward correction

TimeZone conversions
LocalDateTime → Instant

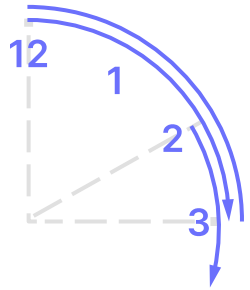
Autumn DST clock shift




```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.OCTOBER, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz)       // 2020-10-25T00:30:00Z
```

TimeZone conversions
LocalDateTime → Instant

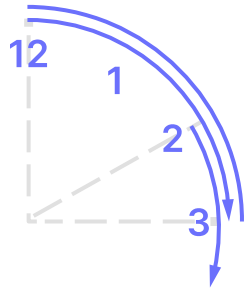
Autumn DST clock shift




```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.OCTOBER, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz)          // 2020-10-25T00:30:00Z

instant.toLocalDateTime(tz)                // 2020-10-25T02:30
```

TimeZone conversions
LocalDateTime → Instant

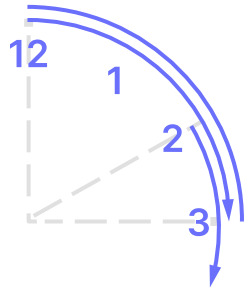
Autumn DST clock shift




```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.OCTOBER, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz)          // 2020-10-25T00:30:00Z

instant.toLocalDateTime(tz)                // 2020-10-25T02:30
(instant + 1.hours).toLocalDateTime(tz)    // 2020-10-25T02:30
```

TimeZone conversions
LocalDateTime → Instant

Autumn DST clock shift




```
val tz = TimeZone.of("Europe/Berlin")
val local = LocalDate(2020, Month.OCTOBER, 29)
                .atTime(2, 30)

val instant = local.toInstant(tz)          // 2020-10-25T00:30:00Z

instant.toLocalDateTime(tz)                // 2020-10-25T02:30
(instant + 1.hours).toLocalDateTime(tz)    // 2020-10-25T02:30
```

The same date and time for different instants

kotlinx.datetime Types to represent temporals

- Flavors of time
- Instant ↔ LocalDateTime conversions
- When to use which type

Using Instant: log timestamp

```
fun log(message: String) {
    val ts: Instant = Clock.System.now()
    println("$ts $message")
}

log("Service started")
```

Using Instant: log timestamp

```
fun log(message: String) {
    val ts: Instant = Clock.System.now()
    println("$ts $message")
}

log("Service started")
```

```
2020-08-17T11:55:15.185Z Service started
```

Using Instant: log timestamp

```
fun log(message: String) {
    val ts: Instant = Clock.System.now()
    println("$ts $message")
}

log("Service started")
```

```
2020-08-17T11:55:15.185Z Service started
```

Using Instant: log timestamp in local TZ

```
private val systemTz = TimeZone.currentSystemDefault()
fun log(message: String) {
    val ts: Instant = Clock.System.now()
    println("${ts.toLocalDateTime(systemTz)} $message")
}

log("Service started")
```

```
2020-08-17T14:55:15.185 Service started
```

Using Instant: log timestamp in local TZ with its offset

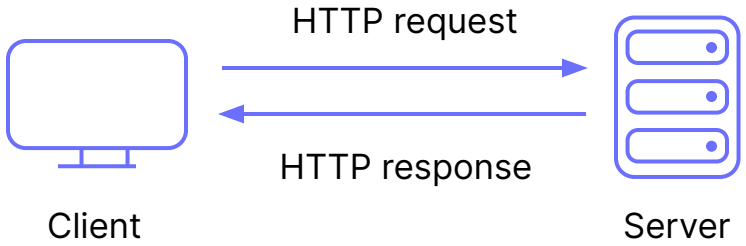
```
private val systemTz = TimeZone.currentSystemDefault()
fun log(message: String) {
    val ts: Instant = Clock.System.now()
    println("${ts.toLocalDateTime(systemTz)}${ts.offsetIn(systemTz)} $message")
}

log("Service started")
```

```
2020-08-17T14:55:15.185+03:00 Service started
```

Using Instant: modification timestamp

▼ **Request Headers**

:authority: youtrack.jetbrains.com
:method: GET
:path: /youtrack.svg
:scheme: https
accept: text/html,application/xhtml+xml,applicat
q=0.9
accept-encoding: gzip, deflate, br
accept-language: en-US,en;q=0.9,ru;q=0.8
cache-control: max-age=0
dnt: 1
if-modified-since: Mon, 10 Aug 2020 16:14:16 GMT



Using Instant: modification timestamp

```
if-modified-since: Mon, 10 Aug 2020 16:14:16 GMT
```

```
val modifiedSince: Instant = call.request.ifModifiedSince() // 2020-08-10T16:14:16Z
```

Using Instant: modification timestamp

```
if-modified-since: Mon, 10 Aug 2020 16:14:16 GMT
```

```
val modifiedSince: Instant = call.request.ifModifiedSince() // 2020-08-10T16:14:16Z

// checking that resource has changed since the last request
if (resource.lastModified <= modifiedSince) {
   call.respond(HttpStatusCode.NotModified)
}
```

Using Instant: modification timestamp

```
if-modified-since: Mon, 10 Aug 2020 16:14:16 GMT
```

```
val modifiedSince: Instant = call.request.ifModifiedSince() // 2020-08-10T16:14:16Z

// checking that resource has changed since the last request
if (resource.lastModified <= modifiedSince) {
   call.respond(HttpStatusCode.NotModified)
} else {
   call.respondText(resource.content)
}
```

Future Instants: deadline

▼ **Response Headers**

accept-ranges: bytes

access-control-allow-origin: *

age: 51912

cache-control: public, max-age=315360000, s-maxage=86400, immutable

content-disposition: inline; filename="LyonText-Italic-Web.woff"

content-length: 64793

content-type: application/font-woff

Future Instants: deadline

```
cache-control: max-age=31536000
```

Future Instants: deadline

```
cache-control: max-age=31536000
```

```
val expires: Instant = Clock.System.now() + response.cacheControl.maxAge.seconds
```

Future Instants: deadline

```
cache-control: max-age=31536000
```

```
val expires: Instant = Clock.System.now() + response.cacheControl.maxAge.seconds
cachedResource.expires = expires
cachedResource.content = response.content
```

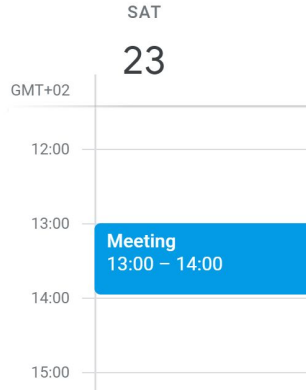
Future Instants: deadline

```
cache-control: max-age=31536000
```

```
val expires: Instant = Clock.System.now() + response.cacheControl.maxAge.seconds
cachedResource.expires = expires
cachedResource.content = response.content


// ...

if (Clock.System.now() > cachedResource.expires) {
   // refresh expired content
} else {
   return cachedResource.content
}
```

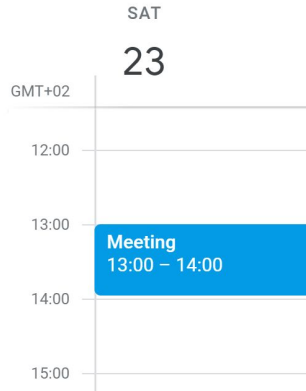
Future Instants: scheduled events?

```
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+2
val meetingStarts = LocalDateTime(2025, Month.AUGUST, 23, 13, 00)
val startInstant = meetingStarts.toInstant(localTz) // 2025-08-13T11:00:00Z
```



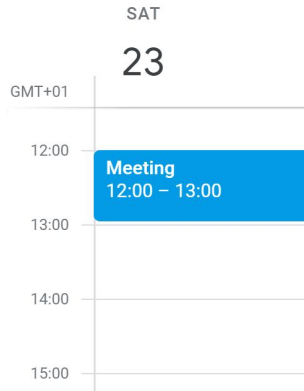
Future Instants: scheduled events?

```
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+2
val meetingStarts = LocalDateTime(2025, Month.AUGUST, 23, 13, 00)
val startInstant = meetingStarts.toInstant(localTz) // 2025-08-13T11:00:00Z

... 5 years later ...

val startInstant = Instant.parse("2025-08-13T11:00:00Z")
val localTz = TimeZone.currentSystemDefault()
val meetingStarts = startInstant.toLocalDateTime(localTz)
```



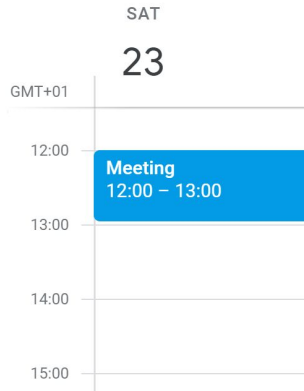
Future Instants: scheduled events?

```
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+2
val meetingStarts = LocalDateTime(2025, Month.AUGUST, 23, 13, 00)
val startInstant = meetingStarts.toInstant(localTz) // 2025-08-13T11:00:00Z

... 5 years later ...

val startInstant = Instant.parse("2025-08-13T11:00:00Z")
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+1
val meetingStarts = startInstant.toLocalDateTime(localTz)
                                                   // 2025-08-13T12:00
```



Future Instants: scheduled events?

```
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+2
val meetingStarts = LocalDateTime(2025, Month.AUGUST, 23, 13, 00)
val startInstant = meetingStarts.toInstant(localTz) // 2025-08-13T11:00:00Z

... 5 years later ...

val startInstant = Instant.parse("2025-08-13T11:00:00Z")
val localTz = TimeZone.currentSystemDefault()      // Europe/Berlin: UTC+1
val meetingStarts = startInstant.toLocalDateTime(localTz)
                                                   // 2025-08-13T12:00
```



- Consider what the source of truth is in your domain
- Preserve it when persisting data

No Zoned/OffsetDateTime?

No Zoned/OffsetDateTime?

OffsetDateTime: LocalDateTime + exact offset from UTC

ZonedDateTime: LocalDateTime + TimeZone + exact offset from UTC

- Can be convenient in date/time calculations

No Zoned/OffsetDateTime?

OffsetDateTime: LocalDateTime + exact offset from UTC

ZonedDateTime: LocalDateTime + TimeZone + exact offset from UTC

- Can be convenient in date/time calculations
- **but**: Problematic comparisons
  2020-10-01 15:00 +01:00 < = > 2020-10-01 16:00 +02:00

No Zoned/OffsetDateTime?

OffsetDateTime: LocalDateTime + exact offset from UTC

ZonedDateTime: LocalDateTime + TimeZone + exact offset from UTC

- Can be convenient in date/time calculations
- **but**: Problematic comparisons
  2020-10-01 15:00 +01:00 < = > 2020-10-01 16:00 +02:00
- **but**: Unclear what zone to use in an operation involving two ZonedDateTimes

No Zoned/OffsetDateTime?

OffsetDateTime: LocalDateTime + exact offset from UTC

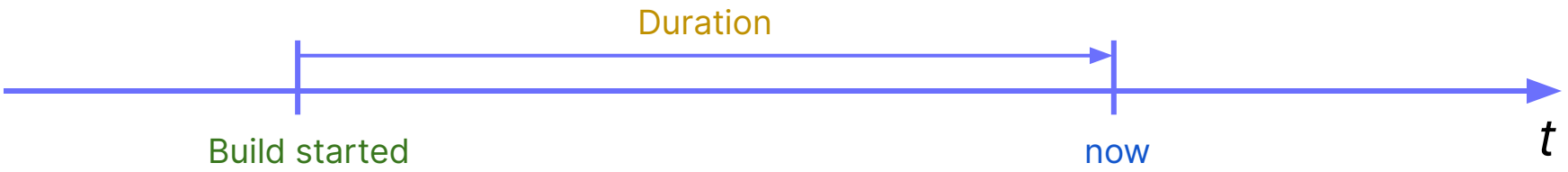
ZonedDateTime: LocalDateTime + TimeZone + exact offset from UTC

- Can be convenient in date/time calculations
- **but**: Problematic comparisons
  2020-10-01 15:00 +01:00 < = > 2020-10-01 16:00 +02:00
- **but**: Unclear what zone to use in an operation involving two ZonedDateTimes
- **but**: Components of a ZonedDateTime may become inconsistent if it is stored for a long time

kotlinx.datetime

- Operations on temporal types

Duration between Instants



Kotlin / Kotlin Dev / Tests / Gradle Integration Tests

#1.4.20-dev-3802 at 20 Aug 16:21

Tests passed: 541, ignored: 32; :kotlin-gradle-plugin-integration-tests:tes

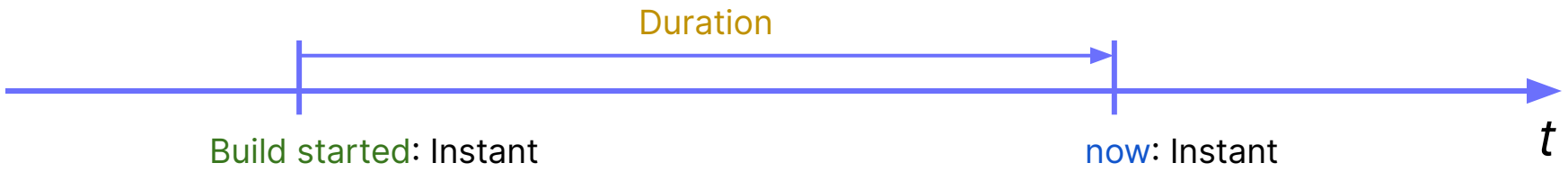
master | Stop... | Actions | Details | Assign investiga

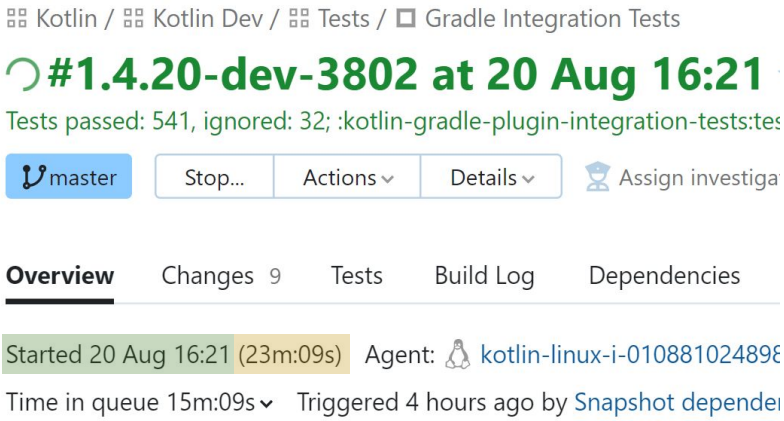
Overview | Changes 9 | Tests | Build Log | Dependencies

Started 20 Aug 16:21 (23m:09s) Agent: kotlin-linux-i-010881024898

Time in queue 15m:09s Triggered 4 hours ago by Snapshot depende

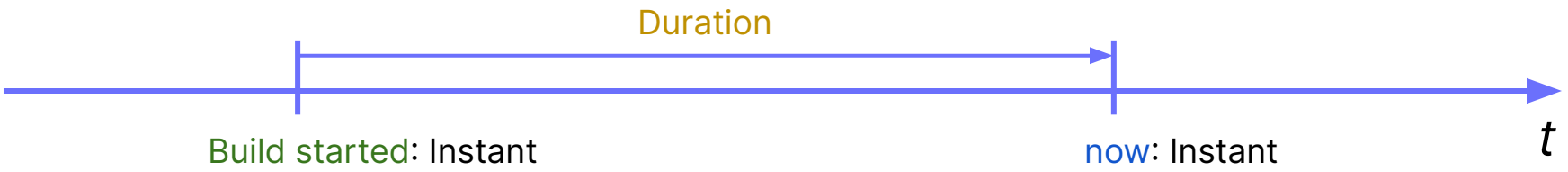
Duration between Instants

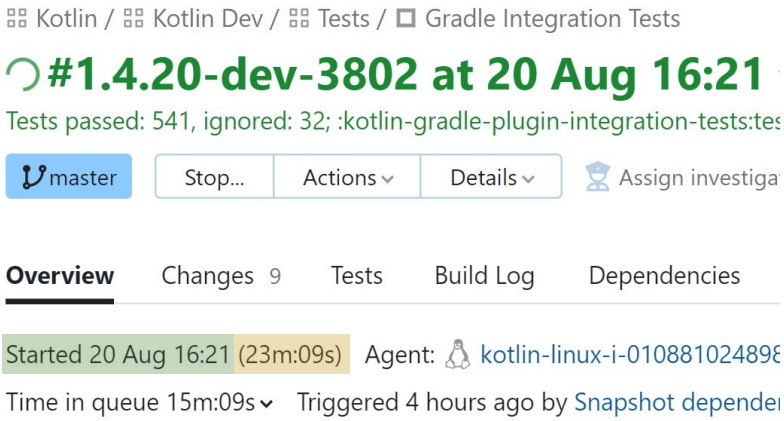





```
val duration: Duration =
    Clock.System.now() - buildStarted

duration    // 23.1m
```

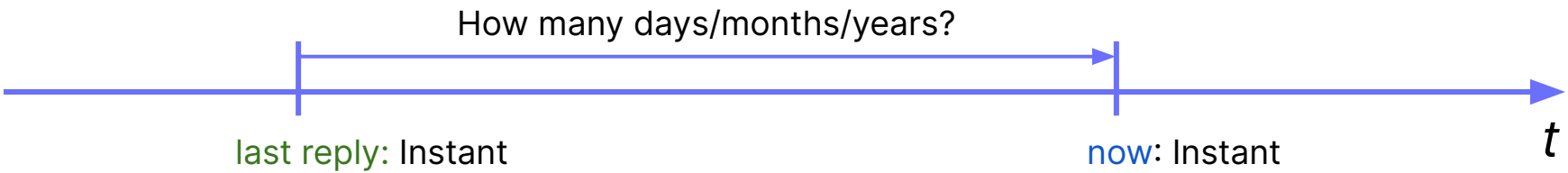
Duration between Instants: units

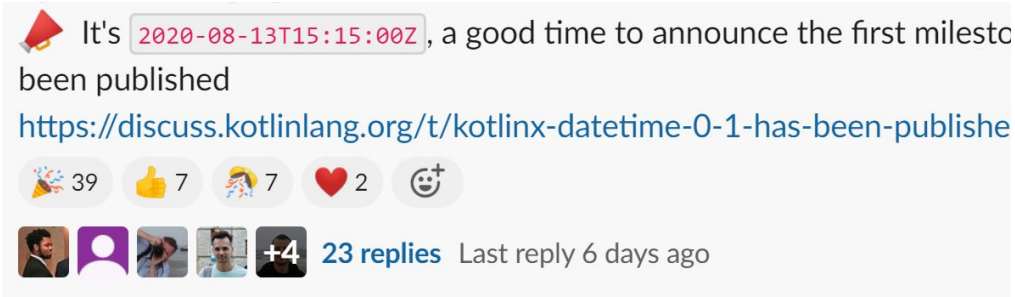





```
val duration: Duration =
    Clock.System.now() - buildStarted

duration.inSeconds // 1389
duration.inMinutes // 23.15
duration.inHours   // 0.385

duration.toComponents { h, m, s, _ ->
}
```

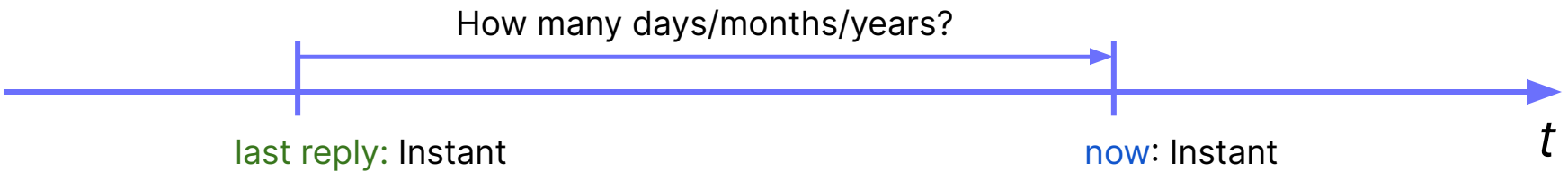
Days/months/years since Instant






```
val inDays: Int = (Clock.System.now() - lastReply) / ?
```

Days/months/years since Instant



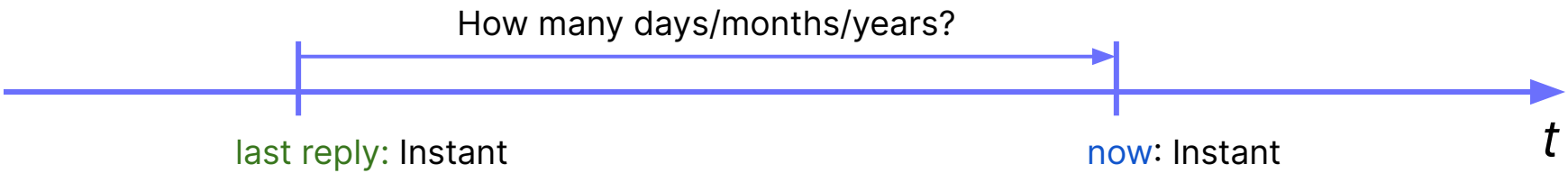
- Months and years can have different lengths in days
- Day is not always 24H 

An Instant can't take into account DST effects (except by passing the zone in to each method call, which is just a horrible user experience).

Stephen Colebourne

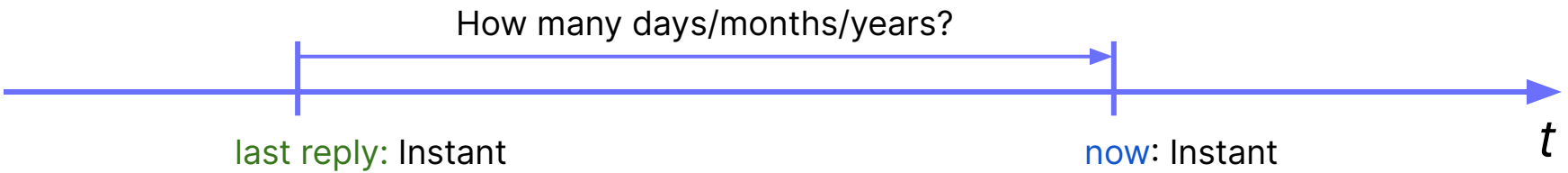
Java Champion, known for Joda projects, JSR-310

Days/months/years since Instant



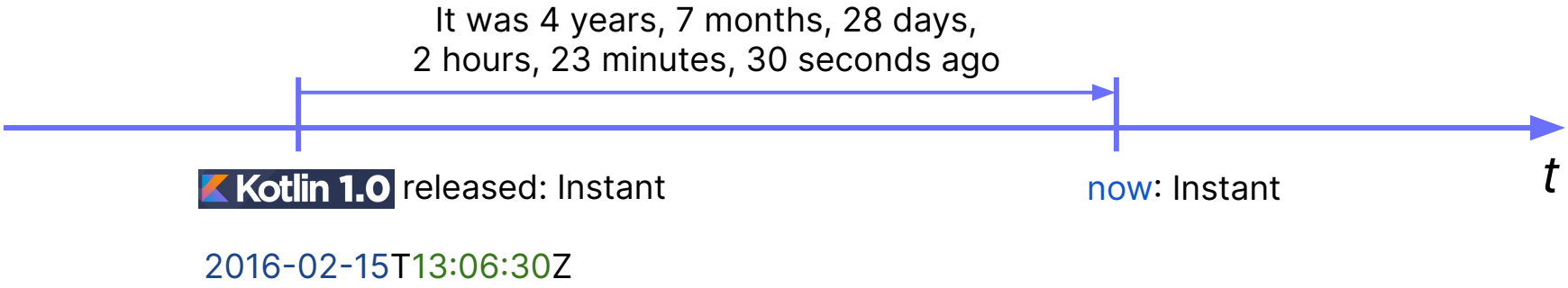

```
val zone = TimeZone.of("Europe/Berlin")
val now = Clock.System.now()

val inDays = lastReply.until(now, DateTimeUnit.DAY, zone)
val inMonths = lastReply.until(now, DateTimeUnit.MONTH, zone)
val inYears = lastReply.until(now, DateTimeUnit.YEAR, zone)
```

Days/months/years since Instant

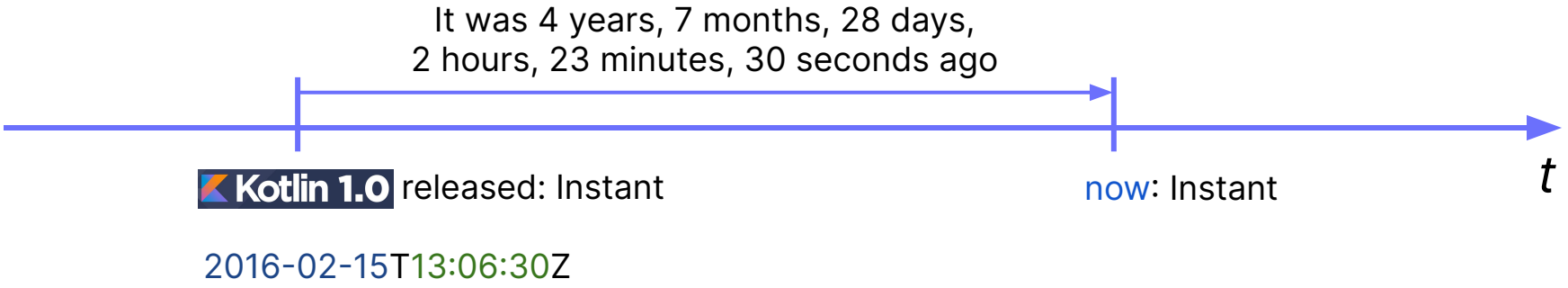



```
val zone = TimeZone.of("Europe/Berlin")
val now = Clock.System.now()

val inDays = lastReply.daysUntil(now, zone)
val inMonths = lastReply.monthsUntil(now, zone)
val inYears = lastReply.yearsUntil(now, zone)
```

Period since Instant

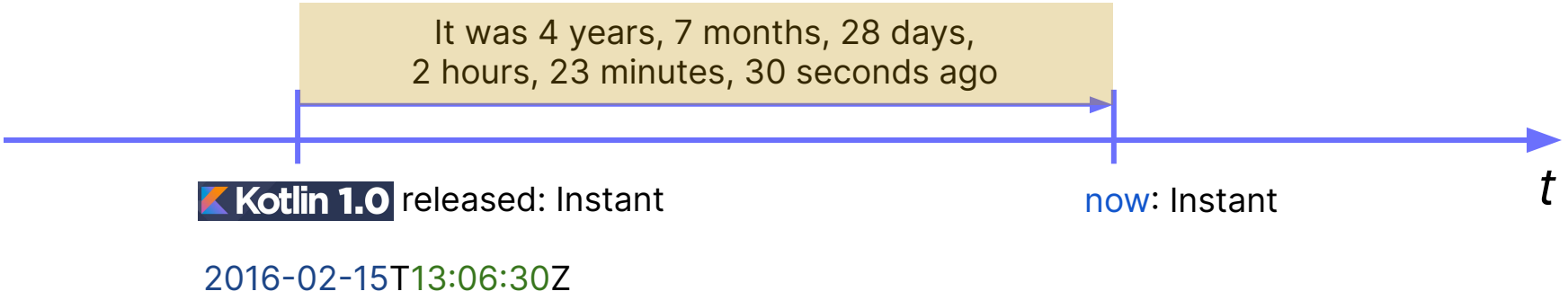


Period since Instant



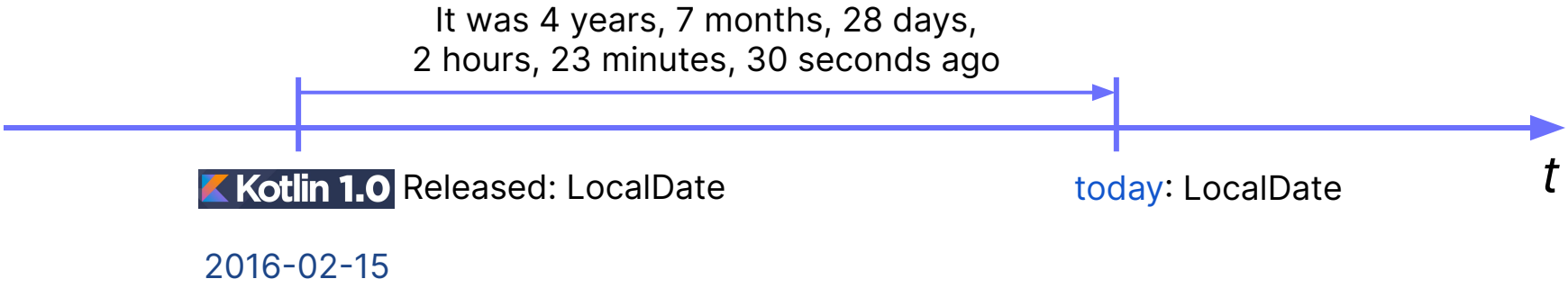

```
val kotlinReleased = Instant.parse("2016-02-15T13:06:30Z")
val now = Clock.System.now()
val zone = TimeZone.currentSystemDefault()
```

Period since Instant




```
val kotlinReleased = Instant.parse("2016-02-15T13:06:30Z")
val now = Clock.System.now()
val zone = TimeZone.currentSystemDefault()

val period: DateTimePeriod = kotlinReleased.periodUntil(now, zone)
with(period) {
    println("It was $years years, $months months, $days days, $hours hours, ... ago")
}
```

Period since Instant




```
val kotlinReleased = LocalDate("2016-02-15")
val today = Clock.System.todayAt(TimeZone.currentSystemDefault())

val period: DatePeriod = kotlinReleased.periodUntil(today)
with(period) {
    println("It was $years years, $months months, $days days ago")
}
```

Kotlin style in API

Kotlin style in API

```
val result = someInstant.toLocalDateTime(zone).date.atTime(12, 0).toInstant(zone)
```

Kotlin style in API with(zone)

```kotlin
val result = someInstant.toLocalDateTime(zone).date.atTime(12, 0).toInstant(zone)
```

```kotlin
val result = with(zone) {
   someInstant.toLocalDateTime().date.atTime(12, 0).toInstant()
}
```

Kotlin style in API with(zone)

```kotlin
val result = someInstant.toLocalDateTime(zone).date.atTime(12, 0).toInstant(zone)
```

```kotlin
val result = with(zone) {
    someInstant.toLocalDateTime().date.atTime(12, 0).toInstant()
}
```

```kotlin
class TimeZone {
    fun Instant.toLocalDateTime(): LocalDateTime
    fun LocalDateTime.toInstant(): Instant
}
```

Kotlin style in API Custom units

```
val inWeeks    = instant1.until(instant2, DateTimeUnit.WEEK, zone)
val inQuarters = instant1.until(instant2, DateTimeUnit.QUARTER, zone)
val inMinutes  = instant1.until(instant2, DateTimeUnit.MINUTE, zone)
```

Kotlin style in API Custom units

```
val inWeeks     = instant1.until(instant2, DateTimeUnit.WEEK, zone)
val inQuarters  = instant1.until(instant2, DateTimeUnit.QUARTER, zone)
val inMinutes   = instant1.until(instant2, DateTimeUnit.MINUTE, zone)

val in4WeekIntervals = instant1.until(instant2, DateTimeUnit.WEEK * 4, zone)
```

Kotlin style in API Custom units

```
val inWeeks    = instant1.until(instant2, DateTimeUnit.WEEK, zone)
val inQuarters = instant1.until(instant2, DateTimeUnit.QUARTER, zone)
val inMinutes  = instant1.until(instant2, DateTimeUnit.MINUTE, zone)

val in4WeekIntervals = instant1.until(instant2, DateTimeUnit.WEEK * 4, zone)
```

```
public sealed class DateTimeUnit {
   public abstract operator fun times(scalar: Int): DateTimeUnit
}
```

java.time
Adding time units to a date

```
val date = LocalDate.of(2020, 3, 15)
date + Duration.ofMinutes(1)

Exception in thread "main"
java.time.temporal.UnsupportedTemporalTypeException:
Unsupported unit: Seconds
    at java.time.LocalDate.plus(LocalDate.java:1247)
```

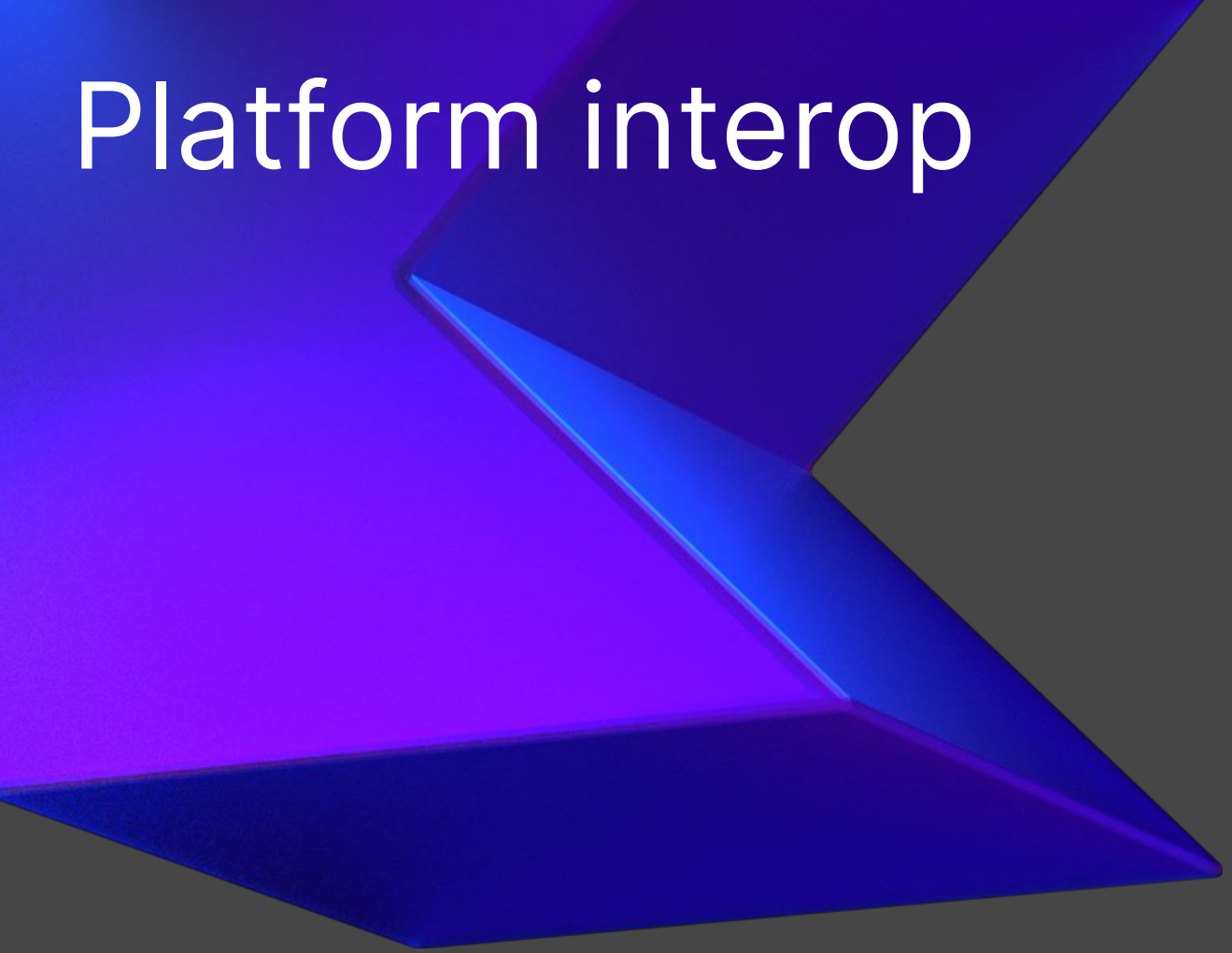
kotlinx.datetime
Adding time units to a date

```
val date = LocalDate(2020, 3, 15)
date + 1.minutes
date.plus(1, DateTimeUnit.MINUTE)
```

kotlinx.datetime
Adding time units to a date

```
val date = LocalDate(2020, 3, 15)
date + 1.minutes
date.plus(1, DateTimeUnit.MINUTE)

fun LocalDate.plus(duration: Duration)

fun LocalDate.plus(value: Int, unit: DateTimeUnit.DateBased): LocalDate
```

Platform interop



kotlinx-datetime
Types and operations: recap

- Types to represent temporals: **Instant**, **LocalDateTime**, **LocalDate**

kotlinx-datetime
Types and operations: recap

- Types to represent temporals: **Instant**, **LocalDateTime**, **LocalDate**
- Date and time arithmetic: **plus**, **minus**, **until**, **periodUntil**

kotlinx-datetime
Types and operations: recap

- Types to represent temporals: **Instant**, **LocalDateTime**, **LocalDate**
- Date and time arithmetic: **plus**, **minus**, **until**, **periodUntil**
- Helper types: **DateTimeUnit**, **DateTimePeriod**/**DatePeriod**, **Month**, **DayOfWeek**

Converters to/from java.time.*

kotlinx.datetime.*

```
Instant.toJavaInstant()

LocalDateTime.toJavaLocalDateTime()

LocalDate.toJavaLocalDate()

DatePeriod.toJavaPeriod()

TimeZone.toJavaZoneId()

ZoneOffset.toJavaZoneOffset()
```

Converters to/from java.time.*

kotlinx.datetime.*	**java.time.***
`Instant.toJavaInstant()`	`Instant.toKotlinInstant()`
`LocalDateTime.toJavaLocalDateTime()`	`LocalDateTime.toKotlinLocalDateTime()`
`LocalDate.toJavaLocalDate()`	`LocalDate.toKotlinLocalDate()`
`DatePeriod.toJavaPeriod()`	`Period.toKotlinDatePeriod()`
`TimeZone.toJavaZoneId()`	`ZoneId.toKotlinTimeZone()`
`ZoneOffset.toJavaZoneOffset()`	`ZoneOffset.toKotlinZoneOffset()`

Formatting with java.time.*

```
val instant: Instant = Clock.System.now()

val formatted = DateTimeFormatter.RFC_1123_DATE_TIME.format(
       instant.toJavaInstant().atZone(TimeZone.UTC.toJavaZoneId()))

// Fri, 21 Aug 2020 10:57:31 GMT
```

Converters to/from platform.Foundation

kotlinx.datetime.*	**platform.Foundation.***
`Instant.toNSDate(): NSDate`	`NSDate.toKotlinInstant()`
`LocalDate.toNSDateComponents()` `LocalDateTime.toNSDateComponents()`	
`TimeZone.toNSTimeZone(): NSTimeZone`	`NSTimeZone.toKotlinTimeZone()`

How to try and what’s next

How to try

```
repositories {
    jcenter()
}
kotlin {
    sourceSets {
        val commonMain by getting {
            dependencies {
                implementation("org.jetbrains.kotlinx:kotlinx-datetime:0.1.0")
            }
        }
    }
}
```

Next plans

- Integration with kotlinx.serialization

Next plans

- Integration with kotlinx.serialization
- Formatting and parsing (at first, without internationalization)

Next plans

- Integration with kotlinx.serialization
- Formatting and parsing (at first, without internationalization)
- Ranges and progressions

```
for (date in startDate..endDate)
for (interval in startTime..endTime step 5.minutes)
```

Next plans

- Integration with kotlinx.serialization
- Formatting and parsing (at first, without internationalization)
- Ranges and progressions

```
for (date in startDate..endDate)
for (interval in startTime..endTime step 5.minutes)
```

- Embedding TZDB

Please share your feedback

- Issue tracker: https://github.com/kotlin/kotlinx-datetime/issues
- Forum: https://discuss.kotlinlang.org/
- Slack: https://kotlinlang.slack.com/ **#kotlinx-datetime**

Thanks!
Have a good time with Kotlin

ilya-g

ilya.gorbunov@jetbrains.com