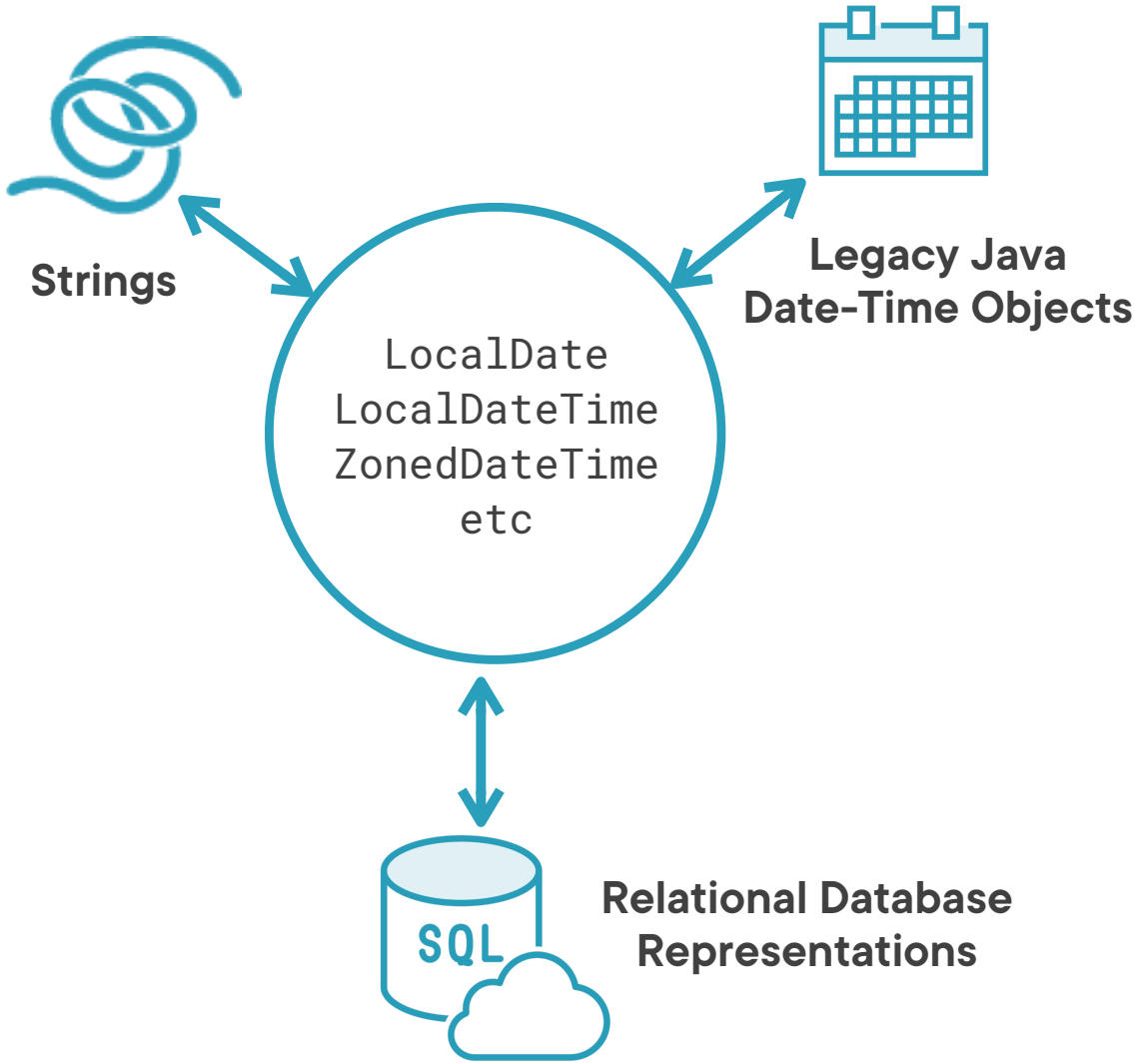
Interconversions and Testing

Maurice Naftalin
Java Champion, JavaOne Rock Star

@mauricenaftalin http://mauricenaftal.in


Strings
Legacy Java
Date-Time Objects
LocalDate
LocalDateTime
ZonedDateTime
etc
SQL
Relational Database
Representations

Module Overview

`TemporalQuery`

Interconversions

- Strings, legacy classes, databases

Unit testing

Demo: testing the methods of the application

“Native” String Interconversion

“Native” String Interconversions

```
ZonedDateTime.of(
   LocalDate.of(2011,12,3),
   LocalTime.of(10,15,30),
   ZoneId.of("Europe/Paris")
)
```

toString()

```
"2011-12-03T10:15:30+01:00[Europe/Paris]"
```

assigned to variable
`docDateTime`

“Native” String Interconversions

```
ZonedDateTime.of(
   LocalDate.of(2011,12,3),
   LocalTime.of(10,15,30),
   ZoneId.of("Europe/Paris")
)
```

toString()

```
"2011-12-03T10:15:30+01:00[Europe/Paris]"
```

```
ZonedDateTime.parse("2011-12-03T10:15:30+01:00[Europe/Paris]")
```

Many Different Formats

```
ZonedDateTime.of(
   LocalDate.of(2011,12,3),
   LocalTime.of(10,15,30),
   ZoneId.of("Europe/Paris")
)
```

2011-12-03T10:15:30Z

3 Dec 2011, 10:15:30 Central European Standard Time

Saturday, 3 December 2011 at 10:15:30 CET

Tue, 3 Jun 2008 11:05:30 GMT

2011-12-03+01:00

2012-337 10:15:30

2011-12-03 10:15:30+01:00

2011-12-03T10:15:30+01:00[Europe/Paris]

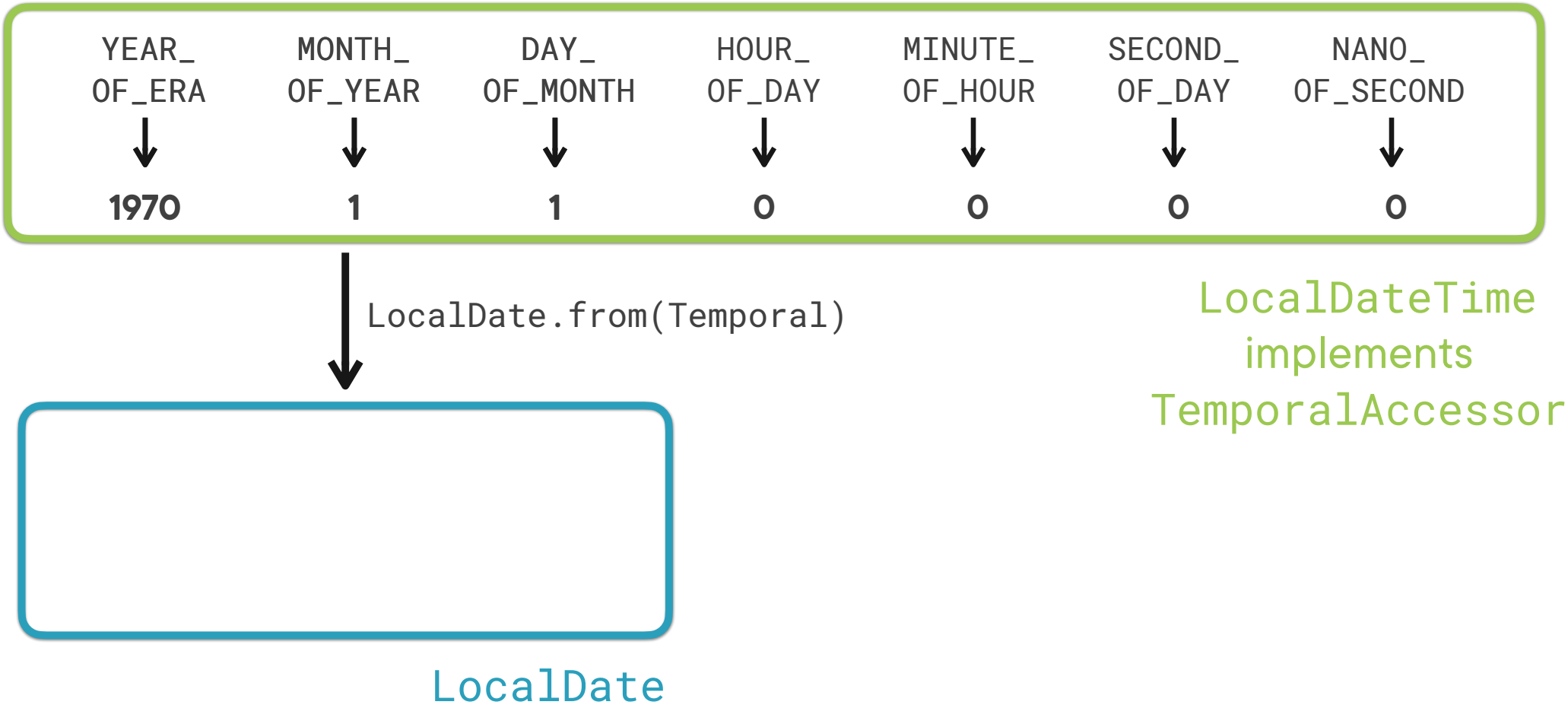
2011-12-03T10:15:30+01:00

2011-12-03T10:15:30

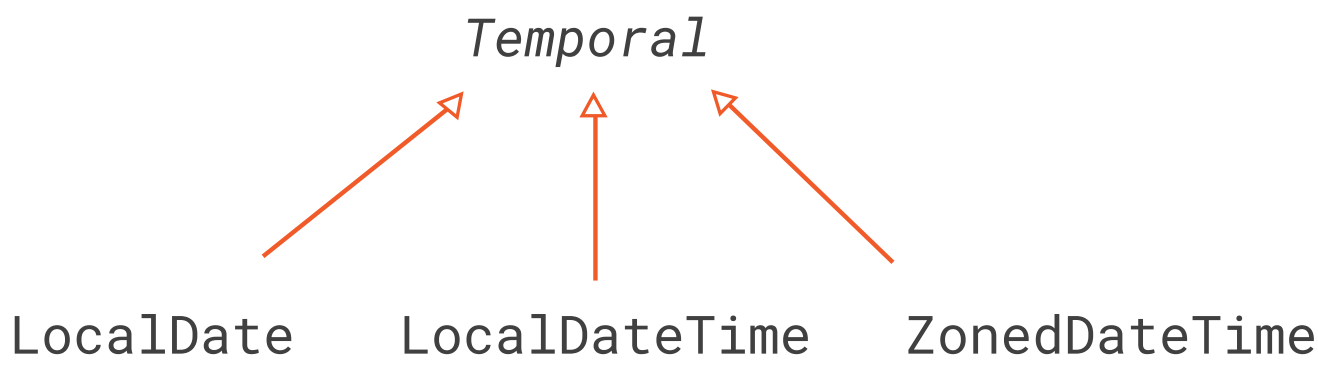
20111203

TemporalQuery

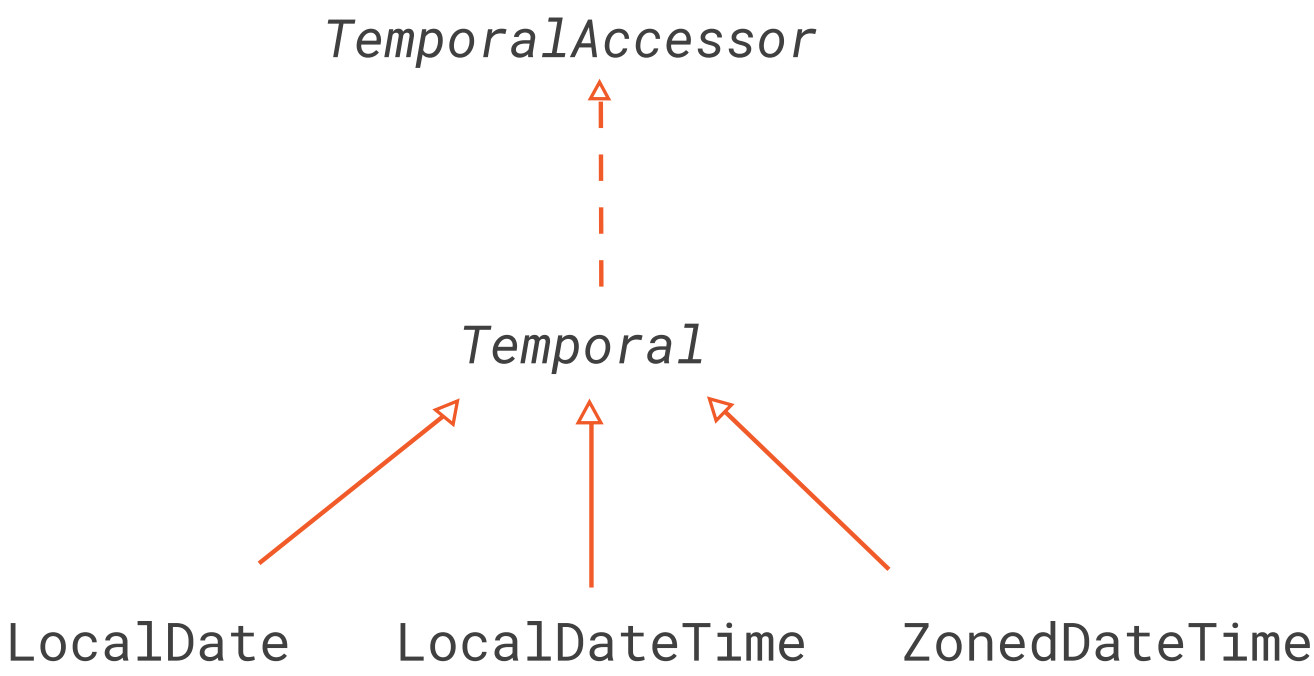
from(...) Methods



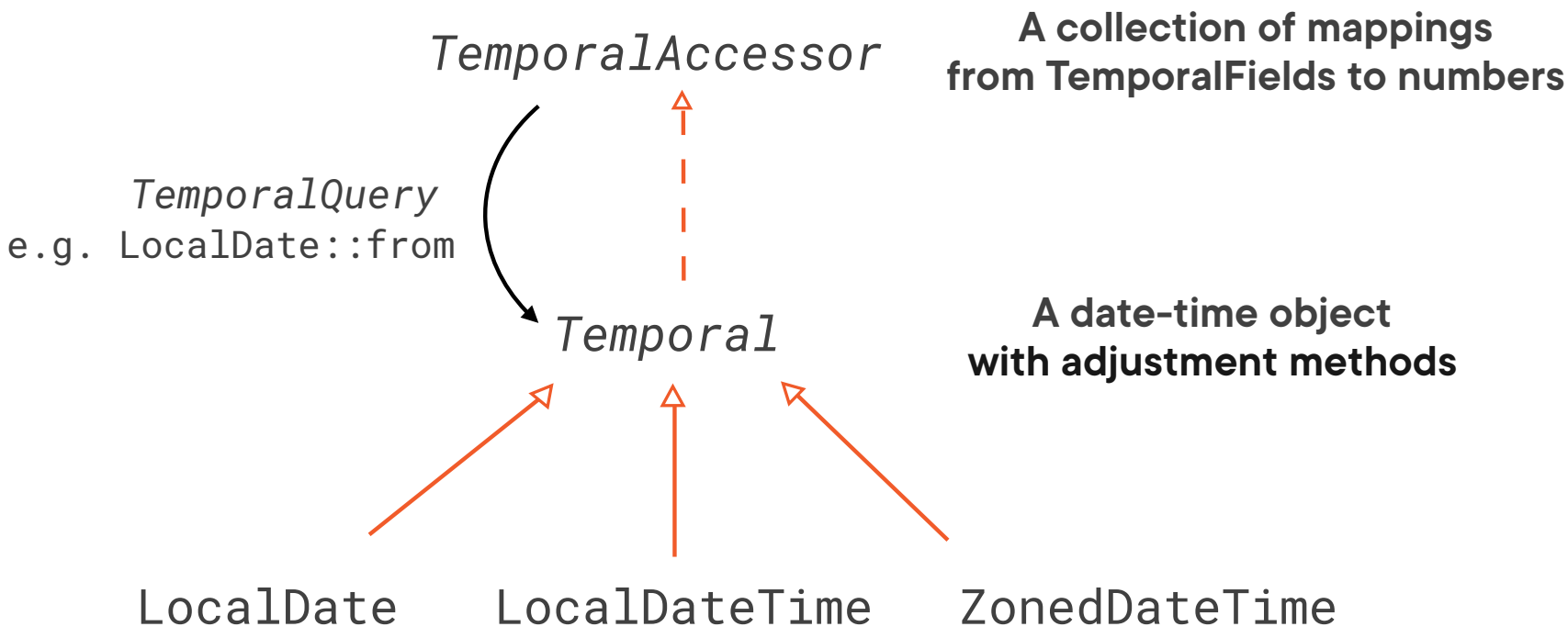
TemporalAccessor



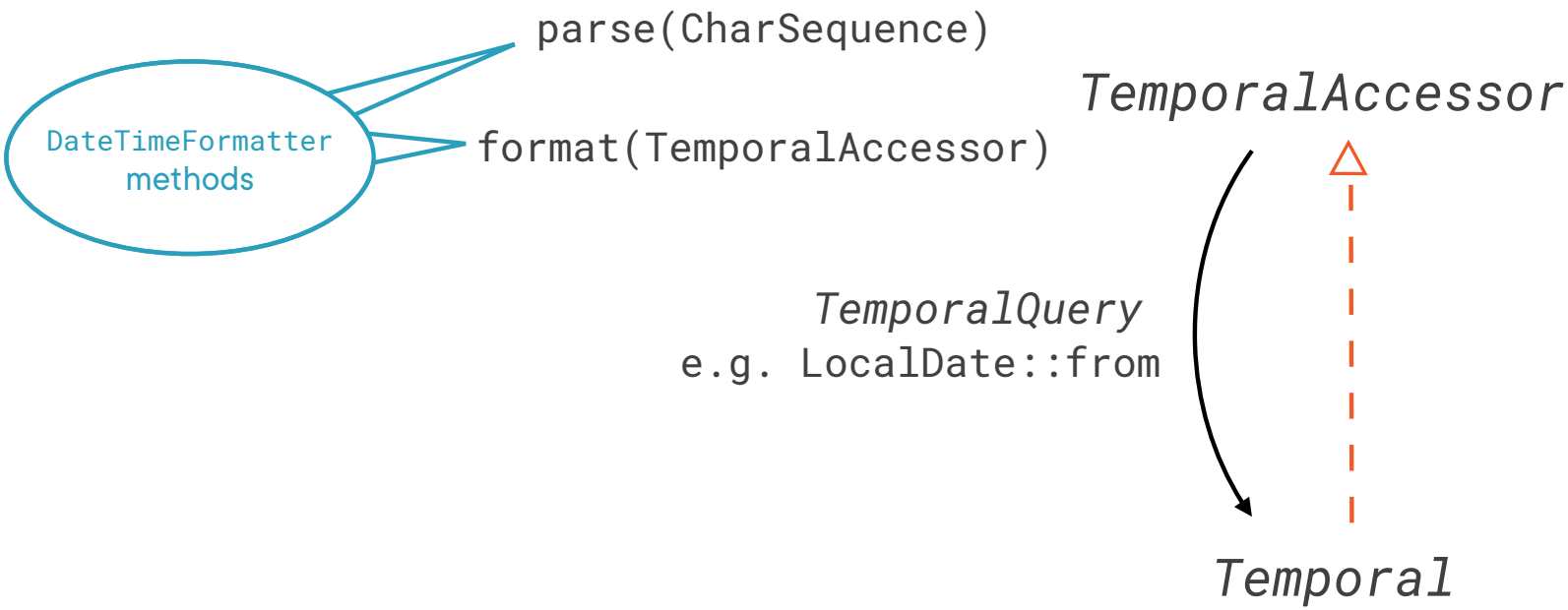
TemporalAccessor



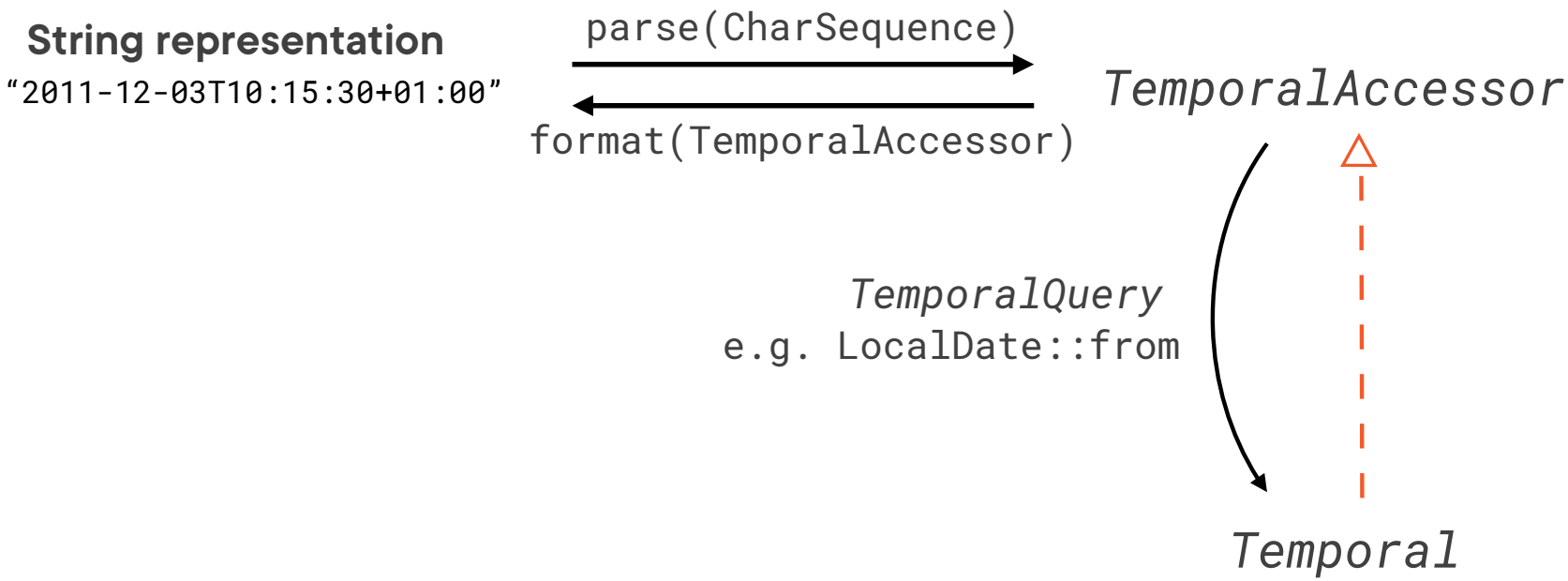
TemporalAccessor



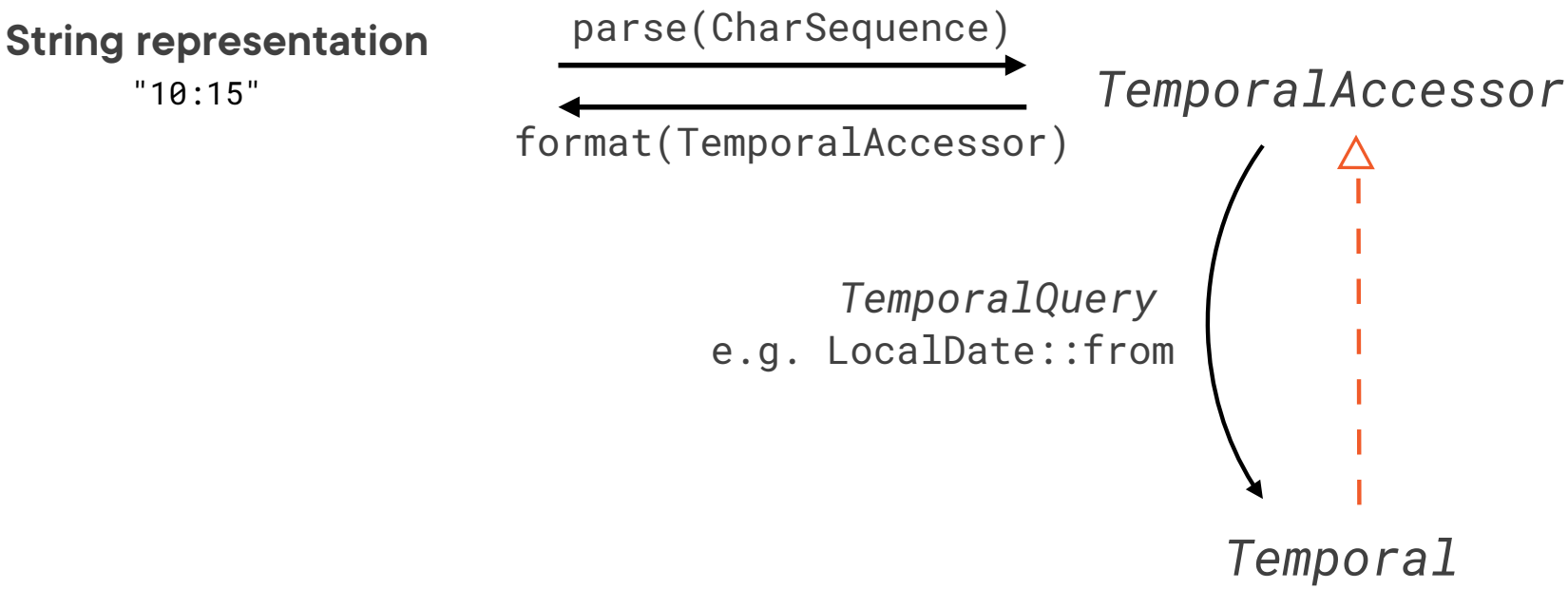
TemporalAccessor



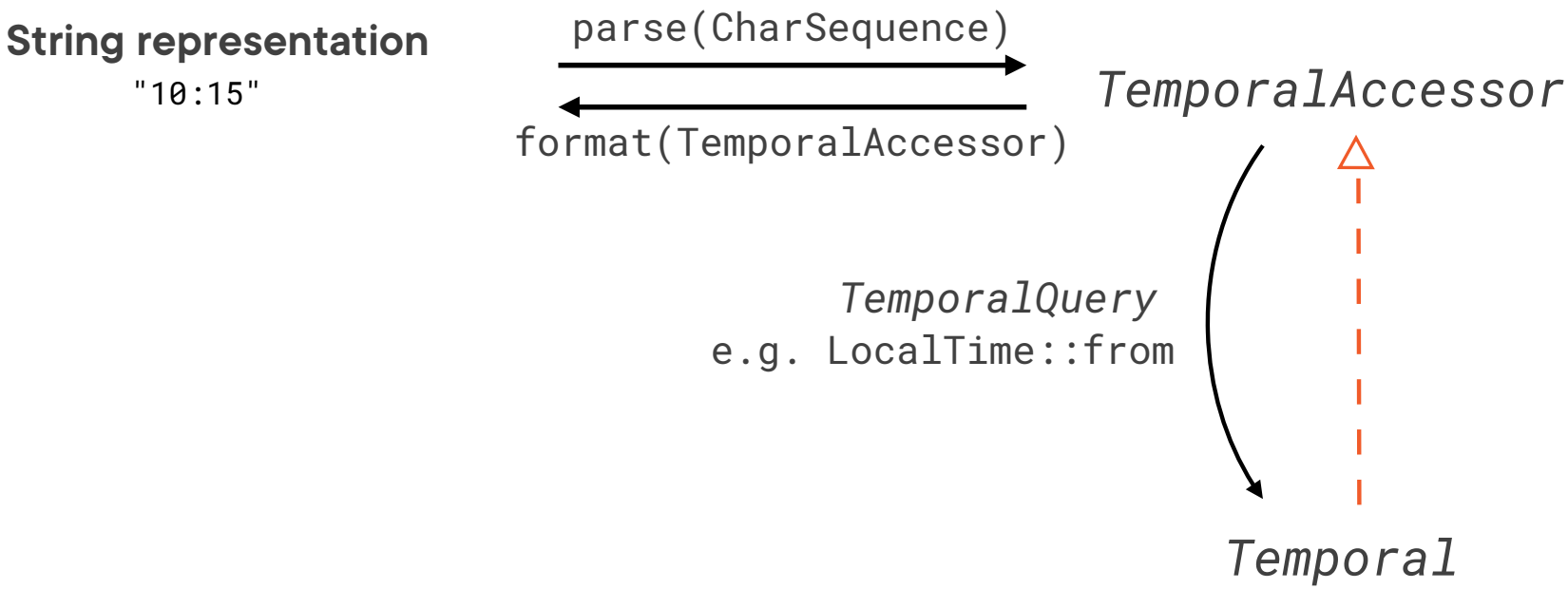
TemporalAccessor



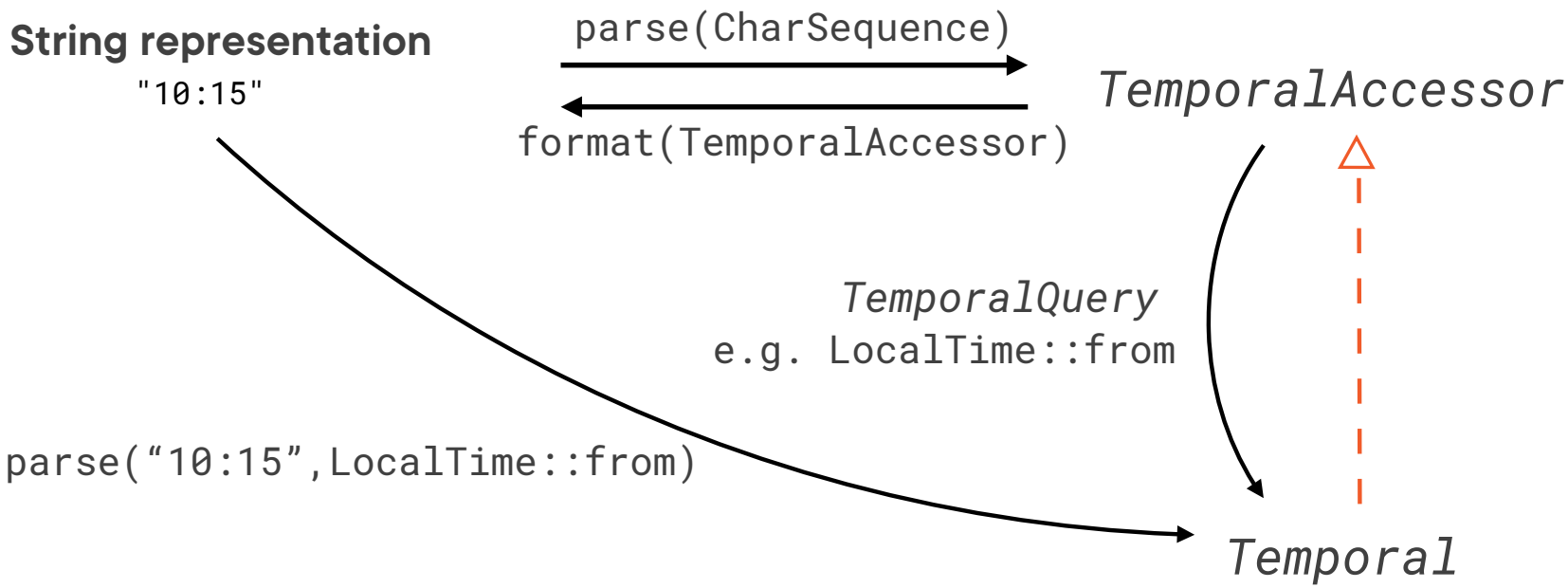
TemporalAccessor



TemporalAccessor

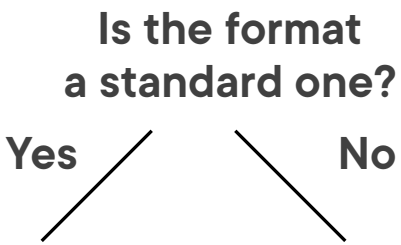


TemporalAccessor



DateTimeFormatter

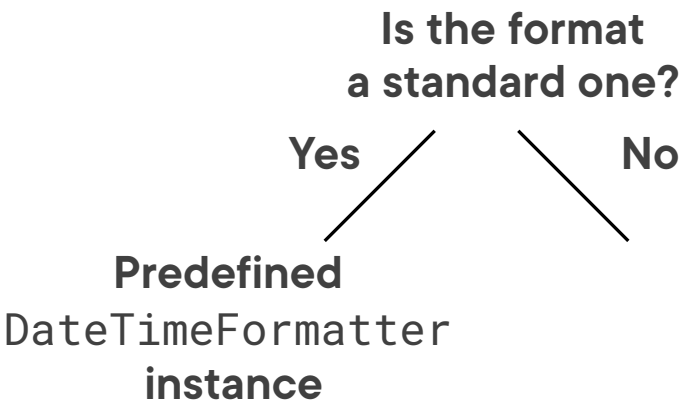
How to Create a `DateTimeFormatter`



Standard Formats

	Example
	'20111203'
	'2011-12-03'
	'2011-12-03+01:00'
	'2011-12-03+01:00'; '2011-12-03'
	'10:15:30'
	'10:15:30+01:00'
	'10:15:30+01:00'; '10:15:30'
	'2011-12-03T10:15:30'
	2011-12-03T10:15:30+01:00'
	'2011-12-03T10:15:30+01:00[Europe/Paris]'
	'2011-12-03T10:15:30+01:00[Europe/Paris]'
	'2012-337'
	2012-W48-6'
	'2011-12-03T10:15:30Z'
	'Tue, 3 Jun 2008 11:05:30 GMT'

How to Create a `DateTimeFormatter`



Standard Formats

	Example
	'20111203'
	'2011-12-03'
	'2011-12-03+01:00'
	'2011-12-03+01:00'; '2011-12-03'
	'10:15:30'
	'10:15:30+01:00'
	'10:15:30+01:00'; '10:15:30'
	'2011-12-03T10:15:30'
	2011-12-03T10:15:30+01:00'
	'2011-12-03T10:15:30+01:00[Europe/Paris]'
	'2011-12-03T10:15:30+01:00[Europe/Paris]'
	'2012-337'
	2012-W48-6'
	'2011-12-03T10:15:30Z'
	'Tue, 3 Jun 2008 11:05:30 GMT'

Predefined `DateTimeFormatters`

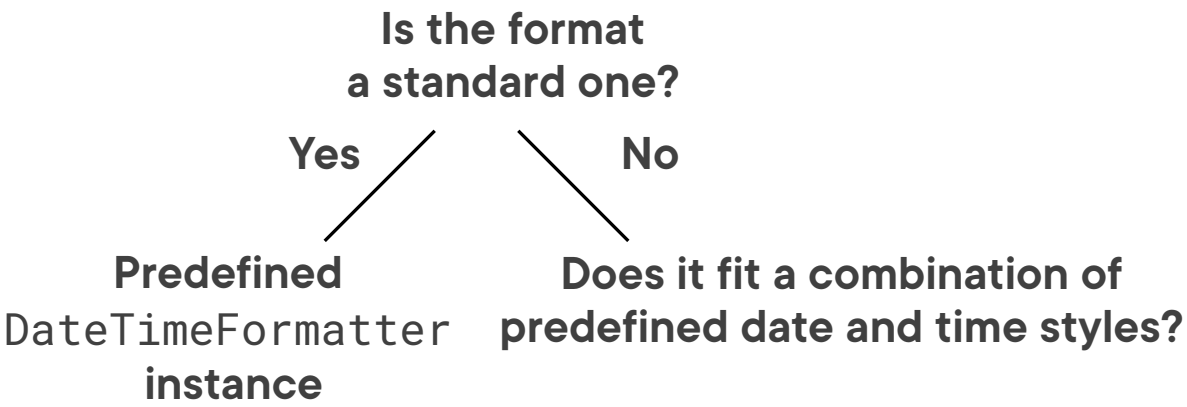
Formatter	Description	Example
BASIC_ISO_DATE	Basic ISO date	'20111203'
ISO_LOCAL_DATE	ISO Local Date	'2011-12-03'
ISO_OFFSET_DATE	ISO Date with offset	'2011-12-03+01:00'
ISO_DATE	ISO Date with or without offset	'2011-12-03+01:00'; '2011-12-03'
ISO_LOCAL_TIME	Time without offset	'10:15:30'

```
jshell> DateTimeFormatter.ISO_LOCAL_DATE.format(docDateTime)
$17 ==> "2011-12-03"

jshell> DateTimeFormatter.ISO_LOCAL_DATE.parse("2011-12-03")
$18 ==> {},ISO resolved to 2011-12-03

jshell> DateTimeFormatter.ISO_LOCAL_DATE.parse("2011-12-03",LocalDate::from)
$19 ==> 2011-12-03
```

Predefined `DateTimeFormatters`

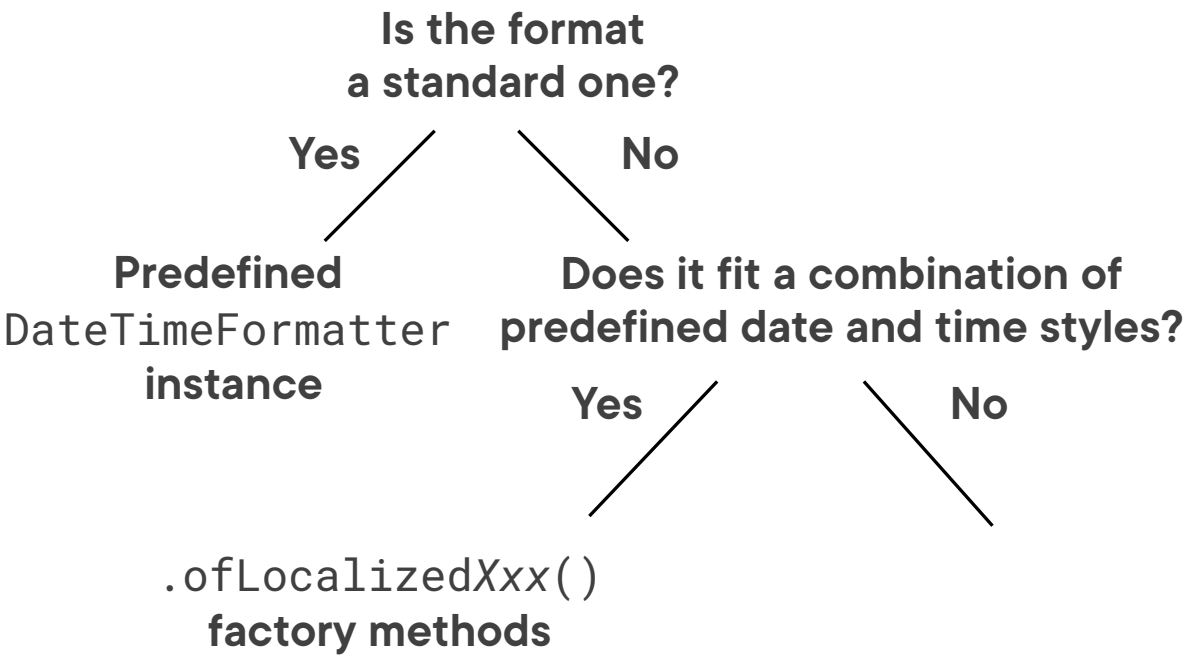
Formatter		Description	Example
BASIC_ISO_DATE		Basic ISO date	'20111203'
ISO_LOCAL_DATE	LocalDate	ISO Local Date	'2011-12-03'
ISO_OFFSET_DATE		ISO Date with offset	'2011-12-03+01:00'
ISO_DATE		ISO Date with or without offset	'2011-12-03+01:00'; '2011-12-03'
ISO_LOCAL_TIME	LocalTime	Time without offset	'10:15:30'
ISO_OFFSET_TIME	OffsetTime	Time with offset	'10:15:30+01:00'
ISO_TIME		Time with or without offset	'10:15:30+01:00'; '10:15:30'
ISO_LOCAL_DATE_TIME	LocalDateTime	ISO Local Date and Time	'2011-12-03T10:15:30'
ISO_OFFSET_DATE_TIME	OffsetDateTime	Date Time with Offset	2011-12-03T10:15:30+01:00'
ISO_ZONED_DATE_TIME	ZonedDateTime	Zoned Date Time	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_DATE_TIME		Date and time with ZoneId	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_ORDINAL_DATE		Year and day of year	'2012-337'
ISO_WEEK_DATE		Year and Week	2012-W48-6'
ISO_INSTANT	Instant	Date and Time of an Instant	'2011-12-03T10:15:30Z'
RFC_1123_DATE_TIME		RFC 1123 / RFC 822	'Tue, 3 Jun 2008 11:05:30 GMT'

How to Create a `DateTimeFormatter`



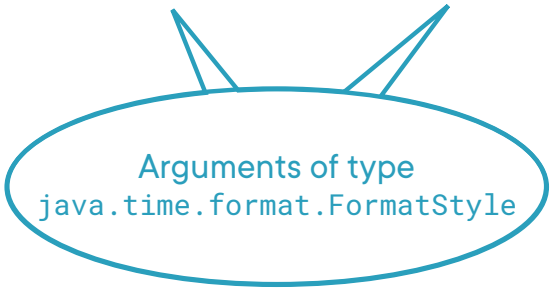
Date	Time
Saturday, 3 December 2011	10:15:30 Central European Standard Time
3 December 2011	10:15:30 CET
3 Dec 2011	10:15:30
03/12/2011	10:15

How to Create a `DateTimeFormatter`



DateTimeFormatter.ofLocalized*Xxx*(...)

```
DateTimeFormatter.ofLocalizedDate(FormatStyle)

DateTimeFormatter.ofLocalizedTime(FormatStyle)

DateTimeFormatter.ofLocalizedDateTime(FormatStyle)

DateTimeFormatter.ofLocalizedDateTime(FormatStyle,FormatStyle)
```



DateTimeFormatter.ofLocalized*Xxx*(...)

	Date	Time
FormatStyle.FULL	Saturday, 3 December 2011	10:15:30 Central European Standard Time
FormatStyle.LONG	3 December 2011	10:15:30 CET
FormatStyle.MEDIUM	3 Dec 2011	10:15:30
FormatStyle.SHORT	03/12/2011	10:15

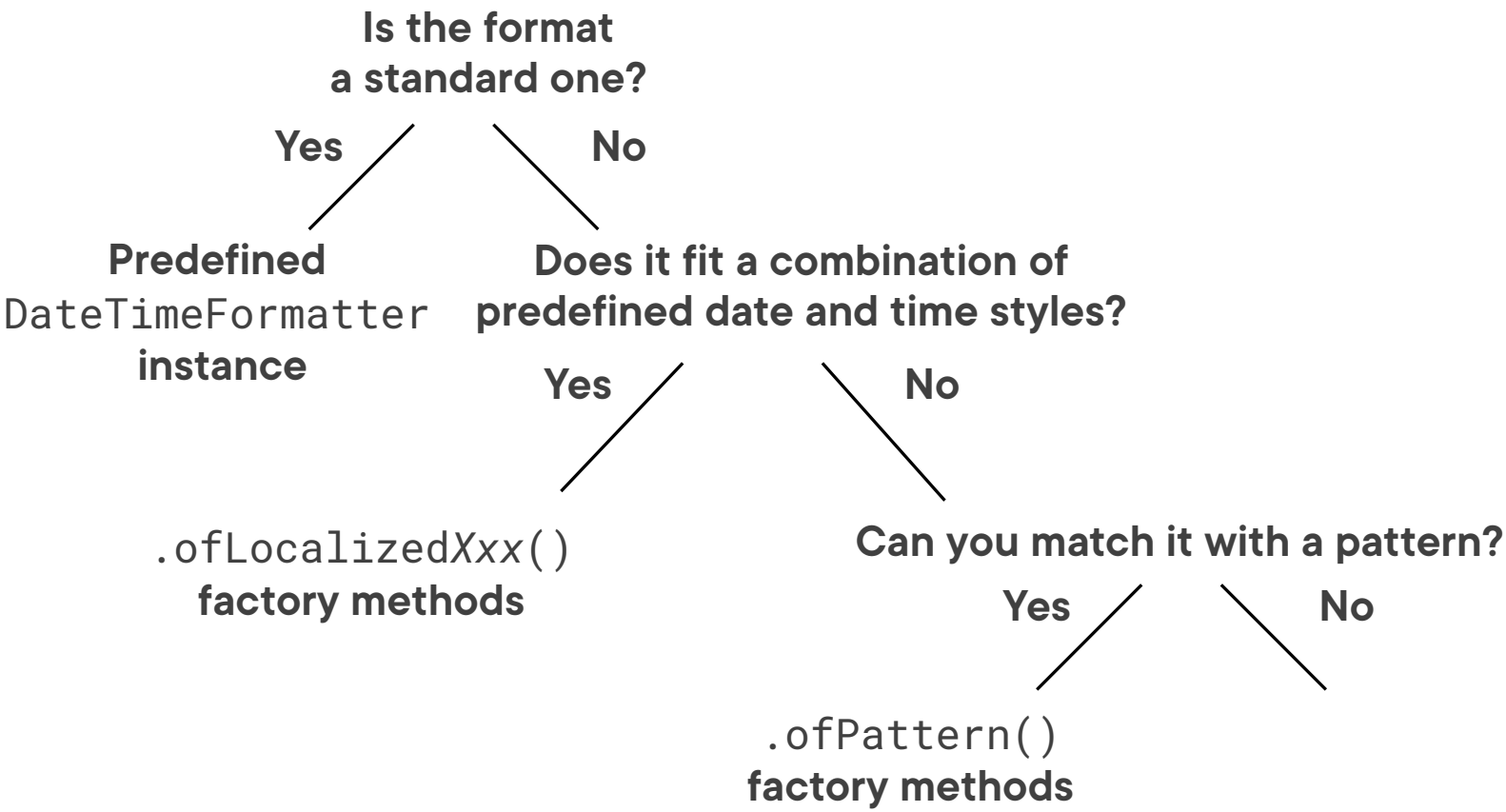
```
jshell> import static java.time.format.FormatStyle.*

jshell> ofLocalizedDate(MEDIUM).format(docDateTime)
$4 ==> "3 Dec 2011"

jshell> DateTimeFormatter.ofLocalizedDateTime(FULL,MEDIUM).format(docDateTime)
$5 ==> "Saturday, 3 December 2011, 10:15:30"
```

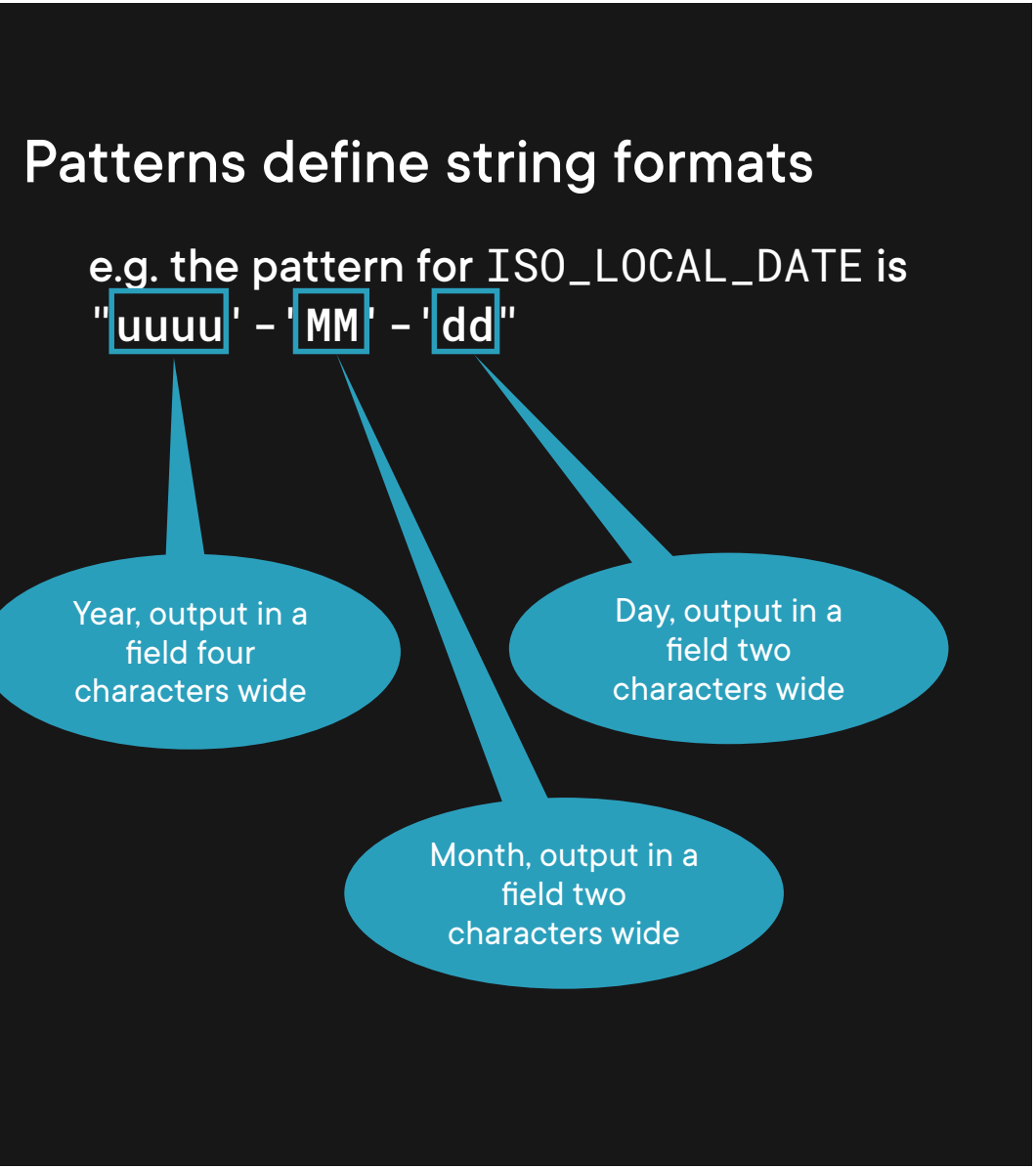
DateTimeFormatter.ofLocalized*Xxx*(...)

	Date	Time
FormatStyle.FULL	Saturday, 3 December 2011	10:15:30 Central European Standard Time
FormatStyle.LONG	3 December 2011	10:15:30 CET
FormatStyle.MEDIUM	3 Dec 2011	10:15:30
FormatStyle.SHORT	03/12/2011	10:15

```
jshell> DateTimeFormatter.ofLocalizedDateTime(FULL,MEDIUM).withLocale(Locale.FRANCE)
$6 ==> Localized(FULL,MEDIUM)

jshell> $6.format(docDateTime)
$7 ==> "samedi 3 décembre 2011 à 10:15:30"
```

How to Create a `DateTimeFormatter`

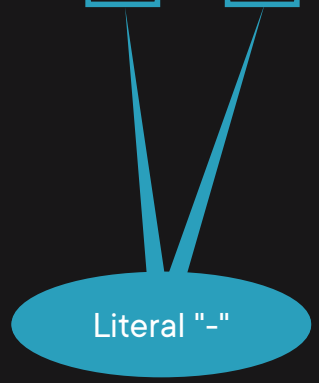




- Add a note here
- Line up text with the corresponding code
- You may need to adjust the amount of spacing “Before Paragraph" under “Spacing”

Patterns define string formats

e.g. the pattern for `ISO_LOCAL_DATE` is
`"uuuu'-'MM'-'dd"`



- Add a note here
- Line up text with the corresponding code
- You may need to adjust the amount of spacing “Before Paragraph" under “Spacing”

Patterns define string formats

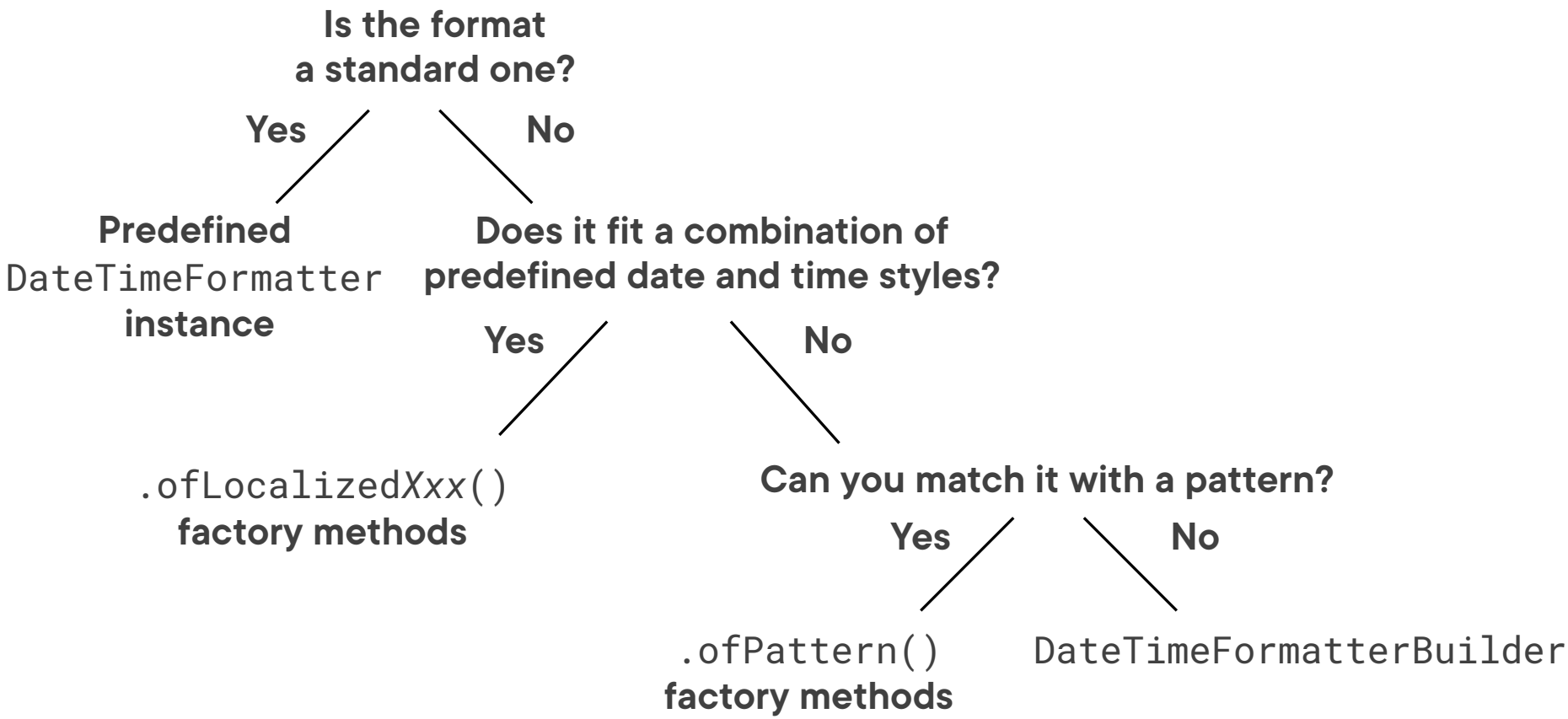
e.g. the pattern for `ISO_LOCAL_DATE` is `"uuuu'-'MM'-'dd"`

A few other pattern symbols:

Symbol	Meaning
K	hour of am/pm
a	am/pm of day
E	day of week
Z	zone offset
[	optional section start
]	optional section end

```
jshell> import static java.time.format.DateTimeFormatt

jshell> ofPattern("E").format(docDateTime)
$3 ==> "Sat"

jshell> ofPattern("EEEE").format(docDateTime)
$4 ==> "Saturday"

jshell> ofPattern("EEEEE").format(docDateTime)
$5 ==> "S"

jshell> ofPattern("HH':'mm").parse("13:06")
$6 ==> {},ISO resolved to 13:06

jshell> ofPattern("HH':'mm[':'ss]").parse("13:06:27")
$7 ==> {},ISO resolved to 13:06:27

jshell> ofPattern("HH':'mm[':'ss]").parse("13:06")
$8 ==> {},ISO resolved to 13:06

jshell> ofPattern("HH':'mm[':'ss[':'nnnnn]]").
   ...> parse("13:06:27:43241")
$9 ==> {},ISO resolved to 13:06:27.000043241
```

How to Create a `DateTimeFormatter`



`DateTimeFormatter` `Builder`

Implementation of the Builder Pattern

Simplifies construction of complex objects

Advanced features

- Default values for fields
- Control case sensitivity of parsing
- Inclusion of other `DateTimeFormatters`

Call `toFormatter()` **on a finished** `Builder`

DateTimeFormatterBuilder

e.g. to parse `LocalDates` in this format:

```
"2011 Dec 03"
```

we could use this formatter:

```
DateTimeFormatter.ofPattern("uuuu' 'MMM' 'dd")
```

To create the equivalent `DateTimeFormatterBuilder` we would write

```
DateTimeFormatterBuilder dtfBuilder = new DateTimeFormatterBuilder()
        .appendValue(YEAR, 4)
        .appendLiteral(" ")
        .appendText(MONTH_OF_YEAR, SHORT)
        .appendLiteral(" ")
        .appendValue(DAY_OF_MONTH, 2);

DateTimeFormatter formatter = dtfBuilder.toFormatter();
```

Other `TextStyle` options:
`NARROW, FULL`

Formatting and Parsing `TemporalAmount`s

The Class `Period`

`toString()` – ISO-8601 format
`parse(String)` – relaxed ISO-8601

For more flexible formatting, can use accessors:

- `getYears()`
- `getMonths()`
- `getDays()`

```
jshell> Period.between(LocalDate.EPOCH, LocalDate.now())
$1 ==> P51Y6M25D

jshell> $1.toString()
$2 ==> "P51Y6M25D"

jshell> Period.parse("P5D")
$3 ==> P5D

jshell> Period.parse("P1Y2M3D")
$4 ==> P1Y2M3D

jshell> Period.parse("P1Y2M3W4D")
$5 ==> P1Y2M25D

jshell> printf("%d years, %d months, %d days",
   ...> $1.getYears(), $1.getMonths(), $1.getDays())
51 years, 6 months, 25 days
```

The Class `Duration`

`toString()` – ISO-8601 format
`parse(String)` – relaxed ISO-8601

In Java 8, the only accessors are

- `getSeconds()`
- `getNano()`

Java 9 provides new methods:

- `toNanosPart()`
- `toMillisPart()`
- `toSecondsPart()`
- `toMinutesPart()`
- `toHoursPart()`
- `toDaysPart()`

```
jshell> Duration.ofSeconds(34567)
$2 ==> PT9H36M7S

jshell> $2.toString()
$3 ==> "PT9H36M7S"

jshell> $2.truncatedTo(ChronoUnit.HOURS)
$4 ==> PT9H

jshell> $2.minus($4).truncatedTo(ChronoUnit.MIN
$5 ==> PT36M

jshell> printf("%d hours, %d minutes %n",
   ...> $4.toHours(), $5.toMinutes())
9 hours, 36 minutes

jshell> printf("%d hours, %d minutes %n",
   ...> $2.toHoursPart(),$2.toMinutesPart())
9 hours, 36 minutes

jshell> DateTimeFormatter.ofPattern("HH:mm:ss")
   ...> format(LocalTime.MIDNIGHT.plus($2))
$8 ==> "09:36:07"
```

Other Java Date-Time Classes

Legacy Date-Time Classes

Legacy Type	`java.time` equivalent	Conversion Methods	
		to `java.time`	from `java.time`
`java.util`			
`java.sql`			
`java.nio.file.attribute.FileTime`			

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date			
	GregorianCalendar			
	SimpleTimeZone			

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	milliseconds since the epoch	seconds and nanos since the epoch		

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	GregorianCalendar	ZonedDateTime	toZonedDateTime() toInstant()	from(ZonedDateTime)

handles historical dates

Gregorian calendar only

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	GregorianCalendar	ZonedDateTime	toZonedDateTime() toInstant()	from(ZonedDateTime)
	SimpleTimeZone	ZoneId	toZoneId()	getTimeZone(ZoneId)

single daylight saving time rule

historical daylight saving time rules

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	GregorianCalendar	ZonedDateTime	toZonedDateTime() toInstant()	from(ZonedDateTime)
	SimpleTimeZone	ZoneId	toZoneId()	getTimeZone(ZoneId)
java.sql	Date	LocalDate	toLocalDate()	valueOf(LocalDate)
	Time	LocalTime	toLocalTime()	valueOf(LocalTime)

milliseconds since the epoch

interpret the java.sql type in the JVM's time zone

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	GregorianCalendar	ZonedDateTime	toZonedDateTime() toInstant()	from(ZonedDateTime)
	SimpleTimeZone	ZoneId	toZoneId()	getTimeZone(ZoneId)
java.sql	Date	LocalDate	toLocalDate()	valueOf(LocalDate)
	Time	LocalTime	toLocalTime()	valueOf(LocalTime)
	Timestamp	Instant	toInstant() toLocalDateTime()	from(Instant)

milliseconds and nanos since the epoch

interprets the absolute time in the JVM's time zone

Legacy Date-Time Classes

Legacy Type		java.time equivalent	Conversion Methods	
			to java.time	from java.time
java.util	Date	Instant	toInstant()	from(Instant)
	GregorianCalendar	ZonedDateTime	toZonedDateTime() toInstant()	from(ZonedDateTime)
	SimpleTimeZone	ZoneId	toZoneId()	getTimeZone(ZoneId)
java.sql	Date	LocalDate	toLocalDate()	valueOf(LocalDate)
	Time	LocalTime	toLocalTime()	valueOf(LocalTime)
	Timestamp	Instant	toInstant() toLocalDateTime()	from(Instant)
java.nio.file.attribute.FileTime		Instant	toInstant()	from(Instant)

FileTime has a greater range than Instant – in theory!

Database Persistence

Database Persistence

JPA 2.1

Use an
`AttributeConverter`

JPA 2.2+

Supported types:

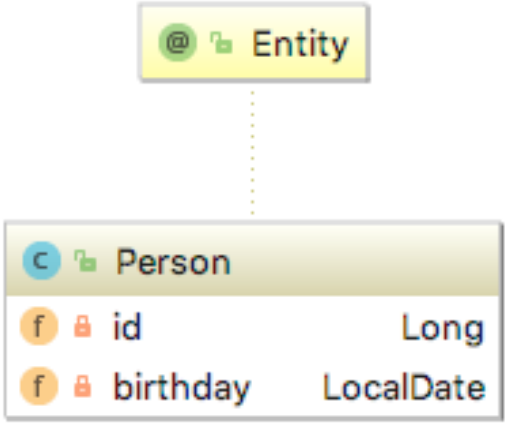
`LocalDate`
`LocalTime`
`LocalDateTime`
`OffsetTime`
`OffsetDateTime`

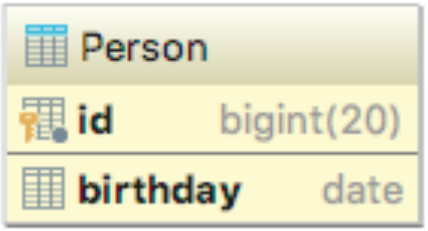
Hibernate 5.3+

Also supported:

`Instant`
`Duration`
`ZonedDateTime`

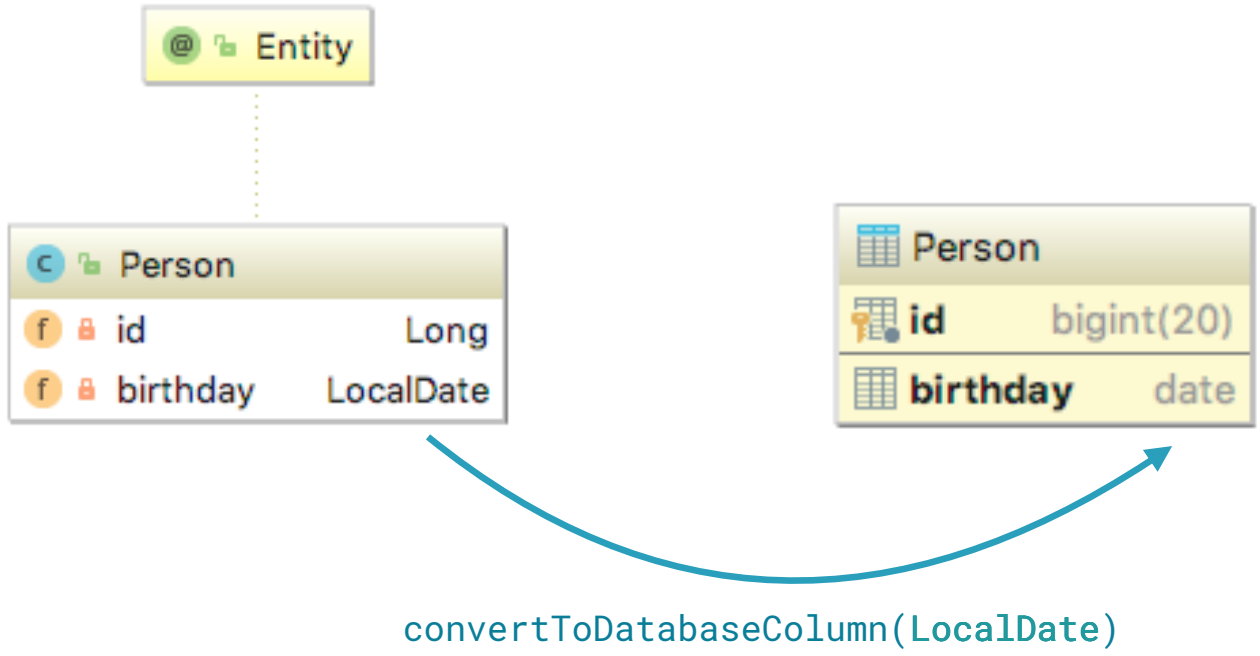
Persistence Without JPA Support





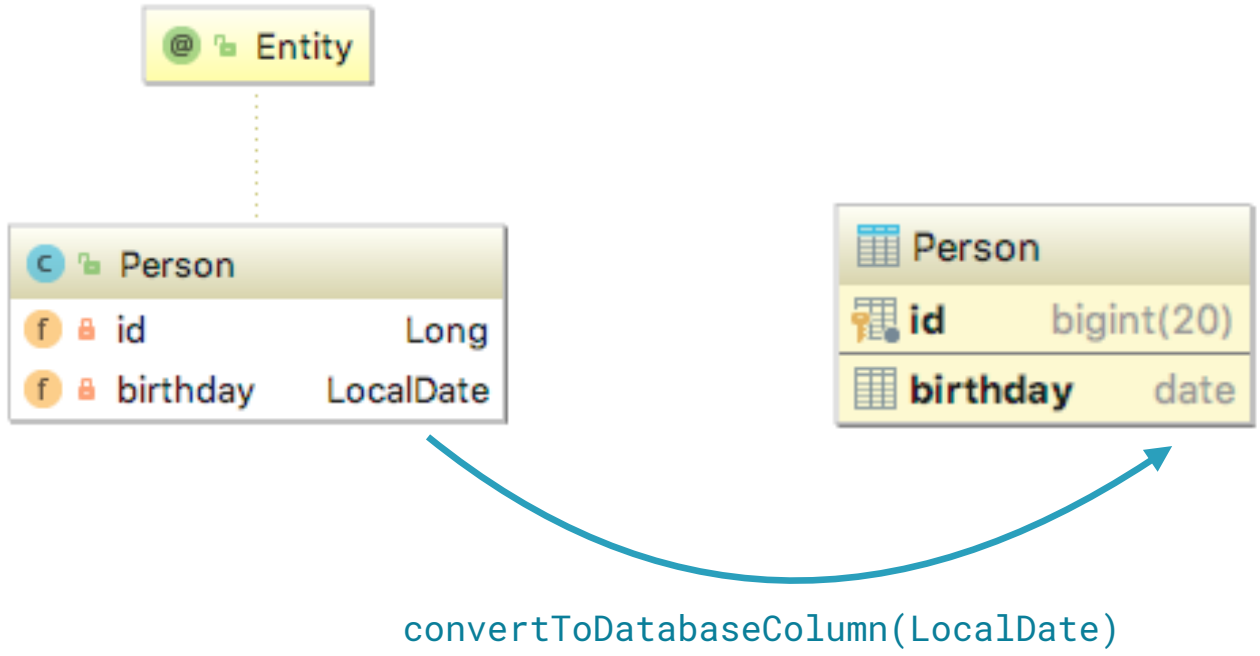

```
interface AttributeConverter<X,Y> {
    public Y convertToDatabaseColumn(X);
    public X convertToEntityAttribute(Y);
}
```

Persistence Without JPA Support



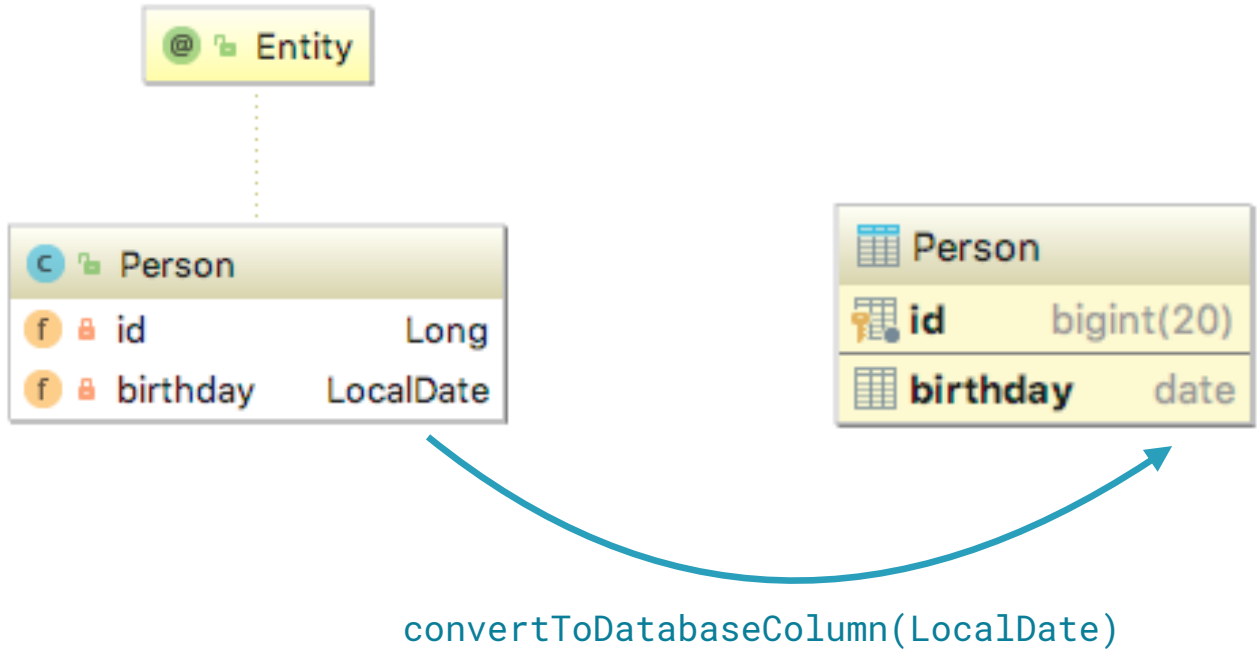

```
interface AttributeConverter<X,Y> {
    public Y convertToDatabaseColumn(X);
    public X convertToEntityAttribute(Y);
}
```

Persistence Without JPA Support



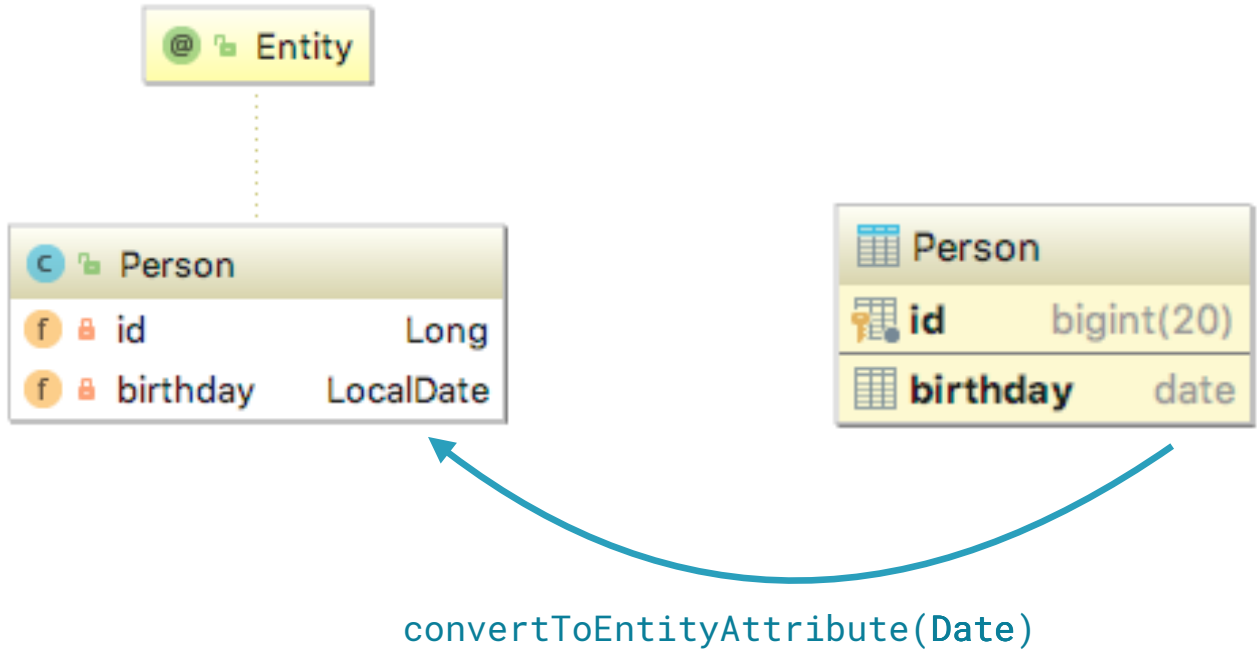

```
interface AttributeConverter<LocalDate,Y> {
    public Y convertToDatabaseColumn(LocalDate);
    public LocalDate convertToEntityAttribute(Y);
}
```

Persistence Without JPA Support



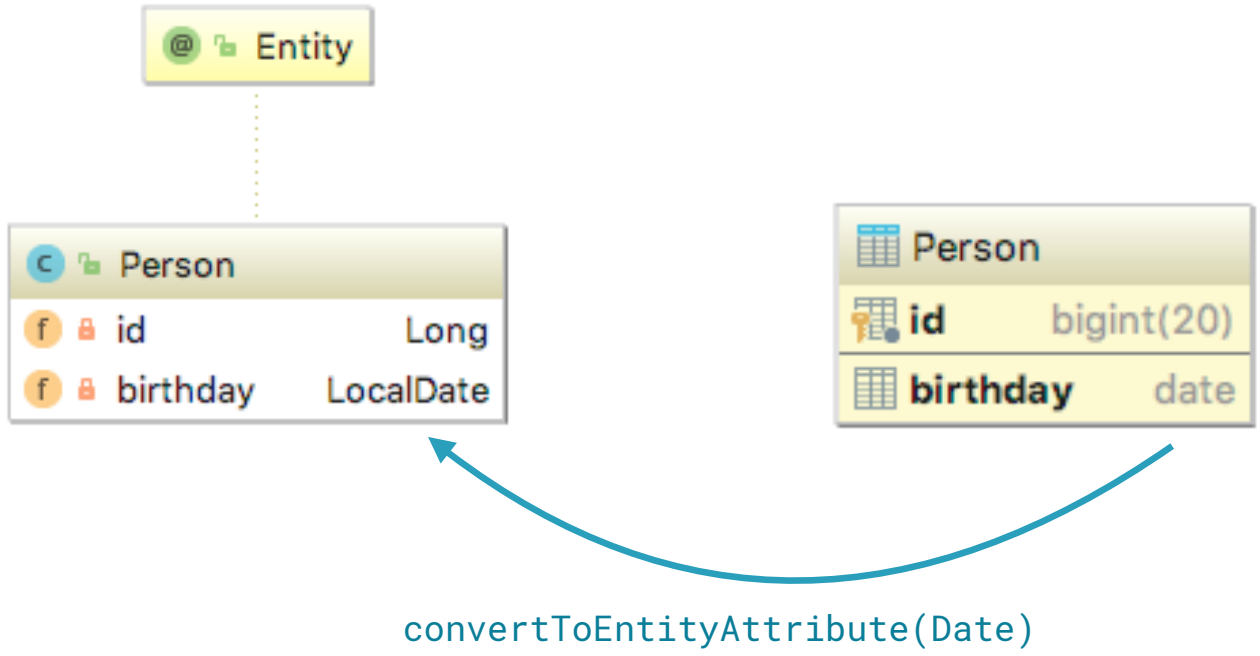

```
interface AttributeConverter<LocalDate,Y> {
    public Y convertToDatabaseColumn(LocalDate);
    public LocalDate convertToEntityAttribute(Y);
}
```

Persistence Without JPA Support




```
interface AttributeConverter<LocalDate,Y> {
    public Y convertToDatabaseColumn(LocalDate);
    public LocalDate convertToEntityAttribute(Y);
}
```

Persistence Without JPA Support




```
interface AttributeConverter<LocalDate,Date> {
    public Date convertToDatabaseColumn(LocalDate);
    public LocalDate convertToEntityAttribute(Date);
}
```

```java
class LocalDateAttributeConverter implements AttributeConverter<LocalDate, Date> {

    @Override
    public Date convertToDatabaseColumn(LocalDate localDate) {
        return (localDate == null ? null : Date.valueOf(localDate));
    }

    @Override
    public LocalDate convertToEntityAttribute(Date sqlDate) {
        return (sqlDate == null ? null : sqlDate.toLocalDate());
    }
}
```

Library of `AttributeConverter` implementations:

https://github.com/marschall/threeten-jpa

JPA 2.2
Jakarta Persistence 3.0+

Supported by
- DataNucleus (v5.1+)
- EclipseLink (v2.7+)
- Hibernate (v5.3+)
- OpenJPA (v3.0.0+)

JAVA TYPE	JDBC TYPE
`java.time.LocalDate`	`DATE`
`java.time.LocalTime`	`TIME`
`java.time.LocalDateTime`	`TIMESTAMP`
`java.time.OffsetTime`	`TIME_WITH_TIMEZONE`
`java.time.OffsetDateTime`	`TIMESTAMP_WITH_TIMEZONE`

Best practice:
use UTC for business logic
and—especially!—for
database persistence

JPA 2.2
Jakarta Persistence 3.0+

Supported by
- DataNucleus (v5.1+)
- EclipseLink (v2.7+)
- Hibernate (v5.3+)
- OpenJPA (v3.0.0+)

Hibernate also supports persistence of `Duration`, `Instant`, and `ZonedDateTime`

JAVA TYPE	JDBC TYPE
`java.time.LocalDate`	`DATE`
`java.time.LocalTime`	`TIME`
`java.time.LocalDateTime`	`TIMESTAMP`
`java.time.OffsetTime`	`TIME_WITH_TIMEZONE`
`java.time.OffsetDateTime`	`TIMESTAMP_WITH_TIMEZONE`
`java.time.Duration`	`BIGINT`
`java.time.Instant`	`TIMESTAMP`
`java.time.ZonedDateTime`	`TIMESTAMP`

Testing

Make your date-time and
time zone dependencies explicit:

Make your date-time and
time zone dependencies explicit:
always use the
`.now(Clock)` and `Clock.getZone()`
methods

Methods of `java.time.Clock`

Factory methods

- Using system time
  - systemDefaultZone()
  - systemUTC, system(ZoneId)
- With different granularity
  - tickSeconds(ZoneId), tickMinutes(ZoneId)
- From an existing Clock
  - tick(Clock, Duration), offset(Clock, Duration)
- Fixed
  - fixed(Instant, ZoneId)

Accessors

- instant()
- millis()
- getZone()

```java
@Test
public void testAllocateOneTaskSuccess() {
    SchedulerCalendar calendar = new SchedulerCalendar();
    LocalDateTime start = LocalDateTime.now();
    calendar.addWorkPeriod(WorkPeriod.of(start, start.plusHours(2)));
    Task task = new Task(120, "");
    calendar.addTask(task);
    Schedule schedule = calendar.createSchedule(start, ZoneId.systemDefault());
    assertTrue(schedule.isSuccessful());
}
```

Testing without a `Clock`

```java
private Clock clock;
@Before
public void setup() {
    clock = Clock.fixed(Instant.EPOCH, ZoneOffset.UTC)
}
@Test
public void testAllocateOneTaskSuccess() {
    SchedulerCalendar calendar = new SchedulerCalendar();
    LocalDateTime start = LocalDateTime.now( );
    calendar.addWorkPeriod(WorkPeriod.of(start, start.plusHours(2)));
    Task task = new Task(120, "");
    calendar.addTask(task);
    Schedule schedule = calendar.createSchedule(start, ZoneId.systemDefault());
    assertTrue(schedule.isSuccessful());
}
```

Adding a `Clock` fixture

```java
private Clock clock;
@Before
public void setup() {
    clock = Clock.fixed(Instant.EPOCH, ZoneOffset.UTC)
}
@Test
public void testAllocateOneTaskSuccess() {
    SchedulerCalendar calendar = new SchedulerCalendar();
    LocalDateTime start = LocalDateTime.now(clock);
    calendar.addWorkPeriod(WorkPeriod.of(start, start.plusHours(2)));
    Task task = new Task(120, "");
    calendar.addTask(task);
    Schedule schedule = calendar.createSchedule(start, clock.getZone());
    assertTrue(schedule.isSuccessful());
}
```

Making time and zone dependencies explicit

ThreeTen Extra

About

ThreeTen-Extra provides additional date-time classes that complement those in Java SE 8.

Not every piece of date/time logic is destined for the JDK. Some concepts are too specialized or too bulky to make it in. This project provides some of those additional classes as a well-tested and reliable jar. It is curated by the primary author of the Java 8 date and time library, Stephen Colebourne.

ThreeTen-Extra is licensed under the business-friendly BSD 3-clause license.

Features

The following features are included:

- `DayOfMonth` - a day-of-month without month or year
- `DayOfYear` - a day-of-year without year
- `AmPm` - before or after midday
- `Quarter` - the four quarters, Q1, Q2, Q3 and Q4
- `YearQuarter` - combines year and quarter, 2014-Q4
- `YearWeek` - a week in a week-based-year, 2014-W06
- `OffsetDate` - combines `LocalDate` and `ZoneOffset`
- `Seconds`, `Minutes`, `Hours`, `Days`, `Weeks`, `Months` and `Years` - amounts of time
- `Interval` - an interval between two instants
- `LocalDateRange` - a range between two dates
- `PeriodDuration` - combines `Period` and `Duration`
- More utilities, such as weekend adjusters
- Extended formatting of periods and durations, including word-based formatting
- Additional calendar systems
- Support for the TAI and UTC time-scales

https://www.threeten.org/threeten-extra/

Demo

Validating a test

- Looking for date-time corner cases

Summary

`TemporalQuery`

Interconversions

- Strings, legacy classes, databases

Unit testing

Demo: testing the methods of the application

Course Windup

@mauricenaftalin
http://mauricenaftal.in
https://github.com/MauriceNaftalin/

Fundamental questions about date-time programming

Basics of the API

Time zones and daylight saving time

Interconversions

Testing java.time programs

Have fun programming In java.time!