Interconversions and Testing

Maurice Naftalin
@mauricenaftalin

Summary

Summary

Summary

Interconversions with Other Representations

Summary

Interconversions with Other Representations

- **Strings – Formatting and Parsing**

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing

String Interconversion

String Interconversion

Date/Time classes expose methods:

String Interconversion

Date/Time classes expose methods:

- `toString()`
- `parse(CharSequence)`

String Interconversion

Date/Time classes expose methods:

- `toString()`
- `parse(CharSequence)`

– these delegate to `DateTimeFormatter` methods:

String Interconversion

Date/Time classes expose methods:

- `toString()`
- `parse(CharSequence)`

– these delegate to `DateTimeFormatter` methods:

- `format(TemporalAccessor)`
- `parse(CharSequence)`

String Interconversion

Date/Time classes expose methods:

- `toString()`
- `parse(CharSequence)`

– these delegate to `DateTimeFormatter` methods:

```
• format(TemporalAccessor)
• parse(CharSequence)
• parse(CharSequence,
          TemporalQuery<T>)
```

Obtaining `DateTimeFormatter` Instances

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

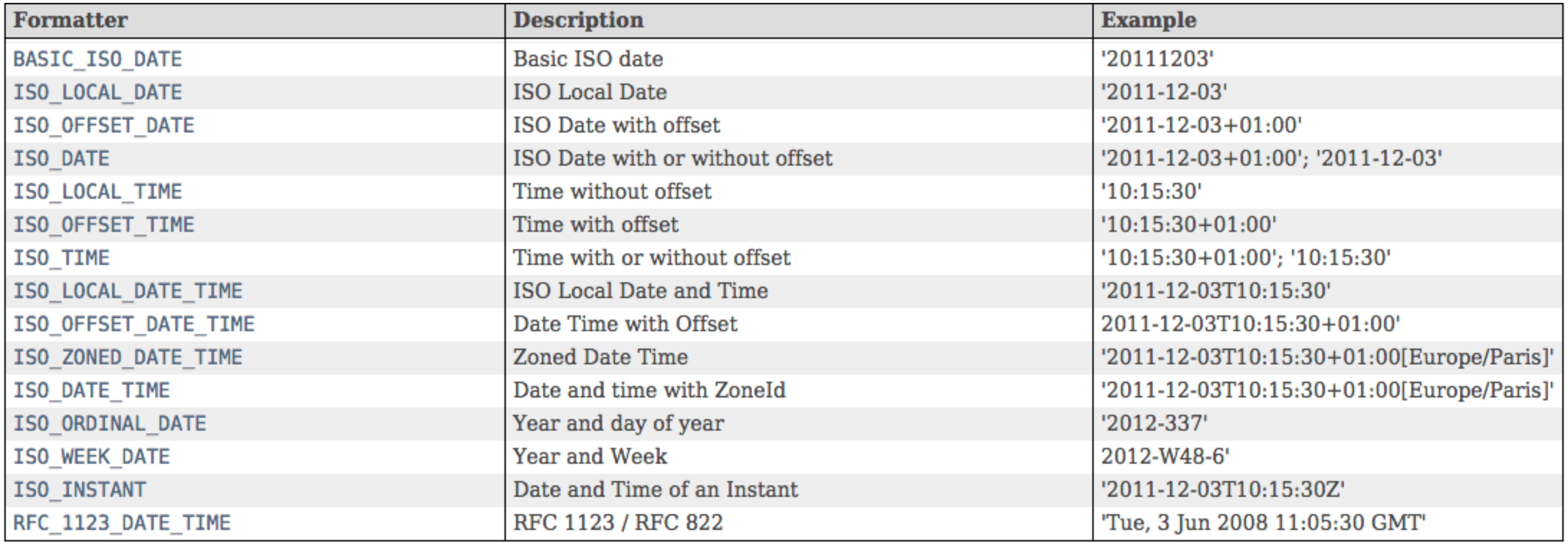
Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

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'
`ISO_OFFSET_TIME`	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`	ISO Local Date and Time	'2011-12-03T10:15:30'
`ISO_OFFSET_DATE_TIME`	Date Time with Offset	2011-12-03T10:15:30+01:00'
`ISO_ZONED_DATE_TIME`	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`	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'

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'

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

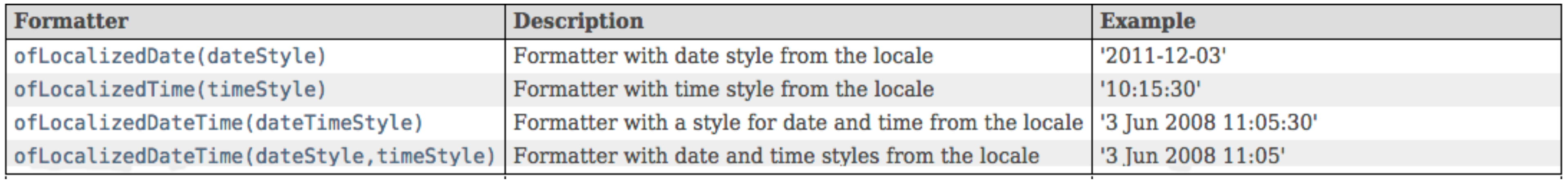
Factory Methods

Using format patterns

DateTimeFormatterBuilder

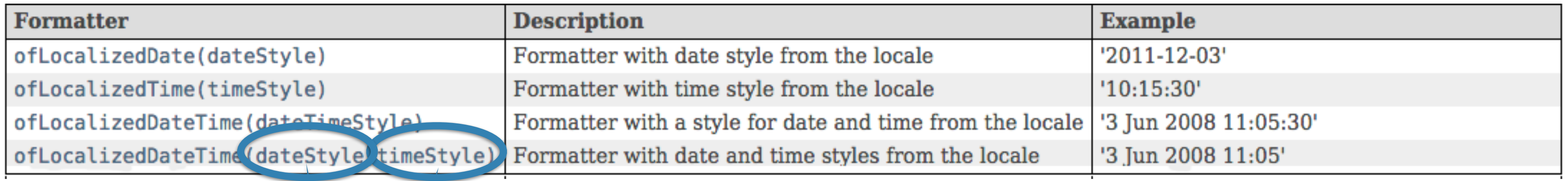
Most flexible way of creating

`DateTimeFormatter` Factory Methods



Formatter	Description	Example
`ofLocalizedDate(dateStyle)`	Formatter with date style from the locale	'2011-12-03'
`ofLocalizedTime(timeStyle)`	Formatter with time style from the locale	'10:15:30'
`ofLocalizedDateTime(dateTimeStyle)`	Formatter with a style for date and time from the locale	'3 Jun 2008 11:05:30'
`ofLocalizedDateTime(dateStyle,timeStyle)`	Formatter with date and time styles from the locale	'3 Jun 2008 11:05'

`DateTimeFormatter` Factory Methods



Formatter	Description	Example
`ofLocalizedDate(dateStyle)`	Formatter with date style from the locale	'2011-12-03'
`ofLocalizedTime(timeStyle)`	Formatter with time style from the locale	'10:15:30'
`ofLocalizedDateTime(dateTimeStyle)`	Formatter with a style for date and time from the locale	'3 Jun 2008 11:05:30'
`ofLocalizedDateTime(dateStyle,timeStyle)`	Formatter with date and time styles from the locale	'3 Jun 2008 11:05'

Arguments of type :
`java.time.format.FormatStyle`

The enum `FormatStyle`

The enum FormatStyle

FormatStyle members:

- FULL
- LONG
- MEDIUM
- SHORT

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

Formatter Patterns

Formatter Patterns

Patterns define string formats

Formatter Patterns

Patterns define string formats

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

Formatter Patterns

Patterns define string formats

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

Year, output in a field at least four characters wide

Formatter Patterns

Patterns define string formats

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

Year, output in a field at least four characters wide

Literal "-"

Formatter Patterns

Patterns define string formats

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

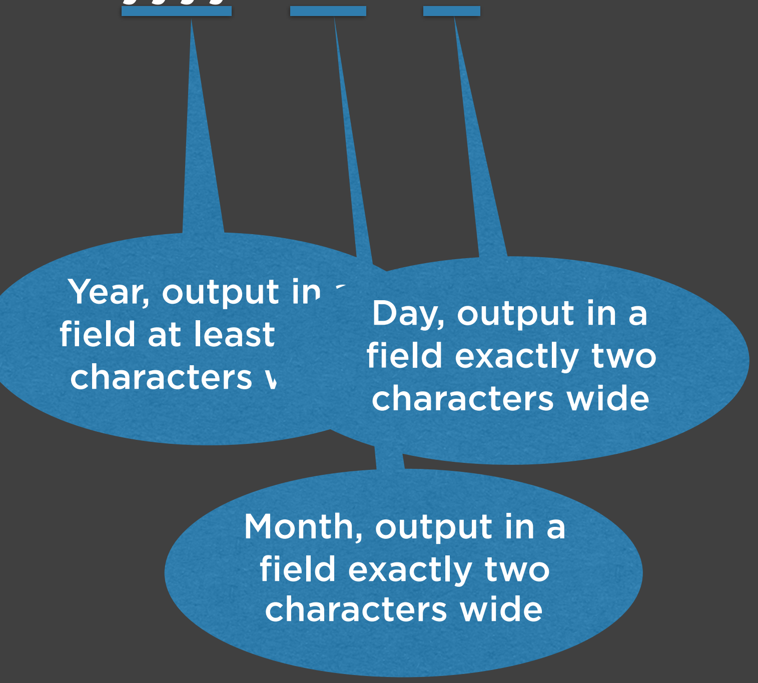
Year, output in a field at least four characters wide

Month, output in a field exactly two characters wide

Formatter Patterns

Patterns define string formats

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


Year, output in
field at least
characters
Day, output in a field exactly two characters wide
Month, output in a field exactly two characters wide

Formatter Patterns

Patterns define string formats

- e.g. the pattern for ISO_LOCAL_DATE is "yyyy'-'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

Formatter Properties

Formatter Properties

Properties of
`java.time.format.DateTimeFormatter`

Formatter Properties

Properties of `java.time.format.DateTimeFormatter`

- Zone
  - Used when a zone is required but not supplied by the parse string or date-time value

Formatter Properties

Properties of `java.time.format.DateTimeFormatter`

- `Zone`
  - Used when a zone is required but not supplied by the parse string or date-time value
- `Locale`
  - Used for localization

Formatter Properties

Properties of `java.time.format.DateTimeFormatter`

- `Zone`
  - Used when a zone is required but not supplied by the parse string or date-time value
- `Locale`
  - Used for localization
- `ResolverStyle`
  - `STRICT`, `LENIENT`, or `SMART`

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

Obtaining `DateTimeFormatter` Instances

Predefined Instances

Factory Methods

Using predefined date and time styles

Factory Methods

Using format patterns

DateTimeFormatterBuilder

Most flexible way of creating

DateTimeFormatter Builder

DateTimeFormatter Builder

- Implementation of the Builder Pattern
  - Simplifies the construction of complex objects

DateTimeFormatter Builder

- Implementation of the Builder Pattern
  - Simplifies the construction of complex objects
- Can set properties:
  - zone
  - locale
  - resolver style

`DateTimeFormatter Builder`

- Implementation of the Builder Pattern
  - Simplifies the construction of complex objects
- Can set properties:
  - zone
  - locale
  - resolver style
- Can append existing `DateTimeFormatters`

`DateTimeFormatter Builder`

- Implementation of the Builder Pattern
  - Simplifies the construction of complex objects
- Can set properties:
  - zone
  - locale
  - resolver style
- Can append existing `DateTimeFormatters`
- Once building is complete, calling `toFormatter()` creates a `DateTimeFormatter`

DateTimeFormatterBuilder

`DateTimeFormatterBuilder`

e.g. to parse `LocalDate`s in this format:

"2018 Aug 23"

DateTimeFormatterBuilder

e.g. to parse `LocalDate`s in this format:

"2018 Aug 23"

we could use a formatter:

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

`DateTimeFormatterBuilder`

e.g. to parse `LocalDate`s in this format:

"2018 Aug 23"

we could use a formatter:

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

To create the equivalent `DateTimeFormatterBuilder` we would write

DateTimeFormatterBuilder

e.g. to parse `LocalDate`**s in this format:**

"2018 Aug 23"

we could use a formatter:

```
DateTimeFormatter.ofPattern("yyyy' '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();
```

DateTimeFormatterBuilder

e.g. to parse `LocalDate`**s in this format:**

"2018 Aug 23"

we could use a formatter:

```
DateTimeFormatter.ofPattern("yyyy' '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

TemporalAmount classes

The Class Period

The Class Period

`toString()` - ISO-8601 format

TemporalAmount classes

The Class Period

`toString()`	- ISO-8601 format
`parse(Period)`	- relaxed ISO-8601

TemporalAmount classes

The Class Period

toString()	- ISO-8601 format
parse(Period)	- relaxed ISO-8601

For more flexible formatting, can use accessors

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

`TemporalAmount` classes

The Class Duration

The Class Duration

`toString()`	- fixed format
`parse(String)`	- fixed format

The Class `Duration`

`toString()`	- fixed format
`parse(String)`	- fixed format

In Java 8, the only accessors are

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

`TemporalAmount` classes

TemporalAmount classes

The Class Duration

`toString()`	– fixed format
`parse(String)`	– fixed format

In Java 8, the only accessors are

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

Java 9 provides new methods:

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

Interconversions and Testing

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing

Demo: Testing the Methods of the Task Scheduler

Legacy Date/Time Classes

Legacy Date/Time Classes

Legacy Date/Time Classes

Legacy Type	`java.time` Equivalent	Conversion Methods	

Legacy Date/Time Classes

Legacy Type	java.time Equivalent	Conversion Methods	
java.util.Date	Instant	toInstant()	from(Instant)

Legacy Date/Time Classes

Legacy Type	java.time Equivalent	Conversion Methods	
java.util.Date	Instant	toInstant()	from(Instant)

java.util.GregorianCalendar	ZonedDateTime	toInstant()
		toZonedDateTime()
			from(ZonedDateTime)

Legacy Date/Time Classes

Legacy Type	`java.time` Equivalent	Conversion Methods	
`java.util.Date`	`Instant`	`toInstant()`	`from(Instant)`
`java.util.GregorianCalendar`	`ZonedDateTime`	`toInstant()` `toZonedDateTime()`	`from(ZonedDateTime)`
`java.util.TimeZone`	`ZoneId`	`toZoneId()`	`getTimeZone(ZoneId)`

Legacy Date/Time Classes

Legacy Type	`java.time` **Equivalent**	**Conversion Methods**	
`java.util.Date`	`Instant`	`toInstant()`	`from(Instant)`
`java.util.GregorianCalendar`	`ZonedDateTime`	`toInstant()` `toZonedDateTime()`	`from(ZonedDateTime)`
`java.util.TimeZone`	`ZoneId`	`toZoneId()`	`getTimeZone(ZoneId)`
`java.sql.Date`	`LocalDate`	`toLocalDate()`	`valueOf(LocalDate)`

Legacy Date/Time Classes

Legacy Type	java.time Equivalent	Conversion Methods	
java.util.Date	Instant	toInstant()	from(Instant)

java.util.GregorianCalendar	ZonedDateTime	toInstant()
		toZonedDateTime()
			from(ZonedDateTime)

java.util.TimeZone	ZoneId	toZoneId()	getTimeZone(ZoneId)
java.sql.Date	LocalDate	toLocalDate()	valueOf(LocalDate)
java.sql.Time	LocalTime	toLocalTime()	valueOf(LocalTime)

Legacy Date/Time Classes

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

Legacy Date/Time Classes

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

Interconversions and Testing

Interconversions with Other Representations

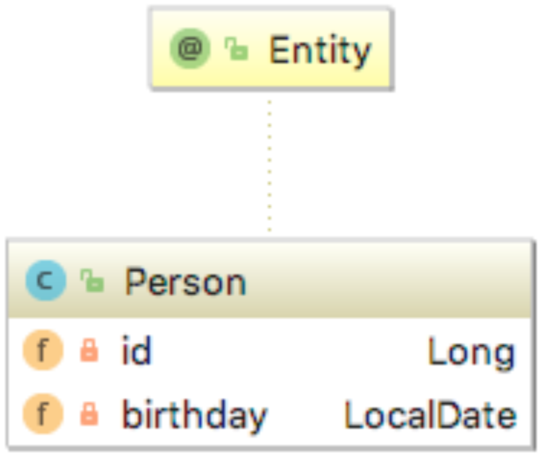
- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing

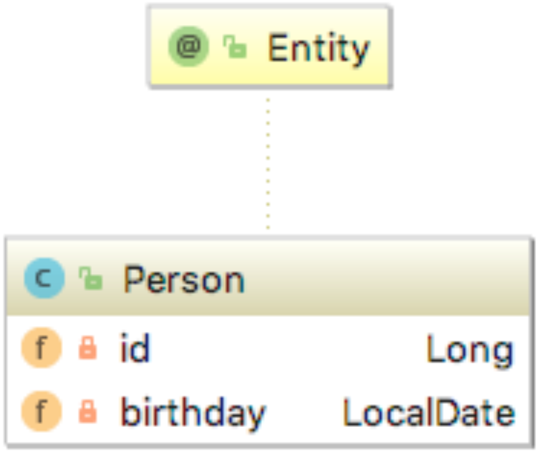
Demo: Testing the Methods of the Task Scheduler

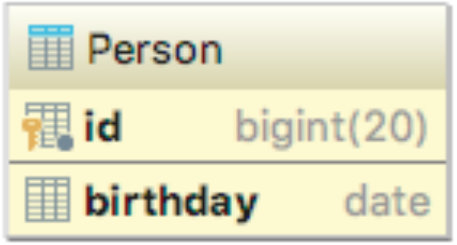
Persistence Without JPA Support

Persistence Without JPA Support

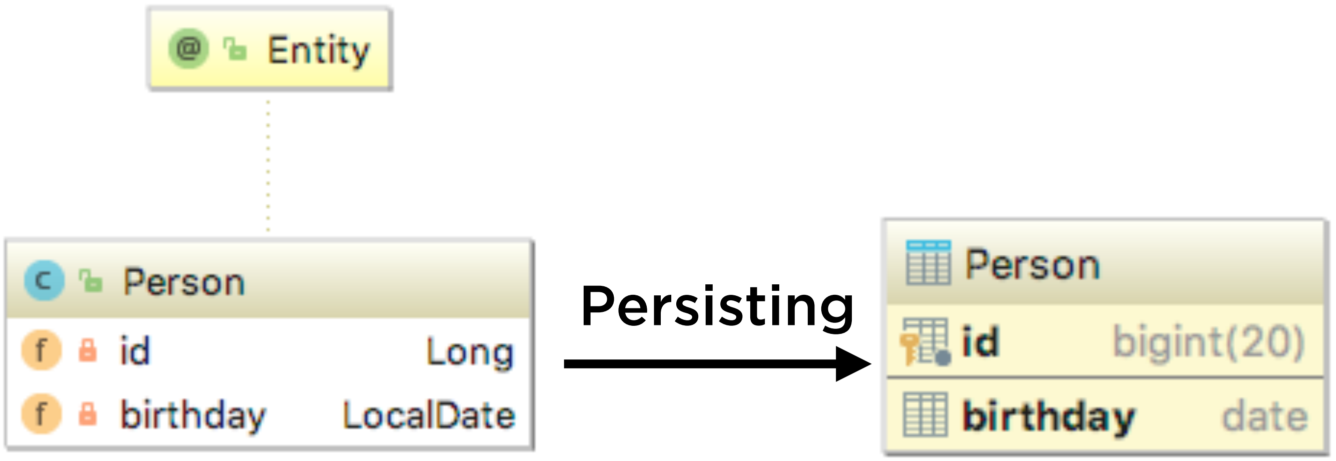


Persistence Without JPA Support

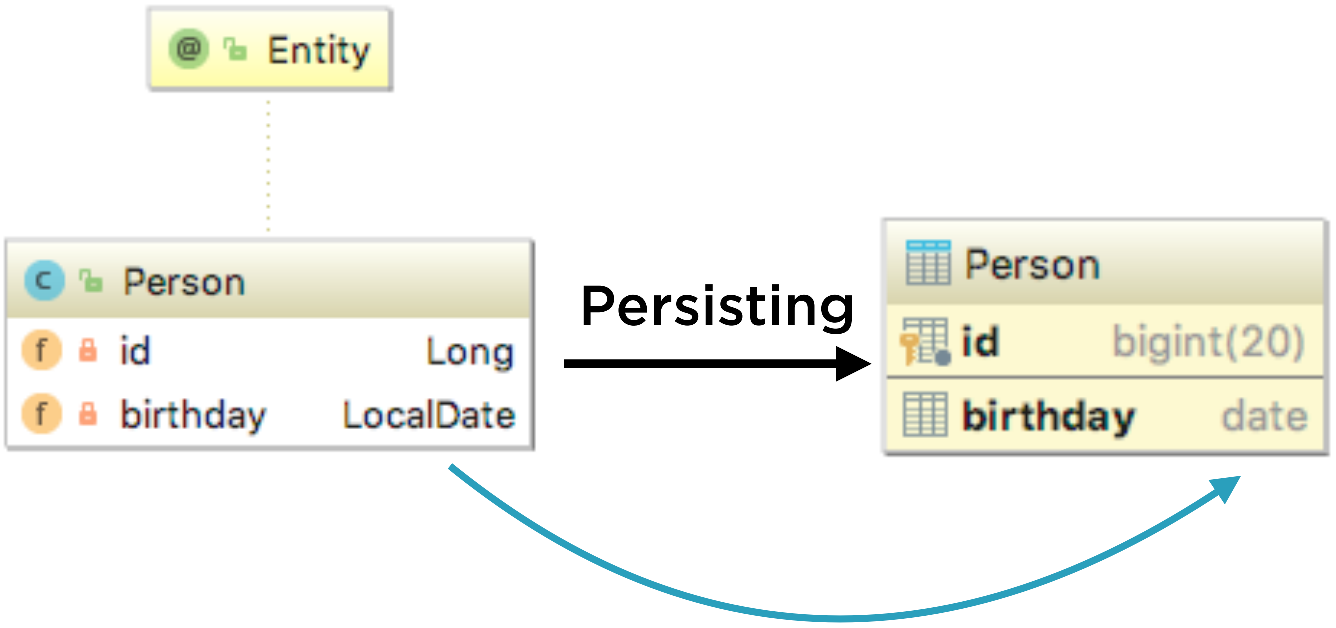




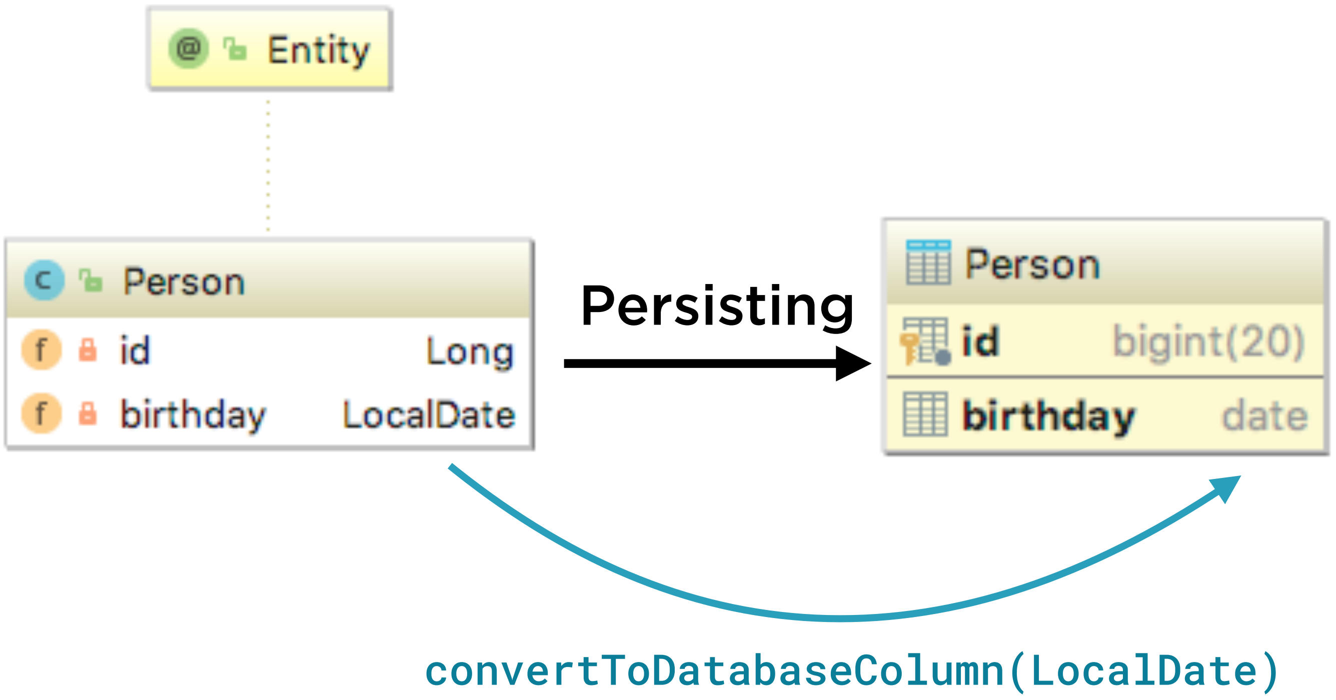
Persistence Without JPA Support



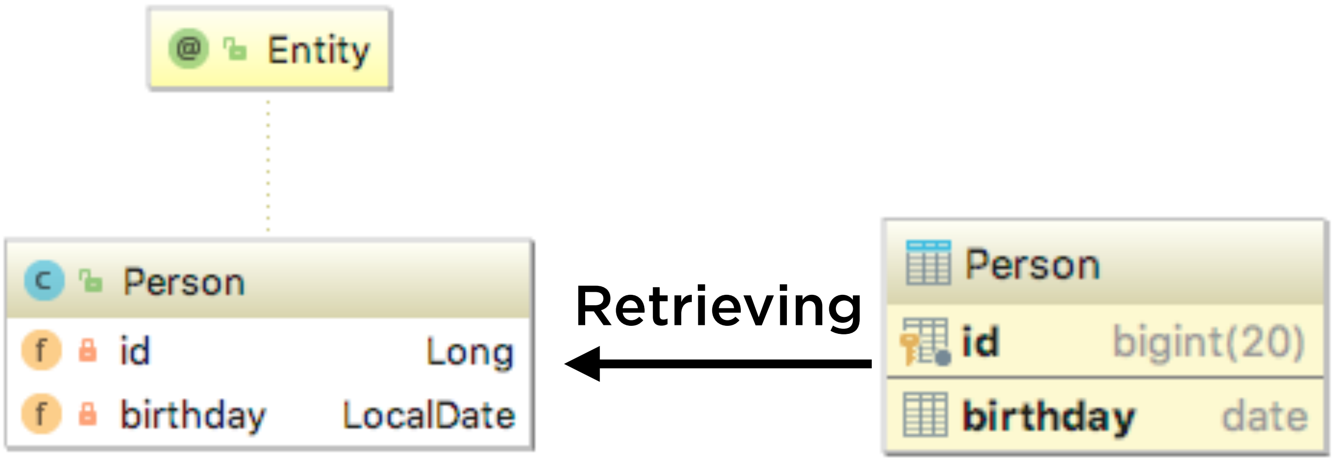
Persistence Without JPA Support



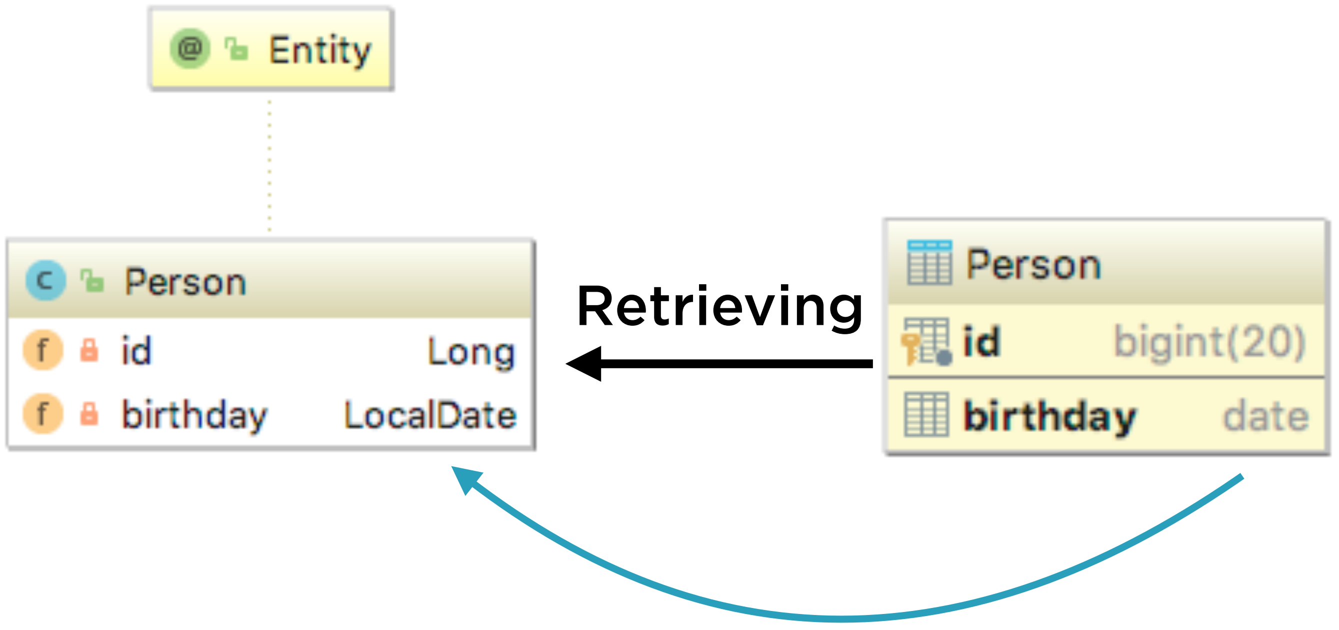
Persistence Without JPA Support



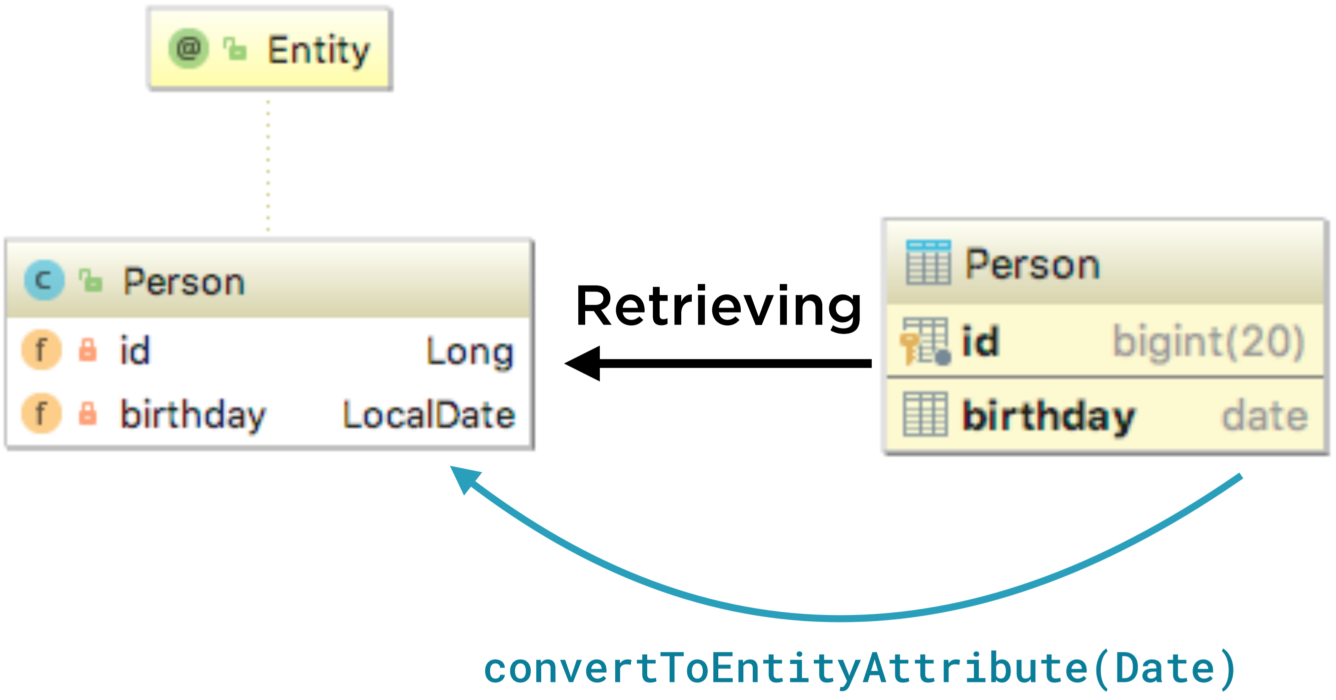
Persistence Without JPA Support



Persistence Without JPA Support



Persistence Without JPA Support



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

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

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

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

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

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

```
public 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());
    }
}
```

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

```java
@Converter(autoApply = true)
public 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());
    }
}
```

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

```java
@Converter(autoApply = true)
public 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());
    }
}
```

Three conversion libraries:

- **https://github.com/perceptron8/datetime-jpa**
- **https://github.com/marschall/threeten-jpa**
- **https://bitbucket.org/montanajava/jpaattributeconverters**

With JPA Support

JPA 2.2

With JPA Support

With JPA Support

JPA 2.2

Supported by

- DataNucleus
- EclipseLink (v2.7+)
- Hibernate (v5.3+)

With JPA Support

JPA 2.2

Supported by
- DataNucleus
- EclipseLink (v2.7+)
- Hibernate (v5.3+)

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

With JPA Support

JPA 2.2

Supported by
- **DataNucleus**
- **EclipseLink (v2.7+)**
- **Hibernate (v5.3+)**

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

With JPA Support

JPA 2.2

Supported by
- **DataNucleus**
- **EclipseLink (v2.7+)**
- **Hibernate (v5.3+)**

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

Interconversions and Testing

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing

Demo: Testing the Methods of the Task Scheduler

Methods of the Class `Clock`

Methods of the Class `Clock`

Factory Methods

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
```

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
```

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
fixed(Instant, ZoneId)
```

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
fixed(Instant, ZoneId)
```

Accessors

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
fixed(Instant, ZoneId)
```

Accessors

```
instant()
```

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
fixed(Instant, ZoneId)
```

Accessors

```
instant()
millis()
```

Methods of the Class `Clock`

Factory Methods

```
systemDefaultZone()
systemUTC()
system(ZoneId)
tickSeconds(ZoneId)
tickMinutes(ZoneId)
fixed(Instant, ZoneId)
```

Accessors

```
instant()
millis()
getZone()
```

```
private Calendar cal;
private ZonedDateTime start;
@Before
public void setup() {
   cal = new Calendar();
   start = ZonedDateTime.now()
}
@Test
public void testNoWorkPeriods() {
    cal.addEvent(Event.of(start, start.plusHours(1),""));
    NavigableSet<WorkPeriod> combined = cal.overwritePeriodsByEvents(ZoneId.systemDefault());
    assertTrue(combined.isEmpty());
}
```

Testing without a `Clock`

```
private Calendar cal;
private ZonedDateTime start;
@Before
public void setup() {
   cal = new Calendar();
   start = ZonedDateTime.now()
}
@Test
public void testNoWorkPeriods() {
    cal.addEvent(Event.of(start, start.plusHours(1),""));
    NavigableSet<WorkPeriod> combined = cal.overwritePeriodsByEvents(ZoneId.systemDefault());
    assertTrue(combined.isEmpty());
}
```

Testing without a `Clock`

```java
private Clock clock;
private Calendar cal;
private ZonedDateTime start;
@Before
public void setup() {
   cal = new Calendar();
   clock = Clock.fixed(Instant.EPOCH, ZoneOffset.UTC);
   start = ZonedDateTime.now(clock);
}
@Test
public void testNoWorkPeriods() {
    cal.addEvent(Event.of(start, start.plusHours(1),""));
    NavigableSet<WorkPeriod> combined = cal.overwritePeriodsByEvents(clock.getZone());
    assertTrue(combined.isEmpty());
}
```

Testing with a Clock

Using a Mocking Framework to Simulate Changed Time in a Method

```
@Test
public void myTest() {
    testObject.methodThatAcceptsAnInstant(clock.instant());

    testObject.methodThatAcceptsAnInstant(clock.instant());
}
```

Using a Mocking Framework to Simulate Changed Time in a Method

```
private Instant currentTime;





@Test
public void myTest() {
    testObject.methodThatAcceptsAnInstant(clock.instant());
    currentTime = currentTime.plus(1, ChronoUnit.DAYS);     // simulate passage of time
    testObject.methodThatAcceptsAnInstant(clock.instant());
}
```

Using a Mocking Framework to Simulate Changed Time in a Method

```
private Instant currentTime;
private Clock clock = Mockito.mock(Clock.class);
@Before
public void setup() {
   currentTime = Instant.EPOCH;     // or other initialisation
   when(clock.instant()).thenAnswer(invocation -> currentTime);
}
@Test
public void myTest() {
    testObject.methodThatAcceptsAnInstant(clock.instant());
    currentTime = currentTime.plus(1, ChronoUnit.DAYS);     // simulate passage of time
    testObject.methodThatAcceptsAnInstant(clock.instant());
}
```

Using a Mocking Framework to Simulate Changed Time in a Method

Interconversions and Testing

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing

Demo: Testing the Methods of the Task Scheduler

Summary

Summary

Summary

Interconversions with Other Representations

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Summary

Interconversions with Other Representations

- Strings – Formatting and Parsing
- Other JDK Date/Time Classes
- Database Persistence

Unit Testing