Working with Time Zones

Maurice Naftalin
@mauricenaftalin

Summary

Summary

Summary

Overview of Time Zones

Summary

Overview of Time Zones

Working with the Time Zone Classes

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset` **and** `ZoneId`

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset` and `ZoneId`
- `OffsetDateTime` and `ZonedDateTime`

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset` and `ZoneId`
- `OffsetDateTime` and `ZonedDateTime`

Temporal Adjusters

Core `java.time` Time Zone Classes

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

The Class `ZoneOffset`

The Class `ZoneOffset`

A `ZoneOffset` is the amount that a time zone differs from standard time
e.g. UTC+05:30, UTC-04:00

The Class `ZoneOffset`

A `ZoneOffset` is the amount that a time zone differs from standard time
e.g. UTC+05:30, UTC-04:00

Standard time is *UTC*, an atomic timescale corrected to solar time by occasional adjustments (*leap seconds*)

String Interconversion

The Class `ZoneOffset`

A `ZoneOffset` is the amount that a time zone differs from standard time
e.g. UTC+05:30, UTC-04:00

The Class `ZoneOffset`

A `ZoneOffset` is the amount that a time zone differs from standard time e.g. UTC+05:30, UTC-04:00

String Interconversion

`of(String)`

The Class `ZoneOffset`

A `ZoneOffset` is the amount that a time zone differs from standard time e.g. UTC+05:30, UTC-04:00

String Interconversion

```
of(String)

getId()
toString()
```

The Class `ZoneOffset`

The Class `ZoneOffset`

Creation Methods

The Class `ZoneOffset`

Creation Methods

```
ofHours(int)
ofHoursMinutes(int,int)
ofHoursMinutesSeconds(int,int,int)
ofTotalSeconds(int)
```

The Class `ZoneOffset`

Creation Methods

```
ofHours(int)
ofHoursMinutes(int,int)
ofHoursMinutesSeconds(int,int,int)
ofTotalSeconds(int)

from(TemporalAccessor)
```

The Class `ZoneOffset`

Field Access

The Class `ZoneOffset`

The Class `ZoneOffset`

Field Access

```
getHour()
getMinute()
getSecond()
getNano()

get(TemporalField)
getLong(TemporalField)
```

The Class ZoneOffset

Field Access

```
getHour()
getMinute()
getSecond()
getNano()

get(TemporalField)
getLong(TemporalField)

query(TemporalQuery<R>)
```

Getting Information from `TemporalAccessor` Objects

Getting Information from `TemporalAccessor` Objects

Static

`from`

methods

Getting Information from `TemporalAccessor` Objects

Static
`from`
methods

Methods to query an individual field
`getXxx()`

Getting Information from `TemporalAccessor` Objects

Static

`from`

methods

Methods to query an individual field

`getXxx()`

`get(TemporalField)`
`getLong(TemporalField)`

Getting Information from `TemporalAccessor` Objects

Static

`from`

methods

Methods to query an individual field

`getXxx()`

`get(TemporalField)`
`getLong(TemporalField)`

The method

`query`

Using the Method `query(…)`

Using the Method `query(…)`

`TemporalAccessor.query(...)`
accepts `TemporalQuery<R>`
objects

Using the Method `query(…)`

`TemporalAccessor.query(...)`
accepts `TemporalQuery<R>`
objects

strategies for extracting information from `TemporalAccessor` **objects**

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

strategies for extracting information from `TemporalAccessor` objects

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

The single abstract method of `TemporalQuery<R>` **is declared as**

```
R queryFrom(TemporalAccessor)
```

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

The single abstract method of `TemporalQuery<R>` **is declared as**

```
R queryFrom(TemporalAccessor)
```

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

The single abstract method of `TemporalQuery<R>` **is declared as**

```
R queryFrom(TemporalAccessor)
```

the lambda or method reference used as an instance must match it

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

The single abstract method of `TemporalQuery<R>` **is declared as**

```
R queryFrom(TemporalAccessor)
```

the lambda or method reference used as an instance must match it
e.g.

```
(TemporalAccessor ta) -> LocalDate.from(ta)
```

Using the Method `query(…)`

`TemporalAccessor.query(...)` **accepts** `TemporalQuery<R>` **objects**

`TemporalQuery<R>` **is a functional interface, so a lambda or method reference can be supplied as an instance**

The single abstract method of `TemporalQuery<R>` **is declared as**

```
R queryFrom(TemporalAccessor)
```

the lambda or method reference used as an instance must match it
e.g.

```
(TemporalAccessor ta) -> LocalDate.from(ta)
```

The Class `TemporalQueries`

The Class `TemporalQueries`

```
TemporalQuery<Chronology> chronology()
```

The Class `TemporalQueries`

```
TemporalQuery<Chronology> chronology()
TemporalQuery<LocalDate> localDate()
TemporalQuery<LocalTime> localTime()
TemporalQuery<ZoneOffset> offset()
```

The Class `TemporalQueries`

```
TemporalQuery<Chronology> chronology()
TemporalQuery<LocalDate> localDate()
TemporalQuery<LocalTime> localTime()
TemporalQuery<ZoneOffset> offset()
TemporalQuery<TemporalUnit> precision()
```

The Class `TemporalQueries`

```
TemporalQuery<Chronology> chronology()
TemporalQuery<LocalDate> localDate()
TemporalQuery<LocalTime> localTime()
TemporalQuery<ZoneOffset> offset()
TemporalQuery<TemporalUnit> precision()
TemporalQuery<ZoneId> zone()
TemporalQuery<ZoneId> zoneId()
```

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

The Class `ZoneId`: Different Forms

The Class `ZoneId`: Different Forms

Same as ZoneOffset

e.g. "Z", "+03:00"

The Class `ZoneId`: Different Forms

Same as ZoneOffset

e.g. "Z", "+03:00"

Offset-style with prefix

prefix is: UTC, UT or GMT

e.g. "GMT+2", "UTC+01:00"

The Class `ZoneId`: Different Forms

Same as ZoneOffset

e.g. "Z", "+03:00"

Offset-style with prefix

prefix is: UTC, UT or GMT

e.g. "GMT+2", "UTC+01:00"

Region-based

e.g.

"Europe/London"
"America/New_York"

The Class `ZoneId`

The Class ZoneId

```
getAvailableZoneIds()
```

- Returns available region-based IDs

The Class ZoneId

```
getAvailableZoneIds()

systemDefault()
```

- Returns available region-based IDs
- **Returns the platform time zone**

The Class `ZoneId`

```
getAvailableZoneIds()

systemDefault()

getRules()
```

- Returns available region-based IDs
- Returns the platform time zone
- **Returns a `ZoneRules` object**

The Class `ZoneId`

`getAvailableZoneIds()`	Returns available region-based IDs
`systemDefault()`	Returns the platform time zone
`getRules()`	Returns a `ZoneRules` object
`normalized()`	**Converts this `ZoneId` to a `ZoneOffset`**

The Class ZoneId

```
getAvailableZoneIds()
```

- Returns available region-based IDs

```
systemDefault()
```

- Returns the platform time zone

```
getRules()
```

- Returns a `ZoneRules` object

```
normalized()
```

- Converts this `ZoneId` to a `ZoneOffset`

```
of(String)
ofOffset(String,ZoneOffset)
of(String,Map<String,String>)
```

- **Factory methods**

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

The Class `OffsetDateTime`

Adjustment Methods

Adjustment Methods

`truncatedTo(TemporalUnit)`

The Class `OffsetDateTime`

Adjustment Methods

`truncatedTo(TemporalUnit)`

`plus...``(int)`
`minus...(int)`
`with...``(int)`

The Class `OffsetDateTime`

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int)
minus...(int)
with...(int)
withOffsetSameInstant(ZoneOffset)
withOffsetSameLocal(ZoneOffset)
```

The Class `OffsetDateTime`

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int)
minus...(int)
with...(int)
withOffsetSameInstant(ZoneOffset)
withOffsetSameLocal(ZoneOffset)

plus/minus(long,TemporalUnit)
plus/minus(TemporalAmount)

with(TemporalField,long)
with(TemporalAdjuster)
```

The Interface `Temporal`

The Interface `Temporal`

Subinterface of `TemporalAccessor`

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**

  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

`Temporal` - adjustment methods
`plus/minus``(long,`**`TemporalUnit`**`),`

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

Temporal - adjustment methods
plus/minus(long,**TemporalUnit),**
plus/minus(TemporalAmount)

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

Temporal - adjustment methods
plus/minus(long,**TemporalUnit**),
plus/minus(TemporalAmount)
with(TemporalField,long)

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

Temporal - adjustment methods
plus/minus(long,**TemporalUnit**),
plus/minus(TemporalAmount)
with(TemporalField,long)
with(TemporalAdjuster)

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

Temporal – adjustment methods
plus/minus(long,TemporalUnit),
plus/minus(TemporalAmount)
with(TemporalField,long)
with(TemporalAdjuster)

Temporal – other methods
isSupported(TemporalUnit)

The Interface `Temporal`

Subinterface of `TemporalAccessor`

- **so inherits** `TemporalAccessor` **methods (for reading)**
  `get(…), getLong(…), isSupported(TemporalField), query(…), range(…)`
- **adds methods for adjusting temporal information**
- **implemented by:** `ZoneOffset`, `Zoned/LocalDateTime`, `LocalDate/Time`, **etc**

Temporal – adjustment methods
plus/minus(long,TemporalUnit),
plus/minus(TemporalAmount)
with(TemporalField,long)
with(TemporalAdjuster)

Temporal – other methods
isSupported(TemporalUnit)
until(Temporal, TemporalUnit)

The Class `OffsetDateTime`

The Class `OffsetDateTime`

Factory Methods

The Class `OffsetDateTime`

Factory Methods

```
now()
now(ZoneId)
now(Clock)
```

- Create an `OffsetDateTime` from the current instant, or the supplied `Clock`

The Class `OffsetDateTime`

Factory Methods

```
now()
now(ZoneId)
now(Clock)

of(int,int,int,int,int,int,int,
                          ZoneOffset)
of(LocalDate,LocalTime,ZoneOffset)
of(LocalDateTime,ZoneOffset)

ofInstant(Instant,ZoneId)
```

- Create an `OffsetDateTime` from the current instant, or the supplied `Clock`
- **Create an `OffsetDateTime` from the supplied values**

The Class `OffsetDateTime`

Conversion Methods

The Class `OffsetDateTime`

The Class `OffsetDateTime`

Conversion Methods

```
toLocalDate()
toLocalTime()
toLocalDateTime()
toInstant()
toZonedDateTime()
```

The Class `OffsetDateTime`

Conversion Methods

```
toLocalDate()
toLocalTime()
toLocalDateTime()
toInstant()
toZonedDateTime()

toEpochSecond()
```

The Class `OffsetDateTime`

Conversion Methods

```
toLocalDate()
toLocalTime()
toLocalDateTime()
toInstant()
toZonedDateTime()

toEpochSecond()

toOffsetTime()
```

The Class `OffsetDateTime`

Conversion Methods

```
toLocalDate()
toLocalTime()
toLocalDateTime()
toInstant()
toZonedDateTime()

toEpochSecond()

toOffsetTime()

atZoneSameInstant(ZoneId)
atZoneSimilarLocal(ZoneId)
```

The Class `OffsetDateTime`

Field Access

The Class `OffsetDateTime`

The Class `OffsetDateTime`

Field Access

`get...()`

The Class `OffsetDateTime`

Field Access

```
get...()
get(TemporalField)
```

The Class `OffsetDateTime`

Field Access

```
get...()
get(TemporalField)
```

Comparison Methods

```
isBefore(OffsetDateTime)
isAfter(OffsetDateTime)
isEqual(OffsetDateTime)
compareTo(OffsetDateTime)
```

The Class `OffsetDateTime`

Field Access

```
get...()
get(TemporalField)
```

Comparison Methods

```
isBefore(OffsetDateTime)
isAfter(OffsetDateTime)
isEqual(OffsetDateTime)
compareTo(OffsetDateTime)
```

Metadata Queries

```
isSupported(TemporalUnit)
isSupported(TemporalField)
range(TemporalField)
```

Using `ZoneOffset` and `OffsetDateTime`

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
LocalDateTime origLocalLandingTime = origOffsetLandingTime.toLocalDateTime();
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
LocalDateTime origLocalLandingTime = origOffsetLandingTime.toLocalDateTime();

List<WorkPeriod> usableWorkPeriods = wps.stream()
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
LocalDateTime origLocalLandingTime = origOffsetLandingTime.toLocalDateTime();

List<WorkPeriod> usableWorkPeriods = wps.stream()
   .filter(wp -> wp.getEndTime().isAfter(LocalDateTime.now()))
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
LocalDateTime origLocalLandingTime = origOffsetLandingTime.toLocalDateTime();

List<WorkPeriod> usableWorkPeriods = wps.stream()
   .filter(wp -> wp.getEndTime().isAfter(LocalDateTime.now()))
   .filter(wp -> wp.getEndTime().isBefore(origLocalLandingTime) ||
                 wp.getStartTime().isAfter(origLocalLandingTime))
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

```
List<WorkPeriod> wps = Utils.generateWorkPeriods(LocalDate.now(), 1);
ZoneOffset origZone = ZoneOffset.of("+0");
ZoneOffset destZone = ZoneOffset.of("-5");
LocalDateTime destLocalLandingTime = LocalDateTime.of(LocalDate.now(), LocalTime.of(11,0));

OffsetDateTime destOffsetLandingTime = OffsetDateTime.of(destLocalLandingTime, destZone);
OffsetDateTime origOffsetLandingTime = destOffsetLandingTime.withOffsetSameInstant(origZone);
LocalDateTime origLocalLandingTime = origOffsetLandingTime.toLocalDateTime();

List<WorkPeriod> usableWorkPeriods = wps.stream()
   .filter(wp -> wp.getEndTime().isAfter(LocalDateTime.now()))
   .filter(wp -> wp.getEndTime().isBefore(origLocalLandingTime) ||
                 wp.getStartTime().isAfter(origLocalLandingTime))
   .collect(toList());
```

Using `ZoneOffset` and `OffsetDateTime`

Which work periods will be usable on my travel day?

- In flight, I want to discard any work periods that have already ended or that contain the time at which my flight lands
- I'm travelling from London to NYC in winter, so my origin zone is GMT
- My destination zone is UTC-05:00, where I'm landing at 11am local time

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

Core `java.time` Time Zone Classes

ZoneOffset

the amount that a time zone differs from standard time

ZoneId

A ZoneOffset, or a region-based identifier

OffsetDateTime

A date-time with an offset from standard time

ZonedDateTime

A date-time with a ZoneId

The Class `ZonedDateTime`

Creation
Methods

The Class `ZonedDateTime`

The Class ZonedDateTime

Creation Methods

`from(TemporalAccessor)`

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
```

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
```

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
```

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
ofInstant(LDT,ZoneOffset,ZoneId)
ofStrict(LDT,ZoneOffset,ZoneId)
ofLocal(LDT,ZoneId,ZoneOffset)
```

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
ofInstant(LDT,ZoneOffset,ZoneId)
ofStrict(LDT,ZoneOffset,ZoneId)
ofLocal(LDT,ZoneId,ZoneOffset)
```

Conversion Methods

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
ofInstant(LDT,ZoneOffset,ZoneId)
ofStrict(LDT,ZoneOffset,ZoneId)
ofLocal(LDT,ZoneId,ZoneOffset)
```

Conversion Methods

```
to…()
```

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
ofInstant(LDT,ZoneOffset,ZoneId)
ofStrict(LDT,ZoneOffset,ZoneId)
ofLocal(LDT,ZoneId,ZoneOffset)
```

Conversion Methods

```
to…()
```

Field Access

The Class ZonedDateTime

Creation Methods

```
from(TemporalAccessor)
now(...)
of(...)
ofInstant(Instant,ZoneId)
ofInstant(LDT,ZoneOffset,ZoneId)
ofStrict(LDT,ZoneOffset,ZoneId)
ofLocal(LDT,ZoneId,ZoneOffset)
```

Conversion Methods

```
to…()
```

Field Access

```
get...()
```

The Class `ZonedDateTime`

Adjustment Methods

The Class `ZonedDateTime`

The Class `ZonedDateTime`

Adjustment Methods

`truncatedTo(TemporalUnit)`

The Class ZonedDateTime

Adjustment Methods

`truncatedTo(TemporalUnit)`

`plus...(int), minus...(int)`

The Class ZonedDateTime

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int), minus...(int)
with...(int)
```

The Class ZonedDateTime

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int), minus...(int)
with...(int)

withEarlierOffsetAtOverlap()
withLaterOffsetAtOverlap()
```

The Class ZonedDateTime

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int), minus...(int)
with...(int)

withEarlierOffsetAtOverlap()
withLaterOffsetAtOverlap()

withZoneSameInstant()
withZoneSameLocal()
```

The Class ZonedDateTime

Adjustment Methods

```
truncatedTo(TemporalUnit)

plus...(int), minus...(int)
with...(int)

withEarlierOffsetAtOverlap()
withLaterOffsetAtOverlap()

withZoneSameInstant()
withZoneSameLocal()

withFixedOffsetZone()
```

Using `ZonedDateTime` and `Interval`

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.
- Calculating overlap is easiest using the ThreeTenExtra class `Interval`

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.
- Calculating overlap is easiest using the ThreeTenExtra class `Interval`
- This code is for demonstration purposes only: it has time complexity of $O(n^2)$

```
// from WorkPeriod
public Interval toInterval(ZoneId zone) {
   return Interval.of(ZonedDateTime.of(startTime, zone).toInstant(), ZonedDateTime.of(endTime, zone).toInstant());
}
```

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.
- Calculating overlap is easiest using the ThreeTenExtra class `Interval`
- This code is for demonstration purposes only: it has time complexity of $O(n^2)$

```
// from WorkPeriod
public Interval toInterval(ZoneId zone) {
   return Interval.of(ZonedDateTime.of(startTime, zone).toInstant(), ZonedDateTime.of(endTime, zone).toInstant());
}
List<WorkPeriod> workPeriods = Utils.generateWorkPeriods(LocalDate.of(2017,10,26), 10);
ZoneId myZone = ZoneId.of("Europe/London");                // in the UK the clocks went back on 29th October
ZoneId theirZone = ZoneId.of("America/Chicago");           // in the US they went back on 5th November
```

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.
- Calculating overlap is easiest using the ThreeTenExtra class `Interval`
- This code is for demonstration purposes only: it has time complexity of $O(n^2)$

```
// from WorkPeriod
public Interval toInterval(ZoneId zone) {
   return Interval.of(ZonedDateTime.of(startTime, zone).toInstant(), ZonedDateTime.of(endTime, zone).toInstant());
}
List<WorkPeriod> workPeriods = Utils.generateWorkPeriods(LocalDate.of(2017,10,26), 10);
ZoneId myZone = ZoneId.of("Europe/London");                // in the UK the clocks went back on 29th October
ZoneId theirZone = ZoneId.of("America/Chicago");           // in the US they went back on 5th November
List<Interval> myIntervals = workPeriods.stream().map(wp -> wp.toInterval(myZone)).collect(toList());
List<Interval> theirIntervals = workPeriods.stream().map(wp -> wp.toInterval(theirZone)).collect(toList());
```

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- I'm in the UK time zone (Europe/London); they're in America/Chicago.
- Calculating overlap is easiest using the ThreeTenExtra class `Interval`
- This code is for demonstration purposes only: it has time complexity of $O(n^2)$

```
// from WorkPeriod
public Interval toInterval(ZoneId zone) {
   return Interval.of(ZonedDateTime.of(startTime, zone).toInstant(), ZonedDateTime.of(endTime, zone).toInstant());
}
List<WorkPeriod> workPeriods = Utils.generateWorkPeriods(LocalDate.of(2017,10,26), 10);
ZoneId myZone = ZoneId.of("Europe/London");                // in the UK the clocks went back on 29th October
ZoneId theirZone = ZoneId.of("America/Chicago");           // in the US they went back on 5th November
List<Interval> myIntervals = workPeriods.stream().map(wp -> wp.toInterval(myZone)).collect(toList());
List<Interval> theirIntervals = workPeriods.stream().map(wp -> wp.toInterval(theirZone)).collect(toList());
Duration intersectDuration = Duration.ZERO;
for (Interval myIntvl : myIntervals) {
   for (Interval theirIntvl : theirIntervals) {
      if (myIntvl.isConnected(theirIntvl)) {
         intersectDuration = intersectDuration.plus(myIntvl.intersection(theirIntvl).toDuration());
      }
   }
}
```

Using `ZonedDateTime` and `Interval`

How much overlap will I have with my colleagues in the American midwest?

- **I'm in the UK time zone (Europe/London); they're in America/Chicago.**
- **Calculating overlap is easiest using the ThreeTenExtra class `Interval`**
- **This code is for demonstration purposes only: it has time complexity of $O(n^2)$**

The Interface

`TemporalAdjuster`

The Interface

`TemporalAdjuster`

Strategy for adjusting a temporal object

The Interface

`TemporalAdjuster`

Strategy for adjusting a temporal object

Externalizes the process of adjustment

The Interface

`TemporalAdjuster`

Strategy for adjusting a temporal object

Externalizes the process of adjustment

For example:

- set a date to avoid weekends
- set a date-time to midnight

The Interface

`TemporalAdjuster`

Strategy for adjusting a temporal object

Externalizes the process of adjustment

For example:

- set a date to avoid weekends
- set a date-time to midnight

Most common use as the argument to

`with(TemporalAdjuster)`

method of `Temporal` **objects**

```
List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.plusDays(1))
         .filter(Utils::isWorkingDay)
         .limit(dayCount)
         .collect(toList());
}
```

When to Use a `TemporalAdjuster`?

This is how we calculated working days in the last module

```
private static boolean isWorkingDay(LocalDate d) {
   return d.getDayOfWeek() != DayOfWeek.SATURDAY &&
          d.getDayOfWeek() != DayOfWeek.SUNDAY;
}
List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.plusDays(1))
         .filter(Utils::isWorkingDay)
         .limit(dayCount)
         .collect(toList());
}
```

When to Use a `TemporalAdjuster`?

This is how we calculated working days in the last module

```
List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.with(nextWorkingDayAdjuster))
         .limit(dayCount)
         .collect(toList());
}
```

Using a `TemporalAdjuster`

```
List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.with(nextWorkingDayAdjuster))
         .limit(dayCount)
         .collect(toList());
}
```

Using a `TemporalAdjuster`

An alternative is to encapsulate the strategy in a reusable `TemporalAdjuster`

```
static TemporalAdjuster nextWorkingDayAdjuster =
      t -> LocalDate.from(t).getDayOfWeek() == FRIDAY
            ? t.with(TemporalAdjusters.next(MONDAY))
            : t.plus(1, java.time.temporal.ChronoUnit.DAYS);

List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.with(nextWorkingDayAdjuster))
         .limit(dayCount)
         .collect(toList());
}
```

Using a `TemporalAdjuster`

An alternative is to encapsulate the strategy in a reusable `TemporalAdjuster`

```
static TemporalAdjuster nextWorkingDayAdjuster =
      t -> LocalDate.from(t).getDayOfWeek() == FRIDAY
            ? t.with(TemporalAdjusters.next(MONDAY))
            : t.plus(1, java.time.temporal.ChronoUnit.DAYS);

List<LocalDate> generateWorkingDays(LocalDate startDate, int dayCount) {
   return Stream.iterate(startDate, d -> d.with(nextWorkingDayAdjuster))
         .limit(dayCount)
         .collect(toList());
}
```

Using a `TemporalAdjuster`

An alternative is to encapsulate the strategy in a reusable `TemporalAdjuster`

- Best practice is to define user-written adjusters as static instances

```
public Optional<WorkPeriod> split() {
   LocalDateTime midnight = startTime.plusDays(1).toLocalDate().atStartOfDay();
   return split(midnight);
}
```

Using a `TemporalAdjuster`

```
public Optional<WorkPeriod> split() {
   LocalDateTime midnight = startTime.plusDays(1).toLocalDate().atStartOfDay();
   return split(midnight);
}
```

Using a `TemporalAdjuster`

Many classes implement `TemporalAdjuster`**:**

- `LocalDateTime`, `LocalDate`, `LocalTime`
- `OffsetDateTime`, `OffsetTime`, `ZoneOffset`
- `Instant`
- `Enum`**s representing days, months, years**

```
public Optional<WorkPeriod> split() {
   LocalDateTime midnight = startTime.plusDays(1).toLocalDate().atStartOfDay();
   return split(midnight);
}

public Optional<WorkPeriod> split() {
   LocalDateTime midnight = startTime.plusDays(1).with(LocalTime.MIDNIGHT);
   return split(midnight);
}
```

Using a `TemporalAdjuster`

Many classes implement `TemporalAdjuster`**:**

- `LocalDateTime`, `LocalDate`, `LocalTime`
- `OffsetDateTime`, `OffsetTime`, `ZoneOffset`
- `Instant`
- `Enum`**s representing days, months, years**

Methods of the Class `TemporalAdjusters`

```
dayOfWeekInMonth(int,DayOfWeek)
firstDayOfMonth()
firstDayOfNextMonth()
firstDayOfNextYear()
firstDayOfYear()
firstInMonth(DayOfWeek)
lastDayOfMonth()
lastDayOfYear()
lastInMonth(DayOfWeek)
next(DayOfWeek)
nextOrSame(DayOfWeek)
ofDateAdjuster(UnaryOperator<LocalDate>)
previous(DayOfWeek)
previousOrSame(DayOfWeek)
```

```
List<Event> createReviews(LocalDateTime start, int count, Duration dur, ZoneId zone) {
   return Stream.iterate(start.toLocalDate(), d -> d.with(nextReviewDate))
      .limit(count)
      .map(d -> ZonedDateTime.of(d, start.toLocalTime(), zone))
      .map(zonedDt -> new Event(zonedDt, dur, "standup"))
      .collect(toList());
}
```

Using the Methods of `TemporalAdjusters`

```
List<Event> createReviews(LocalDateTime start, int count, Duration dur, ZoneId zone) {
   return Stream.iterate(start.toLocalDate(), d -> d.with(nextReviewDate))
      .limit(count)
      .map(d -> ZonedDateTime.of(d, start.toLocalTime(), zone))
      .map(zonedDt -> new Event(zonedDt, dur, "standup"))
      .collect(toList());
}
```

Using the Methods of `TemporalAdjusters`

We want to schedule review meetings for the first Monday of every month

```
List<Event> createReviews(LocalDateTime start, int count, Duration dur, ZoneId zone) {
   return Stream.iterate(start.toLocalDate(), d -> d.with(nextReviewDate))
      .limit(count)
      .map(d -> ZonedDateTime.of(d, start.toLocalTime(), zone))
      .map(zonedDt -> new Event(zonedDt, dur, "standup"))
      .collect(toList());
}
```

Using the Methods of `TemporalAdjusters`

We want to schedule review meetings for the first Monday of every month

- We can compose predefined `TemporalAdjuster` instances

```
static TemporalAdjuster nextReviewDate = startDate -> startDate
     .with(TemporalAdjusters.firstDayOfNextMonth())
     .with(TemporalAdjusters.firstInMonth(DayOfWeek.MONDAY));


List<Event> createReviews(LocalDateTime start, int count, Duration dur, ZoneId zone) {
   return Stream.iterate(start.toLocalDate(), d -> d.with(nextReviewDate))
     .limit(count)
     .map(d -> ZonedDateTime.of(d, start.toLocalTime(), zone))
     .map(zonedDt -> new Event(zonedDt, dur, "standup"))
     .collect(toList());
}
```

Using the Methods of `TemporalAdjusters`

We want to schedule review meetings for the first Monday of every month

- **We can compose predefined** `TemporalAdjuster` **instances**

Demo: Combining Periods and Events

Demo: Combining Periods and Events

Demo: Combining Periods and Events

Events interrupt working time – it's everyone's problem!

Demo: Combining Periods and Events

Events interrupt working time
– it's everyone's problem!

Overlaps between work periods and events
– resolved by preferring events

Demo: Combining Periods and Events

Events interrupt working time
- it's everyone's problem!

Overlaps between work periods and events
- resolved by preferring events

Could compute overlaps by comparing every period with every event
- **as for the schedule intersection example**
- **very inefficient! ($O(n^2)$)**

Summary

Summary

Summary

Overview of Time Zones

Summary

Overview of Time Zones

Working with the Time Zone Classes

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset and ZoneId`

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset` and `ZoneId`
- `OffsetDateTime` **and** `ZonedDateTime`

Summary

Overview of Time Zones

Working with the Time Zone Classes

- `ZoneOffset` and `ZoneId`
- `OffsetDateTime` and `ZonedDateTime`

Temporal Adjusters