Add a Timezone to LocalDateTime with ZonedDateTime in Java 8

{} **codebyamir.com**/blog/add-a-timezone-to-localdatetime-with-zoneddatetime-in-java-8

May 2, 2018 Amir Boroumand

Overview

The `LocalDateTime` class introduced in Java 8 stores the date and time but not the timezone. So we need something else when developing applications that need to be aware of different timezones.

Fortunately, we can use the `ZonedDateTime` class to represent dates and times with timezone information. The class contains static methods that allow us to perform date conversions and calculations easily.

Code Examples

`ZonedDateTime` objects are immutable so the static methods all return a new instance.

Get current date and time in the local timezone

ZonedDateTime.now()

```
ZonedDateTime now = ZonedDateTime.now();
// 2018-05-02T15:45:20.981-04:00[America/New_York]
```

Get current date and time in a different timezone

ZonedDateTime.now(zoneId)

```
ZonedDateTime nowInParis = ZonedDateTime.now(ZoneId.of("Europe/Paris"));
// 2018-05-02T21:45:20.982+02:00[Europe/Paris]
```

TimeZone Id Values

The list of TimeZone id values is documented well here.

We can also retrieve them using a method:

```
Set<String> zoneIds = ZoneId.getAvailableZoneIds();
```

Always use timezone values in the Region/City format (`America/New_York`) and avoid the short abbreviations (`EST`) because they are ambiguous and non-standard.

Create ZonedDateTime from a String

ZonedDateTime.parse(dateString)

```
ZonedDateTime departureTime = ZonedDateTime.parse("2018-07-
01T10:00:00Z[America/New_York]");
ZonedDateTime arrivalTime = ZonedDateTime.parse("2018-07-
01T22:00:00Z[Europe/London]");
```

Convert LocalDateTime to ZonedDateTime

We can convert a LocalDateTime to ZonedDateTime a couple different ways.

```
LocalDateTime ldt = LocalDateTime.parse("2018-07-01T08:00");
ZoneId zoneId = ZoneId.of("Europe/Paris");
```

ZonedDateTime.of(ldt, zoneId)

```
ZonedDateTime zdt = ZonedDateTime.of(ldt, zoneId);
// 2018-07-01T08:00+02:00[Europe/Paris]
```

atZone(zoneId)

```
ZonedDateTime zdt = ldt.atZone(zoneId);
// 2018-07-01T08:00+02:00[Europe/Paris]
```

Note that this doesn't apply any timezone conversion to our LocalDateTime. It simply adds the timezone to whatever date and time was stored in the source object.

Compare ZonedDateTime objects

isBefore(zdt)

```
boolean departureBeforeArrival = departureTime.isBefore(arrivalTime);
```

isAfter(zdt)

```
boolean arrivalAfterDeparture = arrivalTime.isAfter(departureTime);
```

isEqual(zdt)

This compares the actual dates and times and not the object references.

```
LocalDateTime ldt = LocalDateTime.parse("2018-07-01T08:00");

ZonedDateTime zdtParis = ZonedDateTime.of(ldt, ZoneId.of("Europe/Paris"));
// 2018-07-01T08:00+02:00[Europe/Paris]

ZonedDateTime zdtNewYork = ZonedDateTime.of(ldt, ZoneId.of("America/New_York"));
// 2018-07-01T08:00-04:00[America/New_York]

boolean equal = zdtParis.isEqual(zdtNewYork); // false
```

Find the time difference between ZonedDateTime objects

`ChronoUnit` in `java.time.temporal` allows us to calculate the time difference between `ZonedDateTime` objects:

```
long flightTimeInHours = ChronoUnit.HOURS.between(departureTime, arrivalTime); // 7
long flightTimeInMinutes = ChronoUnit.MINUTES.between(departureTime, arrivalTime);
// 420
```

Create ZonedDateTime with new timezone from existing ZonedDateTime object

Given a ZonedDateTime object in timezone A, we can create another ZonedDateTime object in timezone B which represents the same point in time.

withZoneSameInstant(zoneId)

```
ZonedDateTime nowInLocalTimeZone = ZonedDateTime.now();
// 2018-05-02T20:10:27.896-04:00[America/New_York]

ZonedDateTime nowInParis =
nowInLocalTimeZone.withZoneSameInstant(ZoneId.of("Europe/Paris"));
// 2018-05-03T02:10:27.896+02:00[Europe/Paris]
```

Unit Testing Considerations

Suppose we need a method that determines if we can issue a boarding pass to an airline passenger based on some business logic.

It should return `true` when the departure time of the outbound OR return flight is less than 24 hours away.

We could write something like this:

```
package com.example.demo;

import java.time.ZonedDateTime;
import java.time.temporal.ChronoUnit;

public class BoardingPassService {
  private static final int MIN_HOUR_DIFF_BOARDING_PASS = -24;

  public boolean canIssueBoardingPass(Ticket ticket) {
    if (ticket == null || ticket.getOutboundFlight() == null ||
ticket.getReturnFlight() == null) {
      return false;
    }

    ZonedDateTime now = ZonedDateTime.now();
 ZonedDateTime outboundDepartureTime =
ticket.getOutboundFlight().getDepartureTime();
    ZonedDateTime returnDepartureTime = ticket.getReturnFlight().getDepartureTime();

    long diffInHours = ChronoUnit.HOURS.between(outboundDepartureTime, now);

    boolean outboundDepartureInAllowableRange = (diffInHours >=
MIN_HOUR_DIFF_BOARDING_PASS && diffInHours <= 0);

    diffInHours = ChronoUnit.HOURS.between(returnDepartureTime, now);

    boolean returnDepartureInAllowableRange = (diffInHours >=
MIN_HOUR_DIFF_BOARDING_PASS && diffInHours <= 0);

    if (outboundDepartureInAllowableRange || returnDepartureInAllowableRange) {
      return true;
    }

    return false;
  }
}
```

This works fine but is difficult to unit test because of line 14. We cannot mock ZonedDateTime.now() because it's a static method.

java.time.Clock

The now() methods in java.time all accept a Clock argument which we can use to refactor our code so it's testable.

Here's the testable version of the class:

```
package com.example.demo;

import java.time.Clock;
import java.time.ZonedDateTime;
import java.time.temporal.ChronoUnit;

public class BoardingPassService {

  private final Clock clock;
  private static final int MIN_HOUR_DIFF_BOARDING_PASS = -24;

  public BoardingPassService(Clock clock) {
    this.clock = clock;
  }

  public boolean canIssueBoardingPass(Ticket ticket) {
    if (ticket == null || ticket.getOutboundFlight() == null ||
ticket.getReturnFlight() == null) {
      return false;
    }

    ZonedDateTime now = ZonedDateTime.now(clock);
 ZonedDateTime outboundDepartureTime =
ticket.getOutboundFlight().getDepartureTime();
    ZonedDateTime returnDepartureTime = ticket.getReturnFlight().getDepartureTime();

    long diffInHours = ChronoUnit.HOURS.between(outboundDepartureTime, now);

    boolean outboundDepartureInAllowableRange = (diffInHours >=
MIN_HOUR_DIFF_BOARDING_PASS && diffInHours <= 0);

    diffInHours = ChronoUnit.HOURS.between(returnDepartureTime, now);

    boolean returnDepartureInAllowableRange = (diffInHours >=
MIN_HOUR_DIFF_BOARDING_PASS && diffInHours <= 0);

    if (outboundDepartureInAllowableRange || returnDepartureInAllowableRange) {
      return true;
    }

    return false;
  }
}
```

On line 12, we added a single argument constructor which accepts a Clock object. We can then use the clock on line 21.

In the calling code, we'd pass `Clock.systemDefaultZone()` to the the constructor to use the current system time.

In our tests, we'll create a fixed time clock using `Clock.fixed()` and pass that.

Unit Tests

```
package com.example.demo;
```

```
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.time.Clock;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class BoardingPassServiceTest {
  private Ticket ticket;
  private BoardingPassService boardingPassService;

  private final LocalDateTime REFERENCE_DATE_TIME = LocalDateTime.of(2018, 7, 1, 10,
0); // 2018-07-01 at 10:00am
  private final ZoneId defaultZone = ZoneId.systemDefault();
  private final Clock FIXED_CLOCK =
Clock.fixed(REFERENCE_DATE_TIME.atZone(defaultZone).toInstant(), defaultZone);

  @BeforeEach
  public void setUp() {
    ticket = new Ticket();
 boardingPassService = new BoardingPassService(FIXED_CLOCK);
  }

  @Test
  public void testCanIssueBoardingPass_Null() {
    assertFalse(boardingPassService.canIssueBoardingPass(null));
  }

  @Test
  public void testCanIssueBoardingPass_OutboundFlightWithinTimeRange() {
    Flight outboundFlight = new Flight();
 outboundFlight.setDepartureTime(ZonedDateTime.parse("2018-07-
02T07:30:00Z[America/Chicago]"));

 Flight returnFlight = new Flight();
 returnFlight.setDepartureTime(ZonedDateTime.parse("2018-07-
09T11:30:00Z[America/New_York]"));

 ticket.setOutboundFlight(outboundFlight);
 ticket.setReturnFlight(returnFlight);

 assertTrue(boardingPassService.canIssueBoardingPass(ticket));
  }

  @Test
  public void testCanIssueBoardingPass_ReturnFlightWithinTimeRange() {
    Flight outboundFlight = new Flight();
 outboundFlight.setDepartureTime(ZonedDateTime.parse("2018-06-
20T07:30:00Z[America/Chicago]"));

 Flight returnFlight = new Flight();
 returnFlight.setDepartureTime(ZonedDateTime.parse("2018-07-
02T09:30:00Z[America/New_York]"));

 ticket.setOutboundFlight(outboundFlight);
    ticket.setReturnFlight(returnFlight);

    assertTrue(boardingPassService.canIssueBoardingPass(ticket));
  }

  @Test
  public void testCanIssueBoardingPass_NoFlightWithinTimeRange() {
    Flight outboundFlight = new Flight();
 outboundFlight.setDepartureTime(ZonedDateTime.parse("2018-09-
01T07:30:00Z[America/Chicago]"));

 Flight returnFlight = new Flight();
 returnFlight.setDepartureTime(ZonedDateTime.parse("2018-09-
08T11:30:00Z[America/New_York]"));

    ticket.setOutboundFlight(outboundFlight);
 ticket.setReturnFlight(returnFlight);

 assertFalse(boardingPassService.canIssueBoardingPass(ticket));
  }
}
```