Dátum- és időkezelés Java-ban

Jeszenszky Péter
Debreceni Egyetem, Informatikai Kar
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2018. január 15.

Történet

- JDK 1.0 (1996): a `java.lang.System.currentTimeMillis()` metódus és a `java.util.Date` osztály
- JDK 1.1 (1997):
  - A `java.util.Calendar`, `java.util.GregorianCalendar`, `java.util.TimeZone` és `java.util.SimpleTimeZone` osztályok
  - Formázás, elemezés: a `java.text.DateFormat` és a `java.text.SimpleDateFormat` osztályok
  - JDBC: a `java.sql.Date`, `java.sql.Time` és `java.sql.Timestamp` osztályok
- Java SE 5 (2004): `javax.xml.datatype` csomag
- Java SE 8 (2014): *JSR 310: Date and Time API* (a `java.time` csomag és alcsomagjai)

java.util.Date (1)

- Egy időpillanatot ábrázol ezredmásodperc pontossággal. https://docs.oracle.com/javase/8/docs/api/java/util/Date.html
- Problémák:
  - Félrevezető név.
  - A legtöbb konstruktor és metódus elavult (*deprecated*).
  - Hónapok 0 alapú indexelése.
  - Helyi időzóna használata.
  - Változtathatóság (*mutability*).
    - Valahányszor vissza kell adni egy dátumot, egy másolatot adjunk vissza, hogy ne lehessen elrontani a referencián keresztül az eredeti objektumot.
  - Nem szálbiztos (*not thread-safe*).
  - Az osztály nem biztosít műveleteket az objektumok manipulálásához (például 1 nappal későbbi időpillanat előállítása).

java.util.Date (2)

- Példa a használatra:

```
Date date = new Date(1848 - 1900, 3 - 1, 15);
System.out.println(date);   // Wed Mar 15 00:00:00 CET 1848
System.out.println(date.getTimezoneOffset());  // -60

DateFormat format = new SimpleDateFormat("yyyy. MMMM dd.", new Locale("HU"));
System.out.println(format.format(date));   // 1848. március 15.

date = format.parse("2017. január 13.");
System.out.println(date);  // Fri Jan 13 00:00:00 CET 2017

date = new Date();
System.out.println(date);  // Sun Jan 15 11:01:46 CET 2017

date.setTime(0);
System.out.println(date);  // Thu Jan 01 01:00:00 CET 1970
```

JDBC adattípusok (1)

- SQL adattípusoknak megfelelő osztályok:
  - `java.sql.Date`: az SQL `DATE` adattípusát ábrázolja. https://docs.oracle.com/javase/8/docs/api/java/sql/Date.html
  - `java.sql.Time`: az SQL `TIME` adattípusát ábrázolja. https://docs.oracle.com/javase/8/docs/api/java/sql/Time.html
  - `java.sql.Timestamp`: az SQL `TIMESTAMP` adattípusát ábrázolja. https://docs.oracle.com/javase/8/docs/api/java/sql/Timestamp.html
    - Nanomásodperc pontosságú.
- Mindhárom a `java.util.Date` osztályt terjeszti ki.
- Egyiknél sem kerül tárolásra időzóna információ.

JDBC adattípusok (2)

- Példa a használatukra:

```
Date date = Date.valueOf("1848-03-15");
System.out.println(date);         // 1848-03-15

Time time = Time.valueOf("17:49:03");
System.out.println(time);         // 17:49:03

Timestamp timestamp = new Timestamp(System.currentTimeMillis());
System.out.println(timestamp);    // 2017-01-15 11:03:13.407
```

java.util.Calendar (1)

- Egy időpillanatot ábrázol ezredmásodperc pontossággal. https://docs.oracle.com/javase/8/docs/api/java/util/Calendar.html
  - Lehetővé teszi a hozzáférést az év, hónap, nap stb. mezőkhöz.
  - Időzónát is tárol.
  - Metódusok biztosítása az objektumok manipulálásához.
    - Például: `add(...)`, `roll(...)`
- Absztrakt osztály, egyetlen konkrét alosztálya a `java.util.GregorianCalendar.`
  - A Julián naptár és a Gergely-naptár támogatása.

java.util.Calendar (2)

- Problémák továbbra is:
  - Félrevezető név.
  - Hónapok 0 alapú indexelése.
  - Változtathatóság (*mutability*).
  - Nem szálbiztos (*thread-safe*).

java.util.Calendar (3)

- Példa a használatra:

```
Calendar cal = new GregorianCalendar(1848, 2, 15);
System.out.println(cal);
// java.util.GregorianCalendar[time=?,areFieldsSet=false,
// areAllFieldsSet=false,lenient=true,
// ...
// zone=sun.util.calendar.ZoneInfo[id="Europe/Budapest",
// YEAR=1848,MONTH=2,WEEK_OF_YEAR=?,WEEK_OF_MONTH=?,
// DAY_OF_MONTH=15, DAY_OF_YEAR=?,DAY_OF_WEEK=?,
// DAY_OF_WEEK_IN_MONTH=?,AM_PM=0,HOUR=0,HOUR_OF_DAY=0,
// MINUTE=0,SECOND=0,MILLISECOND=?,ZONE_OFFSET=?,DST_OFFSET=?]

int year = cal.get(Calendar.YEAR);                // 1848
int month = cal.get(Calendar.MONTH);              // 2 -> március
int dayOfMonth = cal.get(Calendar.DAY_OF_MONTH);// 15
int dayOfWeek = cal.get(Calendar.DAY_OF_WEEK);  // 4 -> szerda
int dayOfYear = cal.get(Calendar.DAY_OF_YEAR);  // 75

TimeZone tz = cal.getTimeZone();
System.out.println(tz.getDisplayName()); // Central European Time
System.out.println(tz.getID());          // Europe/Budapest
```

java.util.Calendar (4)

- Példa a használatra (folytatás):

```
DateFormat format = SimpleDateFormat.getDateInstance(
    DateFormat.FULL, Locale.GERMAN);

Date date = cal.getTime();
System.out.println(format.format(date)); // Mittwoch, 15. März 1848

cal.add(Calendar.DAY_OF_MONTH, 16);
date = cal.getTime();
System.out.println(format.format(date)); // Freitag, 31. März 1848

cal.roll(Calendar.MONTH, -1);
date = cal.getTime();
System.out.println(format.format(date)); // Dienstag, 29. Februar 1848
```

java.util.Calendar (5)

- Példa a használatra:

```
Calendar now = Calendar.getInstance();
System.out.println(now.getClass().getName());
// java.util.GregorianCalendar

DateFormat format = SimpleDateFormat.getDateTimeInstance(
    DateFormat.FULL, DateFormat.FULL, new Locale("hu"));

Date date = now.getTime();
System.out.println(format.format(date));
// 2017. január 15. 11:06:08 CET

format.setTimeZone(TimeZone.getTimeZone("PST"));
System.out.println(format.format(date));
// 2017. január 15. 2:06:08 PST
```

java.util.Calendar (6)

- Miért baj a változtathatóság?

```
public static long countDays(Calendar start, Calendar end) {
    long count = 0;
    while (start.before(end)) {
        start.add(Calendar.DAY_OF_MONTH, 1);
        ++count;
    }
    return count;
}

Calendar then = new GregorianCalendar(1848, 2, 15);
Calendar now = Calendar.getInstance();
System.out.println(countDays(then, now));  // 61668
System.out.println(countDays(then, now));  // Mi az output?
```

java.util.Calendar (6)

- Miért baj a változtathatóság?

```
public static long countDays(Calendar start, Calendar end) {
    long count = 0;
    while (start.before(end)) {
        start.add(Calendar.DAY_OF_MONTH, 1);
        ++count;
    }
    return count;
}

Calendar then = new GregorianCalendar(1848, 2, 15);
Calendar now = Calendar.getInstance();
System.out.println(countDays(then, now));  // 61668
System.out.println(countDays(then, now));  // 0
```

java.util.Calendar (7)

- Az előző példa helyesen:

```
public static long countDays(Calendar start, Calendar end) {
    long count = 0;
    start = (Calendar) start.clone();
    while (start.before(end)) {
        start.add(Calendar.DAY_OF_MONTH, 1);
        ++count;
    }
    return count;
}

Calendar then = new GregorianCalendar(1848, 2, 15);
Calendar now = Calendar.getInstance();
System.out.println(countDays(then, now));  // 61668
System.out.println(countDays(then, now));  // 61668
```

W3C XML Schema (1)

- Paul V. Biron, Ashok Malhotra (ed). *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. 28 October 2004. https://www.w3.org/TR/xmlschema-2/
  - Szabványos adattípusok definiálása dátum- és időkezeléshez.
- Anders Berglund, Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, Norman Walsh (ed). *XQuery 1.0 and XPath 2.0 Data Model (XDM)*. W3C Recommendation. 23 January 2007. https://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/
  - Két további adattípus bevezetése: `xsd:dayTimeDuration`, `xsd:yearMonthDuration`.

W3C XML Schema (2)

- Az adattípusok értékeinek ábrázolása karakterláncokkal (literálokkal), melyhez szintaxis meghatározása.

W3C XML Schema (3)

- Beépített adattípusok dátum- és időkezeléshez:
  - **`xsd:date`**: naptári dátumokat ábrázoló adattípus.
    - Literálok: `1848-03-15`, `2004-10-28+13:00`, `2016-02-29Z`
  - **`xsd:time`**: naponta ismétlődő időpillanatokat ábrázoló adattípus.
    - Literálok: `12:15:00` (nincs időzóna), `23:05:30Z` (egyezményes koordinált világidő), `00:45:00+01:00` (közép-európai idő)
  - **`xsd:dateTime`**: egyedi időpillanatokat ábrázoló adattípus.
    - Literálok: `1969-07-16T13:32:00Z` (az Apollo-11 fellövésének pillanata)
  - **`xsd:duration`**: időtartamokat ábrázoló adattípus.
    - Literálok: `P1Y2DT5H30M` (1 év, 2 nap, 5 óra és 30 perc), `P120Y6M` (120 év és 6 hónap), `PT9.58S` (9,58 másodperc)

W3C XML Schema (4)

- Beépített adattípusok dátum- és időkezeléshez (folytatás):
  - **`xsd:gDay`**: a Gergely-naptár havonta ismétlődő napjait reprezentáló adattípus.
    - Literálok: `---05` (a hónap ötödik napja)
  - **`xsd:gMonth`**: a Gergely-naptár 12 hónapját reprezentáló adattípus.
    - Literálok: `--03` (március)
  - **`xsd:gYear`**: a Gergely-naptár éveit reprezentáló adattípus.
    - Literálok: `-0753` (Róma alapításának éve a hagyomány szerint), 0476 (a Nyugatrómai Birodalom bukásának éve), 1984
  - **`xsd:gMonthDay`**: a Gergely-naptár évente ismétlődő dátumait reprezentáló adattípus.
    - Literálok: `--05-25` (május 25. napját jelöli, az úgynevezett Törülközőnap)
  - **`xsd:gYearMonth`**: a Gergely-naptár éveinek hónapjait reprezentáló adattípus.
    - Literálok: `1848-03` (1848 márciusa)

W3C XML Schema (5)

- Java támogatás: `javax.xml.datatype` csomag https://docs.oracle.com/javase/8/docs/api/javax/xml/datatype/package-summary.html
  - A 2004-ben kiadott J2SE 5 tartalmazza.
- Példa a használatára:

```
DatatypeFactory factory = DatatypeFactory.newInstance();

Duration duration = factory.newDuration(true, 1, 0, 2, 5, 30, 30);
System.out.println(duration);   // P1Y0M2DT5H30M30S

XMLGregorianCalendar date = factory.newXMLGregorianCalendarDate(1848, 3,
    15, DatatypeConstants.FIELD_UNDEFINED);
System.out.println(date);       // 1848-03-15

date.add(duration);
System.out.println(date);       // 1849-03-17
```

Joda-Time

- Java programkönyvtár dátum- és időkezeléshez. http://www.joda.org/joda-time/
  - Licenc: *Apache License 2.0*
- A Java SE 8 előtt *de-facto* szabvány az iparban.
- A Java SE 8-ban megjelent `java.time` csomag feleslegessé tette.
- Joda-Time és JSR-310:
  - Stephen Colebourne: *Why JSR-310 isn't Joda-Time*. November 20, 2009. http://blog.joda.org/2009/11/why-jsr-310-isn-joda-time_4941.html

JSR 310

- *JSR 310: Date and Time API (Final Release)*. 4 March 2014. https://jcp.org/en/jsr/detail?id=310
  - Csomagok: `java.time`, `java.time.chrono`, `java.time.format`, `java.time.temporal`, `java.time.zone`
- A Java SE 8 tartalmazza.

Az alapul szolgáló szabvány

- ISO 8601:2004: *Data elements and interchange formats – Information interchange – Representation of dates and times* http://www.iso.org/iso/home/standards/iso8601.htm

Tervezési célok

- Tisztaság:
  - Jól definiált, kiszámíthatóan működő API.
- Minden osztály megváltoztathatatlan (*immutable*):
  - Egy objektum a létrehozását követően nem módosítható.
- Fluent API:
  - Cél a jól olvashatóság, metódus-láncolás használata.
- Kiterjeszthetőség:
  - Akár saját naptárrendszer is implementálható.

További jellemzők

- Többféle dátum- és időkezelő osztály biztosítása különféle célokra.
  - Minden osztály megváltoztathatatlan és szálbiztos.
  - Minden osztály támogatja az objektumok sztringekből való létrehozását és formázását.
- Nanomásodperc pontosság.
- I18n és L10n támogatás:
  - Az *Unicode Common Locale Data Repository* (CLDR) használata. http://cldr.unicode.org/
- Időzónák:
  - Az IANA *Time Zone Database* használata. http://www.iana.org/time-zones

Csomagok

- `java.time`:
  - Core API, mely a dátumok, idők, időtartamok, időzónák stb. kezelésére szolgáló osztályokat tartalmazza.
- `java.time.chrono`:
  - API naptárrendszerek kezeléséhez.
- `java.time.format`:
  - API dátum és idő formázáshoz, objektumok sztringekből való létrehozásához.
- `java.time.temporal`:
  - Kiterjesztett API elsősorban keretrendszer és programkönyvtár készítők számára.
- `java.time.zone`:
  - Időzóna kezelés.

Metódusok névkonvenciója

Előtag	Típus	Jelentés
`from`	statikus objektumgyár	Objektum létrehozása a paraméter konvertálásával.
`of`	statikus objektumgyár	Objektum létrehozása a paraméterekből.
`parse`	statikus objektumgyár	Objektum létrehozása sztringből.
`at`	példányszintű	Az objektumot egy másik objektummal kombináló metódus, mint például `LocalDate.atTime(LocalTime).`
`format`	példányszintű	Az objektum formázására szolgáló metódus.
`get`	példányszintű	Valamilyen érték lekérdezésére szolgáló metódus.
`is`	példányszintű	Az objektum állapotát vizsgáló logikai értékű metódus.
`minus`	példányszintű	Valamilyen mennyiség kivonása az objektumból.
`plus`	példányszintű	Valamilyen mennyiség hozzáadása az objektumhoz.
`to`	példányszintű	Az objektumot egy valamilyen más típusra konvertáló metódus.
`with`	példányszintű	Visszaad egy másolatot az objektumról benne egy elemet megváltoztatva, a beállító metódus *immutable* megfelelője.

Dátum és idő osztályok

Class or Enum	Year	Month	Day	Hours	Minutes	Seconds*	Zone Offset	Zone ID	toString Output
Instant						✓			2013-08-20T15:16:26.355Z
LocalDate	✓	✓	✓						2013-08-20
LocalDateTime	✓	✓	✓	✓	✓	✓			2013-08-20T08:16:26.937
ZonedDateTime	✓	✓	✓	✓	✓	✓	✓	✓	2013-08-21T00:16:26.941+09:00[Asia/Tokyo]
LocalTime				✓	✓	✓			08:16:26.943
MonthDay		✓	✓						--08-20
Year	✓								2013
YearMonth	✓	✓							2013-08
Month		✓							AUGUST
OffsetDateTime	✓	✓	✓	✓	✓	✓	✓		2013-08-20T08:16:26.954-07:00
OffsetTime				✓	✓	✓	✓		08:16:26.957-07:00
Duration			**	**	**	✓			PT20H (20 hours)
Period	✓	✓	✓				***	***	P10D (10 days)

* Seconds are captured to nanosecond precision.

** This class does not store this information, but has methods to provide time in these units.

Forrás: https://docs.oracle.com/javase/tutorial/datetime/iso/overview.html

Példa az API alapszintű használatára

```
LocalDate date = LocalDate.now();
System.out.println(date);     // 2017-01-15

date = LocalDate.of(1848, 3, 15);
System.out.println(date);     // 1848-03-15

date = LocalDate.of(1848, Month.MARCH, 15);
System.out.println(date);     // 1848-03-15

LocalTime time = LocalTime.of(14, 52, 26);
System.out.println(time);     // 14:52:26

LocalDateTime dateTime = date.atTime(time);
System.out.println(dateTime); // 1848-03-15T14:52:26

ZonedDateTime zonedDateTime = dateTime.atZone(ZoneId.of("CET"));
System.out.println(zonedDateTime);
// 1848-03-15T14:52:26+01:00[CET]
```

Gazdag és tiszta API

- Példa:

```
LocalDate date = LocalDate.of(2012, 4, 4);
System.out.println(date);                    // 2012-04-04
System.out.println(date.getEra());           // CE
System.out.println(date.getYear());          // 2012
System.out.println(date.getMonth());         // APRIL
System.out.println(date.getMonthValue());    // 4
System.out.println(date.getDayOfMonth());    // 4
System.out.println(date.getDayOfWeek());     // WEDNESDAY
System.out.println(date.lengthOfMonth());    // 30
System.out.println(date.getDayOfYear());     // 95
System.out.println(date.isLeapYear());       // true

date = Year.of(1848).atMonth(Month.MARCH).atDay(15);
System.out.println(date);                    // 1848-03-15
```

Fluent API

- Példa:

```
LocalDate today = LocalDate.now();
LocalDate yesterday = today.minusDays(1);
LocalDate tomorrow = today.plusDays(1);
LocalDate nextWeek = today.plusWeeks(1);
LocalDate nextYear = today.plusYears(1);

LocalTime time = LocalTime.now().plusHours(1).plusMinutes(45);
```

A korábbi feladat megoldása

- Megoldható egyetlen metódushívással:

```
public static long countDays(LocalDate start, LocalDate end) {
    return start.until(end, ChronoUnit.DAYS);
}

LocalDate then = LocalDate.of(1848, Month.MARCH, 15);
System.out.println(then);                    // 1848-03-15

LocalDate now = LocalDate.now();             // 2017-01-15
System.out.println(now);

System.out.println(countDays(then, now));  // 61667
```

Részleges dátumok

- Részleges dátumok kezelésére szolgálnak a `MonthDay`, `Year` és `YearMonth` osztályok.
- Példa a használatukra:

```
MonthDay monthDay = MonthDay.of(Month.JUNE, 22);
System.out.println(monthDay);    // --06-22

LocalDate date = monthDay.atYear(1975);
System.out.println(date);        // 1975-06-22

LocalDate dateOfBirth = LocalDate.of(1809, 1, 19);

Year birthyear = Year.from(dateOfBirth);
System.out.println(birthyear);   // 1809

MonthDay birthday = MonthDay.from(dateOfBirth);
System.out.println(birthday);    // --01-19
```

Időtartamok (1)

- `java.time.Period`: https://docs.oracle.com/javase/8/docs/api/java/time/Period.html
  - Időtartam kifejezése években, hónapokban és napokban.
- `java.time.Duration`: https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html
  - Időtartam kifejezése másodpercekben és nanomásodpercekben.

Időtartamok (2)

- Példa a használatukra:

```
LocalDate dateOfBirth = LocalDate.of(2012, Month.APRIL, 4);
LocalDate today = LocalDate.now();
System.out.println(today);// 2017-01-15

Period age = dateOfBirth.until(today);
System.out.println(age);                    // P4Y9M11D
System.out.println(age.getYears());         // 4
System.out.println(age.getMonths());        // 9
System.out.println(age.getDays());          // 11
System.out.println(age.isNegative());       // false

Instant t1 = Instant.now();
System.out.println(t1);              // 2017-01-15T10:23:18.678Z

Duration duration = Duration.ofMinutes(50).plusSeconds(48);
System.out.println(duration);        // PT50M48S

Instant t2 = t1.plus(duration);
System.out.println(t2);              // 2017-01-15T11:14:06.678Z
```

Dátum és idő módosítás (1)

- A `withXXX(...)` metódusok szolgálnak módosításra:

```
LocalDate date = LocalDate.of(2012, 4, 4);
System.out.println(date);  // 2012-04-04

date = date.withMonth(6);
System.out.println(date);  // 2012-06-04

date = date.withDayOfMonth(22);
System.out.println(date);  // 2012-06-22

date = date.withYear(1975);
System.out.println(date);  // 1975-06-22
```

Dátum és idő módosítás (2)

- A `java.time.temporal.TemporalAdjuster` egy dátum és idő módosítására szolgáló funkcionális interfész, mely a dátum és idő osztályok `with(TemporalAdjuster)` metódusával használható.
  - Beépített implementációk a `java.time.temporal.TemporalAdjusters` osztályban.

Dátum és idő módosítás (3)

- Példa:

```
LocalDate date = LocalDate.of(1848, Month.MARCH, 15);
System.out.println(date); // 1848-03-15

System.out.println(date
    .with(TemporalAdjusters.lastDayOfMonth()));
// 1848-03-31

System.out.println(date
    .with(TemporalAdjusters.firstDayOfNextMonth()));
// 1848-04-01

System.out.println(date
    .with(TemporalAdjusters.firstInMonth(DayOfWeek.SUNDAY)));
// 1848-03-05

System.out.println(date
    .with(TemporalAdjusters.nextOrSame(DayOfWeek.WEDNESDAY)));
// 1848-03-15
```

Lekérdezések (1)

- A `java.time.temporal.TemporalQuery` egy információkinyerésre szolgáló funkcionális interfész, mely a dátum és idő osztályok `query(TemporalQuery)` metódusával használható.
  - Beépített implementációk a `java.time.temporal.TemporalQueries` osztályban.
- Példa:

```
LocalTime now = LocalTime.now();
System.out.println(now.query(TemporalQueries.precision()));
// Nanos
```

Lekérdezések (2)

- Példa:

```
public static Boolean isClassBreak(TemporalAccessor temporal) {
    int hour = temporal.get(ChronoField.HOUR_OF_DAY);
    int minute = temporal.get(ChronoField.MINUTE_OF_HOUR);
    if (hour < 8 || hour > 18) return false;
    return (hour % 2 == 1) && (minute >= 40 && minute < 60);
}

LocalTime time = LocalTime.of(14, 15);
System.out.println(time.query(QueryExample::isClassBreak));
// false

time = LocalTime.of(15, 42);
System.out.println(time.query(QueryExample::isClassBreak));
// true
```

Lekérdezések (3)

- Példa (folytatás):

```
Stream.of("08:45", "09:40", "09:59", "10:00", "11:50",
       "17:45", "19:30")
    .map(LocalTime::parse)
    .map(QueryExample::isClassBreak)
    .forEach(System.out::println);
// false
// true
// true
// false
// true
// true
// false
```

Formázás és elemzés

- Példa:

```
LocalDate date = LocalDate.parse("13 Jan 2017",
    DateTimeFormatter.ofPattern("dd MMM yyyy")
        .withLocale(Locale.ENGLISH));
System.out.println(date); // 2017-01-13

DateTimeFormatter formatter = new DateTimeFormatterBuilder()
    .appendValue(ChronoField.DAY_OF_MONTH, 2)
    .appendLiteral(' ')
    .appendText(ChronoField.MONTH_OF_YEAR, TextStyle.SHORT)
    .appendLiteral(' ')
    .appendValue(ChronoField.YEAR).toFormatter(Locale.ENGLISH);

date = LocalDate.parse("13 Jan 2017", formatter);
System.out.println(date); // 2017-01-13

System.out.println(LocalDate.now().format(formatter));  // 15 Jan 2017
```

Időszámításunk előtti dátumok

- Példa a használatra és ábrázolásra:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("y G",
    Locale.ENGLISH);

Year year = Year.parse("753 BC", formatter);// Róma alapításának éve
System.out.println(year);                   // -752
System.out.println(formatter.format(year)); // 753 BC
```

Instant

- Egy adott időpillanatot ábrázoló osztály. https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html
  - Alkalmas például például alkalmazásokban események időbélyegeinek rögzítéséhez.
- Példa a használatra:

```
Instant now = Instant.now();
System.out.println(now);      // 2017-01-15T11:05:10.525Z

~~LocalDateTime dateTime =~~
   ~~LocalDateTime.from(now);~~   // java.time.DateTimeException

LocalDateTime dateTime = LocalDateTime.ofInstant(now,
   ZoneId.systemDefault());
System.out.println(dateTime); // 2017-01-15T12:05:10.525
```

Clock (1)

- Hozzáférést biztosít az aktuális időpillanathoz, dátumhoz és időhöz egy adott időzónában. https://docs.oracle.com/javase/8/docs/api/java/time/Clock.html
  - Lásd a dátum és idő osztályok `now(Clock)` metódusát, mely a paraméterként adott órát használja az objektum létrehozáshoz.
  - Ez hasznos teszteléshez.

Clock (2)

- Példa a használatra:

```
Clock clock1 = Clock.systemDefaultZone();
Clock clock2 = Clock.systemUTC();
Clock clock3 = Clock.system(ZoneId.of("Asia/Vladivostok"));

System.out.println(clock1);   // SystemClock[Europe/Budapest]
System.out.println(clock2);   // SystemClock[Z]
System.out.println(clock3);   // SystemClock[Asia/Vladivostok]

System.out.println(LocalTime.now(clock1));   // 14:01:26.204
System.out.println(LocalTime.now(clock2));   // 13:01:26.209
System.out.println(LocalTime.now(clock3));   // 23:01:26.209
```

Clock (3)

- Példa a használatra: az idő megállítása (jól jöhet például teszteléshez)

```
Instant instant = Instant.parse("2017-01-13T00:00:00Z");
Clock clock = Clock.fixed(instant, ZoneId.of("UTC"));

System.out.println(ZonedDateTime.now(clock));
// 2017-01-13T00:00Z[UTC]

Thread.sleep(5000);   // 5 másodperc várakozás

System.out.println(ZonedDateTime.now(clock));
// 2017-01-13T00:00Z[UTC]

Instant now = Instant.now(clock);
System.out.println(now.equals(instant)); // true
```

Clock (4)

- A jó gyakorlat az aktuális időpillanatot használó alkalmazások számára:

```
public class MyClass {

    private Clock clock;  // dependency inject

    public void process(LocalDate date) {
        if (date.isBefore(LocalDate.now(clock))) {
            // ...
        } else {
            // ...
        }
    }

}
```

Interoperabilitás a régi dátum és idő osztályokkal

- Lásd az alábbi metódusokat:
  - `java.util.Calendar.toInstant()`
  - `java.util.GregorianCalendar.from(java.time.ZonedDateTime)`
  - `java.util.GregorianCalendar.toZonedDateTime()`
  - `java.util.Date.fromInstant(java.time.Instant)`
  - `java.util.Date.toInstant()`
  - `java.util.TimeZone.toZoneId()`

Példa

- Ha április 1.-én helyi idő szerint 11:45-kor indul egy repülőjárat New Yorkból Tokióba, melynek menetideje 14 óra 10 perc, mikor érkezik Tokióba helyi idő szerint?

```
ZonedDateTime departure = LocalDate.of(2017, Month.APRIL, 1)
    .atTime(11, 45)
    .atZone(ZoneId.of("America/New_York"));
System.out.println(departure);
// 2017-04-01T11:45-04:00[America/New_York]

Duration duration = Duration.ofHours(14).plusMinutes(10);
System.out.println(duration);
// PT14H10M

ZonedDateTime arrival =
departure.withZoneSameInstant(ZoneId.of("Asia/Tokyo"))
    .plus(duration);
System.out.println(arrival);
// 2017-04-02T14:55+09:00[Asia/Tokyo]
```

Példa

- Határozzuk meg, hogy adott év mely napjai estek péntek 13-ára.

```
public static List<LocalDate> friday13th(int year) {
    return IntStream.range(1, 13)
        .mapToObj(i -> LocalDate.of(year, i, 13))
        .filter(d -> d.getDayOfWeek().equals(DayOfWeek.FRIDAY))
        .collect(Collectors.toList());
}

System.out.println(friday13th(2016));
// [2016-05-13]

System.out.println(friday13th(2017));
// [2017-01-13, 2017-10-13]
```

Példa (1)

- Határozzuk meg, hogy melyik nap esik egy adott napot követően legközelebb péntek 13-ára.
  - Naiv megoldás `LocalDate` objektumokra.
  - Általános megoldás `TemporalAdjuster` implementációként.

Példa (2)

- Naiv megoldás `LocalDate` objektumokra:

```
public static LocalDate nextFriday13th(LocalDate date) {
    int dayOfMonth = date.getDayOfMonth();
    if (dayOfMonth != 13) {
        if (dayOfMonth > 13)
            date = date.plusMonths(1);
        date = date.withDayOfMonth(13);
    }
    while (date.getDayOfWeek() != DayOfWeek.FRIDAY)
        date = date.plusMonths(1);
    return date;
}

LocalDate date = LocalDate.of(2017, Month.JUNE, 22);
System.out.println(nextFriday13th(date));  // 2017-10-13
```

Példa (3)

- Általános megoldás:

```
public class NextFriday13th implements TemporalAdjuster {

    public Temporal adjustInto(Temporal temporal) {
        int dayOfMonth = temporal.get(ChronoField.DAY_OF_MONTH);
        if (dayOfMonth != 13) {
            if (dayOfMonth > 13)
                temporal = temporal.plus(1, ChronoUnit.MONTHS);
            temporal = temporal.with(ChronoField.DAY_OF_MONTH, 13);
        }
        while (temporal.get(ChronoField.DAY_OF_WEEK) != 5)
            temporal = temporal.plus(1, ChronoUnit.MONTHS);
        return temporal;
    }

}
```

Példa (4)

- Általános megoldás (folytatás): az előbbi `TemporalAdjuster` révén módosítani lehet `LocalDate`, `LocalDateTime`, `ZonedDateTime` és `OffsetDateTime` objektumokat is.

```
LocalDate date = LocalDate.of(2017, Month.JUNE, 22);
System.out.println(date.with(new NextFriday13th()));
// 2017-10-13

LocalDateTime dateTime = date.atStartOfDay();
System.out.println(dateTime.with(new NextFriday13th()));
// 2017-10-13T00:00
```

ThreeTen-Backport

- Java programkönyvtár, mely a JSR-310 API funkcionalitását nyújtja Java SE 6-hoz és Java SE 7-hez. http://www.threeten.org/threetenbp/
  - Licenc: *New BSD License*
- Csomagok: `org.threeten.bp` és alcsomagjai
  - Az osztályok a lehető legnagyobb mértékben próbálnak megfelelni a JSR-310 API-nak.
- Lásd még:
  - Stephen Colebourne: *ThreeTen-Backport vs Joda-Time*. July 1, 2014. http://blog.joda.org/2014/07/threeten-backport-vs-joda-time.html

ThreeTen-Extra

- A Java SE 8 dátum- és időkezelő osztályait továbbiakkal kiegészítő programkönyvtár. http://www.threeten.org/threeten-extra/
  - Licenc: *New BSD License*
- Példa a használatra:

```
LocalDate date = LocalDate.of(2017, Month.JANUARY, 13);
System.out.println(date);                    // 2017-01-13
System.out.println(date.with(Temporals.nextWorkingDay())); // 2017-01-16

YearQuarter quarter = YearQuarter.from(date);
System.out.println(quarter);                 // 2017-Q1

Interval interval = Interval.of(Instant.now(), Duration.ofHours(1));
System.out.println(interval.getStart()); // 2017-01-15T11:26:53.423Z
System.out.println(interval.getEnd());   // 2017-01-15T12:26:53.423Z
System.out.println(interval.contains(Instant.now().plus(10,
    ChronoUnit.MINUTES)));                   // true
```

További olvasnivaló

- `java.time` (Java Platform SE 8)
  https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html
- *The Java™ Tutorials – Trail: Date Time*
  https://docs.oracle.com/javase/tutorial/datetime/