Java 8 - práce s datem a časem

Michal Lauer
4it101

2019/2020 - Letní semestr

Obsah

1 Problémy ve verzí Java 7

Práce s datem a časem byla ve verzi 7 velmi kostrbatá, a to nejen z toho důvodu, že s datem a časem pracovala pomocí jedné třídy. Nebyla stejná časová zóna (defaultně UTC, ale mohla se lišit podle JVM), instance byli neintuitivní, indexování a pozice byli nekonzistentní a byl "proměnlivý"(ang. *mutable*). Pojďme si uvést nějaké příklady

- **Date(int rok, int měsíc, int den)**
  - rok - počítá se od roku 1900. Proto pro vytvoření roku 2020 jste museli zadat rok 120 (2020 - 1900)
  - Měsíc - počítání začíná na 0, proto prosinec = 11
- **Proměnlivost**
  - Při kopírování se zkopíroval pouze odkaz na dané datum - úprava jednoho data ovlivňila i datum druhé
- **SQL**
  - Pro vložení data do SQL se musel Date převést na java.sql.Date, který reprezentuje **pouze jeden den**.

Jak je vidět, datum a čas v Javě 7 byl velmi chaotický, proto přechod na nové API byl nevyhnutelný. (Oracle, 2020a)

2 Java 8

V této sekci si představíme nejzákladnější třídy, které využijeme při práci s časem a datem . *LocalDate*, *LocalTime*, *LocalDateTime*. Rozdělujeme zde údaj na údaj s časovou zónou (**"zoned"** a údaj bez časové zóny *"local"*. Pokud budeme chtít zjistit délku mezi daty, použijeme jednu z následujících tříd

- *Duration* - pro rozdíl v čase
- *Period* - rozdíl v datu
- *ChronoUnit* - rozdíl mezi datem a časem najednou (lze použít i jednotlivě)

Dále existuje možnost čas i datum formátovat pomocí třídy *DateTimeFormatter*, kde nalezneme předdefinované styly. Pokud nebudou stačit přednastavené styly, můžeme si vytvořit vlastní. (AlBlue, 2020)

Pro vytvoření se používají z pravidla statické metody. Lze získat aktuální čas/datum, čas/datum z jiné časové zóny nebo čas/datum dle vlastního uvážení (vlastní nastavění, podle čas. zóny).Třídy si prve musíme naimportovat

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
```

a tak dále pro ostatní třídy.

V textu níže si představíme již zmíněné třídy a metody k nim. Pokud chcete dále rozšířit své znalosti, podívejte se do oficiální dokumentace pro Javu 8.

2.1 LocalDate

LocalDate je třída, která představuje **pouze informaci o datu.**. Třída kromě data (YYYY-MM-DD) ukládá informace o dni v kalendáři (den v týdnu, den v roce aj.) Pro vytvoření instance použijeme jeden z následujících atributů či metod

- **MIN**
  - Vrátí nejmenší možné datum (-999999999-01-01).
- **MAX**
  - Vrátí největší možné datum (+999999999-12-31)
- **now([*ZoneId* zóna])**[1]
  - získá aktuální datum v časovém pásmu. Můžeme předat nepovinný parametr časové zóny
- **of(int rok, int měsíc, int den)**
  - vytvoří datum podle stanovených hodnot
- **parse(CharSequence text [, DateTimeFormater formát])**
  - vytvoří datum podle zadaného textu a popř. formátu [2]
- **ofYearDay(int rok, int denVRoce)**
  - Vrátí den podle počtu uplynulých dní od začátku roku

[1] *LocalDate* sice s čas. zónou nepracuje, ale v tomto případě jenom předáváme časovou zónu pomocí parametru - vzniklá instance bude bez časové zóny

[2] defaultně se bere v potaz formátování ISO-8601 - YYYY-MM-DD

Na instanci můžeme použít například tyto metody

- **minusYears(long odečíst), plusYears(long odečíst)** atd.
  - odečte/přičte požadovanou složku data
- **withDayOfMonth(int měsíc), withDayOfyear(int denVRoce)**
  - nastaví den v měsíci nebo den v roce
- **withMonth(int měsíc), withYear(int denVRoce)**
  - nastaví měsíc nebo rok
- **getDayOfMonth(), getMonthValue()**
  - vrátí enum složku dnu v týdnu či měsíce
- **compareTo(*LocalDate* datum2)**
  - porovná pár dat, tj. datum1 - datum2
  - -1 -> datum2 je v budoucnu; 0 -> data jsou stejná; > 0 –> datum2 je v minulosti

Pro reprezentaci měsíce můžeme použít buď čísla, nebo enum Months. Metody se změní tímto způsobem

- **withMonth(Month měsíc)**
- **.of(int rok, Month měsíc, int den)**

dále můžeme reprezentovat den v týdnu, což je hlavně užitečně s metodou get

- **getDayOfWeek(Weekday den)**

(Oracle, 2020e)

2.1.1 LocalDate kódově

```
LocalDate ldNow = LocalDate.now();
LocalDate ldAustralia = LocalDate.now(ZoneId.of("Australia/Canberra"));
System.out.println("Den u tebe: " + ldNow.getDayOfWeek());
System.out.println("Den v Austrálii: " + ldAustralia.getDayOfWeek());
System.out.println("Jsi pozadu? --> " + ldNow.isBefore(ldAustralia));

LocalDate ldCustom = LocalDate.of(1234, 7,18);
System.out.println("Den v týdnu: " + ldCustom.getDayOfWeek());
System.out.println("Den v roce: " + ldCustom.getDayOfYear());
System.out.println("Měsíc v roce: " + ldCustom.getMonth());
System.out.println("Délka měsíce: " + ldCustom.lengthOfMonth());
System.out.println("Éra: " + ldCustom.getEra());
```

2.2 LocalTime

Jak jsme se již dozvěděli, v osmé verzi Javy je čas, datum a časová zóna rozdělená. Ve třídě *LocalTime* proto najdeme **pouze lokální čas (bez čas. zóny) s přesností na nanosekundu.** Jelikož má třída *LocalTime* privátní konstruktor, musíme vytvořit proměnnou a inicializovat tam hodnotu, kterou nám třída nabízí. Prve se podíváme na 4 vlastnosti třídy

- **MIN** - nejmenší podporovaná hodnota (00:00)
- **MAX** - největší podporovaná hodnota (23.59.99)
- **MIDNIGHT** - čas o půlnoci (stejné jako MIN)
- **NOON** - čas o pravém poledni

Proměnnou dále můžeme naplnit metodami

- **now([*ZoneId* zóna])**[3]
  - získá aktuální čas. Můžeme předat nepovinný parametr časové zóny
- **of(int hodina, int minuta [, int sekunda] [, int nanosekunda])**
  - vytvoří čas podle stanovených hodnot
- **parse(CharSequence text [, DateTimeFormater formát])**
  - vytvoří čas podle zadaného textu a popř. formátu [4]

S časem můžeme manipulovat použitím metod

- **minusHours(long odečíst), plusHours(long odečíst)** atd.
  - odečte/přičte požadovaný čas
- **withHour(int hodina), withMinute(hodina)** atd.
  - nastaví požadovaný časový údaj
- **getHour(), getMinute()** atd.
  - vrátí požadovanou časovou jednotku
- **compareTo(*LocalTime* čas2)**
  - porovná dva časové údaje, tj. čas1 - čas2
  - -1 -> čas2 je v budoucnu; 0 -> časy jsou stejné; > 0 –> čas2 je v minulosti

(Oracle, 2020g)

[3] *LocalTime* sice s čas. zónou nepracuje, ale v tomto případě jenom předáváme časovou zónu pomocí parametru - vzniklá instance bude bez časové zóny

[4] defaultně se bere v potaz formátování ISO-8601 - HH:MM[:SS]

2.2.1 LocalTime kódově

```
LocalTime ltNow = LocalTime.now();
System.out.println("ltNow = " + ltNow);
ltNow.minusHours(9);
ltNow.plusMinutes(53);
System.out.println("Čas u tebe: " + ltNow);

LocalTime ltLondon = LocalTime.now(ZoneId.of("Europe/London"));
System.out.println("Čas v Londýně: " + ltLondon);
ltLondon = ltLondon.withHour(LocalTime.now().getHour());
System.out.println("Čas v Londýně ale jakoby v Praze: " + ltLondon);

if (LocalTime.now().compareTo(LocalTime.NOON) == -1){
    System.out.println("Měl by jsi jít ještě spát, je před polednem!");
}
else if (LocalTime.now().compareTo(LocalTime.NOON) == 1){
    System.out.println("Už jdi spát, je dávno po poledni!");
}
else{
    System.out.println("Dej si šlofíka, je pravé poledne!");
}
```

2.3 LocalDateTime

Jak již bylo naznačeno, čas a datum je v Java 8 rozdělen do tříd *LocalDate* a *LocalTime*. Třída *LocalDateTime* kombinuje obě třídy a přidává nám možnost uložit datum i čas do jednoho objektu pořád **(bez časové zóny).** Lze vytvořit jednou z následujících metod

- **now([*ZoneId* zóna])**[5]
  - získá aktuální datum a čas v časovém pásmu. Můžeme předat nepovinný parametr časové zóny
- **of(int rok, int měsíc, int den, int hodina, int minuta [, int sekunda, int nanosekunda]**
  - vytvoří instanci s požadovaným datem a časem
- **if(*LocalDate* datum, *LocalTime* čas)**
  - vytvoří *LocalDateTime* podle tříd *LocalDate* a *LocalTime*
- **parse(CharSequence text [, *DateTimeFormatter* formát])**
  - instance se vytvoří podle zadaného textu, popř. formátu

Pokud bychom měli již zadaný *LocalTime* a chtěli k němu přidat datum (vytvořit třídu *LocalDateTime*), můžeme použít na čas funkci čas.atDate(*LocalDate* datum). Samozřejmě nový *LocalDateTime* musíme někam uložit

```
LocalDateTime ldt = LocalTime.now().atDate(LocalDate.now());
```

V opačném případě - tedy tvoření *LocalDateTime* z data - lze použít metoda .atTime(int hodina, int minuta[, int sekunda, int nanosekunda]), resp.

```
LocalDateTime ldt = LocalDate.now().atTime(12,30,34,12);
```

Finální instance třídy obsahuje mnoho metod - mimo jiné i metody ze tříd *LocalDate* a *LocalTime*. Rád bych ale zmínil metody, které sice **obsahují třídy *LocalDate* a *LocalTime*, ale nebyly zmíněny.**

- **isAfter(ChronologicalDateTime druhýLDT)**
- **isBefore(ChronologicalDateTime druhýLDT)**
- **isEqual(ChronologicalDateTime druhýLDT)**
  - ChronologicalDateTime je interface, ale můžeme místo něj dosadit třídu *LocalDateTime*

(Oracle, 2020f)

[5] *LocalDateTime* sice s čas. zónou nepracuje, ale v tomto případě jenom předáváme časovou zónu pomocí parametru - vzniklá instance bude bez časové zóny

2.3.1 LocalDateTime kódově

```
LocalTime lt = LocalTime.now();
LocalDate ld = LocalDate.now();
LocalDateTime ltdOfInstances = LocalDateTime.of(ld, lt);
LocalDateTime ltdSingapore = LocalDateTime.now(ZoneId.of("Asia/Singapore"));
System.out.println("Čas a datum u tebe: " + ltdOfInstances);
System.out.println("Čas a datum v Singapuru: " + ltdSingapore);

ltdOfInstances = ltdOfInstances.plusDays(6);
System.out.println(ltdOfInstances.isBefore(ltdSingapore));
```

2.4 ZonedDateTime

Třída *ZonedDateTime* obsahuje datum, čas a přidanou časovou zónu. Pro určení časové zóny budeme používat funkci .of(String časZóna) (Oracle, 2020j)

```
ZoneId zID = ZoneId.of("Europe/Prague");
```

Nyní stačí vytvořit instanci pomocí metod, které jsme si již představili, akorát přidáme parametr pro časovou zónu. Důležité je poznamenat, že *ZoneID* **převede čas jenom s kombinací .now(*ZoneID* id).**. Při vytváření metodou .of() přidá pouze časový rozdíl od GMT/UTC jako text, proto je důležité **přiřadit správnou časovou zónu!** Podívejme se na příklad

```
ZonedDateTime ldtNow = ZonedDateTime.now();
//2020-04-26T17:35:45.284+02:00[Europe/Prague]
ZonedDateTime ldtNowNY = ZonedDateTime.now(ZoneId.of("America/New_York"));
//2020-04-26T11:35:45.284-04:00[America/New_York]
LocalDate ld = LocalDate.now();
LocalTime lt = LocalTime.now();
ZoneId zId = ZoneId.of("America/New_York");
ZonedDateTime ldtOfZoneID = ZonedDateTime.of(ld, lt, zId);
//2020-04-26T17:35:45.284-04:00[America/New_York]
```

Ostatní instanční funkce jsou více méně stejné, akorát stačí přidat *ZoneId* parametr. Pro příklady se podívejte na *LocalDateTime* v kódu.(Oracle, 2020i)

2.5 Duration, Period a ChronoUnit

Duration

Pomocí *duration* můžeme měřit čas. Použijeme metodu .between(*LocalTime* lt1, *LocalTime* lt2) a dále metodu .getSeconds(). Pro zjištění minut či hodin musíme hodnotu vydělit. (Oracle, 2020c)

```
LocalTime lt1 = LocalTime.now();
LocalTime lt2 = LocalTime.now().plusHours(4);
Duration d = Duration.between(lt1, lt2);
System.out.println("Rozdíl v sekundách: " + d.getSeconds());
System.out.println("Rozdíl v minutách: " + d.getSeconds()/60);
System.out.println("Rozdíl v hodinách: " + d.getSeconds()/3600);
```

Period

Třída *Period* se používá na měření rozdílu v datech. Použijeme metodu .between(*LocalDate* ld1, *LocalDate* ld2). Výsledek lze zjistit pomocí .getDays(), getMonths(), .getYears(). (Oracle, 2020h)

```
LocalDate ld1 = LocalDate.now();
LocalDate ld2 = LocalDate.now().plusYears(3).plusDays(40);
Period p = Period.between(ld1, ld2);
System.out.println("Rozdíl ve dnech: " + p.getDays());
System.out.println("Rozdíl ve měsících: " + p.getMonths());
System.out.println("Rozdíl ve letech: " + p.getYears());
```

ChronoUnit

Pokud chceme zjistit rozdíl mezi *LocalDateTime*, slouží nám k tomu třída *ChronoUnit*. Díky ní lze zjistit nejen rozdíl mezi *LocalDateTime*, ale i **mezi *LocalDate* a *LocalTime***. Pro zjištění výsledku použijeme *ChronoUnits.hodnota, kterou chceme zjistit*. Dále klasicky použijeme .between(*LocalDateTime* ldt1, *LocalDateTime* ldt2). (Oracle, 2020d)

```
LocalTime lt1 = LocalTime.now(ZoneId.of("America/New_York"));
LocalDate ld1 = LocalDate.now(ZoneId.of("Australia/Canberra"));
LocalDateTime ldt1 = LocalDateTime.now();
LocalDateTime ldt2 = LocalDateTime.now().plusDays(62).plusHours(13);
System.out.println("Rozdíl ve dnech: " + ChronoUnit.DAYS.between(ldt1, ldt2));
System.out.println("Rozdíl v hodinách: " + ChronoUnit.HOURS.between(ldt1, ldt2));
System.out.println("Rozdíl v sekundách: " + ChronoUnit.SECONDS.between(lt1, ldt1));
System.out.println("Rozdíl ve dnech: " + ChronoUnit.DAYS.between(ld1, ldt2));
```

2.6 DateTimeFormatter

Pro formátování datumu a času používáme třídu *DateTimeFormatter*. Máme zde několik přednastavených formátů

- ISO_LOCAL_DATE –> 2011-12-03
- ISO_LOCAL_TIME –> 10:15:30
- a mnoho dalších

Dále můžeme nastavit vlastní formátování díky metodě *DateTimeFormatter.ofPattern(**pattern**)*. V kódu to může vypadat následovně

```
    LocalDateTime ldt = LocalDateTime.now();
    System.out.println(ldt.format(DateTimeFormatter.ofPattern("H:MM YYYY/MM/dd")));
```

Všechny zkratky naleznete v dokumentaci.

Hlídání chyb

Pokud chceme převést text (např. vstup od uživatele) na časovou či datumovou hodnotu, používáme metodu .parse(). Nikdy si ale nemůžeme být jisti, jestli uživatel zadal správný formát. Proto při parsování textu používáme klauzuli try-catch s výjimkou DateTimeParseException. Použití je následující

```
Scanner scan = new Scanner(System.in);
System.out.println("Zadejte datum ve formátu dd-MM-yyyy");
String date = scan.nextLine();
LocalDate ld = null;
try {
    ld = LocalDate.parse(date, DateTimeFormatter.ofPattern("dd-MM-yyyy"));
} catch (DateTimeParseException e){
    System.out.println(e.getMessage());
}
```

(Oracle, 2020b)

3 Příklady k procvičení

3.1 Procvičení LocalDate

Zadání

Uživatel vloží jeden pár dat ve formátu ”YYYY-MM-DD”. Zjistěte rozdíl mezi datumy ve dnech.

Rada: Scanner, try-catch

Řešení

```
Scanner scan = new Scanner(System.in);
System.out.println("Zadejte první datum ve formátu YYYY-MM-DD");
String d1 = scan.nextLine();
System.out.println("Zadejte druhé datum ve formátu YYYY-MM-DD");
String d2 = scan.nextLine();
try{
    LocalDate ld2 = LocalDate.parse(d2);
    LocalDate ld1 = LocalDate.parse(d1);
    System.out.println("Rozdíl ve dnech je: " + Period.between(ld1, ld2).getDays());
} catch(DateTimeParseException e){
    System.out.println(e.getLocalizedMessage());
}
```

3.2 Procvičení LocalTime

Zadání

Uživatel vloží čas ve formátu HH:MM. Po ověření správného formátu bude uživatel dotázán, aby přidal takový čas, který po přičtení k původnímu vytvoří poledne. Pokud zadá špatný čas, bude znovu dotázán na přidání času.

Řešení

```
Scanner scan = new Scanner(System.in);
System.out.println("Zadejte čas ve formátu HH:MM >> ");
String t1 = scan.nextLine();
LocalTime lt = null;
try{
    lt = LocalTime.parse(t1);
} catch(DateTimeParseException e){
    System.out.println(e.getMessage());
}
System.out.println("Zadaný čas je: " + lt);
while(!lt.equals(LocalTime.NOON)){
    System.out.println("Zadejte čas tak, aby bylo poledne >> ");
    String novyT = scan.nextLine();
    try{
        LocalTime temp = LocalTime.parse(novyT);
        lt = lt.plusHours(temp.getHour()).plusMinutes(lt.getMinute());
    } catch (DateTimeParseException e){
        System.out.println(e.getMessage());
    }
    if (lt.equals(LocalTime.NOON)){
        System.out.println("Super, splněno!");
    } else{
        System.out.println("Výsledný čas: " + lt + "\nZkus to ještě jednou:)");
    }
}
```

3.3 Procvičení LocalDateTime, DateTimeFormatter

Zadání

Vypište všechny dny v aktuálním měsíci. Pokud se den rovná aktuálnímu dni, vypište celé datum ve formátu ”dd-MM-YYYY”. Dále vypište každou hodinu. Pokud se hodina rovná aktuální hodině, vypište čas ve formátu ”hh:mm”.

Rada: for, *DateTimeFormatter*

Řešení

```
LocalDateTime dnes = LocalDateTime.now();
for(int i = 1; i <= dnes.getMonth().maxLength(); i++){
    if(dnes.getDayOfMonth() == i){
        System.out.println(">> " + dnes.format(DateTimeFormatter.ofPattern("dd-MM-YYYY")));
    } else {
        System.out.println(dnes.withDayOfMonth(i).getDayOfMonth());
    }
}
for(int i = 0; i < 24;i++){
    if(dnes.getHour() == i){
        System.out.println(">> " + dnes.format(DateTimeFormatter.ofPattern("hh:mm")));
    } else {
        System.out.println(dnes.withHour(i).getHour());
    }
}
```

3.4 GitHub

Pro lepší práci s kódem jsou všechny ukázky kódu a příklady k procvičení nahrané na mém github účtu.

Odkazy

ALBLUE. *datetime - What's wrong with Java Date & Time API?* [online] [cit. 2020-04-27]. Dostupné z: https://stackoverflow.com/questions/1969442/whats-wrong-with-java-date-time-api.

ORACLE, a. *Date (Java Platform SE 7)* [online] [cit. 2020-05-02]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/Date.html.

ORACLE, b. *DateTimeFormatter (Java Platform SE 8)* [online] [cit. 2020-04-27]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html.

ORACLE, c. *Duration (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html.

ORACLE, d. *ChronoUnit (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/temporal/ChronoUnit.html.

ORACLE, e. *LocalDate (Java Platform SE 8)* [online] [cit. 2020-04-25]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html.

ORACLE, f. *LocalDateTime (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html.

ORACLE, g. *LocalTime (Java Platform SE 8)* [online] [cit. 2020-04-25]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html.

ORACLE, h. *Period (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/Period.html.

ORACLE, i. *ZonedDateTime (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/ZonedDateTime.html.

ORACLE, j. *ZoneId (Java Platform SE 8)* [online] [cit. 2020-04-26]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/time/ZoneId.html.