Javaプログラミング実習

28. 日時の扱い

株式会社ジードライブ

今回学ぶこと

- Date and Time APIを利用した日時の扱い
- Dateクラスを利用した日時の扱い(補足)

Date and Time API とは

- java.timeパッケージに含まれる日時を扱うためのクラス群で、ISO-8601という国際規格をベースにしている

クラス	説明
Instant	日時の情報を1970年1月1日を基準とした秒数で扱うクラス
Duration	時間ベース(秒単位)で期間を扱うためのクラス。日付ベース(日単位)で期間を扱うクラスとしてPeriodが存在する
LocalDateTime	日時情報をタイムゾーンを考慮しない形で扱うことのできるクラス。関連するクラスとして、日付のみを扱うLocalDate、時間のみを扱うLocalTimeなども存在する
ZonedDateTime	タイムゾーンを含む日時情報を扱うためのクラス
DateTimeFormatter	日時を任意の表記形式で扱うためのクラス

Instantクラス

- ある瞬間の日時情報を保持するクラス
  - 1970年1月1日0時0分0秒からの経過時間をナノ秒の単位まで保持している
  - 1秒 = 1,000,000,000ナノ秒

UNIX時間と呼ぶ

```
Instant instant = Instant.now(); //現在日時の情報をもつオブジェクトを作成
System.out.println(instant.getEpochSecond() + "秒");
System.out.println(instant);
```

表示例

```
1651838270秒
2022-05-06T11:57:50.651110300Z
```

1970/1/1からの経過秒数

ISO 8601形式：最後のZはUTC(協定世界時)を表す

Instantクラス

- メソッドの一例

メソッド	説明
`static Instant now()`	現在日時の情報をもつInstantを返す ※ コンストラクタに相当するメソッド
`static` `Instant ofEpochSecond(long sec)`	引数として指定された日時の情報をもつInstantを返す。引数として渡す日時は、1970/1/1からの秒数で表す ※ 引数をもつコンストラクタに相当
`static` `Instant parse(String dateTime)`	引数として指定された日時の情報をもつInstantを返す。引数として渡す日時は、 「2020-12-05T10:30:00.00Z」という形式で表す ※ 引数をもつコンストラクタに相当
`long getEpochSecond()`	オブジェクトのもつ日時情報を1970/1/1からの秒数で返す
`String toString()`	オブジェクトのもつ日時情報をISO-8601形式で返す

Instantクラス：https://docs.oracle.com/javase/jp/17/docs/api/java.base/java/time/Instant.html

Durationクラス

- 期間を扱うためのクラス
  - 時間ベースで期間の情報を保持する
  - 日時の差分を算出する際に利用することができる

メソッド	説明

`static Duration ofSeconds(long sec)`
`static Duration ofMinutes(long min)`
`static Duration ofHours(long hours)`
`static Duration ofDays(long days)`
	引数として指定された期間(秒、分、時間、日数)の情報をもつDurationを返す
	※ コンストラクタに相当するメソッド

`static Duration between(Temporal from, Temporal to)`	引数として指定された2つの日時の期間の情報をもつDurationを返す ※ 引数をもつコンストラクタに相当

`long toSeconds()`
`long toMinutes()`
`long toHours()`
`long toDays()`
	期間を整数(秒数、分数、時間数、日数)で返す

`String toString()`	期間をISO-8061ベースの文字列表現で返す

Durationクラス：https://docs.oracle.com/javase/jp/17/docs/api/java.base/java/time/Duration.html

Durationクラス

- 利用例：処理時間を計測する

```
Instant ins1 = Instant.now();

StringBuilder sb = new StringBuilder("あいうえお");
for(int i = 1; i <= 100000; i++) {
  sb.append("あいうえお");
}

// 同様の処理をStringで行う場合
/* String str = "あいうえお";
for(int i = 1; i <= 100000; i++) {
  str += "あいうえお";
} */

Instant ins2 = Instant.now();
System.out.println(Duration.between(ins1, ins2));
```

2つのInstantの差分を算出

表示例

```
PT0.0091963S
```

練習

- 練習28-1

LocalDateTimeクラス

- タイムゾーンのない日時情報を扱うクラス
  - 以下のような関連クラスも存在する

クラス	説明
LocalDate	日付のみを扱う
LocalTime	時間のみを扱う
Year	年のみを扱う
YearMonth	年月のみを扱う
Month	月のみを扱う
MonthDay	月日のみを扱う

これらのクラスは共通のインターフェースを実装しているので、
LocalDateTimeと同じように扱うことができる

LocalDateTimeクラス

- メソッドの一例

メソッド	説明
`static LocalDateTime now()`	現在日時の情報をもつLocalDateTimeを返す ※ コンストラクタに相当するメソッド
`static LocalDateTime of(int year, int month, int day, int hour, int minute, int second)`	引数(年、月、日、時、分、秒)で指定された日時の情報をもつLocalDateTimeを返す。第６引数(秒)は省略可能 ※ 引数をもつコンストラクタに相当
`LocalDate toLocalDate()` `LocalTime toLocalTime()`	LocalDateTimeオブジェクトから 日付や時刻を抽出するためのメソッド

`int getYear()`
`int getMonthValue()`
`int getDayOfMonth()`
	LocalDateTimeオブジェクトから
	年、月、日(1～31)を抽出するためのメソッド

LocalDateTimeクラス: https://docs.oracle.com/javase/jp/17/docs/api/java.base/java/time/LocalDateTime.html

LocalDateTimeクラス

- メソッドの一例

メソッド	説明
boolean equals(Object datetime) boolean isBefore(LocalDateTime datetime) boolean isAfter(LocalDateTime datetime)	引数として渡される日時と、比較を行うためのメソッド
LocalDateTime plusSeconds(long seconds) LocalDateTime plusMinutes(long minutes) LocalDateTime plusHours(long hours) LocalDateTime plusDays(long days) LocalDateTime plusMonths(long months) LocalDateTime plusYears(long years)	日時情報を操作するためのメソッドで、時間数や日数などを加算したLocalDateTimeオブジェクトを返す。 減算用の minus***() というメソッドも存在する
LocalDateTime withSecond(int sec) LocalDateTime withMinute(int minute) LocalDateTime withHour(int hour) LocalDateTime withDayOfMonth(int day) LocalDateTime withMonth(int month) LocalDateTime withYear(int year)	日時情報を操作するためのメソッドで、時間や日付などの一部分を変更したLocalDateTimeオブジェクトを返す

LocalDateTimeクラス

- 利用例：「3年前の今日」と「東京五輪開会式」の日数の差分を算出

```
// 現在日時の情報をもつLocalDateTimeの作成
LocalDateTime day1 = LocalDateTime.now();

// 3年前に設定
day1 = day1.plusYears(-3);

// 東京五輪の開会式(2021年7月23日 午前8:00)
LocalDateTime day2 = LocalDateTime.of(2021, 7, 23, 8, 0, 0);

// 差分の確認
System.out.println(Duration.between(day1, day2).toDays() + "日");
```

表示例

807日

練習

- 練習28-2

ZonedDateTimeクラス

- タイムゾーンを含めた日時情報を扱うクラス
  - 一部の国や地域で導入されているサマータイムの情報も持ち合わせている
  - 基本的なメソッドは、LocalDateTimeと同じ

メソッド	説明
`static ZonedDateTime of(int year, int month, int day, int hour, int minute, int sec, int nano, ZoneId zoneId)`	引数(年、月、日、時、分、秒、ナノ秒、タイムゾーン)で指定された日時の情報をもつZonedDateTimeを返す。 第１～７引数は、LocalDateTimeにまとめることも可能
`ZoneOffset getOffset()` `ZoneId getZone()`	ZonedDateTimeオブジェクトから、標準時との差やタイムゾーンの情報を返すメソッド
`ZonedDateTime withZoneSameInstant(ZoneId zoneId)`	タイムゾーンを変更し、時刻を調整したZonedDateTimeを返す
`ZonedDateTime withZoneSameLocal(ZoneId zoneId)`	タイムゾーンを変更したZonedDateTimeを返す(タイムゾーン変更に伴う時刻調整はしない)

ZonedDateTimeクラス：https://docs.oracle.com/javase/jp/17/docs/api/java.base/java/time/ZonedDateTime.html

ZonedDateTimeクラス

- 利用例：タイムゾーンの変更

```
ZonedDateTime datetime = ZonedDateTime.now();
System.out.println(datetime);

// タイムゾーンを変更し、時刻を調整する
ZonedDateTime london1 =
datetime.withZoneSameInstant(ZoneId.of("Europe/London"));
System.out.println(london1);

// タイムゾーンのみを変更する
ZonedDateTime london2 =
datetime.withZoneSameLocal(ZoneId.of("Europe/London"));
System.out.println(london2);
```

表示例

```
2022-05-07T16:34:45.616971+09:00[Asia/Tokyo]
2022-05-07T08:34:45.616971+01:00[Europe/London]
2022-05-07T16:34:45.616971+01:00[Europe/London]
```

練習

- 練習28-3

DateTimeFormatterクラス

- 日時の出力形式を指定するクラス

メソッド	説明
`static DateTimeFormatter ofPattern(String pattern)`	コンストラクタに相当するメソッドで、引数で日時のパターンを指定する。 パターンは「**y**年**M**月**d**日」のような特殊なアルファベットを含む文字列で表す
`String format(TemporalAccessor temporal)`	引数として、日時情報のオブジェクト(LocalDateTimeなど)を受け取り、フォーマットした文字列を返す

パターンで使用可能なアルファベット

y	年
M	月
d	日

H	時間。0～23
h	時間。1～12
m	分

s	秒
E	曜日
G	年号

DateTimeFormatterクラス：https://docs.oracle.com/javase/jp/17/docs/api/java.base/java/time/format/DateTimeFormatter.html

DateTimeFormatterクラス

- 利用例：日時をフォーマットして表示する

```
LocalDateTime today = LocalDateTime.now();

DateTimeFormatter dtf1 =
        DateTimeFormatter.ofPattern("y/MM/dd HH:mm:ss");
DateTimeFormatter dtf2 =
        DateTimeFormatter.ofPattern("y年M月d日(E)");

System.out.println(dtf1.format(today));
System.out.println(dtf2.format(today));
```

「MM」のように重ねて記述すると
1桁の場合、頭に0が付く

表示例

```
2022/05/07 17:25:27
2022年5月7日(土)
```

和暦の扱い

- 日本の年号を出力するには、DateTimeFormatter#withChronology()で暦を指定する

```
LocalDateTime dt = LocalDateTime.now();

// 通常形式での元号の表記
DateTimeFormatter dtf1 =
  DateTimeFormatter.ofPattern("Gy年M月d日")
    .withChronology(JapaneseChronology.INSTANCE);
System.out.println(dtf1.format(dt));

// 省略形式での元号の表記
DateTimeFormatter dtf2 =
  DateTimeFormatter.ofPattern("GGGGGy年M月d日")
    .withChronology(JapaneseChronology.INSTANCE);
System.out.println(dtf2.format(dt));
```

表示例

令和4年5月7日
R4年5月7日

練習

- 練習28-4
- 練習28-5

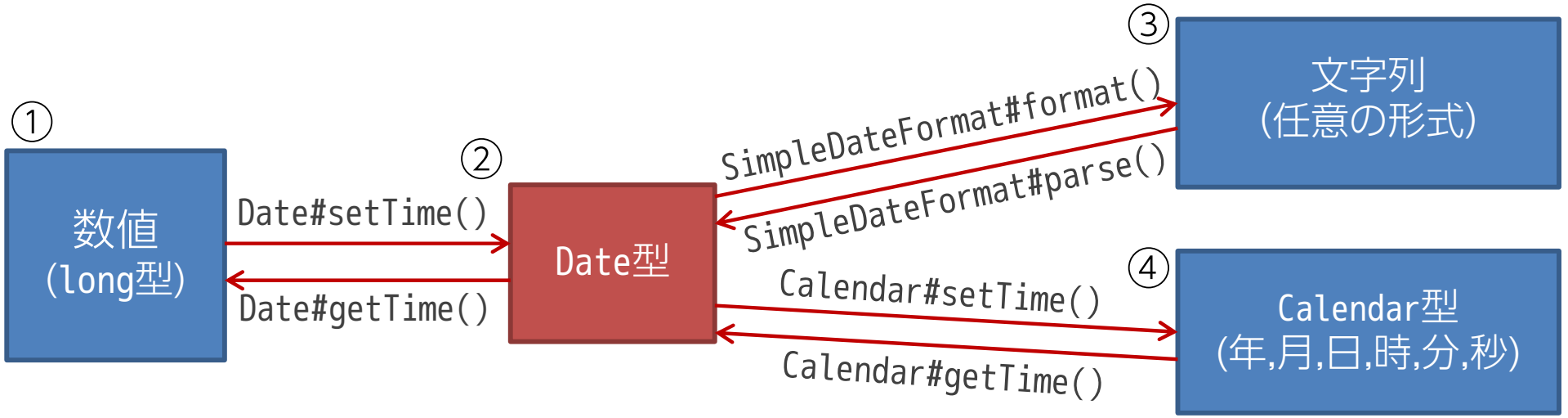
[補足]
java.util.Dateによる日時の扱い

Dateを中心とした日時の扱い

- Javaでは、以下のように日時を扱うことができる

① 数値（long型）
② Dateクラス
} コンピュータが扱いやすい

③ 文字列(SimpleDateFormatクラス)
④ Calendarクラス
} 人間にとってわかりやすく、扱いやすい



数値（long型）

- 1970年1月1日0時00分00秒からのミリ秒数で日時を示す

  UNIXエポックと呼ぶ

  - System.currentTimeMillis() で現在日時の情報を取得することができる(long型)
  - 1ミリ秒 ＝ 1／1000秒　　1000ミリ秒 ＝ 1秒

例：処理時間を計測する

```
long start = System.currentTimeMillis();
String tx = "";
for(int i = 0; i < 10000; i++) {
   tx += "あいうえお";
}
long end = System.currentTimeMillis();

long process = end - start;
System.out.println("処理時間：" + process + "ミリ秒");
```

Dateクラス

- 特定の日時情報を扱うことのできるクラス
  - 内部でlong型の数値を扱う
  - java.util.Dateをインポートして利用する

  ⇒java.sql.Dateはデータベース利用時に使うクラス

```
// 現在日時の情報をもつインスタンスを生成
Date date = new Date();
System.out.println(date);

// ミリ秒の情報を取得
long millis = date.getTime();
System.out.println(millis);

// 現在より10日前の日時をセット
long past = System.currentTimeMillis() - (10 * 86400 * 1000);
date.setTime(past);  // ミリ秒で設定
System.out.println(date);
```

1日の秒数
(24時間×60分×60秒)

ミリ秒数への変換

```
Thu Dec 20 13:44:23 JST 2018
1545281063804
Mon Dec 10 13:44:23 JST 2018
```

SimpleDateFormatクラス

- 任意の書式で日時(Date)を表現できる
  - y, M, d など特別な意味をもつアルファベット(次頁参照)を使い、フォーマットを指定する
  - SimpleDateFormat#format(Date date) を利用する
- 文字列をDate型に変換する
  - SimpleDateFormat#parse(String source) を利用する

```
public static void main(String[] args) throws ParseException {
  Date now = new Date();
  SimpleDateFormat fmt = new SimpleDateFormat("y年MM月dd日(E)");
  System.out.println(fmt.format(now));

  // 文字列からDate型へ
  Date date = fmt.parse("2018年12月12日(水)");
  System.out.println(date);
}
```

この部分は
後日説明します

フォーマットを
一致させる

```
2018年12月20日(木)
Wed Dec 12 00:00:00 JST 2018
```

SimpleDateFormatの書式

- フォーマットで使えるアルファベット
  - MM や dd のようにすると2桁表記になる

記号	意味	例
y	年	2021
M	月	7　(MMとした場合は、07となる)
d	日	27
E	曜日	土
a	午前／午後	午前
H	時 (0 - 23)	20
h	時 (1 - 12)	8
m	分	53
s	秒	23

練習

- 練習28-6

Calendarクラス

- 様々な日時関連処理メソッドを備えた高機能なクラス
  - ミリ秒だけでなく、年・月・日・時・分・秒ごとに日時を操作することができる
- new演算子ではなく、Calendar#getInstance()を使ってインスタンスを取得する

```
Calendar c = Calendar.getInstance();
```

Calendarのメソッド

メソッド	意味
`void set(int year, int month, int date, int hour, int minute, int second)`	Calendarオブジェクトに日時を設定する(monthは0が1月)
`void set(int year, int month, int date)`	Calendarオブジェクトに日付を設定する
`Date getTime()`	このCalendarオブジェクトの時間値を表すDateオブジェクトを返す
`void add(int field, int amount)` 次頁のフィールドを参照	指定された時間量を指定されたカレンダーフィールドに加算（マイナスの場合は減算）する
`boolean after(Objet when)`	このCalendarオブジェクトが指定されたObjectより後の時間なら true を返す
`boolean before(Object when)`	このCalendarオブジェクトが指定されたObjectより前の時間なら true を返す

Calendarクラスのフィールド

- 以下は全てstaticフィールド

フィールド	意味
YEAR	年
MONTH	月
DATE / DAY_OF_MONTH	日
HOUR	時（12時間）
HOUR_OF_DAY	時（24時間）
MINUTE	分
SECOND	秒
MILLISECOND	ミリ秒

Calendarを使った日時計算

例：2020年1月31日と、その1ヵ月後の日付を表示する

```
// 表示用フォーマット
SimpleDateFormat fmt = new SimpleDateFormat("y/M/d (E)");

// 2020年1月31日の情報をもつCalendarオブジェクトを生成
Calendar c = Calendar.getInstance();
c.set(2020, 0, 31);  // 0 ⇒ 1月
Date d = c.getTime(); // 表示用にDateを取得
System.out.println(fmt.format(d));

// Calendarに1カ月足す
c.add(Calendar.MONTH, 1);
d = c.getTime();
System.out.println(fmt.format(d));
```

```
2020/1/31 (金)
2020/2/29 (土)
```

練習

- 練習28-7

和暦の扱い

- SimpleDateFormatオブジェクトの生成時に**ja_JP_JP**というロケール(地域設定)を指定すると和暦(明治～)を使用可能になる
  - 上記設定により、SimpleDateFormatの書式で **GGGG** が漢字での元号を表すようになる
  - **G** とした場合は H (平成), S (昭和) などの省略表記を表すようになる

和暦の扱い

例：西暦と和暦による表示

```
Date now = new Date();

// 西暦のフォーマットで出力
SimpleDateFormat seireki = new SimpleDateFormat("GGGGy年M月d日");
System.out.println(seireki.format(now));

// 和暦用ロケールの設定
// 引数1：言語　引数2：国　引数3：variant(異形)
Locale locale = new Locale("ja", "JP", "JP");
SimpleDateFormat wareki = new SimpleDateFormat("GGGGy年M月d日", locale);
System.out.println(wareki.format(now));
```

```
西暦2020年12月21日
令和2年12月21日
```

練習

- 練習28-8