3-13.java.textパッケージ

■ java.textパッケージ

はじめに

java.textパッケージには、日付や数値に代入された

```
int dateTime = 2018-01-01 00:00:00;
String dateTime = "2018/01/01 00:00:00";
```

などの日付・時刻形式データを、任意の日付・時刻フォーマットに変換するクラスが提供されています。
簡単に言うと「このデータは、この形式で日時データとして解釈できますよ」と教えてあげるための書き方になります。

Step1: SimpleDateFormat

日時フォーマットの変換には `SimpleDateFormat` クラスを使用します。
フォーマットパターン の指定により、様々なフォーマットで日付・時刻を表示できるようになります。

下記の表は、`SimpleDateFormat` クラスで用意されているパターン文字一覧です。

パターン文字	説明	例
G	紀元	AD
y	年	2004, 04
M	月	July, Jul, 07
w	年における週	15
W	月における週	2
D	年における日	167
d	月における日	22
F	月における曜日	3
E	曜日	Saturday, Sat
a	午前/午後	AM
H	一日における時(0〜23)	2
k	一日における時(1〜24)	23
K	午前/午後の時(0〜11)	6
h	午前/午後の時(1〜12)	10
m	分	15
s	秒	11
S	ミリ秒	908
z	タイムゾーン	Pacific Standard Time, PST, GMT-08:00
Z	タイムゾーン	-0800

?

上記のパターン文字を使用して、日時を整形し表現します。
特に、年月日と時間を表す

パターン文字	説明	例
y	年	2004, 04
M	月	July, Jul, 07
D	年における日	167
H	一日における時(0～23)	2
m	分	15
s	秒	11

の6パターンはよく使うので要チェックです。

では、日時形式のデータをどのように **日付・時刻データ** へ変換するのか実際に見ていきましょう。

※intやStringに代入されているデータは、あくまでも **「数値」と「文字列」** です。
「2018/01/01 23:59:00」といった日付・時刻形式でデータが代入されていても、それは
「日時の形をしているデータ」に過ぎないので、Javaは日時データとして認識してくれません。
日時データとして認識させてあげるには、
必ず **「日時データ」へ変換する処理** を書いてあげる必要があるのです。

例)
冒頭で登場したStringクラスの日時形式データ、

```
String dateTime = "2018/01/01 00:00:00"; //yyyy/MM/dd HH:mm:ss形式
```

を日時データに変換したい場合は、

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
```

といったように、SimpleDateFormatの引数へ **日時データに変換したいデータと同じパターン** を記述し、
Stringクラスの日時形式データを **日時データへ変換する準備** をします。

※1：今回は「String dateTime = "2018/01/01 00:00:00"」日時データに変換したいため、
同一パターンの「yyyy/MM/dd HH:mm:ss」を記述しています。

※2：ここでパターンの記述を間違えると、parseExceptionエラー(解析ができませんといった旨のエラー)が
発生してしまうので気をつけましょう。

次に、作成したSimpleDateFormat型の変数「sdf」を使用して、
日時データであることを表す **Date型** への変換を行います。

```
Date date = sdf.parse(dateTime);
```

SimpleDateFormat.parseの引数へ、Date型に変換したい値を渡すことで、
引数に指定したデータのパターン

```
String dateTime = "2018/01/01 00:00:00"; // yyyy/MM/dd HH:mm:ss形式
```

と、SimpleDateFormatに記述したデータパターン

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
```

が一致する場合、Date型に変換後の値が代入されます。

最後に、変換後の値が代入されたDate型変数「date」の中身を見てみましょう。

```
System.out.println(date);
```

【出力結果】

```
Mon Jan 01 00:00:00 JST 2018
```

Stringクラスの日時形式データが、Date型の日時データに変換されました。

【サンプルコード】

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class test {
    public static void main(String[] args) throws ParseException {

        String dateTime = "2018/01/01 00:00:00";

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
        Date date = sdf.parse(dateTime);

        System.out.println(date);
    }
}
```

以上が、日時型データを日時データに変換する処理の簡単な流れとなります。

一例として、実際の作業手順としては、、、

1. ユーザーが入力した日時をStringで受け取る
2. 受け取った値を `SimpleDateFormat` で解析してDate型に **変換**
3. 変換したDate型変数から年だけを取り出し、取り出した年を用いて何らかの処理を記述する

といった流れになります。

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class test {
    public static void main(String[] args) throws ParseException {

        SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
        SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy");
        Date date = new Date();
        System.out.println(sdf1.format(date));
            // 現在時刻が出力されます
        System.out.println(sdf2.format(date));
            // 年のみ出力されます
    }
}
```

【出力結果】

```
2018/12/22 08:22:13
2018
```

上記の例は一例ですが、SimpleDateFormatは

- 年や月といったデータを処理中で使用したい場合
- ライブラリを使用する際、所定の形式へ変換する必要がある場合

に使用するケースがほとんどです。

SimpleDateFormatを使用して作成したデータの用途「こうやって使うこともあるんだな」くらいに覚えておけば大丈夫ですが、
記述する機会も多いため、今のうちにしっかりと把握しておきましょう。

おまけ DateTimeFormatterについて

`SimpleDateFormat` とよく似た機能として、 `DateTimeFormatter` が存在します。
日時パターンへ変換するという処理は `SimpleDateFormat` と同様ですが、いくつか異なる点もありますのでご紹介しましょう。

SimpleDateFormatを使用する際は、

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
```

といったように自インスタンスを生成しますが、DateTimeFormatterの場合は

```
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                                          //ofPattern()に任意のフォーマットパターンを指定します。
```

の一行でパターンの指定が可能です。 **newしない分消費メモリの節約ができます** 。
その後 `LocalDateTime` を使用することで、上記で作成したフォーマットパターンを日時型データに反映させ、
日時型データを日時データへ変換します。

```
LocalDateTime ldt1 = LocalDateTime.parse("2018/12/29 12:00:00", dtf);
```

ちなみに、.parse内の第一引数は固定値でなくても大丈夫です。

```
String date = "2018/12/30 13:00:00";

ocalDateTime ldt2 = LocalDateTime.parse(date, dtf);
                                    // Stringクラスの変数dateを指定しています。
```

【出力結果】

```
2018/12/29 12:00:00
2018/12/30 13:00:00
```

書き方は多少異なっていても、それぞれ日時形式のデータが出力されていることを確認できます。

処理の流れは `SimpleDateFormat` も `DateTimeFormatter` も大きな違いはありませんし、
現場によって使用を推奨されるクラスが異なるケースもあります 。
(`SimpleDateFormat` は `new` でインスタンスを生成するため、
性能要件を満たすには `DateTimeFormatter` を使うほうが望ましい…などなど)
どちらを使うことになっても迷わないよう、「こういう書き方もあるんだな」と
今のうちに覚えておいてしまいましょう。

課題

提出課題はありませんので、一通り学習が終わったら次の章に進んで下さい。

最終更新日時: 2022年 08月 21日(日曜日) 17:01