format()	×	保持情報を "2010/04/12" のような文字列に変換する。文字列書式は DateTimeFormatter で指定する
get ～ ()	×	格納する年や月の情報を取得する。「～」部分には、Year、Month、DayOfMonth、Hour、Minute、Second、Nano 等が入る
isAfter() / isBefore()	×	引数で渡したインスタンスとの前後関係を判定する
plus ～ () / minus ～ ()	×	指定したぶんだけ未来または過去の時点を返す。「～」部分には、Years、Months、Days、Hours、Minutes、Seconds、Nanos 等が入る
plus() / minus()	×	指定した時間間隔（後述の Period や Duration）のぶんだけ未来または過去の時点を返す

基本的に now()、of()、parse() を使ってインスタンスを取得する。new はできないことに注意してほしい。

これらのメソッドを用いたサンプルプログラムをリスト 2-4 に示します。

リスト 2-4　各種日時クラスのメソッド利用例

Main.java

```
import java.time.*;
import java.time.format.*;

public class Main {
  public static void main(String[] args) {
    // 文字列からLocalDateを生成
    DateTimeFormatter f = DateTimeFormatter
        .ofPattern("yyyy/MM/dd");
    LocalDate d =
      LocalDate.parse("2011/08/21", f);

    // 1000日後を計算する
    d = d.plusDays(1000);
    String str = d.format(f);
    System.out.println("1000日後は" + str);

    // 現在日付との比較
    LocalDate now = LocalDate.now();
    if (now.isAfter(d)) {
      System.out.println("nowはdより新しい");
    }
  }
}
```


2.3.5　時間や期間を表すクラス

Java で表現しやすくなったのは、日付や時刻だけじゃないんだ。

Java には長らく「2 つの日付の間隔」や「2 つの時刻の間隔」を格納する標準的な API がありませんでした。そこで Java8 以降で加わったのが Duration クラスと Period クラスです。

主に「時・分・秒」の単位で収まる比較的短い間隔を表す場合は Duration を使いましょう。一方、サマータイムや閏年なども考慮しながら日数ベースで期間を管理する必要がある場合は、Period クラスを使ってください。

両クラスとも、静的メソッド between() や ofDays()、ofMonths() を使うことで、インスタンスを取得することができます。また、表 2-3 にあるように、LocalDateTime 等の plus() や minus() メソッドの引数として利用することもできます（リスト 2-5）。

リスト 2-5　Period クラスの利用例

Main.java

```
import java.time.*;
import java.time.format.*;

public class Main {
  public static void main(String[] args) {
```

```
6      LocalDate d1 = LocalDate.of(2012,1,1);
7      LocalDate d2 = LocalDate.of(2012,1,4);
8
9      // 3日間を表すPeriodを2通りの方法で生成
10     Period p1 = Period.ofDays(3);
11     Period p2 = Period.between(d1, d2);
12
13     // d2のさらに3日後を計算する
14     LocalDate d3 = d2.plus(p2);
15   }
16 }
```

Java7 以前でも便利な日時 API を利用するには

2.3 節で紹介している API は Java8 以降でしか利用できません。しかし、Joda-Time ライブラリ（http://www.joda.org/joda-time）を利用すれば、Java7 以前でもほぼ同等のクラスを利用できます。

2.4 この章のまとめ

基本的な日付の取り扱い

- Java における日付情報は基本的に java.util.Date 型で扱う。
- その他、必要に応じて long 値、6 つの int、String 型に変換して用いる。
- 「年・月・日・時・分・秒」の 6 つの int 値から Date インスタンスを得るためには、Calendar クラスを使う。
- Date インスタンスの内容を任意の書式で文字列に整形したい場合は、SimpleDateFormat クラスを使う。

Java8 以降で利用可能な API

- Java8 以降で java.time パッケージに追加された新しい API を用いることで、より便利かつ安全に日付や時刻を扱うことができる。
- 厳密な時刻を格納するためには、Instant クラスや ZonedDateTime クラスを用いる。一方、日常的に利用する日時情報の格納には、LocalDateTime クラスが適している。
- 「年・月・日・時・分・秒・ナノ秒・タイムゾーン」のうち、いくつかを保持せず曖昧な日時を表現するために、LocalDate や YearMonth など複数のクラスが準備されている。
- Duration クラスや Period クラスを用いることで、2 つの日時の間隔を格納することができる。

2.5 練習問題

練習 2-1

main() メソッドのみを持つクラス Main を定義し、以下の手順を参考にして「現在の 100 日後の日付」を「西暦 2011 年 09 月 24 日」という形式で表示するプログラムを作成してください。なお、回答にあたり、Date 型や Calendar 型の各 API の使用方法については、API リファレンスや『スッキリわかる Java 入門』などを参照してください。

① 現在の日時を Date 型で取得します。
②取得した日時情報を Calendar にセットします。
③Calendar から「日」の数値を取得します。
④取得した値に 100 を足した値を Calendar の「日」にセットします。
⑤Calendar の日付情報を Date 型に変換します。
⑥SimpleDateFormat を用いて、指定された形式で Date インスタンスの内容を表示します。

練習 2-2

練習 2-1 と同様の動作を行うプログラムを、Java8 以降で利用可能になった新しい Time API を用いて記述し直してください。

2.6 練習問題の解答

練習 2-1 の解答

Main.java

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
public class Main {
  public static void main(String[] args) {
    // ①現在の日時をDate型で取得
    Date now = new Date();
    Calendar c = Calendar.getInstance();
    // ②取得した日時情報をCalendarにセット
    c.setTime(now);
    // ③Calendarから「日」の情報を取得
    int day = c.get(Calendar.DAY_OF_MONTH);
    // ④取得した値に100を足してCalendarの「日」にセット
    day += 100;
    c.set(Calendar.DAY_OF_MONTH, day);
    // ⑤Calendarの日付情報をDate型に変換
    Date future = c.getTime();
    // ⑥指定された形式で表示
    SimpleDateFormat f =
        new SimpleDateFormat("西暦yyyy年MM月dd日");
    System.out.println(f.format(future));
  }
}
```

練習 2-2 の解答

Main.java

```
1  import java.time.*;
2  import java.time.format.*;
3
4  public class Main {
5    public static void main(String[] args) {
6      LocalDate now = LocalDate.now();
7      LocalDate future = now.plusDays(100);
8      DateTimeFormatter f =
9        DateTimeFormatter.ofPattern("西暦yyyy年MM月dd日");
10     System.out.println(future.format(f));
11   }
12 }
```

7 行目で 100 日後の日付を求めるために、plusDays() メソッドを利用している部分については、次のように Period を用いても構いません。

```
7      LocalDate future = now.plus(Period.ofDays(100));
```

第3章

コレクション

Java には、ひとまとまりのデータを扱う方法として「配列」というデータ構造がありますが、本格的で複雑なアプリケーション開発では、それ以外により多くのデータ構造を駆使することになるでしょう。

この章では、強力で柔軟なデータ構造の利用を可能にする API クラス群「コレクションフレームワーク」を紹介します。