1 对象与对象变量

要想使用对象，首先必须构造对象，并指定其初始状态，然后对对象应用方法。

使用**构造器**构造新实例。构造器是一种特殊的方法，用来构造并初始化对象。下面以标准 Java 库中的`Date`类为例，介绍构造器的使用方法。

构造器的名字与类名相同，在构造器前面加上`new`操作符，就可以构造一个对象。例如：

```
new Date();
```

这个表达式构造了一个新对象，并被初始化为当前的日期和时间。

可以将新对象传递给一个方法，例如：

```
System.out.println(new Date());
```

也可以对新对象应用一个方法，例如：

```
String s = new Date().toString(); // 返回日期的字符串描述
```

在上面的例子中，构造的对象只能使用一次。为了让对象可以多次使用，需要将对象存放在一个变量中。例如：

```
Date birthday = new Date();
```

对象变量不等于对象。对象变量并不包含对象，它只是引用一个对象。当对象变量没有引用任何对象时，不能在这个变量上使用方法，否则会产生编译错误。例如：

```
Date deadline;
String s = deadline.toString(); // 错误
```

必须首先初始化变量，可以让它引用一个新构造的对象，也可以引用一个已有的对象。例如：

```
Date deadline = new Date();
Date date = deadline; // 此时变量 deadline 和 date 引用同一个对象
```

可以显式地将对象变量设置为`null`，指示这个对象变量没有引用任何对象。例如：

```
Date time = null;
```

在 Java 中，任何对象变量的值都是对存储在另外一个地方的某个对象的引用，`new` 操作符的返回值也是一个引用。可以将 Java 中的对象变量看做类似于 C++ 的对象指针。

2 `LocalDate` 类

上面提到的 `Date` 类用来表示时间点，而 `LocalDate` 类用日历表示法表示日期。下面列出 `LocalDate` 类的一些常用方法。

```
/* java.time.LocalDate */

// 构造一个表示当前日期的对象
static LocalDate now()

// 构造一个表示给定日期的对象
static LocalDate of(int year, int month, int dayOfMonth)

// 得到当前对象指定日期的年
int getYear()

// 得到当前对象指定日期的月
int getMonthValue()

// 得到当前对象指定日期的日
int getDayOfMonth()

// 得到当前对象指定日期的星期数，返回 DayOfWeek 类的一个实例
// 对 DayOfWeek 类的实例调用 getValue() 方法得到 1~7 之间的一个数，1 表示星期一，7 表示星期日
DayOfWeek getDayOfWeek()

// 生成当前对象指定日期之后 daysToAdd 天的日期
LocalDate plusDays(long daysToAdd)

// 生成当前对象指定日期之前 daysToSubtract 天的日期
LocalDate minusDays(long daysToSubtract)
```

只访问对象而不修改对象的方法称为**访问器方法**，修改对象的方法称为**更改器方法**。例如，上面的 `plusDays()` 方法得到当前对象指定日期之后 n 天的日期，但并不改变当前对象，因此它是访问器方法。

应用实例：打印当前月的日历

```java
import java.time.*;

public class Calendar
{
    public static void main(String[] args)
    {
        LocalDate date = LocateDate.now(); // 构造一个对象，并用当前日期初始化
        int month = date.getMonthValue(); // 得到当前月份
        int today = date.getDayOfMonth(); // 得到当前日期

        date = date.minusDays(today - 1); // 将 date 设置为这个月的第一天
        int value = date.getDayOfWeek().getValue(); // 得到这个月的第一天是星期几

        System.out.println("Mon Tue Wed Thu Fri Sat Sun"); // 打印表头
        for (int i = 1; i < value; i++)
        {
            System.out.print("    "); // 打印第一行的缩进，内容为 4 个空格
        }

        while (date.getMonthValue() == month) // 还在这个月内，继续打印
        {
            System.out.printf("%3d", date.getDayOfMonth()); // 打印日期值
            if (date.getDayOfMonth() == today)
            {
                System.out.print("*"); // 如果是当前日期，打印 * 号
            }
            else
            {
                System.out.print(" "); // 否则，打印空格
            }

            date = date.plusDays(1); // 加一天
            if (date.getDayOfWeek().getValue() == 1)
            {
                System.out.println(); // 如果到达新一周的第一天，则换行
            }
        }

        if (date.getDayOfWeek().getValue() != 1)
        {
            System.out.println();
        }
    }
}
```

运行结果（2020 年 11 月）：

```
Mon Tue Wed Thu Fri Sat Sun
                        1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15*
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30
```

在这个例子中，不需要考虑这个月有几天，不需要考虑星期数如何计算，只需要调用方法，而不用考虑方法的具体实现，这体现了封装性的优势。