8.8 Composition

A class can have references to objects of other classes as members. This is called **composition** and is sometimes referred to as a ***has-a* relationship**. For example, an `AlarmClock` object needs to know the current time *and* the time when it's supposed to sound its alarm, so it's reasonable to include *two* references to `Time` objects in an `AlarmClock` object. A car *has-a* steering wheel, a break pedal, an accelerator pedal and more.

Class `Date`

This composition example contains classes `Date` (Fig. 8.7), `Employee` (Fig. 8.8) and `EmployeeTest` (Fig. 8.9). Class `Date` (Fig. 8.7) declares instance variables `month`, `day` and `year` (lines 5–7) to represent a date. The constructor receives three `int` parameters. Lines 15–18 validate the `month`—if it's out-of-range, lines 16–17 throw an exception. Lines 21–25 validate the `day`. If the day is incorrect based on the number of days in the particular `month` (except February 29th which requires special testing for leap years), lines 23–24 throw an exception. Lines 28–29 perform the leap year testing for February. If the month is February and the day is 29 and the `year` is not a leap year, lines 30–31 throw an exception. If no exceptions are thrown, then lines 34–36 initialize the `Date`'s

instance variables and line 38 output the `this` reference as a `String`. Since `this` is a reference to the current `Date` object, the object's `toString` method (lines 42–44) is called *implicitly* to obtain the object's `String` representation. In this example, we assume that the value for `year` is correct—an industrial-strength `Date` class should also validate the year.

```
   1  // Fig. 8.7: Date.java
   2  // Date class declaration.
   3
   4  public class Date {
   5     private int month; // 1-12
   6     private int day; // 1-31 based on month
   7     private int year; // any year
   8
   9     private static final int[] daysPerMonth =
  10        {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31
  11
  12     // constructor: confirm proper value for mont
  13     public Date(int month, int day, int year) {
  14        // check if month in range
  15        if (month <= 0 || month > 12) {
  16           throw new IllegalArgumentException(
  17              "month (" + month + ") must be 1-12"
  18        }
  19
  20        // check if day in range for month
  21        if (day <= 0 ||
  22           (day > daysPerMonth[month] && !(month =
  23           throw new IllegalArgumentException("day
  24              ") out-of-range for the specified mo
  25        }
  26
  27        // check for leap year if month is 2 and d
  28        if (month == 2 && day == 29 && !(year % 40
  29              (year % 4 == 0 && year % 100 != 0)))
  30           throw new IllegalArgumentException("day
```

```
31                ") out-of-range for the specified mo
                  32          }
                       33
           34           this.month = month;
             35           this.day = day;
            36           this.year = year;
                         37
38           System.out.printf("Date object constructor
                    39       }
                         40
41       // return a String of the form month/day/year
         42       public String toString() {
43          return String.format("%d/%d/%d", month, da
                    44       }
                     45    }
```

Fig. 8.7

`Date` class declaration.

Class `Employee`

Class `Employee` (Fig. 8.8) has reference-type instance variables `firstName` (`String`), `last-Name` (`String`), `birthDate` (`Date`) and `hireDate` (`Date`), showing that a class can have as instance variables references to objects of other classes. The `Employee` constructor (lines 11–17) takes four parameters representing the first name, last name, birth date and hire date. The objects referenced by the parameters are assigned to an `Employee` object's instance variables.

When `Employee`'s `toString` method is called, it returns a `String` containing the employee's name and the `String` representations of the two `Date` objects. Each of these `String`s is obtained with an *implicit* call to the `Date` class's `toString` method.

```
1   // Fig. 8.8: Employee.java
2   // Employee class with references to other objec
3
4   public class Employee {
5      private String firstName;
6      private String lastName;
7      private Date birthDate;
8      private Date hireDate;
9
10     // constructor to initialize name, birth date
11     public Employee(String firstName, String last
12        Date hireDate) {
13        this.firstName = firstName;
14        this.lastName = lastName;
15        this.birthDate = birthDate;
16        this.hireDate = hireDate;
17     }
18
19     // convert Employee to String format
20     public String toString() {
21        return String.format("%s, %s Hired: %s Bir
22           lastName, firstName, hireDate, birthDat
23     }
24  }
```

Fig. 8.8

`Employee` class with references to other objects.

Class `EmployeeTest`

Class `EmployeeTest` (Fig. 8.9) creates two `Date` objects to represent an `Employee`'s birthday and hire date, respectively. Line 8 creates an `Employee` and initializes its instance variables by passing to the constructor two `String`s (representing the `Employee`'s first and last names) and two `Date` objects (representing the birthday and hire date). Line 10 *implicitly* invokes the `Employee`'s `toString` method to display the values of its instance variables and demonstrate that the object was initialized properly.

```
1   // Fig. 8.9: EmployeeTest.java
2   // Composition demonstration.
3
4   public class EmployeeTest {
5      public static void main(String[] args) {
6         Date birth = new Date(7, 24, 1949);
7         Date hire = new Date(3, 12, 1988);
8         Employee employee = new Employee("Bob", "Bl
9
10        System.out.println(employee);
11     }
12  }
```

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob Hired: 3/12/1988 Birthday: 7/24/1949
```

Fig. 8.9

Composition demonstration.