14 DATE AND CALENDAR CLASSES

Terry Marris 26 May 2001

14.1 OBJECTIVES

By the end of this lesson the student should be able to

- understand the concept of a calendar
- get today's date from the host operating system
- display dates in a number of formats
- add days, months or years to a given date
- determine the interval between two dates
- use *Calendar*, *Date* and *DateFormat* methods

14.2 PRE-REQUISITES

The student should be comfortable with implementing associations (Chapter 13).

14.3 PREVIEW

There are many kinds of calendars - the Hebrew calendar, the Chinese calendar, the Hindu calendar. The Gregorian calendar is the one that most of us use everyday.

Creating and manipulating dates requires the use of three classes.

The *Date* class is used to represent a specific instant in time.

The *Calendar* class is used to convert a *Date* to a set of integer fields representing day, month and year, and back again, according the rules of some specific calendar e.g. *GregorianCalendar*.

The *DateFormat* class is used to format *Dates* in a number of styles suitable for display e.g. 26-May-01 and 26th May 2001.

We look at some example programs. We list the more useful fields and methods of the *Date*, *Calendar*, *GregorianCalendar* and *DateFormat* classes. We look at a simple application using dates.

14.4 INTRODUCTION

Working with dates is an important part of any business activity. Questions such as when does an account become overdue for payment, what bills must I pay in the next month and how many years have I got left before the lease on my premises runs out need to be answered. As programmers, we need to be able to supply the answers.

14.5 DISPLAYING A DATE

The first example program retrieves today's date from the operating system and displays it on the monitor.

We need to use the

- *GregorianCalendar* class to create a calendar instance initialised with today's date
- *Date* class to create a *Date* object initialised with today's calendar date
- *DateFormat* class to create, from the *Date* object, a string containing the date in an easy-to-read format

The *Date* and *Calendar* classes are found in the *java.util* package. The *DateFormat* class is found in the *java.text* package. So we import both these packages.

```
import java.util.*;
import java.text.*;
```

We create a new *GregorianCalendar* instance named *today*. It is automatically initialised with today's date, as supplied by the computer's operating system.

```
GregorianCalendar today = new GregorianCalendar();
```

We get a *DateFormat* instance and name it *df*.

```
DateFormat df = DateFormat.getDateInstance();
```

Then we get a *Date* instance and initialise it with today's calendar date using a call to *getTime()*.

```
Date date = today.getTime();
```

Finally, we obtain a formatted *Date* instance by sending it as a parameter to the *DateFormat* object's *format()* method.

```
String s = df.format(date);
```

Here is the entire program and its output.

```
/* TodaysDate.java
   Terry Marris  27 May 2001
*/

import java.util.*;
import java.text.*;

public class TodaysDate {
  public static void main(String[] args)
  {
    GregorianCalendar today = new GregorianCalendar();
    DateFormat df = DateFormat.getDateInstance();
    Date date = today.getTime();
    String s = df.format(date);
    System.out.println(s);
  }
}
```

Output:

```
27-May-01
```

The associations between *GregorianCalendar*, *Date* and *DateFormat* are shown in the class diagram (Figure 16.1).

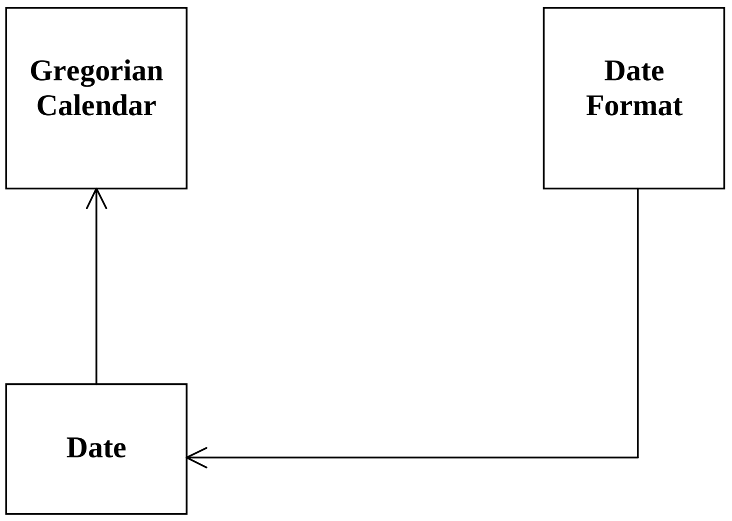

Figure 14.1 A Date sends the message *getTime()* to a *GregorianDate* instance. A *DateFormat* instance has a *Date* from which it creates a date string.

14.6 DISPLAYING A DATE IN DIFFERENT STYLES

The output for the following program is

```
Default: 27-May-01
Long: 27 May 2001
Medium: 27-May-01
Short: 27/05/01
```

Today's date is displayed in various styles.

```
/* TodaysDate2.java
   Terry Marris  27 May 2001
*/

import java.util.*;
import java.text.*;


public class TodaysDate2 {
  public static void main(String[] args)
  {
    GregorianCalendar today = new GregorianCalendar();
    DateFormat df = DateFormat.getDateInstance();
    Date date = today.getTime();
    String s = df.format(date);
    System.out.println("Default: " + s);

    DateFormat dfLong =
               DateFormat.getDateInstance(DateFormat.LONG);
    System.out.println("Long: " + dfLong.format(date));

    DateFormat dfMed =
               DateFormat.getDateInstance(DateFormat.MEDIUM);
    System.out.println("Medium: " + dfMed.format(date));

    DateFormat dfShort =
               DateFormat.getDateInstance(DateFormat.SHORT);
    System.out.println("Short: " + dfShort.format(date));
  }
}
```

The style, long, medium or short, is specified in the creation of the *DateFormat* instance.

14.7 ADD DAYS TO A DATE

The next program adds a number of days to a given date.

We create a given date, 30 June 2001, with

```
GregorianCalendar billingDate = new
                    GregorianCalendar(2001, 6-1, 30);
```

The awkward thing here is that *GregorianCalendar* counts months from zero. So January is month 0, February is month 1, and June is month (6-1) = 5.

We create a cloned copy of the billing date, and then add 28 days to it, to arrive at a due date.

```
     GregorianCalendar dueDate = new GregorianCalendar();
     dueDate = (GregorianCalendar)billingDate.clone();
     dueDate.add(Calendar.DATE, 28);
```

Cloning is necessary because we want the two dates to have a totally independent existence. *Calendar.DATE* refers to the day-in-month number. It is this number, 30, which we want to add 28 days to. We rely on the *add()* method to sensibly manage the change in month and year fields.

The output from the program is

```
     Billing date: 30-Jun-01
     Due date: 28-Jul-01
```

The entire program is shown below.

```
/* Date3.java - adds 28 days to a billing date
   Terry Marris  27 May 2001
*/


import java.util.*;
import java.text.*;


public class Date3 {
  public static void main(String[] args)
  {
    DateFormat df = DateFormat.getDateInstance();

    GregorianCalendar billingDate = new
                    GregorianCalendar(2001, 6-1, 30);
                                      // 30 June 2001

    GregorianCalendar dueDate = new GregorianCalendar();
    dueDate = (GregorianCalendar)billingDate.clone();
    dueDate.add(Calendar.DATE, 28);

    Date date = billingDate.getTime();
    System.out.println("Billing date: " + df.format(date));
    date = dueDate.getTime();
    System.out.println("Due date: " + df.format(date));
  }
}
```

14.8 DIFFERENCE BETWEEN TWO DATES

How many days are there between two dates? We create the two *GregorianCalendar* dates:

```
     GregorianCalendar startOfTerm = new
                    GregorianCalendar(2001, 4-1, 23);
                                  // 23 April 2001
    GregorianCalendar endOfTerm = new
                    GregorianCalendar(2001, 7-1, 6);
                                  // 6 July 2001
```

Then we create and initialise two *Date* instances, one for each *GregorianCalendar* date.

```
     Date startDate = startOfTerm.getTime();
     Date endDate = endOfTerm.getTime();
```

The *getTime()* method of the *Date* class returns the number of milliseconds that have passed since 1 January 1970. We find the number of milliseconds between the two dates.

```
     long millisecond = endDate.getTime() -
                              startDate.getTime();
```

long is a primitive data type. It represents an integer (just like an *int*) but the largest value that can be held by a *long* variable is very much bigger than that which can be held by an *int* variable. The largest *int* value is 2 157 483 647 - just over 2 billion. The largest *long* value is 9 223 372 036 854 775 807. Apart from that, *long* behaves just like an *int*.

Now we convert the milliseconds into days. There are 1000 milliseconds in a second, 60 seconds in a minute, 60 minutes in an hour and 24 hours in a day.

```
     long day = millisecond / (1000 * 60 * 60 * 24);
```

Here is the entire program along with its output.

```
/* Date4.java - the difference between two dates
   Terry Marris  27 May 2001
*/


import java.util.*;
import java.text.*;


public class Date4 {
  public static void main(String[] args)
  {
    GregorianCalendar startOfTerm = new
                      GregorianCalendar(2001, 4-1, 23);
                                     // 23 April 2001
    GregorianCalendar endOfTerm = new
                      GregorianCalendar(2001, 7-1, 6);
                                     // 6 July 2001
    Date startDate = startOfTerm.getTime();
    Date endDate = endOfTerm.getTime();

    long millisecond = endDate.getTime() -
                               startDate.getTime();
    long day = millisecond / (1000 * 60 * 60 * 24);

    DateFormat df = DateFormat.getDateInstance();
    String s = df.format(startDate);
    System.out.println("The start of term is: " + s);
    s = df.format(endDate);
    System.out.println("The end of term is: " + s);
    System.out.println("The number of days between them is: "
                        + day);
  }
}
```

Output:

```
The start of term is: 23-Apr-01
The end of term is: 06-Jul-01
The number of days between them is: 74
```

14.9 THE DATE CLASS

The *Date* class represents an instant in time with millisecond precision.

java.util.Date

Methods		
long	getTime()	returns the number of milliseconds since 1 January 1970 for this *Date*.

Note: *long* is a primitive data type. It represents an integer (just like an *int*) but the largest value that can be held by a *long* variable is very much bigger than that which can be held by an *int* variable. The largest *int* value is 2 157 483 647 - just over 2 billion. The largest *long* value is 9 223 372 036 854 775 807.

14.10 THE CALENDAR CLASS

Calendar is used for converting between a *Date* object and a set of integer fields such day, month and year.

Calendar is really a Date/Time class, but we focus on just the date aspects.

Calendar is the parent class for *GregorianCalendar*. The constants, fields and methods of the *Calendar* class also apply to the *GregorianCalendar* class.

Fields		
static int	DATE, DAY_OF_MONTH	field for the *get()* method indicating the day of the month.
static int	DAY_OF_YEAR	field for the *get()* method indicating the day in year number.
static int	MONTH	field for the *get()* method indicating the month of the year.
static int	YEAR	field for the *get()* method indicating the year.

Methods		
boolean	after(Object obj)	returns true if the given object is a calendar and this calendars date comes after the object's date.
boolean	before(Object obj)	returns true if the given object is a calendar and this calendar's date comes before the given object's date.
Object	clone()	returns a copy of this calendar object.
boolean	equals(Object obj)	returns true if the given object is the same as this calendar object.
int	get(int field)	returns the value for the given field
int	getActualMaximum(int field)	returns the maximum value that the given field can have.
static Calendar	getInstance()	returns a calendar with the default time zone and locale.
Date	getTime()	returns this calendar's current time.
void	set(int year, int month, int day)	sets the values for the given fields, year, month and day in month. January is month zero. Year must be four digits.

14.11 THE GREGORIAN CALENDAR CLASS

The *GregorianCalendar* class provides the standard calendar used by most of the world.

GregorianCalendar is a kind of *Calendar*. The fields, constants and methods described in the *Calendar* class also applies to the *GregorianCalendar* class.

java.util.GregorianCalendar

Constructors	
GregorianCalendar()	initialises a GregorianCalendar object with the current date and time obtained from the operating system.
GregorianCalendar(int year, int month, int day)	initialises a GregorianCalendar with the given year, month and day in month. Year must be four digits. Month zero represents January.

Methods		
void	add(int field, int amount)	adds the given amount to the given field; the result is based on the calendar's rules.
boolean	equals(Object obj)	returns true if this GregorianCalendar is the same as the given calendar object.
boolean	isLeapYear(int year)	returns true if the given year is a leap year.

14.12 THE DATE FORMAT CLASS

The *DateFormat* class is used for formatting dates so that they can be displayed in various styles.

Fields		
static int	LONG	long date format style e.g. 26 August 2001
static int	MEDIUM	medium date format style e.g. 26-Aug-01
static int	SHORT	short date format style e.g. 16/8/01

Methods		
String	format(Date d)	returns a string representing the given date.
static DateFormat	getDateInstance()	returns a date formatter with the default (MEDIUM) formatting style.
static DateFormat	getDateInstance(int style)	returns a date formatter with the given style.

14.13 THE BIRTHDAY BOOK

This example first appeared in SPIVEY *The Z Notation*.

A birthday book is just a collection of birthdays, where a birthday has a name and a date of birth.

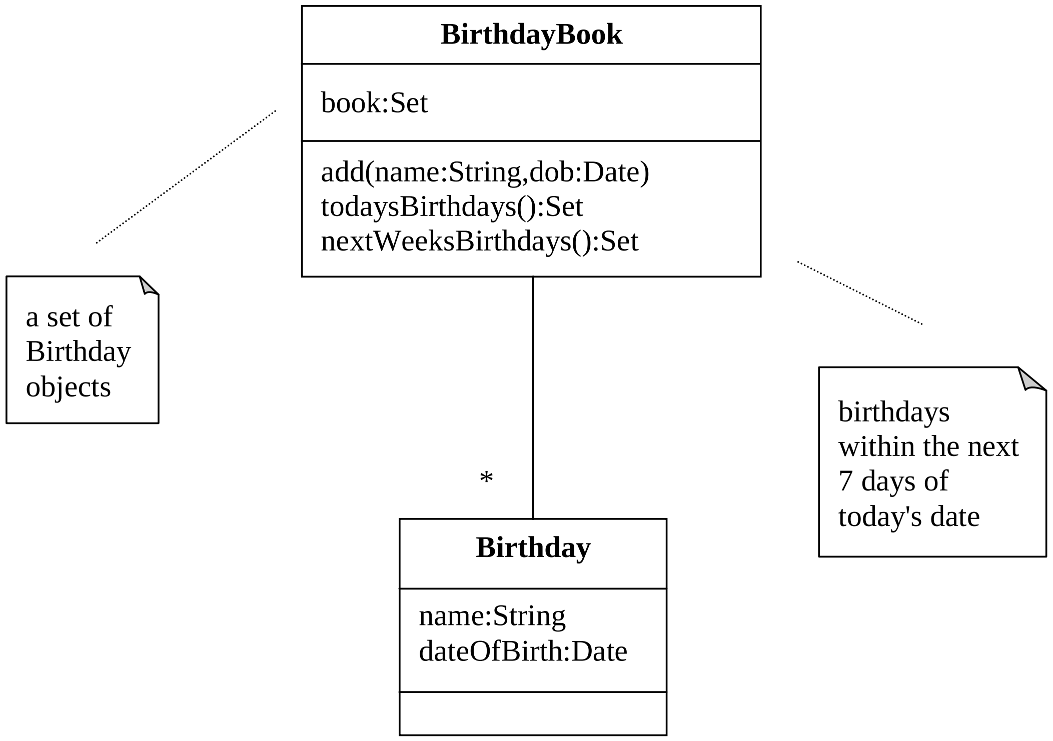

Figure 16.2 *The BirthdayBook*

Set and *Date* are both specify what is required. We shall use *HashSet* to implement *Set*, and *GregorianCalendar* to implement *Date*.

The *Birthday* class is very straightforward.

```
/* Birthday.java
   Terry Marris  27 May 2001
*/


import java.util.*;
import java.text.*;


public class Birthday {
  private String name;
  private GregorianCalendar dateOfBirth;

  public Birthday(String aName, int day, int month, int year)
  {
    name = aName;
    dateOfBirth = new GregorianCalendar(year, month-1, day);
  }


  public GregorianCalendar getDateOfBirth()
  {
    return (GregorianCalendar)dateOfBirth.clone();
  }


  public String toString()
  {
    DateFormat df = DateFormat.getDateInstance();
    Date d = dateOfBirth.getTime();
    String string = df.format(d);
    return "Name: " + name + ", Date of birth: " + string;
  }
}
```

Exercise: Explain each line of the *Birthday* class shown above.

We turn our attention to the *BirthdayBook* class. First, we look at the *birthdaysToday()* method. It returns a *HashSet* containing all those people whose birthday is today.

We get today's date from the operating system and create a new, empty *HashSet* to contain today's birthdays.

```
GregorianCalendar today = new GregorianCalendar();
HashSet todaysBirthdays = new HashSet();
```

book is a *HashSet* that contains all the *Birthday* instances that make up the birthday book. We look at each *Birthday* instance in turn.

```
Iterator it = book.iterator();
while (it.hasNext()) {
 Birthday aBirthday = (Birthday)it.next();
```

We retrieve the *GregorianCalendar* date of birth from the *Birthday* instance we are currently looking at.

```
 GregorianCalendar birthDate = new GregorianCalendar();
 birthDate = aBirthday.getDateOfBirth();
```

We determine whether today's date is the same as the birth date - we consider just the day-in-month and month numbers and ignore the years. If the dates match we add the birthday instance to our set of today's birthdays.

```
 if (today.get(Calendar.DATE) ==
               birthDate.get(Calendar.DATE) &&
     today.get(Calendar.MONTH) ==
               birthDate.get(Calendar.MONTH))
   todaysBirthdays.add(aBirthday);
```

Calendar.DATE refers to the day-in-month number, e.g. the 29 in 29th. May.
Calendar.MONTH refers to the month number.

Finally, we return the *HashSet* containing today's birthdays.

```
return todaysBirthdays;
```

Since a *HashSet* always comes with its own iterator, we rely on the user to examine the contents of the *HashSet*. Here is an extract from the *main()* method.

```
     BirthdayBook birthdayBook = new BirthdayBook();
     birthdayBook.add("jerry", 26, 5, 1956);
     birthdayBook.add("tom", 27, 5, 1959);
     birthdayBook.add("may", 28, 5, 1972);

     ...

     System.out.println("Today's birthdays are:");
     HashSet todaysBirthdays = new HashSet(
                                birthdayBook.birthdaysToday());
     Iterator it = todaysBirthdays.iterator();
     while (it.hasNext()) {
       System.out.println(it.next());
    }
```

The entire *BirthdayBook* class, along with some output, is shown below.

```
/* BirthdayBook.java
   Terry Marris  27 May 2001
*/


import java.util.*;
import java.text.*;

public class BirthdayBook {
  private HashSet book;

  public BirthdayBook()
  {
    book = new HashSet();
  }


  public void add(String name, int day, int month, int year)
  {
    Birthday birthday = new Birthday(name, day, month, year);
    book.add(birthday);
  }


  public HashSet getBook()
  {
    return (HashSet)book.clone();
  }


  public HashSet birthdaysToday()
  {
    GregorianCalendar today = new GregorianCalendar();

    HashSet todaysBirthdays = new HashSet();

    Iterator it = book.iterator();
    while (it.hasNext()) {
      Birthday aBirthday = (Birthday)it.next();
      GregorianCalendar birthDate = new GregorianCalendar();
      birthDate = aBirthday.getDateOfBirth();

      if (today.get(Calendar.DATE) ==
                   birthDate.get(Calendar.DATE) &&
          today.get(Calendar.MONTH) ==
                   birthDate.get(Calendar.MONTH))
        todaysBirthdays.add(aBirthday);
    }
    return todaysBirthdays;
  }
```

```
public HashSet birthdaysNextWeek()
{
  GregorianCalendar today = new GregorianCalendar();
  GregorianCalendar nextWeek = new GregorianCalendar();
  nextWeek.add(Calendar.DATE, 8);

  HashSet comingBirthdays = new HashSet();

  Iterator it = book.iterator();
  while (it.hasNext()) {
    Birthday aBirthday = (Birthday)it.next();
    GregorianCalendar birthDate = new GregorianCalendar();
    birthDate = aBirthday.getDateOfBirth();

    GregorianCalendar birthDay = new GregorianCalendar();
    birthDay.set(today.get(Calendar.YEAR),
                 birthDate.get(Calendar.MONTH),
                 birthDate.get(Calendar.DATE));

    if (birthDay.after(today) && birthDay.before(nextWeek))
      comingBirthdays.add(aBirthday);

  }
  return comingBirthdays;
}


private static String
        stringFromCalendarDate(GregorianCalendar calendar)
{
  DateFormat df = DateFormat.getDateInstance();
  Date date = calendar.getTime();
  return df.format(date);
}
```

```
  public static void main(String[] s)
  {
    BirthdayBook birthdayBook = new BirthdayBook();
    birthdayBook.add("jerry", 26, 5, 1956);
    birthdayBook.add("tom", 27, 5, 1959);
    birthdayBook.add("may", 28, 5, 1972);

    birthdayBook.add("hank", 2, 6, 1981);
    birthdayBook.add("anne", 3, 6, 1963);
    birthdayBook.add("flo", 4, 6, 1972);

    System.out.println("Contents of birthday book:");
    HashSet book = new HashSet(birthdayBook.getBook());
    Iterator it = book.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
    System.out.println();

    GregorianCalendar today = new GregorianCalendar();
    System.out.println("Today is: " +
                  BirthdayBook.stringFromCalendarDate(today));
    System.out.println();

    System.out.println("Today's birthdays are:");
    HashSet todaysBirthdays = new HashSet(
                            birthdayBook.birthdaysToday());
    it = todaysBirthdays.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
    System.out.println();

    GregorianCalendar nextWeek = new GregorianCalendar();
    nextWeek.add(Calendar.DATE, 7);
    System.out.println("Date next week is: " +
               BirthdayBook.stringFromCalendarDate(nextWeek));
    System.out.println();

    System.out.println("Birthdays in the coming 7 days are:");
    HashSet comingBirthdays = new HashSet(
                          birthdayBook.birthdaysNextWeek());
    it = comingBirthdays.iterator();
    while (it.hasNext()) {
      System.out.println(it.next());
    }
  }
}
```

Output:

```
Contents of birthday book:
Name: tom, Date of birth: 27-May-59
Name: hank, Date of birth: 02-Jun-81
Name: anne, Date of birth: 03-Jun-63
Name: flo, Date of birth: 04-Jun-72
Name: may, Date of birth: 28-May-72
Name: jerry, Date of birth: 26-May-56

Today is: 27-May-01

Today's birthdays are:
Name: tom, Date of birth: 27-May-59

Date next week is: 03-Jun-01

Birthdays in the coming 7 days are:
Name: hank, Date of birth: 02-Jun-81
Name: anne, Date of birth: 03-Jun-63
Name: tom, Date of birth: 27-May-59
Name: may, Date of birth: 28-May-72
```

Exercise:

1 Explain each line of the *birthdaysNextWeek()* method.

2 Is the output entirely what you would expect?

14.14 FURTHER READING

HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 117