

2c Programming with Static Methods

1 Example: Printing a calendar

We tackle a modestly large programming problem, of a size and complexity that requires us to structure it as a collection of methods. We will write a program which prints a calendar for any year. We will restrict the year to the range 1900 to 2099, just to simplify the leap year calculations. The following is an outline of the program:

```
class Calendar {

    static void yearHeader(int y) {
        // Display calendar header for year y, such as
        //           Calendar 2001
        .....
    }

    static void monthHeader(int m) {
        // Display month header for month M, such as
        //           February
        // Sun Mon Tue Wed Thu Fri Sat
        .....
    }

    static int firstDay(int y) {
        // Calculate day of week on which 1/1/y falls
        // Sunday = 0, Monday = 1, etc.
        .....
    }

    static int daysInMonth(int m, int y) {
        // Calculate number of days in month m in year y.
        // (Note year is only significant for month 2 (i.e. February))
    }
}
```

```

    .....
}

static void putMonth(int d, int s) {
    // Display the calendar for a month containing s days, where
    // the first of the month falls on day d (Sunday = 0, etc.).
    // E.g. for a month of 28 days beginning on a Thursday:
    //
    //           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
    .....
}

static void putCalendar(int y) {
    // Display a calendar for year y, y>=1900.
    .....
}

public static void main(String[] args) {
    // Read a year (from 1900) and print its calendar
    .....
}
}

```

The complete program is presented below.

```

class Calendar {

    static void yearHeader(int y) {
        // Display calendar header for year y
        System.out.print("        CALENDAR " + y);
        System.out.println(); System.out.println();
    }

    static void monthHeader(int m) {

```

```

// Display month header for month m
String[] month = {"February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November",
    "December"};
System.out.println("    " + month[m-1]);
System.out.println(" Sun Mon Tue Wed Thu Fri Sat");
}

```

```

static int firstDay(int y) {
// Day of week on which 1/1/y falls (Sunday = 0, etc.).
return (((y-1900)*365 // elapsed days since 1/1/1900
    + (y-1901)/4 // not forgetting leap days
    + 1 // 1/1/1900 fell on a Monday
    )%7); // 7 days in a week
}

```

```

static int daysInMonth(int m, int y) {
// Number of days in month m in year y
if (m==9 || m==4 || m==6 || m==11) return(30);
else if (m==2) { // catch leap year
    if (y%4==0 && y!=1900) return(29);
    else return(28);
}
else return(31);
}

```

```

static void putMonth(int d, int s) {
// Display a month of s days, the first of the month
// falling on day d (Sunday = 0, etc.).
// indent first line of month by d positions
for (int k=1; k<=d; k++) System.out.print("  ");
int day = d; // day of week
for (int date=1; date<=s; date++) {
    System.out.printf(" %2d", date);
    day = (day+1)%7;
    if (day==0 && date<s) // another line of days to come
        System.out.println();
}
System.out.println(); System.out.println();
}

```

```

static void putCalendar(int y) {
// Display a calendar for year y, y>=1900.
    yearHeader(y);
    int d = firstDay(y);
    for (int m=1; m<=12; m++) {
        monthHeader(m);
        int s = daysInMonth(m, y);
        putMonth(d, s);
        d = (d+s)%7;
    }
}

public static void main(String[] args) {
// Read a year (from 1900) and print its calendar
    System.out.print("Enter a year in range 1900..: ");
    int y = Console.readInt();
    putCalendar(y);
}
}

```

2 Local and global variables

A variable that is declared within a method is said to be *local* to the method. It cannot be accessed by code in a different method. Consider:

```

class MyClass {

    public static void main(String[] args) {
        int x;
        x = 0; // this is ok
        .....
    }

    static void p() {
        x++; // this is illegal - x belongs to a different method
        .....
    }
}

```

Local variables are created each time the method is invoked, and destroyed each time it terminates. They do not carry over their value from one invocation to the next. Consider:

```
class MyClass {  
  
    public static void main(String[] args) {  
        silly(); silly(); silly();  
    }  
  
    static void silly() {  
        int count = 0;  
        count++;  
        System.out.print(count + " ");  
    }  
}
```

The above program does not cause 1 2 3 to be printed, but rather 1 1 1.

Static variables

Variables can be declared in a class rather than locally in a method; such variables are said to be *global* to the methods in the class (otherwise the variable is *local* to the method in which it is declared). For example, variable `numCalls` below is global to both methods in the class, i.e. it is accessible from both methods, and it carries over its value between method invocations:

```
class MyClass {  
  
    static int numCalls = 0; // for number of method invocations  
  
    static void p() {  
        numCalls++; // record another invocation  
        .....  
    }  
  
    static void q() {  
        numCalls++; // record another invocation  
        .....  
    }  
}
```

```
public static void main(String[] args) {  
    ..... p(); .....; q(); ..... p(); ....  
    ..... q(); .....; q(); ..... p(); ....  
    .....  
    System.out.print("Number of invocations = ", numCalls);  
}  
  
}
```

Note that `numCalls` has the attribute `static`, just as for the methods. Global variables can also be dynamic (also called “instance variables”), just as for methods, and we will meet these later.