```
//  Additive Pattern for numeric classes
//  ------------------------------------

//  Instructions:

//  1.  Paste the code below into the class being defined.
//  2.  Substitute the class name for "Class"
//  3.  Substitute the name of any interacting pure number type
//      for "PURE_TYPE
//  4.  Fill in the bodies of the incomplete functions below.


public: Class  addSet(Class     rs) {                    return this;}
public: Class  addSet(PURE_TYPE rs) {                    return this;}
public: Class  subSet(Class     rs) {                    return this;}
public: Class  subSet(PURE_TYPE rs) {                    return this;}
public: Class  mpySet(PURE_TYPE rs) {                    return this;}
public: Class  divSet(PURE_TYPE rs) {                    return this;}
public: Class  modSet(Class     rs) {                    return this;}
public: Class  modSet(PURE_TYPE rs) {                    return this;}
public: PURE_TYPE div(Class  rs)    {return;                        }

public: Class  minus()              {return;                        }



//  Completely defined functions -- no fill-in needed:
//  ---------------------------

public: Class  add(Class     rs) {return new Class(this).addSet(rs);}
public: Class  sub(Class     rs) {return new Class(this).subSet(rs);}
public: Class  mpy(PURE_TYPE rs) {return new Class(this).mpySet(rs);}
public: Class  div(PURE_TYPE rs) {return new Class(this).divSet(rs);}
public: Class  mod(Class     rs) {return new Class(this).modSet(rs);}
public: Class  mod(PURE_TYPE rs) {return new Class(this).modSet(rs);}
```

```
/* Calendar information   */  package ididates;
// --------------------   (Copyright 1997, Information Disciplines, Inc.)

//  This pseudo-class:
//     -  collects constants and tables that describe the Gregorian calendar,
//     -  provides certain calendar-related functions that are independent
//        of the Date class and of the representation of Date objects.

//  All members (both data and functions) are static (non-instance) members.

public class CalendarInfo {

  public static final short DAYS_PER_YEAR      =    365;
  public static final short DAYS_PER_4_YEARS   =   1461;
  public static final int   DAYS_PER_100_YEARS =  36524;
  public static final int   DAYS_PER_400_YEARS = 146097;

  public static final byte  DAYS_IN_MONTH []   = {0, 31, 28, 31, 30, 31, 30,
                                                      31, 31, 30, 31, 30, 31};

  public static final short DAYS_BEFORE_MONTH[]= {0,  0, 31, 59, 90,120,151,
                                                 181,212,243,273,304,334,365};

  public static short centuryBreak = 20; //  2-digit year is 19yy if yy >
                                         //                  20yy if yy <=

//  Language-specific names
//  ----------------------

  public static final String  MONTH_NAME[]
    ={"","January", "February", "March"    , "April"  ,  "May"    , "June"   ,
         "July"   , "August"  , "September", "October", "November", "December"};

  public static final String  DAY_NAME[]
    = {"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"};
//  Utility functions
//  -----------------

  public static boolean isLeapYear(int yyyy)
  {return (0 == yyyy % 4) && (!(0 == yyyy % 100) || (0 == yyyy % 400));}

  public static boolean isLegalYMD(int yyyy, int mm, int dd)
  {return mm > 0 &&  mm <= 12
       && dd > 0 && (dd <= DAYS_IN_MONTH [mm]
                    || (dd == 29 && mm == 2 && isLeapYear(yyyy)));
  }

  public static short dayNumber(int yyyy, int mm, int dd)
  {if  (!isLegalYMD(yyyy, mm, dd)) return 0;
   int  ddd = DAYS_BEFORE_MONTH [mm] + dd;
   if  (isLeapYear(yyyy) && mm > 2) ddd++;
   return (short) ddd;
  }

}
```

```
/* Date class                     */  package ididates;
// ----------    (Copyright 1997, Information Disciplines, Inc.)

public class Date {

//  NOTE:  The class name duplicates the name of a class in "java.util".  While
//  we normally avoid conflicts with library components, that misnamed class is
//  so poorly conceived and user-hostile that we would never use it in any IDI
//  application, and we strongly recommend against using it anywhere.




//  Internal representation
//  -----------------------

  int  value;                          //  Number of days since origin

  static final int bias                // Origin date is December 30, 1899,
   = - CalendarInfo.DAYS_PER_400_YEARS * 5 //  for compatibility with Lotus
     + CalendarInfo.DAYS_PER_100_YEARS     //  1-2-3 and various other
     + CalendarInfo.DAYS_PER_YEAR + 2;     //  software products

  static final short bias_weekday = 6; // Origin date was a Saturday


//  Note that:

//    1.  Some conversion functions assume the Gregorian calendar,
//        even for dates before that calendar was adopted.

//    2.  B.C. dates can be generated by arithmetic operations, but
//        are not necessarily supported by conversion functions.
//  Constructors
//  ------------

 public Date(int yyyy, int ddd)             //  Year and day number
 {int  years = yyyy - 1;
  value = bias + ddd                        //  Start with base date
      + years * CalendarInfo.DAYS_PER_YEAR  //  Convert years to days
      + years/4 - years/100 + years/400;    //  Apply leap year adjustments
 }


 public Date(int yyyy, int mm, int dd)      //  Year, month, and day of month
 {this(yyyy,CalendarInfo.dayNumber(yyyy,mm,dd));}


 public Date(Date d) {value = d.value;}    //  Copy constructor


 public Date() {}                          //  No default value

 public Date(String s)                      //  YYMMDD (ANSI character repr.)
 {String yymmdd = new String(s.trim());     //    Result is undefined if illegal
  if (yymmdd.length() != 6) {return;}
  char charVal[] = new char[6];             //  Numeric value of each character
  for (int i = 0; i < 6; i++)               //  Decompose string to into
      charVal[i] = (char) (yymmdd.charAt(i) - '0'); //  character values
  int y = charVal[0] * 10 + charVal[1];
  int m = charVal[2] * 10 + charVal[3];
  int d = charVal[4] * 10 + charVal[5];
  this.set(new Date(y+(y<CalendarInfo.centuryBreak ?
                              2000 : 1900), m, d));
 }
```

```
//  Pseudo assignment operator
//  --------------------------

 public Date set(Date rs) {value = rs.value; return this;}




//  Accessors
//  ---------

 public short weekday()
       {return (short)(((value + bias_weekday) % 7 + 7) % 7);}

//  To avoid redundant computation and still provide an independent accessors
//  for each component, the private decomposition function, ymd, caches its
//  result.  If the user then invokes more accessors for the same date before
//  invoking any for a different date, the saved values are retrieved.

 public synchronized int   year () {ymd(); return y;}
 public synchronized byte  month() {ymd(); return m;}
 public synchronized byte  day  () {ymd(); return d;}
 public synchronized int   dayno() {ymd(); return ddd;}

 static int   y;                  //  Set by ymd function (below) and
 static byte  m, d;               //    retrieved by accessors (above)
 static int   ddd;
 static int   cur_value = bias;   //  Flag to lock above values
```

```
//  Decompose a date into calendar components (See above accessors)
//  -----------------------------------------

synchronized void ymd()
{if (value == cur_value) return;  //  Do nothing if unchanged
  cur_value = value;              //  Save value for next time
  int ngrps;

  ddd    = value - bias;          //  Strip off origin date

  ngrps  = ddd   / CalendarInfo.DAYS_PER_400_YEARS;
  y      = ngrps * 400;
  ddd   -= ngrps * CalendarInfo.DAYS_PER_400_YEARS;

  ngrps  = ddd   / CalendarInfo.DAYS_PER_100_YEARS;
  y     += ngrps * 100;
  ddd   -= ngrps * CalendarInfo.DAYS_PER_100_YEARS;

  ngrps  = ddd / CalendarInfo.DAYS_PER_4_YEARS;
  y     += ngrps * 4;
  ddd   -= ngrps * CalendarInfo.DAYS_PER_4_YEARS;

  if (ddd == 0)                              //  End-of-year correction
        ddd=CalendarInfo.isLeapYear(y) ? 366 : 365;
  else {ddd  += CalendarInfo.DAYS_PER_YEAR - 1;
        y    += ddd / CalendarInfo.DAYS_PER_YEAR;
        ddd  %= CalendarInfo.DAYS_PER_YEAR; ++ddd;
      }

//  At this point y is the year and ddd is the day number

  int dx = ddd;                       //  (Leap-year corrected day number)
  if (CalendarInfo.isLeapYear(y))     //  Handle February 29 as special case
      if (ddd > 60) --dx;             //  Adjust day number
      else if (ddd == 60) {m = 2; d = 29; return;}

  m = (byte)((dx + 28) / 29);         //  Estimate the month, then adjust
  if  (dx <= CalendarInfo.DAYS_BEFORE_MONTH[m]) m--;
  d = (byte)(dx  - CalendarInfo.DAYS_BEFORE_MONTH[m]);
  if  (m == 13) {m = 1; y++;}

 }

//  Conversion functions
//  --------------------

 public String toString()           //  Default external representation
 {String mm=new String(((month() < 10) ? "0":"") + month()); // Insert
  String dd=new String(((day  () < 10) ? "0":"") + day  ()); //   leading zero
  return year() + "-" + mm + '-' + dd;}

 public String toEnglish()          //  Standard English (non-American)
 {return day() + " " + CalendarInfo.MONTH_NAME[month()] + " " + year();}



//************ NOTE:
//   From here on Java demands repetition of code which in C++ we would
//   package for reuse.  Although this imposes major maintenance and
//   testing burdens, Java offers no practical solution as of March, 1997.


//  Relational operators  (implements "ordered")
//  --------------------

 public boolean equals     (Date rs) {return value == rs.value;}
 public boolean lessThan   (Date rs) {return value <  rs.value;}
 public boolean greaterThan(Date rs) {return value >  rs.value;}
```

```
//  Arithmetic operators  (implements "point")
//  --------------------

 public Date add  (Days rs)  {return new Date(this).addSet(rs);}
 public Date sub  (Days rs)  {return new Date(this).subSet(rs);}
 public Days sub  (Date rs)  {return new Days(value - rs.value);}
 public Date add  (int  rs)  {return new Date(this).addSet(rs);}
 public Date sub  (int  rs)  {return new Date(this).subSet(rs);}

 public Date addSet(Days rs)  {value += rs.toInt(); return this;}
 public Date subSet(Days rs)  {value -= rs.toInt(); return this;}
 public Date addSet(int  rs)  {value += rs;         return this;}
 public Date subSet(int  rs)  {value -= rs;         return this;}



// Special function to get the current date
// ---------------- (The only method that uses Java library standard class)
 public static Date today()
    {java.util.GregorianCalendar d = new java.util.GregorianCalendar();
     return new Date(d.get(java.util.Calendar.YEAR),
                     d.get(java.util.Calendar.MONTH)+1,
                     d.get(java.util.Calendar.DATE));
    }

}
```