import java.util.*;

public class Holidays
{

    public static Date NewYearsDayObserved (int nYear)
    {
        int nX;
        int nMonth = 0;                 // January
        int nMonthDecember = 11;        // December
        Date dtD;
        dtD = new Date(nYear, nMonth, 1);
        nX = dtD.getDay();
        if (nYear > 1900)
        {
            nYear -= 1900;
        }
        switch(nX)
        {
            case 0 : // Sunday
                return new Date(nYear, nMonth, 2);
            case 1 : // Monday
            case 2 : // Tuesday
            case 3 : // Wednesday
            case 4 : // Thursday
            case 5 : // Friday
                return new Date(nYear, nMonth, 1);
            default :
                // Saturday, then observe on friday of previous year
                return new Date(--nYear, nMonthDecember, 31);
        }
    }

    public static Date NewYearsDay (int nYear)
    {
        // January 1st
        int nMonth = 0; // January
        return new Date(nYear, nMonth, 1);
    }

    public static Date RobertELeeDay (int nYear)
    {
        int nMonth = 0; // January
        return new Date(nYear, nMonth, 18);
    }

    public Date MartinLutherKingObserved (int nYear)
    {
        // Third Monday in January
        int nX;
        int nMonth = 0; // January
    }
}
Date dtD;

dtD = new Date(nYear, nMonth, 1);
nX = dtD.getDay();
switch(nX)
{
    case 0 : // Sunday
        return new Date(nYear, nMonth, 16);
    case 1 : // Monday
        return new Date(nYear, nMonth, 15);
    case 2 : // Tuesday
        return new Date(nYear, nMonth, 21);
    case 3 : // Wednesday
        return new Date(nYear, nMonth, 20);
    case 4 : // Thursday
        return new Date(nYear, nMonth, 19);
    case 5 : // Friday
        return new Date(nYear, nMonth, 18);
    default : // Saturday
        return new Date(nYear, nMonth, 17);
}

public static Date GroundhogDay (int nYear)
{
    int nMonth = 1; // February
    // February 8th
    return new Date(nYear, nMonth, 8);
}

public static Date AbrahamLincolnsBirthday (int nYear)
{
    int nMonth = 1; // February
    // February 12th
    return new Date(nYear, nMonth, 12);
}

public static Date ValentinesDay (int nYear)
{
    int nMonth = 1; // February
    // February 14th
    return new Date(nYear, nMonth, 14);
}

public static Date SusanBAAnthonyDay (int nYear)
{
    int nMonth = 1; // February
    // February 15th
    return new Date(nYear, nMonth, 15);
}

public static Date PresidentsDayObserved (int nYear)
{
    // Third Monday in February
    int nX;
int nMonth = 1; // February
Date dtD;

dtD = new Date(nYear, nMonth, 1);
nX = dtD.getDay();
switch(nX)
{
    case 0 : // Sunday
        return new Date(nYear, nMonth, 16);
    case 1 : // Monday
        return new Date(nYear, nMonth, 15);
    case 2 : // Tuesday
        return new Date(nYear, nMonth, 21);
    case 3 : // Wednesday
        return new Date(nYear, nMonth, 20);
    case 4 : // Thursday
        return new Date(nYear, nMonth, 19);
    case 5 : // Friday
        return new Date(nYear, nMonth, 18);
    default : // Saturday
        return new Date(nYear, nMonth, 17);
}

public static Date SaintPatricksDay (int nYear)
{
    int nMonth = 2; // March
    return new Date(nYear, nMonth, 17);
}

public static Date GoodFridayObserved(int nYear)
{
    // Get Easter Sunday and subtract two days
    int nEasterMonth    = 0;
    int nEasterDay     = 0;
    int nGoodFridayMonth  = 0;
    int nGoodFridayDay  = 0;
    Date dEasterSunday;

    dEasterSunday = EasterSunday(nYear);
nEasterMonth = dEasterSunday.getMonth();
nEasterDay = dEasterSunday.getDate();
if (nEasterDay <= 3 && nEasterMonth == 3) // Check if <= April 3rd
{
    switch(nEasterDay)
    {
        case 3 :
            nGoodFridayMonth = nEasterMonth - 1;
            nGoodFridayDay   = nEasterDay - 2;
            break;
        case 2 :
            nGoodFridayMonth = nEasterMonth - 1;
            nGoodFridayDay   = 31;
            break;
        case 1 :
GoodFridayMonth = nEasterMonth - 1;
nGoodFridayDay = 31;
break;
default:
nGoodFridayMonth = nEasterMonth;
nGoodFridayDay = nEasterDay - 2;
}
}
else {
   
nGoodFridayMonth = nEasterMonth;
nGoodFridayDay = nEasterDay - 2;
}
}
return new Date(nYear, nGoodFridayMonth, nGoodFridayDay);
}

public static Date EasterSunday(int nYear) {
   /*
   Calculate Easter Sunday
   Written by Gregory N. Mirsky

   Source: 2nd Edition by Peter Duffett-Smith. It was originally from
   Butcher's Ecclesiastical Calendar, published in 1876. This
   algorithm has also been published in the 1922 book General
   Astronomy by Spencer Jones; in The Journal of the British
   Astronomical Association (Vol.88, page 91, December 1977); and in

   This algorithm holds for any year in the Gregorian Calendar, which
   (of course) means years including and after 1583.

   a=year%19
   b=year/100
   c=year%100
   d=b/4
   e=b%4
   f=(b+8)/25
   g=(b-f+1)/3
   h=(19*a+b-d-g+15)%30
   i=c/4
   k=c%4
   l=(32+2*e+2*i-h-k)%7
   m=(a+11*h+22*l)/451
   Easter Month =((h+l-7*m+114)/31 [3=March, 4=April]
   p=(h+l-7*m+114)%31
   Easter Date=p+1 (date in Easter Month)

   Note: Integer truncation is already factored into the
   calculations. Using higher precision variables will cause
   inaccurate calculations.
   */
   
   int nA = 0;
```java
int nB = 0;
int nC = 0;
int nD = 0;
int nE = 0;
int nF = 0;
int nG = 0;
int nH = 0;
int nI = 0;
int nK = 0;
int nL = 0;
int nM = 0;
int nP = 0;
int nYY = 0;
int nEasterMonth = 0;
int nEasterDay = 0;

// Calculate Easter
nYY = nYear;
if (nYear < 1900)
{
    // if year is in java format put it into standard
    // format for the calculation
    nYear += 1900;
}

nA = nYear % 19;
int nB = nYear / 100;
int nC = nYear % 100;
int nD = nB / 4;
int nE = nB % 4;
int nF = (nB + 8) / 25;
int nG = (nB - nF + 1) / 3;
int nH = (19 * nA + nB - nD - nG + 15) % 30;
int nI = nC / 4;
int nK = nC % 4;
int nL = (32 + 2 * nE + 2 * nI - nH - nK) % 7;
int nM = (nA + 11 * nH + 22 * nL) / 451;

// [3=March, 4=April]
nEasterMonth = (nH + nL - 7 * nM + 114) / 31;
nP = (nH + nL - 7 * nM + 114) % 31;

// Date in Easter Month.
nEasterDay = nP + 1;

// Uncorrect for our earlier correction.
nYear = nYear - 1900;

// Populate the date object...
return new Date(nYear, nEasterMonth, nEasterDay);
}

public static Date EasterMonday (int nYear)
{
    int nEasterMonth = 0;
    ```
int nEasterDay   = 0;
int nMonthMarch  = 2; // March
int nMonthApril  = 3; // April
Date dEasterSunday = EasterSunday(nYear);
nEasterMonth = dEasterSunday.getMonth();
nEasterDay = dEasterSunday.getDay();
if (nEasterMonth == nMonthMarch || nEasterDay == 31)
    { return new Date(nYear, nMonthApril, 1); }
else
    { return new Date(nYear, nEasterMonth, ++nEasterDay); }
}

public static Date CincoDeMayo (int nYear)
{
    int nMonth = 4; // May
    // May 5th
    return new Date(nYear, nMonth, 5);
}

public static Date MemorialDayObserved (int nYear)
{
    // Last Monday in May
    int nX;
    int nMonth = 4; // May
    Date dtD;
    dtD = new Date(nYear, nMonth, 31);
    nX = dtD.getDay();
    switch(nX)
    { case 0 : // Sunday
              return new Date(nYear, nMonth, 25);
        case 1 : // Monday
              return new Date(nYear, nMonth, 31);
        case 2 : // Tuesday
              return new Date(nYear, nMonth, 30);
        case 3 : // Wednesday
              return new Date(nYear, nMonth, 29);
        case 4 : // Thursday
              return new Date(nYear, nMonth, 28);
        case 5 : // Friday
              return new Date(nYear, nMonth, 27);
        default : // Saturday
              return new Date(nYear, nMonth, 26);
        }
}

public static Date IndependenceDayObserved (int nYear)
{
    int nX;
    int nMonth = 6; // July
    "}
Date dtD;

dtD = new Date(nYear, nMonth, 4);
nX = dtD.getDay();
switch(nX)
{
    case 0 : // Sunday
        return new Date(nYear, nMonth, 5);
    case 1 : // Monday
    case 2 : // Tuesday
    case 3 : // Wednesday
    case 4 : // Thursday
    case 5 : // Friday
        return new Date(nYear, nMonth, 4);
    default :
        // Saturday
        return new Date(nYear, nMonth, 3);
}

public static Date IndependenceDay (int nYear)
{
    int nMonth = 6; // July
    // July 4th
    return new Date(nYear, nMonth, 4);
}

public static Date CanadianCivicHoliday (int nYear)
{
    // First Monday in August
    int nX;
    int nMonth = 7; // August
    Date dtD;

dtD = new Date(nYear, nMonth, 1);
nX = dtD.getDay();
switch(nX)
{
    case 0 : // Sunday
        return new Date(nYear, nMonth, 2);
    case 1 : // Monday
        return new Date(nYear, nMonth, 1);
    case 2 : // Tuesday
        return new Date(nYear, nMonth, 7);
    case 3 : // Wednesday
        return new Date(nYear, nMonth, 6);
    case 4 : // Thursday
        return new Date(nYear, nMonth, 5);
    case 5 : // Friday
        return new Date(nYear, nMonth, 4);
    default : // Saturday
        return new Date(nYear, nMonth, 3);
}
}
public static Date LaborDayObserved (int nYear)
{
    // The first Monday in September
    int nX;
    int nMonth = 8; // September
    Date dtD;

    dtD = new Date(nYear, 9, 1);
    nX = dtD.getDay();
    switch(nX)
    {
        case 0 : // Sunday
            return new Date(nYear, nMonth, 2);
        case 1 : // Monday
            return new Date(nYear, nMonth, 7);
        case 2 : // Tuesday
            return new Date(nYear, nMonth, 6);
        case 3 : // Wednesday
            return new Date(nYear, nMonth, 5);
        case 4 : // Thursday
            return new Date(nYear, nMonth, 4);
        case 5 : // Friday
            return new Date(nYear, nMonth, 3);
        default : // Saturday
            return new Date(nYear, nMonth, 2);
    }
}

public static Date ColumbusDayObserved (int nYear)
{
    // Second Monday in October
    int nX;
    int nMonth = 9; // October
    Date dtD;

    dtD = new Date(nYear, nMonth, 1);
    nX = dtD.getDay();
    switch(nX)
    {
        case 0 : // Sunday
            return new Date(nYear, nMonth, 9);
        case 1 : // Monday
            return new Date(nYear, nMonth, 15);
        case 2 : // Tuesday
            return new Date(nYear, nMonth, 14);
        case 3 : // Wednesday
            return new Date(nYear, nMonth, 13);
        case 4 : // Thursday
            return new Date(nYear, nMonth, 12);
        case 5 : // Friday
            return new Date(nYear, nMonth, 11);
        default : // Saturday
            return new Date(nYear, nMonth, 10);
    }
}
public static Date Halloween (int nYear)
{
    int nMonth = 9; // October 31st
    return (new Date(nYear, nMonth, 31));
}

public static Date USElectionDay (int nYear)
{
    // First Tuesday in November
    int nX;
    int nMonth = 10; // November
    Date dtD;
    dtD = new Date(nYear, nMonth, 1);
    nX = dtD.getDay();
    switch(nX)
    {
        case 0 : // Sunday
            return new Date(nYear, nMonth, 3);
        case 1 : // Monday
            return new Date(nYear, nMonth, 2);
        case 2 : // Tuesday
            return new Date(nYear, nMonth, 1);
        case 3 : // Wednesday
            return new Date(nYear, nMonth, 7);
        case 4 : // Thursday
            return new Date(nYear, nMonth, 6);
        case 5 : // Friday
            return new Date(nYear, nMonth, 5);
        default : // Saturday
            return new Date(nYear, nMonth, 4);
    }
}

public static Date VeteransDayObserved (int nYear)
{
    //November 11th
    int nMonth = 10; // November
    return new Date(nYear, nMonth, 11);
}

public static Date RememberenceDayObserved (int nYear)
{
    // Canadian version of Veterans Day
    return VeteransDayObserved(nYear);
}

public static Date ThanksgivingObserved(int nYear)
{
    int nX;
    int nMonth = 10; // November
    Date dtD;
dtD = new Date(nYear, nMonth, 1);
nX = dtD.getDay();
switch(nX)
{
    case 0 : // Sunday
        return new Date(nYear, nMonth, 26);
    case 1 : // Monday
        return new Date(nYear, nMonth, 25);
    case 2 : // Tuesday
        return new Date(nYear, nMonth, 24);
    case 3 : // Wednesday
        return new Date(nYear, nMonth, 23);
    case 4 : // Thursday
        return new Date(nYear, nMonth, 22);
    case 5 : // Friday
        return new Date(nYear, nMonth, 28);
    default : // Saturday
        return new Date(nYear, nMonth, 27);
}

public static Date ChristmasDayObserved (int nYear)
{
    int nX;
    int nMonth = 11; // December
    Date dtD;
    dtD = new Date(nYear, nMonth, 25);
    nX = dtD.getDay();
    switch(nX)
    {
        case 0 : // Sunday
            return new Date(nYear, nMonth, 26);
        case 1 : // Monday
            return new Date(nYear, nMonth, 25);
        case 2 : // Tuesday
            return new Date(nYear, nMonth, 24);
        case 3 : // Wednesday
            return new Date(nYear, nMonth, 23);
        case 4 : // Thursday
            return new Date(nYear, nMonth, 22);
        case 5 : // Friday
            return new Date(nYear, nMonth, 28);
        default :
            // Saturday
            return new Date(nYear, nMonth, 24);
    }
}

public static Date ChristmasDay (int nYear)
{
    int nMonth = 11; // December
    // December 25th
    return new Date(nYear, nMonth, 25);
}

//*****************************************************************************
// Miscellaneous other holidays are left as an exercise for the reader.
First, calculate the Golden Number G. This is fundamental to the
calculation of both the date of Easter and the Date of Rosh Hashanah.
It is intimately connected with the Metonic Cycle. For any year Y, the
Golden Number is defined as

\[ G = \text{Remainder}(Y|19) + 1. \]

Don't forget to add the 1!!!

The following rules are also due to John Horton Conway, of Princeton
University. In the Gregorian year Y of the Common Era, Rosh Hashanah
normally falls on September N, where

\[ N + \text{fraction} = \{[Y/100] - [Y/400] - 2\} + \]
\[ 765433/492480*\text{Remainder}(12G|19) + \text{Remainder}(Y|4)/4 - (313Y+89081)/98496 \]

Here, G is the Golden Number, and * means multiply. However, if certain
conditions are satisfied, Rosh Hashanah is postponed by one or even two
days, as follows:

***Postponement rules***

1. If the day calculated above is a Sunday, Wednesday, or Friday, Rosh
Hashanah falls on the next day (Monday, Thursday or Saturday,
respectively).

2. If the calculated day is a Monday, and if the fraction is greater
than or equal to 23269/25920, and if \( \text{Remainder}(12G|19) \) is greater than
11, Rosh Hashanah falls on the next day, a Tuesday.
3. If it is a Tuesday, and if the fraction is greater than or equal to 1367/2160, and if $\text{Remainder}(12G|19)$ is greater than 6, Rosh Hashanah falls two days later, on Thursday (NOT WEDNESDAY!!).

Once you have determined the date of Rosh Hashanah, it is easy to calculate the date of Passover in the same (Gregorian or Julian) year. Let $M =$ the number of days from September 6 to Rosh Hashanah. In the example for 1996, $M=$September 14 - September 6 = 8 days.

Count $M$ days from March 27. That is the date of Passover. It actually begins at sundown on the previous evening. In the example for 1996, 8 days after March 27 is April 4 (there are 31 days in March), so Passover begins at sundown on April 3.

public static Date Passover(int nYear) {
    Source: William H. Jefferys, Department of Astronomy, University of Texas Austin, TX 78712
    http://quasar.as.utexas.edu
    Once you have determined the date of Rosh Hashanah, it is easy to calculate the date of Passover in the same (Gregorian or Julian) year. Let $M =$ the number of days from September 6 to Rosh Hashanah. In the example for 1996, $M=$September 14 - September 6 = 8 days.
    Count $M$ days from March 27. That is the date of Passover. It actually begins at sundown on the previous evening. In the example for 1996, 8 days after March 27 is April 4 (there are 31 days in March), so Passover begins at sundown on April 3.
}

public static Date DominionDay (int nYear) {
    // 01 July YYYY
}

public static Date BoxingDay (int nYear) {
    // Day after Christmas, December 26th...
}

public static String getClassInfo() {
    return ("Name: Holidays\n    "Author: Gregory N. Mirsky\n    "Updated: John D. Mitchell\n    "Version 1.02\n    "Copyright 1997, All rights reserved.");
}