AP Computer Science — Project 4 - Operators
Dr. Paul Bailey — Monday, August 26, 2019

Create one directory called `P4_Operators` to store the `.java` files. Unless explicitly told to do so, *do not use any of the Java standard library*. Please format your source code using appropriate indentation.

The purpose of this project is to practice our use of arithmetic operators, to translate written language and equations into code, and to begin to understand the complexity of date computations. In the next project we will use these methods to practice the material from Savitch sections 3.1 - 3.3.

We wish to implement the concept of an *internal date*, which is an integer defined to be the number of days since a given (fixed) date. We wish to be able to convert between internal date (an integer) and external date (a string of the form MM/DD/YYYY). Using this, we will be able to state the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat) of an internal date, and consequently, of an external date.

In order to convert an internal date into an external date, one needs to select the calender type, which "overlays" the raw number of days to give each day a name. We give a bit of history regarding calenders (from Wikipedia).

- The Roman calender had ten months with various numbers of days in each month. These days per month did not add up to a year, so there were continual adjustments and proposed modifications. Other civilizations at the time had differing calender schemes.
- The Julian calender is the result of a reform instigated by Julius Caesar in 46 B.C. He added two months (which, after some rearranging, later became July (for Julius Caesar) and August (for Augustus Caesar). This calender added one leap day every four years, and so, after many decades, the calender became out of sync with the seasons.
- The Gregorian calender was introduced by Pope Gregory XIII in October, 1582, and refined the rule for leap years to adjust for the discrepancy of the Julian calender. This is the calender in modern use.

One standardized form of internal date is *Julian date*, as explained at

`http://en.wikipedia.org/wiki/Julian_day`.

To quote this source, "Julian day number 0 is assigned to the day starting at noon on Monday, January 1, 4713 BC, proleptic Julian calendar."

We will call Julian day number "Julian date", and use it as one for of internal date. We will use Gregorian date as our external date.

Program 0. Create a new class `Julian`. In it we will store some static methods which deal with Julian dates. Include the given method to format a date.

```
public class Julian
{
    public static String formatDate(int M, int D, int Y)
    {
        return String.format("%02d/%02d/%04d", M, D, Y);
    }
}
```

For testing, it may be useful to know some Julian dates.

Gregorian Date	Julian date
July 4, 1776	2369916
January 1, 1900	2415021
January 1, 1970	2440588
August 25, 2019	2458722

Program 1. According to Wikipedia, we may convert Gregorian date to Julian date by the following computation:

> **Converting Gregorian calendar date to Julian Day Number** [edit]
>
> The algorithm[60] is valid for all (possibly proleptic) Gregorian calendar dates after November 23, −4713.
>
> JDN = (1461 × (Y + 4800 + (M − 14)/12))/4 +(367 × (M − 2 − 12 × ((M − 14)/12)))/12 − (3 × ((Y + 4900 + (M - 14)/12)/100))/4 + D − 32075

In the `Julian` class, write a method

```
public static int julianDate(int M, int D, int Y),
```

which takes the Gregorian date and returns the Julian date. Here, `M` is month (1 through 12), `D` is the day of the month (1 through 31), and `Y` is the year (1000 through 2999).

Program 2. According to Wikipedia, we may convert Julian date to Gregorian date by the following computation:

> **Julian or Gregorian calendar from Julian day number** [edit]
>
> This is an algorithm by Richards to convert a Julian Day Number, **J**, to a date in the Gregorian calendar (proleptic, when applicable). Richards states the algorithm is valid for Julian day numbers greater than or equal to 0.[63][64] All variables are integer values, and the notation "*a* div *b*" indicates integer division, and "mod(*a*,*b*)" denotes the modulus operator.
>
> **Algorithm parameters for Gregorian calendar**
>
> | variable | value | variable | value |
> |---|---|---|---|
> | *y* | 4716 | *v* | 3 |
> | *j* | 1401 | *u* | 5 |
> | *m* | 2 | *s* | 153 |
> | *n* | 12 | *w* | 2 |
> | *r* | 4 | *B* | 274277 |
> | *p* | 1461 | *C* | −38 |
>
> For Julian calendar:
>
> 1. $f = \mathbf{J} + j$
>
> For Gregorian calendar:
>
> 1. $f = \mathbf{J} + j + (((4 \times \mathbf{J} + B) \text{ div } 146097) \times 3) \text{ div } 4 + C$
>
> For Julian or Gregorian, continue:
>
> 2. $e = r \times f + v$
> 3. $g = \text{mod}(e, p) \text{ div } r$
> 4. $h = u \times g + w$
> 5. $\mathbf{D} = (\text{mod}(h, s)) \text{ div } u + 1$
> 6. $\mathbf{M} = \text{mod}(h \text{ div } s + m, n) + 1$
> 7. $\mathbf{Y} = (e \text{ div } p) - y + (n + m - \mathbf{M}) \text{ div } n$
>
> **D**, **M**, and **Y** are the numbers of the day, month, and year respectively for the afternoon at the beginning of the given Julian day.

In the `Julian` class, write a method

```
public static String gregorianDate(int J),
```

which takes the Julian date and returns the Gregorian date. Here, `J` is the Julian day number. Once you compute `M`, `D`, and `Y`, use `formatDate` to format the return value.