Sprichwort

Aufgabe 1

Laurenz Grote

Inhaltsverzeichnis

Meine Umsetzung für „Rhinozelfant“ erfolgte unter Arch Linux mit Java 1.8. Der Programmcode liegt sowohl als ausführbare JAR-Datei als auch als Quellcode vor.

1 Lösungsidee

1.1 Definition des Osterdatums

> Als Osterdatum wurde im Jahre 325 auf dem Konzil von Nicäa der erste Sonntag nach dem ersten Vollmond im Frühling (Datum des Frühlingsvollmondes), der frühestens am 21. März stattfinden kann, festgelegt. Der früheste Ostersonntag fällt folglich auf den 22. März, der späteste auf den 25. April.
>
> –Wikipedia, https://de.wikipedia.org/wiki/Osterdatum

All diese Bedingungen wurden von Carl Friedrich Gauß in der Gaußschen Osterformel[1] zusammengefasst. In meinem Programm nutze ich die im Wikipedia-Artikel vorgestellte Fassung von Heiner Lichtenberg.

1.2 Unterschied zwischen gregorianischem und julianischem Kalender

Das Weihnachtsfest wird in beiden Religionen am 25. Dezember gefeiert. Die russisch-orthodoxe Kirche hat allerdings die 1852 durchgeführte Kalenderreform durch Papst Gregor nicht angewandt. Der deshalb dort noch genutzte Vorgängerkalender, der julianische Kalender, stimmt allerdings mit dem heutigen Kalender bis auf die Schaltjahresregelung komplett überein.

Im julianischen sowie dem gregorianischen Kalender ist jedes durch 4 restlos teilbare Jahr ein Schaltjahr. Allerdings sind im gregorianischem Kalender Jahre, die durch 100 restlos teilbar sind, ausgenommen und damit regelmäßige Jahre. Von dieser Ausnahmeregelung sind wiederum Jahre, die restlos durch 400 teilbar sind ausgenommen und damit doch wieder Schaltjahre.

Ferner wurden im Reformjahr 1852 10 Kalendertage übersprungen. Deshalb driften die beiden Kalendersysteme alle hundert Jahre um durchschnittlich 0.75 Tage auseinander. Aus beiden Bedingungen lässt sich folgende Formel[2] für den Abstand in Tagen zwischen gregorianischem und julianischem Kalender ableiten, wobei x die Jahreszahl ist. Außerdem sind wiederum alle Divisionen ohne Rest auszuführen! Für die Monate Januar und Februar gelten Ausnahmen, die für uns aber irrelevant sind (25. Dez. und Ostern zwischen März und April). **ECHT?**

$$f(x) = x \div 100 - x \div 400 - 2$$

[1] `https://de.wikipedia.org/wiki/Gau%C3%9Fsche_Osterformel`
[2] `https://de.wikipedia.org/wiki/Umrechnung_zwischen_julianischem_und_gregorianischem_Kalender`

1.3 Lösungsidee

Mit diesen Sachinformationen lässt sich das Problem in recht wenig Zeit mittels Brute Force lösen: Von 2016 an berechnet man alle katholischen Osterfeste und orthodoxen Weihnachtsfeste, bis die Ergebnisse sich gleichen. Eine weitere Optimierung ist nicht möglich, da das Osterdatum unregelmäßig schwankt. Höchstens ist das Caching des orthodoxen Weihnachtsdatums für 100 Jahre möglich. Da also das Überprüfen einer richtigen Lösung sehr schnell geht (es muss nur ein Osterdatum berechnet und umgerechnet werden), das berechnen einer Lösung aber nicht Zeiteffizient möglich ist, ist das Problem **NP-Vollständig**.

Das andere Datum lässt sich dementsprechend finden, indem man alle orthodoxen Osterfeste berechnet, bis das Ergebnis dem 25. Dezember des gregorianischen Kalenders entspricht.

JAHRESOFFSETS!

2 Umsetzung

2.1 Projektstruktur

Zunächst müssen die Daten in Java durch eine Datenstruktur abgebildet waren. Ich habe mich entschieden, allgemeine Eigenschaften eines Datums in einer abstrakten Oberklasse und spezifischen Unterklassen für die jeweiligen Kalendersysteme abzubilden. Hier ein UML-Diagramm:

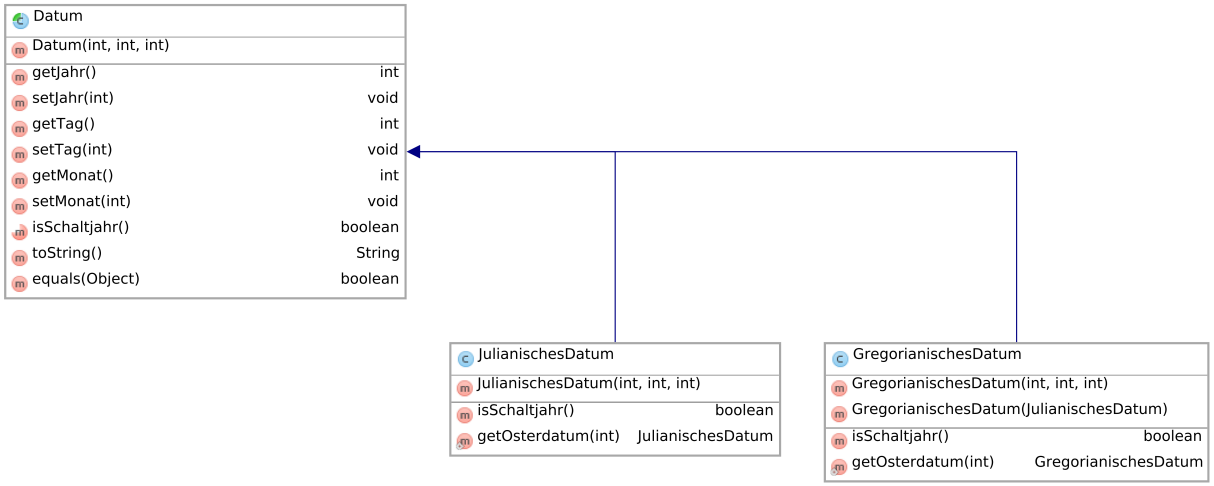

Abbildung 1: UML-Diagramm der Implementation

Um beide Aufgabenstellungen bearbeiten zu können, muss nun für beide Kalnendersysteme getrennt die Osterformel implementiert werden. Da die Ausgabe im gregorianischen Kalendersystem gefordert wird und Daten verglichen werden müssen, müssen Daten aus dem julianischen Kalender in den gregorianischen Kalender umgerechnet werden können. Dazu gibt es in der gregorianischen Klasse einen überladenen Konstruktor, der ein julianisches Datum entgegennimmt und entsprechend umrechnet.

2.2 Implementierungen der beiden Kalendersysteme

2.2.1 Implementierung des julianischen Datums

Für die Umsetzung des julianischen Datums habe ich das obengenannte Schaltjahrkriterium folgendermaßen nach Java übersetzt:

```
1     public boolean isSchaltjahr() {
2         return getJahr() % 4 == 0;
3     }
```

Außerdem habe ich die Osterformel von Wikipedia in Java umgesetzt:

```
1     public static JulianischesDatum getOsterdatum(int jahr) {
2         int m = 15;
3         int s = 0;
4         int a = jahr % 19;
5         int d = (19 * a + m) % 30;
6         int r = (d + a / 11) / 29;
7         int og = 21 + d - r;
8         int sz = 7- (jahr + jahr / 4 + s) % 7;
9         int oe = 7- (og - sz) % 7;
10        int os = og + oe;
11
12        if (os > 31) {
13            return new JulianischesDatum(os - 31, 4, jahr);
14        } else {
15            return new JulianischesDatum(os, 3, jahr);
16        }
17    }
```

2.2.2 Implementierung des gregorianischen Datums

Auch hier habe ich das Schaltjahkriterium folgendermaßen nach Java übersetzt:

```
1     public boolean isSchaltjahr() {
2         return (getJahr() % 4 == 0 && getJahr() % 100 != 0) || (getJahr() % 400 ==
              0);
3     }
```

Ebenso habe ich die für den gregorianischen Kalender angepasste Osterformel umgesetzt.

```
1     public static GregorianischesDatum getOsterdatum(int jahr) {
2         int k = jahr / 100;
3         int m = 15 + (3 * k + 3) / 4 - (8 * k + 13) / 25;
4         int s = 2 - (3 * k + 3) / 4;
5         int a = jahr % 19;
6         int d = (19 * a + m) % 30;
7         int r = (d + a / 11) / 29;
8         int og = 21 + d - r;
9         int sz = 7- (jahr + jahr / 4 + s) % 7;
10        int oe = 7- (og - sz) % 7;
11        int os = og + oe;
12
13        if (os > 31) {
```

```
14              return new GregorianischesDatum(os - 31, 4, jahr);
15          } else {
16              return new GregorianischesDatum(os, 3, jahr);
17          }
18      }
```

Außerdem gibt es in dieser Klasse noch die Umrechnungsfunktion von dem julianischen in den gregorianischen Kalender. Zunächst wird dabei das julianischem Datum übernommen. Danach wird mit der in der Lösungsidee genannten Formel der Abstand zwischen den beiden Kalendersystemen in dem Jahr des Datums berechnet. Dieser Abstand wird dem Tag hinzuaddiert. Danach wird die Funktion korrigiereUeberhang() aufgerufen. Diese korrigiert das Datum falls durch das hinzuaddieren der Tagesdifferenz die Monatsläge überschritten wurde.

Zunächst ermittelt diese Funktion die für das Jahr des Datums zutreffenden Monatslängen (Schaltjahr). Danach wird berechnet, um wie viele Tage das gesetzte Datum die Monatslänge des Datums unter-/überschreitet.

Wenn die Monatslänge überschritten wird, wird folgendermaßen das Datum korrigiert:

Da die Tage die Monatslänge überschritten haben, muss das Datum mindestens im nächsten Monat liegen. Daher wird der Monat des Datums um eins erhöht. Ist der aktuelle Monat ein Dezember, wird dementsprechend das Jahr um 1 erhöht, der Monat auf Januar gesetzt und der Schaltjahresregel entsprechend die für das neue Jahr gültigen Monatslängen gesetzt.

Unterschreiten oder gleichen die überhängenden Tage die Monatslänge des neuen Monates, ist die Umrechnung beendet. Die noch überhangenden Tage sind dann der Tag des neuen Monates.

Überschreiten die überhägenden Tage die Monatslänge des neuen Monates, wird auch dieser Monat, wie zwei Absätze zuvor beschrieben, übersprungen. Die Monatslänge kann von den überhangenden Tagen dann subtrahiert werden.

Dies wird solange wiederholt, bis das Datum erfolgreich umgerechnet wurde.

```
1       public GregorianischesDatum(JulianischesDatum julianischesDatum) {
2           // Daten übernehmen
3           super(julianischesDatum.getTag(), julianischesDatum.getMonat(),
                julianischesDatum.getJahr());
4
5           // Abstand zwischen Kalendersystemen
6           int abstandInTagen = (getJahr() / 100) - (getJahr() / 400) - 2;
7
8           // Auf den Tag dazuaddieren
9           setTag(getTag() + abstandInTagen);
10
11          // Jetzt evt. Überschreitung der Monatslänge korrigieren
12          korrigiereUeberhang();
13      }
14      private void korrigiereUeberhang () {
15          int[] gueltigeMonatslaengen;
16
17          // Monatslängen für diese Jahr festlegen
18          if (isSchaltjahr()) {
19              gueltigeMonatslaengen = monatslaengenSchaltjahr;
20          } else {
21              gueltigeMonatslaengen = monatslaengen;
```

```
22          }
23
24          // Um wieviele Tage über-/unterschreitet das Datum die Monatslänge?
25          int ueberhang = getTag() - gueltigeMonatslaengen[getMonat()];
26
27          // SOLANGE es zu viele sind
28          while (ueberhang > 0) {
29              // Schiebe den Monat (bei Dez. Jahr) um eins voran
30              if (getMonat() == 12) {
31                  setJahr(getJahr() + 1);
32                  setMonat(1);
33
34                  // gueltigeMonatslaengen festlegen
35                  if (isSchaltjahr()) {
36                      gueltigeMonatslaengen = monatslaengenSchaltjahr;
37                  } else {
38                      gueltigeMonatslaengen = monatslaengen;
39                  }
40              } else {
41                  setMonat(getMonat() + 1);
42              }
43              // Passen die übriggebliebenen Tage in den aktuellen Monat
44              if (ueberhang <= gueltigeMonatslaengen[getMonat()]) {
45                  // Wen ja ist der verbleibende Überhang der Monatstag
46                  setTag(ueberhang);
47                  // Und danach ist kein Überhang merh da
48                  ueberhang = 0;
49              } else {
50                  // Sonst wird der Überhang um die Monatslänge verringert
51                  ueberhang -= gueltigeMonatslaengen[getMonat()];
52              }
53          }
54      }
```

2.3 Quelltext Suchcode

```
1     private static String wannKatholischeOsternOrthodoxeWeihnacht() {
2         int i = 2015;
3         GregorianischesDatum katholischeOstern, orthodoxeWeihnacht;
4         do {
5             i++; // Los gehts also bei 2016
6             katholischeOstern = GregorianischesDatum.getOsterdatum(i);
7             // Das orth. Weihnachten müssen wir vom Vorjahr nehmen, da Weihnachten
                  vom 25. Dez. in das neue Jahr hinein verschoben wird
8             // Außerdem konvertieren wir es direkt zu einem gregorianischen Datum
9             orthodoxeWeihnacht = new GregorianischesDatum(new JulianischesDatum
                  (25, 12, i - 1));
10        } while (!katholischeOstern.equals(orthodoxeWeihnacht));
11
12        // Welche Variable wir ausgeben ist egal, sind ja inhaltsgleich....
13        return katholischeOstern.toString();
14    }
15    private static String wannKatholischeWeihnachtOrthodoxeOstern() {
16        int i = 2015;
17        GregorianischesDatum katholischeWeihnacht, orthodoxeOstern;
18        do {
19            i++; // Los gehts also bei 2016
20            katholischeWeihnacht = new GregorianischesDatum(25, 12, i);
21            orthodoxeOstern = new GregorianischesDatum(JulianischesDatum.
                  getOsterdatum(i));
22        } while (!katholischeWeihnacht.equals(orthodoxeOstern));
23
24        // Welche Variable wir ausgeben ist egal, sind ja inhaltsgleich....
25        return katholischeWeihnacht.toString();
26    }
```

3 Beispiel

Die Programmausgabe lautet:

```
$ java -jar Sprichwort.jar
Kath./Ev. Ostern und Orth. Weihnachten finden am folgendem Datum gleichzeitig statt:
23.3.11919
Kath./Ev. Weihnachten und Orth. Ostern finden am folgendem Datum gleichzeitig statt:
25.12.32839
```

Die Ausgaben habe ich dann noch manuell überprüft:

3.1 Kath./Ev. Ostern und Orth. Weihnachten

Die Lösung 23 März 11919^{gre}/25. Dez. 11918^{jul} ist eine korrekte Lösung, weil:

3.1.1 Kath./Ev. Ostern 11919

Nach der Gaußschen Osterformel von 1816.

$j = 11919$
$a = j \bmod 19 = 6$
$b = j \bmod 4 = 3$
$c = j \bmod 7 = 3$
$k = j \div 100 = 119$
$p = (8 \times k + 13) \div 25 = 38$
$q = k \div 4 = 29$
$M = (15 + k - p - q) \bmod 30 = 7$
$d = (19 \times a + M) \bmod 30 = 1$
$N = (4 + k - q) \bmod 7 = 3$
$e = (2 \times b + 4 \times c + 6 \times d + N) \bmod 7 = 0$
$O = 22 + d + e = 23.\text{März}$

Ostern fällt 11919 auf den 23 März$^{\text{gre}}$!

3.1.2 Orthodoxes Weihnachtsdatum 11918

Orthodoxes Weihnachtsdatum: 25. Dez. 11918^{jul}. Umrechnung in den gregorianischen Kalender:

$JH = 119$
$a = JH \div 4 = 29$
$b = JH \bmod 4 = 3$
$TD = 3 \times a + b - 2 = 88$

Vom 25. Dez. 11918 bis zum 1. Jan 11919 sind es 6 Tage, es verbleiben 82 Tage.

Jan 11919: -31	R: 51
Feb 11919: -28 (kein Schaltjahr!)	R: 23

23 Tage Rest, also liegt der 25. Dez. 11918^{jul} auf dem 23 Mär. 11919^{gre}!

3.2 Kath./Ev. Weihnachten und Orthodoxe Ostern

Die Lösung 25. Dez. 32839^{gre}/25. Apr. 32839^{jul} ist eine korrekte Lösung, weil:

3.2.1 Kath./Ev. Weihnachtsfest 32839

32839 findet Weihnachten natürlich am 25. Dez. 32839^{gre} statt.

3.2.2 Orth. Ostern 32839

Nach der Gaußschen Osterformel von 1816

$j = 32839$
$a = j \bmod 19 = 7$
$b = j \bmod 4 = 3$
$c = j \bmod 7 = 3$
$k = j \div 100 = 328$
$M = (15 + k - p - q) \bmod 30 = 15$
$d = (19 \times a + M) \bmod 30 = 28$
$N = (4 + k - q) \bmod 7 = 6$
$e = (2 \times b + 4 \times c + 6 \times d + N) \bmod 7 = 6$
$O = 22 + d + e = 56.\text{März} \mathrel{\widehat{=}} 25.\text{Apr}$

Osterdatum : 25. Apr. 32839^{jul}

Umrechnung in den Gregorianischen Kalender:

$JH = 328$
$a = JH \div 4 = 82$
$b = JH \bmod 4 = 0$
$TD = 3 \times a + b - 2 = 244$

Vom 25. Apr. 32839 bis zum 1. Mai sind es 5 Tage, es verbleiben 239 Tage:

Mai 32839: -31	R: 208
Jun 32839: -30	R: 178
Jul 32839: -31	R: 147
Aug 32839: -31	R: 116
Sep 32839: -30	R: 86
Okt 32839: -31	R: 55
Nov 32839: -30	R: 25

25 Tage Rest, also liegt der 25. Apr. 32839^{jul} auf dem 25. Dez. 32839^{gre}

4 Überlegungen zur Laufzeit

Die Programmlaufzeit steigt mit der Entfernung zwischen Startjahr der Berechnung und dem Ergebnisjahr in etwa linear, da die Osterberechnung eine konstante Laufzeit hat. Zwar werden durchschnittlich alle 3000 Jahre ein weiterer Schleifendurchlauf bei der Umrechnung nötig, dies kann aber vernachlässigt werden. Die Laufzeit ist also, wenn n die Differenz zwischen Startjahr und Eregbnisjahr darstellt:

$$\mathcal{O}(n)$$