Brandenburgische
Technische Universität
Cottbus - Senftenberg

Brandenburgische Technische Universität Cottbus – Senftenberg
Institut für Informatik
PD Dr.-Ing. habil. Thomas Hinze

Sommersemester 2016

Abgabe der Lösungen: bis Mittwoch, 01.06.2016

Einführung in die Programmierung
Übungsblatt 3

Hinweise

- Erstellen Sie bitte eine schriftliche Lösung der nachfolgenden Aufgaben und erzeugen Sie daraus eine pdf-Datei. Sie können auch Ihre (lesbare!) handschriftliche Lösung als pdf einscannen. Achten Sie bitte darauf, dass am Beginn der ersten Seite des pdfs Ihr *Name*, Ihre *Matrikelnummer* und der Name Ihres *Tutors* (Laborübungsgruppe) vermerkt ist. Jede Aufgabe, die durch **(Laborübung)** gekennzeichnet ist, beinhaltet als Lösung auch ein Java-Programm als Quelltextdatei. Der Quelltextdateiname muss dem Namen der *Klasse* entsprechen, die die `main`-Methode enthält. Bitte packen Sie Ihr pdf gemeinsam mit allen Java-Quelltextdateien (nur `*.java`, keine `*.class`) in eine zip-Datei, die Sie termingemäß im moodle hochladen.
- Sie bekommen eine Bewertung Ihrer Lösung mit der Information, ob sie ausreichend zum Bestehen des Übungsblattes ist. Im Falle einer nichtausreichenden Lösung haben Sie Gelegenheit, eine verbesserte Version zeitnah, aber spätestens bis 15.07.2016, im moodle nachzureichen. Plagiate werden nicht als ausreichende Lösungen anerkannt.

Aufgabe 1

Geben Sie für die Gleitkommazahlen $x = \pm m \cdot 2^e$

$0.5625 \qquad 1.5 \cdot 2^{127} \qquad -181.25 \cdot 2^{-23} \qquad 9.81$

vom Typ `float` jeweils ihre Binärdarstellung $v, z(e), z(m)$ mit $x = (-1)^v \cdot z(m) \cdot 2^{z(e)}$ im Standard IEEE 754 single an. Dort stehen insgesamt 32 Bit für die Binärdarstellung zur Verfügung, wovon 1 Bit auf das Vorzeichen v entfällt, 8 Bit auf den charakteristischen Exponenten $z(e)$ und 23 Bit auf die normalisierte Mantisse $z(m)$. Es gilt: $v = \begin{cases} 0 & \text{falls } x \geq 0 \\ 1 & \text{falls } x < 0 \end{cases}$ sowie $z(e) = (e + 127_{(10)})_{(2)}$.

Aufgabe 2

- Erläutern und vergleichen Sie die Schleifenkonstrukte `while`, `do-while` und `for`.
- **(Laborübung)** Betrachten Sie folgende vier Quelltextauszüge. Welche der Schleifen terminieren? Wie oft werden die terminierenden Schleifen jeweils durchlaufen, und welche Werte haben die Variablen `i` und `j` nach dem Verlassen der Schleife?

```
/* --- (1) --- */
int i = 1;
int j = 1;
do {
    i = i + j;
    j++;
} while (i < 20);
```

```
/* --- (2) --- */
int i = 100;
int j = 27;
while (i != j) {
    i = i / 20;
    j = j / 3;
}
```

```
/* --- (3) --- */
int i = 1;
int j = 20;
while (i+j > i) {
    i = i + 2;
    j--;
}
```

```
/* --- (4) --- */
int i = 26;
int j = 24, x = 0;
for(x=0; x<1000; x++) {
    i = i/12 + 23*x;
    j = (x--)+j+5;
}
```

Aufgabe 3

(Laborübung) Die Kreiszahl π kann auf vielerlei Arten berechnet werden. Hier betrachten wir ein probabilistisches Verfahren, welches auch in einer Zuckerbäckerei Anwendung finden könnte. Als Utensilien genügen ein quadratisches Backblech und eine darauf gestellte kreisförmige Kuchenform, deren Durchmesser der Kantenlänge des Blechs gleicht. Nun werden aus einiger Höhe möglichst zufällig Zuckerstreusel auf das Blech verteilt. Danach zählt (oder wiegt) man die Streusel, die auf dem Blech in die Kuchenform gefallen sind. Aus dem Verhältnis zur Zahl der Streusel, die insgesamt auf das Blech gefallen sind, kann die Kreiszahl π errechnet werden:

$$\frac{\#_{\text{Kreis}}}{\#_{\text{Quadrat}}} = \frac{\pi \cdot r^2}{(2 \cdot r)^2} \qquad \text{und somit} \qquad \pi = 4 \cdot \frac{\#_{\text{Kreis}}}{\#_{\text{Quadrat}}}$$

Schreiben Sie ein Java-Programm, welches das beschriebene Verfahren umsetzt. Wählen Sie der Einfachheit halber ein Quadrat der Kantenlänge Eins, auf dem eine Viertelkreisscheibe mit Radius Eins liegt. Dann können Sie mit der Methode `Math.random()` eine x- und eine y-Koordinate jeweils zwischen 0 und 1 (pseudo)zufällig erzeugen. x- und y-Koordinate definieren zusammen ein Zuckerstreusel, der auf das Blech fällt. Für Zuckerstreusel, die in den Viertelkreis mit Radius 1 gefallen sind, gilt: $x^2 + y^2 \leq 1$. Gestalten Sie Ihr Programm so, dass insgesamt $1\,000\,000$ Streusel fallen, bevor die ermittelte Näherung für π ausgegeben wird. Vergleichen Sie Ihre Näherung für π mit dem als Konstante `Math.PI` verfügbaren Wert.

Aufgabe 4

(Laborübung) Miss Marple plant einen gemütlichen Bridge-Abend und möchte als Mitspielerinnen drei ihrer Freundinnen Anita, Beate, Chris, Dunja und Evita einladen. Bei der Einladung muss sie allerdings auf die Sympathien und Antipathien der Damen achten, sonst kommt es zum Eklat!

(1) Wenn Anita eingeladen wird, dann auch Chris, sonst ist sie beleidigt.
(2) Dunja kommt nur, wenn auch Chris eingeladen wird.
(3) Beate und Dunja können sich nicht ausstehen und dürfen auf keinen Fall beide eingeladen werden.
(4) Evita und Beate kommen immer gemeinsam oder gar nicht.

Helfen Sie Miss Marple bei der Erstellung der Einladungen, und schreiben Sie ein Java-Programm, das alle möglichen Kombinationen der Damen ausprobiert und nur die zulässigen Kombinationen ausgibt.

Aufgabe 5

(Laborübung) Nach der *ergänzten Gaußschen Osterformel* lässt sich das Datum das Ostersonntags für das Jahr x im Gregorianischen Kalender (erstmals zu Ostern gültig 1583) mithilfe der nachfolgenden Funktionen schrittweise berechnen (div ist die ganzzahlige Division).
Implementieren Sie jede dieser Funktionen als eigenständige Methode, der die benötigten Parameter übergeben werden und die den jeweiligen Funktionswert zurückgibt. Schreiben Sie darüber hinaus eine `main`-Methode, die eine Jahreszahl (z.B. 2016) als Tastatureingabe einliest, auf Plausibilität prüft, bei einem gültigen Wert die Ostersonntagsberechnung per parametrisiertem Methodenaufruf anstößt und das Ergebnis im Format `dd. Maerz` bzw. `dd. April` ausgibt. Vereinen Sie alle Methoden in einer gemeinsamen Java-Klasse.

1. die Säkularzahl ..... $k(x) = x \operatorname{div} 100$
2. die säkulare Mondschaltung ..... $m(k) = 15 + (3k + 3) \operatorname{div} 4 - (8k + 13) \operatorname{div} 25$
3. die säkulare Sonnenschaltung ..... $s(k) = 2 - (3k + 3) \operatorname{div} 4$

4. den Mondparameter ..... $a(x) = x \bmod 19$
5. den Keim für den ersten Vollmond im Frühling ..... $d(a, m) = (19a + m) \bmod 30$
6. die kalendarische Korrekturgröße ..... $r(d, a) = (d + a \operatorname{div} 11) \operatorname{div} 29$
7. die Ostergrenze ..... $og(d, r) = 21 + d - r$
8. den ersten Sonntag im März ..... $sz(x, s) = 7 - (x + x \operatorname{div} 4 + s) \bmod 7$
9. die Entfernung des Ostersonntags von der Ostergrenze ..... $oe(og, sz) = 7 - (og - sz) \bmod 7$
10. das Datum des Ostersonntags als Märzdatum (32. März = 1. April usw.) ..... $os(og, oe) = og + oe$

Testen Sie Ihr Programm für mindestens vier verschiedene gültige Jahreszahlen aus, darunter 2016. Fällt von 2000 bis zum Jahr 2500 der Ostersonntag auf das frühestmögliche Datum am 22. März und auf das spätestmögliche Datum am 25. April und falls ja, wann ist dies jeweils das nächste Mal der Fall? Ergänzen Sie dafür Ihre `main`-Methode in geeigneter Weise.