FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme

Einsendeaufgaben zum Kurs „Einführung in die Objektorientierte Programmierung“ (01618)

Kurseinheit 01 im SS 2014

Letzte Einsendemöglichkeit: 13.04.2014, 23:59 Uhr

1. Die Abgabe erfolgt ausschließlich über das online-Übungssystem.
2. Besuchen Sie dazu die Website
   `https://online-uebungssystem.fernuni-hagen.de/ps/KursStartSeite/01618/SS14`
3. Als Beleger des Kurses 01618 können Sie mit Ihrem FernUni-Account Ihre Lösungen dort einstellen.
4. Sobald eine Korrektur der Aufgaben für Sie bereitsteht, werden Sie per E-Mail informiert.
5. Sollten Sie in Sonderfällen das online-Übungssystem nicht nutzen können (z.B. wegen körperlicher Einschränkungen oder als JVA-Insasse), nehmen Sie bitte direkt Kontakt mit unserem Sekretariat unter (02331 / 987 - 2998) auf.

Aufgabe 1: Kontrollstrukturen (20 Punkte)

Die Studentin Daniela K. hat zum Training ein Programm geschrieben, in dem sie verschiedene Java-Kontrollstrukturen verwendet hat. Leider hat sie bei der Programmerstellung vergessen, das Programm zu kommentieren und versteht jetzt ihren eigenen, zugegeben etwas sonderbaren Quellcode nicht mehr.

a) Versehen Sie das Programm mit Kommentaren, die Ihnen helfen, es zu verstehen. **(5 Punkte)**

b) Testen Sie das Programm mit verschiedenen Eingaben und vollziehen Sie die jeweiligen Ausgaben nach. In welchen Fällen lautet die Ausgabe „`Fehler!`“? Gibt es Eingaben, für die Sie die Ausgabe „`Das kann nicht sein!`“ erhalten? Für welche Eingabe erhalten Sie eine nicht abgefangene Exception? **(10 Punkte)**

c) Erweitern Sie die Implementierung, so dass keine nicht abgefangenen Exceptions mehr geworfen werden. **(5 Punkte)**

```
class ZahlAusdenken {
    public static void main(String[] args) {
        int startzahl = Integer.parseInt(args[0]);
        int ergebnis = startzahl;
        if (ergebnis > 0) {
            ergebnis = ergebnis + 2;
        } else {
            ergebnis = ergebnis * (-1) + 2;
        }
        while (ergebnis > 2) {
            ergebnis = ergebnis - 2;
        }
        int arbeiten = 4;
        for (int i = 2; i <= 20; i = i + 2) {
            arbeiten = arbeiten + startzahl;
        }
        switch (ergebnis) {
        case 0:
            System.out.println("Das kann nicht sein!");
            break;
        case 1:
            System.out.println("Die urspruengliche Zahl war ungerade!");
            break;
        case 2:
            System.out.println("Die urspruengliche Zahl war gerade!");
            break;
        default:
            System.out.println("Fehler!");
        }
        System.out.println("startzahl = " + startzahl);
    }
}
```

Aufgabe 2: Fallunterscheidungen (20 Punkte)

Der Student Klaus K. hat beschlossen, mit seinen ersten Java-Kenntnissen ein kleines Geometrie-Programm zu schreiben. In der ersten Ausbaustufe gibt seine Implementierung zu einer geometrischen Figur lediglich einige elementare Eigenschaften aus. Wird das Programm mit dem Parameter „`Quadrat`“ aufgerufen, lautet die Ausgabe

```
Vier Ecken
Vier Symmetrieachsen
Ist punktsymmetrisch
Ist geschlossen
```

In seiner Implementierung benutzt Klaus größtenteils `if`-Abfragen zur Fallunterscheidung und `equals()`-Aufrufe zum Test, ob zwei Strings identisch sind.

a) Schreiben Sie ein solches Programm, wie Klaus es getan hat. Es sollte für insgesamt 10 verschiedene geometrische Formen z.B. „`Strecke`“, „`Dreieck`“, „`RechtwinkligesDreieck`“, „`Kreis`“, „`Ellipse`“, „`Viereck`“... je vier Ausgabesätze liefern können. **(15 Punkte)**

b) Halten Sie Ihr (und Klaus) Vorgehen für praxistauglich? Wo sehen Sie Fehlerquellen? **(5 Punkte)**
(Eine Lösung zur Vermeidung dieser Fehlerquellen brauchen Sie nicht benennen. Diese werden Sie in Kurseinheit 3 mit der Vererbung kennenlernen.)

Aufgabe 3: Gaußsche Summenformel (20 Punkte)

Einer überlieferten Geschichte nach soll es sich eines Tages um das Jahr 1785 ein Mathematiklehrer Namens Büttner besonders einfach gemacht haben, indem er die Schüler im Unterricht die natürlichen Zahlen von 1 bis 100 hat aufaddieren lassen. Doch die Annahme, dass die Jungen mit dieser Aufgabe einige Zeit beschäftigt wären, schlug fehl: Schon nach kurzer Zeit konnte ein Schüler das richtige Ergebnis verkünden. Dieser Schüler wurde uns später als Carl Friedrich Gauß bekannt, die Summenformel für die arithmetische Reihe, welche dem Jungen geholfen hat, ist daher heute auch noch unter dem Namen *Kleiner Gauß* bekannt. Wir wollen die Berechnung der arithmetische Reihe, also die Summe

$$1 + 2 + 3 + 4 + \ldots + n = \sum_{i=1}^{n} i$$

auf verschiedene Arten in Java implementieren. Ergänzen Sie unten stehendes Programmfragment, indem Sie ...

a) ... innerhalb der Methode `forSumme(int startWert)` die Summe aller natürlichen Zahlen von 1 bis `startWert` mit Hilfe einer `for`-Schleife aufsummieren. **(8 Punkte)**

b) ... innerhalb der Methode `kleinerGauss(int startWert)` die Gaußsche Summenformel anwenden: **(4 Punkte)**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

c) ... mit der Methode `rekursiveSumme(int startWert)` die Summe aller natürlichen Zahlen von 1 bis `startWert` rekursiv die Methode `rekursiveSumme` aufrufend aufsummieren. **(8 Punkte)**

d) In Java ist es keine sonderlich gute Idee, die Berechnung der arithmetischen Reihe rekursiv durchzuführen. Erklären Sie, warum. (Tip: lassen Sie Ihr Programm mit großen Eingaben laufen) **(Bonus)**

```
public class Summierer {

    public static void main(String[] args) {
        Summierer summierer = new Summierer();
        int startWert = Integer.parseInt(args[0]);

        System.out.println("Ergebnis mit for-Schleife: "
                + summierer.forSumme(startWert));
        System.out.println("Ergebnis kleiner Gauss: "
                + summierer.kleinerGauss(startWert));
        System.out.println("Ergebnis rekursiv: "
                + summierer.rekursiveSumme(startWert));
    }

    int forSumme(int startWert) { /* ... */ }

    int rekursiveSumme(int startWert) { /* ... */ }

    int kleinerGauss(int startWert) { /* ... */ }
}
```

Aufgabe 4: Game of Life (20 Punkte)

Das Spiel *Game of Life* ist ein Ende der 60er Jahre vom Mathematiker John Horton Conway erstmalig vorgeschlagener und detailliert untersuchter zellulärer Automat. Gegeben ist dabei ein rechteckiges Feld mit n Zeilen und m Spalten, dessen Komponenten Zellen genannt werden. Jede Zelle kann zwei Zustände annehmen: lebend oder tot. Die zu Beginn lebenden Zellen bilden die Anfangsgeneration. Eine Folgegeneration entsteht nach folgenden Regeln:

- Eine lebende Zelle lebt auch in der Folgegeneration, wenn genau zwei oder drei der acht benachbarten Zellen in der aktuellen Generation leben.
- Eine lebende Zelle stirbt in der Folgegeneration an Einsamkeit oder Überbevölkerung, wenn in der aktuellen Generation weniger als zwei oder mehr als drei der acht Nachbarzellen leben.
- Eine tote Zelle wird in der Folgegeneration lebendig, wenn genau drei ihrer acht Nachbarn in der aktuellen Generation lebendig sind.
- Unabhängig von den obigen drei Regeln sind die Randzellen (also dienjenigen Zellen, die nicht genau acht Nachbarn haben) immer tot.

Unten stehendes Programmfragment implementiert einen Teil der Funktionalität des Game of Life. Die aktuellen Zustände der Zellen sind in der zweidimensionalen Array-Variable `feld` gespeichert, der Wahrheitswert `true` steht dabei für eine lebende Zelle, der Wert `false` für eine tote Zelle. Die Methode `print()` gibt die aktuelle Spielsituation auf der Konsole aus.
Ihre Aufgabe ist es, die Methode `nextGeneration()` derart zu implementieren, dass `feld` nach Aufruf der Methode die Zellzustände der Folgegeneration entsprechend der obigen vier Regeln beinhaltet.

Hinweise:

- Da Sie für die Berechnung aller Zellen der Folgegeneration noch Zugriff auf die Zustände der aktuellen Generation benötigen, sollten Sie die Zustände der Folgegeneration zunächst in ein neues, zweidimensionales `boolean`-Array gleicher Größe schreiben.
- Ihre Implementierung sollte nach Möglichkeit unabhängig von der Größe der Zellkultur funktionieren; Sie dürfen aber von einer rechteckigen Zellkultur ausgehen.

Prüfen Sie anschließend durch Aufruf der `main`-Methode, wie sich die vorgegebene Kultur über 10 Generationen hinweg entwickelt.

```
public class GameOfLife {

    boolean[][] feld = { { false, false, false, false, false },
            { false, false, true, false, false },
            { false, false, true, false, false },
            { false, false, true, false, false },
            { false, false, false, false, false } };

    public static void main(String[] args) {
        GameOfLife myGame = new GameOfLife();
        for (int i = 0; i < 10; i++) {
            myGame.nextGeneration();
            myGame.print();
            System.out.println();
        }
    }

    void print() {
        for (int i = 0; i < feld.length; i++) {
            for (int j = 0; j < feld[i].length; j++) {
                if (feld[i][j] == true)
                    System.out.print("o ");
                else
                    System.out.print(". ");
            }
            System.out.println();
        }
    }

    void nextGeneration() {
        /* ... */
    }
}
```

Aufgabe 5: Ausnahmebehandlung (20 Punkte)

a) Wie heißt das Motto, das von dem unten angegebenen Programm bei Ausführung ausgegeben wird? **(5 Punkte)**

b) Beschreiben Sie detailliert, was bei der Ausführung des Programms passiert, d.h. welche Exceptions auftreten und wo diese abgefangen und behandelt werden! **(15 Punkte)**

```
public class Mathematiker {
    public static void main(String[] args) {
        try {
            try {
                int i = 7 % 5;
                if ((i / (i % 2)) == 1) {
                    throw new Exception();
                }
                System.out.println("Ich mag");
            }
            catch (Exception e) {
                System.out.println("Ich liebe");
                try {
                    if ((7 % 6 / (7 % 6 % 2)) == 1) {
                        throw new Exception();
                    }
                    System.out.println("nichts mehr, als");
                } catch (Exception u) {
                    System.out.println("es,");
                }
            }
            System.out.println("wenn");
            try {
                int i = true & false ? 0 : 1;
                switch (i) {
                case 0:
                    System.out.println("eine Formel");
                case 1:
                    System.out.println("ein Programm");
                default:
                    throw new Exception();
                }
            } catch (ArithmeticException e) {
                System.out.println("abbricht.");
            } catch (Exception e) {
                System.out.println("funktioniert.");
            }
            finally {
                int i = false && true ? 0 : 2;
                switch (i) {
                case 1:
                    System.out.println(";)");
                default:
                    throw new Exception();
                }
            }
        } catch (ArithmeticException e) {
            System.out.println(":(");
        } catch (Exception e) {
            System.out.println(" ");
        }
    }
}
```