3. Übung (Wochenendzettel) zur Veranstaltung *Proinformatik: Objektorientierte Programmierung*

Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Informatik, SoSe 2013
Prof. Dr. Marco Block-Berlitz

1. Mit Conway's Game of Life haben wir uns Schritt für Schritt ein lauffähiges Projekt erarbeitet. Dazu haben wir eine Grobarchitektur entworfen. Analog dazu sollen Sie sich das Projekt JavaGotchi noch einmal vornehmen und auch dort einen sauberen Entwurf angeben, Funktionen und eine Programmschleife identifizieren. Damit die Funktionen zu den jeweiligen Zuständen nicht so klein sind, erweitern Sie es. So könnte Ihr Grundgerüst aussehen:

```
public class JavaGotchi {
   static enum Zustand {GLÜCKLICH, HUNGRIG, ESSEN, VERHUNGERT};
   static Zustand aktuellerZustand;
   static int hungerGrad;

   // ... vielleicht noch weitere Funktionen

   static void verhaltenHungrig() {
      // ...
   }

   static void verhaltenVerhungert() {
      // ...
   }

   static void verhaltenEssen() {
      // ...
   }

   static void verhaltenGlücklich() {
      // ...
   }
```

```
    static void aktualisierung() {
        // ...
    }

    static void warte(int ms) {
        // ...
    }

    static void starteSpiel() {
        aktuellerZustand = Zustand.GLÜCKLICH;
        hungerGrad = 0;
        boolean spielLaeuft = true;
        while (spielLaeuft) {
            switch (aktuellerZustand) {
            case HUNGRIG:
                verhaltenHungrig();
                break;
            case VERHUNGERT:
                verhaltenVerhungert();
                spielLaeuft = false;
                break;
            case ESSEN:
                verhaltenEssen();
                break;
            case GLÜCKLICH:
                verhaltenGlücklich();
            }
            aktualisierung();
            warte(400);
        }
    }

    public static void main(String[] args) {
        starteSpiel();
    }
}
```

Es bietet sich an, die drei Parameter `aktuellerZustand`, `hungerGrad` und `spielLaeuft` global zu deklarieren, um von jeder Funktion darauf zuzugreifen und sie gegebenenfalls verändern zu können.

2. Schreiben Sie ein Programm, das neben der `main`-Funktion eine weitere Funktion `invertiereListe` enthält. Erzeugen Sie in der `main` zwei identisch gefüllte `int`-Listen der Länge 10. Übergeben Sie eine der beiden Listen per Parameterübergabe an die Funktion. Die Funktion soll die Reihenfolge der Werte innerhalb dieses Parameters umdrehen, aber keinen Rückgabewert liefern. Geben Sie die beiden Listeninhalte in der `main` nach Verwendung der Funktion wieder aus.
   Hat Sie das Ergebnis überrascht? Den Grund erfahren Sie später.

3. Werten Sie das folgende Programm aus und geben Sie die Belegungen für `c`, `d`, `e` und `f` nach Abarbeitung des Programms an:

```
public static boolean[][] falsiere (boolean[][] l){
   boolean[][] kopie = l.clone();
   for (boolean[] d1:kopie)
      for (boolean d2:d1)
         d2 = false;
   return kopie;
}

public static void main(String[] args) {
   boolean[][] a = {{true},{true,false},{false,false}};
   boolean[][] b = falsiere(a);
   boolean c = a==b;
   b[2][1] = true;
   b[1]=a[0];
   boolean d = a[0][0] == b[0][0];
   boolean e = a[2][1] != b[2][1];
   a[0][0]=true;
   boolean f = b[1][0]!=b[2][0];
}
```

4. Wie lassen sich die folgenden Zeilen jeweils im Speicher interpretieren. Geben Sie dafür Zeichnungen an:

```
int[][] a = new int[n][m];
int[][] c = {{1,2},{3}};
```

5. Versuchen Sie eine eigene Methode zu schreiben, die das häufigste Symbol in einer Zeichenkette bestimmt (analog zu `haeufigstesSymbol` im Projekt Codeknacker).

6. In einer früheren Übung sollte die Summe $\sum_{i=1}^{100} \frac{i*(i+1)}{2}$ in einer Funktion berechnet werden. Leider liefert der folgende Programmcode nicht das gewünschte Ergebnis. Finden Sie die Fehler:

```
public static double sum1(){
   int i, startwert=1;
   double d, h;
   for (i=--startwert; i>100; i++)
      System.out.println(d);
      {h=(i*i*i)/2;
      d=d+h;
   }
   return d;
}
```

7. Das **Wallis-Produkt** wurde von John Wallis 1655 formuliert und gibt eine Näherung für $\frac{\pi}{2}$ mit der folgenden Vorschrift:

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \ldots = \prod_{i=1}^{\infty} \frac{4i^2}{(2i-1)\cdot(2i+1)}$$

Implementieren Sie dieses Verfahren und geben Sie damit eine Näherung für π an. Wenn Ihnen das keine Schwierigkeiten bereitet hat, dann erweitern Sie Ihre Funktionssammlung zur Näherung von π doch noch um die beiden folgenden:

$$1 + \frac{1}{3} - \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \frac{1}{11} - \ldots = \frac{\pi}{2 \cdot \sqrt{2}} \, (Newton)$$

$$1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{5} + \frac{1}{7} - \frac{1}{8} + \ldots = \frac{\pi}{3 \cdot \sqrt{3}} \, (Euler)$$

Wenn Sie das Thema der Annährungen von π noch weiter fesselt, dann schauen Sie sich die sehenswerte Wikipedia-Seite[1] dazu an und implementieren Sie weitere Verfahren.

8. Die trigonometrische Funktion $sin(x)$ kann durch die folgende unendliche Potenzreihe dargestellt werden:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$$

Wenn diese Reihe nur bis zu einem bestimmten Glied auswertet wird, erhält man einen Näherungswert für $sin(x)$. Entwickeln Sie ein entsprechendes Programmfragment und testen es mit verschiedenen Werten für x im Bereich $[0, \frac{\pi}{2}]$.

[1] http://en.wikipedia.org/wiki/Approximations_of_%CF%80