RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

Lehrstuhl für Hochleistungsrechnen
RWTH Aachen · D-52074 Aachen · GERMANY
http://programmierung.informatik.rwth-aachen.de

Prof. Christian Bischof, Ph.D.
Andre Vehreschild, Jakob T. Valvoda

Übung *Programmierung WS 06/07* – Blatt 9

Lösungen müssen bis zum **8. Januar 2007, 17:00 Uhr** in den Kasten Ihrer Übungsgruppe eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Bitte vergessen Sie nicht, die **Nummer Ihrer Übungsgruppe**, sowie die **Namen** und **Matrikelnummern** der Mitglieder Ihrer 2er-Gruppe auf jedes Lösungsblatt Ihrer Abgabe zu schreiben. Bitte heften Sie ihre Blätter zusammen.

WEIHNACHTSÜBUNG

Aufgabe 1 (2 + 1 + 1 + 1 = 5 Punkte)

Beantworten Sie die folgenden Fragen:

a) Was ist der Unterschied zwischen einer normalen, einer abstrakten Klasse und einem Interface?

b) Beschreiben Sie die Unterschiede zwischen den Ausnahmen `Exception` und `RuntimeException`.

c) Mit Hilfe der Konzepte *Vererbung* und *Generics* ist die Implementierung von abstrakten Datentypen möglich, deren Elemente je nach Bedarf auch unterschiedliche Typen haben können. Denken Sie hierbei insbesondere an die in der Vorlesung vorgestellte `Liste` und das Interface `Vergleichbar`. Was sind in diesem Zusammenhang die Vor- und Nachteile der unterschiedlichen Konzepte?

d) Wie unterscheidet sich die Programmierung eines Computers mit gemeinsamem Speicher von der Programmierung eines Computers mit verteiltem Speicher insbesondere im Hinblick auf den Speicherzugriff?

Aufgabe 2 (5 Punkte)

Gegeben sei folgende Zusammenstellung von Klassen und Interfaces:

```
public interface I {
  char c='a';
}

public class Int {
  private int v;

  public Int(int i) {
    v = i;
  }

  public String toString() {
    return Integer.toString(v);
  }
}
public class S extends Int implements K {
  private double v;

  public S(int i) {
    super((int) K.i);
    v = 13.42;
  }

  public String toString() {
    return ""+gV();
  }

  public int gV(){
      return new Double(v).intValue();
  }
}
```

```
public interface K {
  int gV();
  double i=42.7;
}

public abstract class C implements K {
  public int gI() {
    return (int)i;
  }
}
```

```
public class F extends C implements I {
    private char v;

    public F() {
        v= 'd';
    }

    public String gV(int w) {
        String v="";
        for(int i=0; i<w; ++i)
            v= v+ c+ this.v;
        return v;
    }

    public int gV() { return v; }
}
```

und das Hauptprogramm:

```
public static void main(String[] args) {
  S s= new S(1);
  F f= new F();
  I i= f;      // #1
  K k= s;      // #2
  Int zahl= k;    // #3
  C c= new C();   // #4

  System.out.println(f.gV()); // #5
  System.out.println(f.gV(3)); // #6
  System.out.println(s);  // #7
  System.out.println(zahl); // #8
  System.out.println(c.gI()); // #9
```

```
  System.out.println(k.gV()); //#10
}
```

Analysieren Sie die durch `#1` bis `#10` markierten Anweisungen und beschreiben Sie was passiert. Sollte eine Anweisung nicht valides Java darstellen, dann begründen Sie warum und welcher Fehler auftritt und geben Sie eine Alternative an, die die Semantik des Programms möglichst wenig verändert.

Aufgabe 3 (10 Punkte)

Auf dem quadratischen zweidimensionalen Planeten WoToBa leben die Spezies der Wonky-Tonky und die Basto in einem Beute-Räuber-Verhältnis. Die Wonky-Tonky stellen dabei die Beute dar. Der Planet WoToBa ist der Einfachheit halber in Quadrate aufgeteilt. Jedes Quadrat kann ein Individuum beinhalten oder leer sein. Mit einer *einfachen Nachbarschaft* sind die horizontal sowie vertikal benachbarten Quadrate gemeint, während eine *volle Nachbarschaft* auch die diagonal angrenzenden Quadrate meint.

Die Individuen haben bestimmte Fähigkeiten: Ein Wonky wird nach 5 Zeitschritten geschlechtsreif und damit zu einem Tonky. Treffen zwei Tonkys in einfacher Nachbarschaft aufeinander, so vermehren sie sich und es werden drei neue Wonkys gezeugt. Die gezeugten Wonkys werden auf die voll benachbarten Quadrate beider Tonkys zufällig verteilt. Dabei muss beachtet werden, dass kein Quadrat doppelt belegt wird. Sowohl ein Tonky wie auch ein Wonky kann sich in einem Zeitschritt zufallsgesteuert auf eines seiner Nachbarfelder (volle Nachbarschaft) bewegen oder stehen bleiben.

Ein Basto ist ein Räuber und ernährt sich von Wonkys und Tonkys. Er kann sich in einem Zeitschritt um 2 Felder in jeder Richtung bewegen (volle Nachbarschaft). Am Ende seiner Bewegung prüft der Basto die einfach benachbarten Felder und wenn er mindestens ein Wonky oder Tonky findet so frisst er immer eines davon (egal welches). Treffen sich zwei Bastos (einfache Nachbarschaft) so paaren auch sie sich und zeugen ein neues Basto, welches genau wie die Wonkys auf die angrenzenden Felder gesetzt wird. Ein Basto verhungert, wenn er in 15 aufeinander folgenden Zeitschritten nichts zu Fressen findet, oder stirbt an Verfettung, wenn er in 20 aufeinander folgenden Zeitschritten immer Nahrung findet.

Es paaren sich immer nur zwei Individuen. Treffen also drei Individuen in einfacher Nachbarschaft aufeinander, so findet nur zwischen zweien eine Paarung statt.

a) Definieren Sie eine (abstrakte) Basisklasse für ein Individuum. Die Basisklasse soll folgende Aktionen modelieren:

   - Sie soll nicht instanziierbar sein.
   - Sie kennt das Feld in dem das Individuum lebt.
   - Sie hat die aktuelle Koordinate des Individuums in dem Feld (mit Getter-Methode).
   - Eine Methode realisiert die Bewegung um einen Schritt. Wählen sie solange zufällig eine Bewegungsrichtung und versuchen sie das Individuum in das Zielquadrat zu

setzen (siehe nächsten Aufgabenteil), bis dies erfolgreich ist. (Optionale Erweiterung: Stellen Sie sicher, dass das Individuum vorwärts kommt und nicht immer nur zwischen zwei Feldern hin- und herspringt.)

- Deklarieren Sie eine Methode `aktion()`, welche zusätzliche Aktionen innerhalb eines Zeitschritts ausführen soll.
- Jedes Individuum soll als oder in einem Thread laufen. Der Thread soll terminieren, wenn das Individuum stirbt.
- Synchronisieren Sie alle Individuen über die Zeitschritte. Das heisst, ein Individuum soll sich in einem Zeitschritt bewegen und eine Aktion ausführen (paaren, fressen, sterben). Und dann auf die Aufforderung zur Ausführung des nächsten Zeitschritts warten.
- Deklarieren Sie eine Methode, welche die Art eines Individuums für eine Ausgabe zurückgibt (also z.Bsp. `toString()` oder `kind`).

b) Designen Sie die Klasse WoToBa, welche die zweidimensionale quadratische Welt der Individuen darstellt. Die Größe des Feldes soll bei der Erzeugung einstellbar sein, aber nachher nicht mehr veränderbar. Stellen Sie Methoden zur Verfügung, die

- das Feld erzeugen (Konstruktor),
- die Größe des Feldes auslesen,
- ein Individuum an die angegebene Position setzen (geben sie einen Fehlerwert zurück, wenn die Position schon besetzt war),
- den Inhalt eines Feldes zurückgeben, und
- die das Feld ausgeben (z.Bsp. mit Hilfe der Methoden aus Übung 5 Aufgabe 3).

c) Erweitern Sie jetzt die Basisklasse aus dem ersten Aufgabenteil, um die zwei Arten von Individuen korrekt zu modelieren. Beachten Sie, daß ein Wonky nur ein junges Tonky ist und daher beide in einer Klasse modelliert werden können.

d) Testen Sie ihre Implementierung in dem Sie ein Feld der Grösse 20×20 erzeugen und darauf 20 Tonky sowie 5 Basto zufällig verteilen. Eine Steuermethode übernimmt hierbei die Erzeugung aller Objekte und die Freigabe eines Zeitschritts und die Ausgabe des aktuellen Feldzustands. Iterieren Sie die Evolution der Population für 100 Zeitschritte. (Ein alternatives Abbruchkriterium ist die Ausrottung einer der beiden Arten Tonky oder Basto.)

Hinweise:

- Benutzen Sie Conways Game of Life verfügbar auf den Webseiten zur *Vorlesung* $->$ *Folien* als Vorlage.
- Im neuen Jahr wird auf den Seiten zur Übung ein FAQ eingerichtet. Fragen zur Aufgabe können per Mail an vehreschild@sc.rwth-aachen.de gerichtet werden und werden dann im FAQ für alle beantwortet.