Technische Universität Darmstadt
Fachbereich Informatik
Prof. Dr. Andreas Koch

Allgemeine Informatik 1 im WS 2007/08

Übungsblatt 10

Bearbeitungszeit: 14.01. bis 20.01.2008

Aufgabe 1: Gefangenenlager 2

Wir erinnern uns: Der Roboter **`hans`** steckt im Gefängnis. Der Gefängnisdirektor ist nicht besonders nett und lässt die Gefangenen heute zur sinnlosen Strafarbeit antreten: sie müssen ein komplettes Feld von Beepern einsammeln und danach wieder hinlegen. Und natürlich muss es schön ordentlich zugehen: die Gefangenen stehen alle nebeneinander und sollen beim Sammeln im Gleichschritt laufen.

Laden Sie sich von der Übungswebseite auch die Datei **uebung10-1.task** herunter, diese enthält das Grundgerüst der Aufgabe. Die Klasse **`Gefangener`** besteht aus einem zu definierenden Attribut, einem Konstruktor (wird bei Erzeugung eines Roboters automatisch aufgerufen) und verschiedene noch leere Methoden.

a) Bitte stellen Sie sicher, dass die Klasse **`Gefangener`** ein Attribut **`linkerNachbar`** vom Typ **`Gefangener`** enthält.

b) Implementieren Sie außerdem einen Konstruktor. Dieser soll dafür sorgen, dass neue Gefangene immer ohne Beeper, nach Norden schauend und in der ersten Straße erzeugt werden. Die Avenue ist variabel (Parameter **`int avenue`**), außerdem wird mit diesem Konstruktor das Attribut **`linkerNachbar`** initialisiert (Parameter **`Gefangener g`**).

   Dieser Parameter enthält entweder einen Roboter vom Typ **`Gefangener`** oder den Wert **`null`** (bedeutet, dass der erzeugte Roboter keinen linken Nachbarn hat).

   Der Konstruktor legt auch fest, wie der Roboter erzeugt wird. In diesem Fall nicht mit z.B. **`new Robot(1, 1, 0, North)`**, sondern mit z.B. **`new Gefangener(7, null)`**.

c) Im **`task`** wird schon ein Beeperfeld der Größe 7 x 5 erzeugt, Sie brauchen also 5 Roboter, um es aufzusammeln. Erzeugen Sie diese 5 Roboter, wobei der in der ersten Avenue keinen linken Nachbarn hat und der in der 5. Avenue unser Bekannter **`hans`** ist.

d) Implementieren Sie nun die Methode **`void sammeln()`**. In ihr soll der Roboter einen Beeper aufsammeln, einen Schritt gehen und dann seinem linken Nachbarn sagen, er soll ebenfalls **`sammeln()`** – das aber nur, wenn er einen linken Nachbarn hat, **`linkerNachbar`** also ungleich **`null`** ist.

e) Jetzt testen Sie im **`task`** die Funktion Ihrer Methode: solange **`hans`** auf einem Beeper steht (**`while`**), soll er **`sammeln()`**. Bei der Ausführung des Programms sollte dadurch nicht nur **`hans`** sollte eine Beeperreihe einsammeln, sondern auch alle seine Nachbarn.

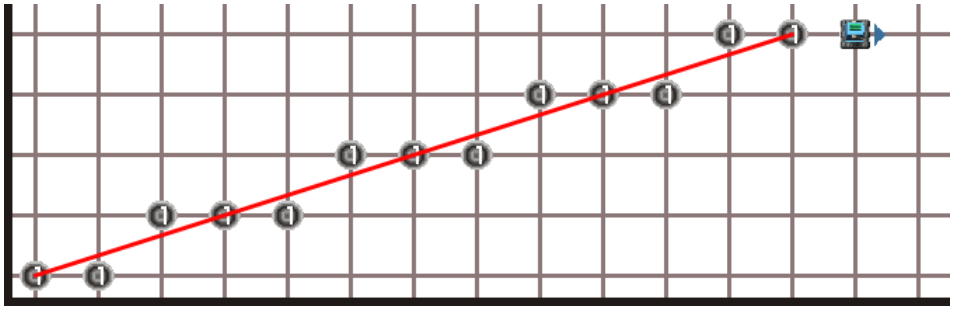
f) Ergänzen Sie jetzt analog zu **`sammeln()`** die Methoden **`umdrehen()`** (um 180 Grad drehen und dem Nachbar Bescheid sagen) und **`ablegen()`** (einen Schritt gehen, einen Beeper ablegen, Nachbar Bescheid sagen).

g) Erweitern Sie nun auch den **`task`**: nach dem Einsammeln soll sich **`hans umdrehen()`**, solange er Beeper hat, sie **`ablegen()`** und sich dann wieder **`umdrehen()`**. Wenn das funktioniert, können Sie das Sammeln und Ablegen auch mittels **`loop(100) {...}`** wiederholen lassen, damit **`hans`** auch wirklich lernt, dass man keine Beeper klauen soll...

Aufgabe 2: Linien aus Beepern

Gegeben ist eine Datei **uebung10-2.task** (auf unserer Webseite, Adresse siehe oben), die Sie erweitern sollen **(also keine eigene Datei anfangen!)**.

Sie sollen die Methode **`void drawLine(int street, int avenue)`** fertig implementieren. Beim Aufruf dieser Methode soll ein Roboter, der auf (1, 1) steht, eine „Linie“ aus Beepern bis zur Kreuzung (**`street`**, **`avenue`**) legen. In der folgenden Abbildung ist eine solche Linie zur Kreuzung (13, 5) und in rot die Ideallinie zu sehen:



Es fällt auf: von links nach rechts geht in jeder Avenue die rote Ideallinie entweder genau durch eine Kreuzung oder sie schneidet die Avenue zwischen zwei Streets. Im ersten Fall liegt der Beeper auf der geschnittenen Kreuzung, im zweiten Fall liegt er auf der Kreuzung, die den kleineren senkrechten Abstand zur Ideallinie hat.

Ein Algorithmus sieht zum Zeichnen einer Linie aus Beepern von (1, 1) sieht also folgendermaßen aus: der Roboter geht immer einen Schritt nach Osten (Hauptrichtung), wodurch der Abstand zur Ideallinie um die Steigung größer wird. Wenn dieser Abstand größer als ½ wird, geht der Roboter zusätzlich einen Schritt nach Norden (Nebenrichtung), wodurch der Abstand zur Ideallinie um 1 kleiner wird. Der Algorithmus ist beendet, wenn der Roboter die Kreuzung (**`street`**, **`avenue`**) erreicht hat.

Das funktioniert allerdings exakt so nur, wenn die Steigung der Geraden kleiner 1 ist. Wenn die Steigung größer ist, werden aber einfach Haupt- und Nebenrichtung vertauscht und es wird mit dem Kehrwert der Steigung gerechnet.

Teilaufgaben:

a) Prüfen Sie, ob die Steigung der Linie größer 1 ist oder nicht (dazu genügt es, **`street`** mit **`avenue`** zu vergleichen!). Dementsprechend initialisieren Sie die Steigung (Differenzenquotient: $m = \frac{street - 1}{avenue - 1}$ bzw. $m = \frac{avenue - 1}{street - 1}$, auf jeden Fall muss $m \leq 1$ gelten) und die Haupt- und Nebenrichtung mit **`East`** und **`North`**. Benutzen Sie die vorgegebenen Variablennamen! Achtung: bei der Berechnung der Steigung benötigen Sie einen **Typecast** nach **`(double)`**!

b) Zeichnen Sie die Linie (benutzen Sie dabei die Roboter-Statusabfrage **`boolean areYouHere(int s, int a)`** und die vorgegebene Methode **`void turnTo()`**!):

```
1. Solange der Roboter nicht bei (street, avenue) ist: {
2.     Beeper legen
3.     Einen Schritt in die Hauptrichtung d1 gehen
4.     Abstand zur Ideallinie (fehler) steigt um steigung
5.     Wenn der Abstand zur Ideallinie größer als 0.5 ist: {
6.         Einen Schritt in die Nebenrichtung d2 gehen
7.         Abstand zur Ideallinie wird um 1 kleiner
8.     }
9. }
```

c) Abschließend muss noch der Beeper auf der Zielkreuzung gelegt werden, außerdem soll der Roboter noch einen Schritt in die Hauptrichtung gehen.

d) Testen Sie Ihr Programm mit verschiedenen Zielkoordinaten!