Technische Universität Darmstadt
Fachbereich Informatik
Prof. Dr. Andreas Koch

Allgemeine Informatik 1 im WS 2007/08

Programmierprojekt

Bearbeitungszeit: 03.12.2007 bis 11.01.2008

Organisatorisches (WICHTIG!):

- Das Programmierprojekt ist in der Gruppe zu bearbeiten, in der Sie sich zum Projekt angemeldet haben (also mit ein oder zwei Gruppenmitgliedern).
- Dieses Programmierprojekt ist mit 20 Punkten angesetzt, das sind 20% ihrer Endnote im Fach Allgemeine Informatik 1. Die Abschlussklausur macht die restlichen 80% aus.
- Verwenden Sie als Vorlage die Datei **propro.task**, die Sie auf unserer Webseite finden, und erweitern Sie diese. Fangen Sie keine neue Datei an!
- Tragen Sie als erstes an den dafür vorgesehenen Stellen am Anfang der Datei Name, Matrikelnummer und RBG-Login aller Gruppenmitgliedern ein.
- Die Übung wird per KarelJIDE-Submit-Funktion an uns gesendet (in der KarelJIDE auf **Submit** klicken), wobei für „Exercise sheet“ und „Exercise number“ jeweils „1“ angegeben werden muss. **Vorher ist allerdings unbedingt der Einstellungsassistent auf** http://www.dpsg-eltville.de/ai/ **zu benutzen** (dort finden Sie auch Informationen für die Arbeit von zu Hause)**!**
- Abgabeschluss ist spätestens der 11. Januar 2008.
- Bitte formatieren und kommentieren Sie ihren Quellcode hinreichend.
- **Achtung:** „Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe ihrer Lösung bestätigen Sie, dass Sie der alleinige Autor / die alleinigen Autoren des gesamten Materials sind. Bei Unklarheiten zu diesem Thema finden Sie weiterführende Informationen auf http://www.informatik.tu-darmstadt.de/Plagiarism oder sprechen Sie Ihren Betreuer an.“

Grundlegendes:

In diesem Programmierprojekt werden Roboter konstruiert, die „Run Length Encoding“ („Lauflängenkodierung“), ein simples Datenkompressionsverfahren, in einer vereinfachten Form implementieren.

Beim RLE werden Daten nicht in ihrer ursprünglichen Form gespeichert, sondern als Blöcke mit einer Angabe, wie oft sie wiederholt werden. Aus „**333336666**“ wird also „**5346**“ (5x „3“, 4x „6“) – für dieselben Daten werden nun nur 4 statt 10 Zeichen benötigt.

In den Aufgaben werden Klassen- und Methodenstrukturen vorgegeben – halten Sie sich unbedingt daran. Sie können allerdings weitere Methoden schreiben, wenn Sie das für sinnvoll halten.

Wir erwarten, dass Sie sinnvoll Schleifen und Methodenaufrufe verwenden!

Aufgabe 1: EnhancedRobot

In dieser Aufgabe sollen Sie eine eigene Roboterklasse **EnhancedRobot** schreiben, die von der Klasse **Robot** erbt und um verschiedene nützliche Methoden erweitert wird:

- **`void goTo(int street, int avenue, direction dir)`**
  Der Roboter soll sich an die Position (**street**, **avenue**) bewegen und anschließend in die Richtung **dir** schauen. Benutzen Sie **while** und die Statusabfragen der Klasse **UrRobot**!
- **`void putBeepers(int n)`**
  Der Roboter soll **n** Beeper ablegen. Sie müssen nicht prüfen, ob er genügend dabei hat.
- **`int countBeepers()`**
  Der Roboter soll die Anzahl der Beeper auf der aktuellen Kreuzung zählen und als Ergebnis der Methode zurückgeben. Hinterher sollen natürlich genau so viele Beeper auf der Kreuzung liegen wie vor der Zählung!

Aufgabe 2: RLE

Schreiben Sie eine Roboterklasse **RLE**, die von **EnhancedRobot** erbt. Diese Klasse soll folgendes beinhalten:

- ein Attribut **data** vom Typ **`int[]`**
  Dieses enthält im weiteren Verlauf die zu komprimierenden Originaldaten („**333336666**“) bzw. die dekodierten Daten („**333336666**“ → „5346“ → „**333336666**“). Die Daten bestehen aus Ziffern zwischen 1 und 9 – keine Null!
- einen Konstruktor **`RLE(int st, int av)`**
  Dieser soll den Roboter an der Position (**st**, **av**) mit 999999 Beepern und Blickrichtung **East** erzeugen (nutzen Sie dazu den Konstruktor der **super**-Klasse!). Mit dem Array **data** soll noch nichts passieren.
- einen Konstruktor **`RLE(int st, int av, int[] dat)`**
  Dieser soll den Roboter ebenfalls an der Position (**st**, **av**) mit 999999 Beepern und Blickrichtung **East** erzeugen, allerdings soll anschließend das Attribut **data** mit dem Array **dat** initialisiert werden.
- eine Methode **`void write()`**
  Diese soll den Inhalt des Arrays **data** ausgeben können. Dazu soll der Roboter für jedes Element des Arrays entsprechend viele Beeper legen und jeweils einen Schritt nach vorne gehen.

Testen Sie die Klasse, indem Sie im **Task** mit Hilfe des in der Vorlage schon vorhandenen Arrays **numbers** einen **RLE**-Roboter bei (**1**, **1**) erzeugen (welchen Konstruktor müssen Sie also benutzen?). Nachdem dieser sein Array ausgegeben hat, soll die Welt folgendermaßen aussehen:

Aufgabe 3: Encoder

In dieser Aufgabe geht es um die Kompression, also den Schritt von „**333336666**“ zu „5346“. Schreiben Sie dazu eine Klasse **Encoder**, die von **RLE** erbt und folgendes beinhaltet:

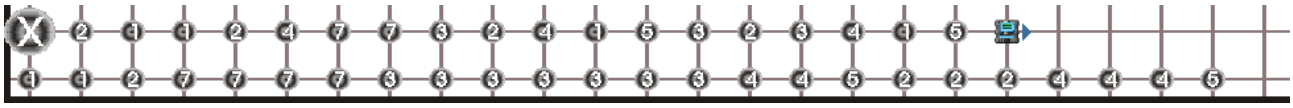
- einen Konstruktor, der Straße, Avenue und ein **`int`**-Array erwartet und die Erzeugung des Roboters an den entsprechenden Konstruktor der **super**-Klasse übergibt.

- eine Methode **`void encode()`**
  Diese soll vor der eigentlichen Kodierung zuerst so viele Beeper legen, wie das Array **`data`** Elemente enthält (**`data.length`**) und dann einen Schritt nach vorne gehen – das ist für die Dekodierung wichtig (siehe Aufgabe 4). Sofern das ursprüngliche Array mehr als neun Elemente enthält, zeigt die KarelJIDE ein „X“ anstatt der Anzahl der Beeper an.

  Anschließend soll die Methode die RLE-Kodierung des Arrays **`data`** berechnen und sie mit Beepern legen (siehe Abbildung). Dazu müssen Sie das Array durchlaufen und dabei herausfinden, wie viele gleiche Array-Elemente aufeinander folgen. Wie Sie das tun, bleibt Ihnen überlassen…

  Hinweise: paarweise Vergleiche! Beim erstem und/oder letztem Array-Element müssen Sie außerhalb der Schleife auch etwas tun.

Ersetzen Sie nun im **`Task`** den **`RLE`**-Roboter durch einen **`Encoder`**. Nachdem jetzt dieser das ursprüngliche Eingabe-Array mit den unkomprimierten Daten ausgegeben hat, soll er sich nach (**2**, **1**) begeben und dort Richtung **`East`** das Array RLE-kodiert ausgeben. Danach soll die Welt folgendermaßen aussehen:



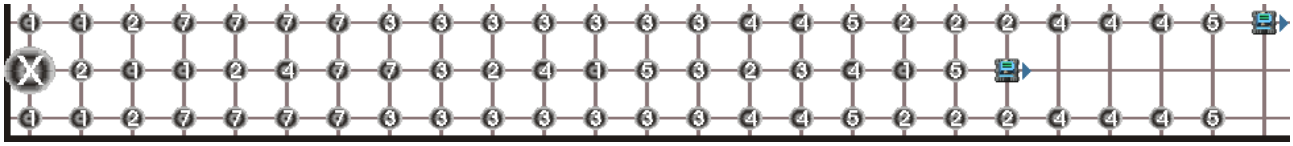
Aufgabe 4: Decoder

Natürlich wollen wir überprüfen, ob die Dekompression (also „5346“ ➔ „333336666“) wieder die ursprünglichen Daten ergibt. Schreiben Sie dazu eine Klasse **`Decoder`**, die von **`RLE`** erbt und folgendes beinhaltet:

- einen Konstruktor, der Straße und Avenue erwartet. Nachdem mit dem entsprechenden Konstruktor der **`super`**-Klasse ein Roboter erzeugt wurde, sollen noch die Beeper auf der aktuellen Kreuzung gezählt werden, das Array **`data`** mit dieser Größe neu angelegt werden und ein Schritt nach vorne gegangen werden.
- eine Methode **`void decode()`**
  In dieser soll der Roboter von seiner aktuellen Position aus geradeaus laufen und dabei sein Array **`data`** füllen, indem er die vor ihm liegenden Beeper-Daten dekomprimiert: wenn auf der aktuellen Kreuzung 7 Beeper liegen und auf der nächsten 5, müssen die nächsten 7 Werte des Arrays mit dem Wert 5 gefüllt werden. Da die Ziffer „0“ nicht vorkommt, wissen Sie, wie weit der Roboter laufen muss.

  Hinweis: Sie werden dabei eine Variable benötigen, um den Index des Arrays zu verwalten.

Erzeugen Sie nun im **`Task`** einen **`Decoder`**-Roboter an der Position (**2**, **1**). Der Roboter soll nun die zweite Straße dekodieren und die dekodierten Daten auf der dritten Straße (in Richtung **`East`**) ausgeben. Die Welt sieht anschließend so aus:



Aufgabe 5: Kompressionsrate

Nun wollen wir genau wissen, wie sehr wir die Originaldaten komprimiert haben – oder ob die RLE-Kodierung sogar mehr Platz braucht. Dazu bestimmen wir die Kompressionsrate in Prozent (ohne Nachkommastellen, also als **`int`**):

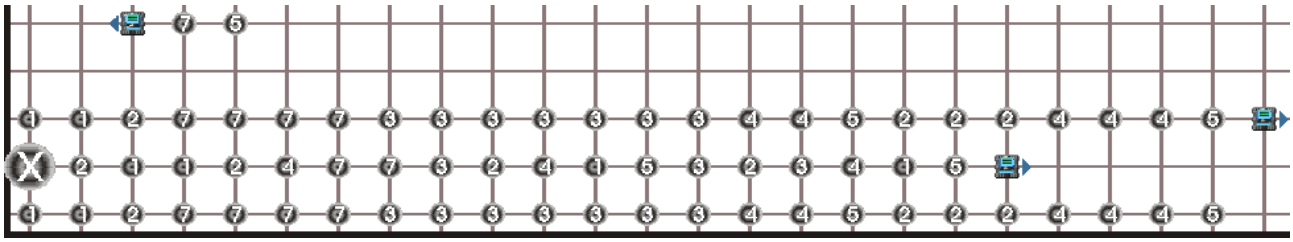
ratio = 100 * anzahl_elemente_kodiert / anzahl_elemente_original

anzahl_elemente_original kennen Sie aus der letzten Aufgabe. Um ***anzahl_elemente_kodiert*** zu bestimmen, hilft Ihnen die Position des Roboters, der noch in der zweiten Straße steht. Achtung: den ersten Beeperstapel nicht mitzählen, er gehört nicht zur eigentlichen RLE-Kodierung! Wenn Sie zwei **int**-Werte dividieren, fallen die Nachkommastellen weg, das wollen wir aber erst im Endergebnis! Sie brauchen also **TypeCasts** nach **`double`** (und für das Ergebnis zurück nach **`int`**).

Um die Kompressionsrate mit Beepern in die Welt zu schreiben, benötigen wir nun eine neue Klasse **`Decimal`**, die von **`EnhancedRobot`** erbt.

Der Konstruktor soll Straße, Avenue und einen dritten **`int`**-Wert **`num`** übergeben bekommen, den Roboter mit 999999 Beepern und Blickrichtung **`West`** erzeugen (**`super`**!) und die Zahl **`num`** mit Beepern in die Welt schreiben. Zuerst die Einerstelle (10^0), dann die Zehnerstelle (10^1), die Hunderterstelle (10^2) und so weiter. Um die Stellen nacheinander zu betrachten, helfen Ihnen der Modulo-Operator **`%`** (**`a % 10`** liefert den Rest bei einer ganzzahligen Division von **`a`** durch 10), Division und eine **`while`**-Schleife mit geeigneter Abbruchbedingung.

Erstellen Sie im **`Task`**, nachdem Sie die Kompressionsrate bestimmt haben, einen **`Decimal`**-Roboter bei (**`5`**, **`5`**), der die Rate ausgibt. Das sieht dann so aus:



Aufgabe 6: Histogramm

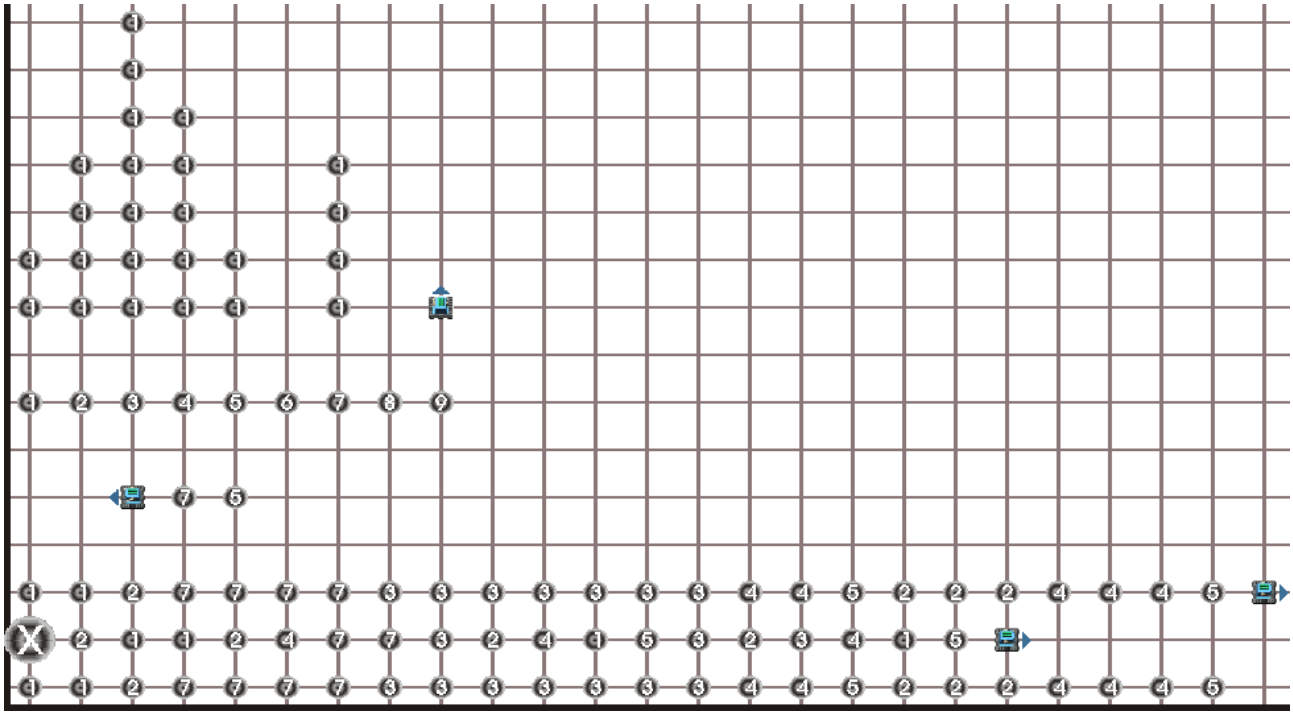
Zuletzt interessiert uns, wie oft jede mögliche Ziffer in den Originaldaten vorkommt. Schreiben Sie daher eine Klasse **`Stats`**, die von **`EnhancedRobot`** erbt und folgendes enthält:

- ein Attribut **`count`** vom Typ **`int[]`**
  In diesem werden im weiteren Verlauf die Häufigkeiten der einzelnen Ziffern gezählt.
- einen Konstruktor **`Stats(int st, int av, int[] dat)`**
  Dieser soll den Roboter an der Position (**`st`**, **`av`**) mit 999999 Beepern und Blickrichtung **`North`** erzeugen. Außerdem soll das Array **`count`** in einer geeigneten Größe angelegt werden.

  Anschließend sollen im Konstruktor die Ziffernhäufigkeiten des Arrays **`dat`** gezählt werden. Die einzelnen Elemente von **`count`** wurden beim Erzeugen des Arrays automatisch mit dem Wert „0“ initialisiert, sie können also für jedes Vorkommen der entsprechenden Ziffer einfach um eins erhöht werden.

  Ebenfalls noch im Konstruktor soll die Häufigkeitsverteilung grafisch dargestellt werden. Dazu soll der Roboter in der Straße **`st`** ab der Avenue **`av`** die Ziffern 1 bis 9 mit Beepern legen und darüber als Säulen aus Beepern die jeweiligen Häufigkeiten (siehe Abbildung).

Im **`Task`** erzeugen Sie dann einen **`Stats`**-Roboter bei (**7**, **1**). Nach Abschluss des gesamten Projekts soll die Welt folgendermaßen aussehen:



Testen Sie Ihr Programm auch mit anderen Eingabe-Arrays – wir tun das auch! Denken Sie auch daran, Ihr Programm vernünftig zu formatieren und zu kommentieren.

Sollten Sie eine Aufgabe nicht bearbeiten können: versuchen Sie trotzdem, die folgenden Aufgaben sinnvoll zu bearbeiten – es gibt auch Teilpunkte.

Viel Spaß und Erfolg bei der Bearbeitung des Projekts!