Technische Universität Darmstadt
Fachbereich Informatik
Prof. Dr. Johannes Fürnkranz

Allgemeine Informatik 1 im WS 2006/07

Programmierprojekt

Bearbeitungszeit: 13.11.2006 bis 13.01.2007

Organisatorisches:

- Das Programmierprojekt ist **in der Gruppe** zu bearbeiten, in der Sie sich zu Beginn des Semesters angemeldet haben (also ein oder zwei Gruppenmitglieder).
- Dieses Programmierprojekt ist mit **20 Punkten** angesetzt. Dies entspricht 20% ihrer Endnote im Fach Allgemeine Informatik 1. Die Abschlussklausur wird die restlichen 80% ausmachen.
- Verwenden Sie als Vorlage die Datei **programmierprojekt.task**, die Sie auf unserer Webseite finden, und erweitern Sie diese. Fangen Sie keine neue Datei an!
- Tragen Sie als erstes in die Datei Name, Matrikelnummer und RBG-Login von **allen** 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.ke.informatik.tu-darmstadt.de/lehre/ws0607/ai1/kareljide.php zu benutzen!
- Abgabeschluss ist spätestens der **13. Januar 2007**.
- Bitte kommentieren Sie ihren Quellcode hinreichend.
- **Wichtig:**
  “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.”

>> **WIR BEHALTEN UNS VOR, DAS PROGRAMMIERPROJEKT MIT 0 PUNKTEN ZU BEWERTEN, WENN OBEN GENANNTE PUNKTE MISSACHTET WERDEN!**

Aufgabe 1: ExtendedRobot

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

a) Implementieren Sie die Methoden **`void turnAround()`** und **`void turnRight()`**: der Roboter dreht sich um 180 Grad bzw. dreht sich um 90 Grad nach rechts.

b) Bitte implementieren Sie die Methode **`void moveToWall()`**: Lassen Sie den Roboter solange geradeaus laufen, bis er auf eine Wand trifft.

c) Bitte implementieren Sie die Methode **`int numBeepers()`**: Diese Methode soll alle Beeper auf der aktuellen Kreuzung aufnehmen, zählen und die Anzahl zurückgeben. **Achtung**: die Beeper sollen **nicht!** wieder abgelegt werden.

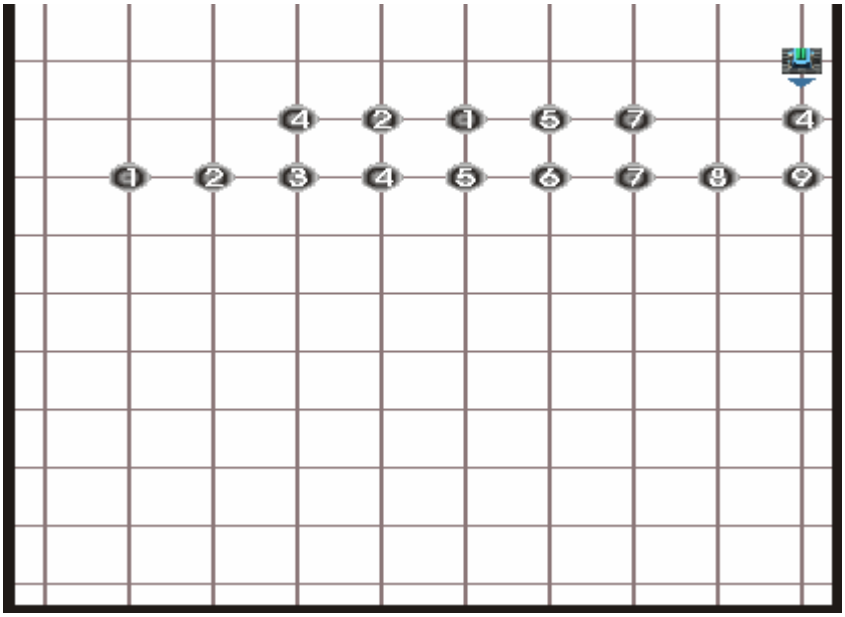
d) Testen Sie die Methoden ausgiebig. Dieser Test ist nicht Bestandteil der Abgabe, sichert jedoch die Funktionalität für die nächsten Aufgaben im Vorfeld.

Aufgabe 2: Welt vorbereiten

Dieser Teil der Übung dient dazu, Ihnen zu erklären, wie Sie unser Testszenario in KarelJ programmieren können.

Anfangsszenario:

- Es gibt eine senkrechte Wand, die sich über die ersten zehn Straßen erstreckt. Im unserem Szenario ist diese rechts von der 10. Avenue.
- Die erste Kreuzung direkt links neben der Wand stellt die Einerstelle (10^0) dar. Die zweite links von der Wand dann die Zehnerstelle (10^1), die dritte die Hunderterstelle (10^2) usw.
- Die Beeper auf der achten und neunten Straße stellen jeweils eine Dezimalzahl dar. Dabei bedeutet eine unbelegte Kreuzung die Ziffer 0. ***Sie können davon ausgehen, dass auf (8, 1) und (9, 1) keine Beeper liegen!***
- Zum Verständnis: die anderen Straßen (eins bis sieben) werden in den folgenden Aufgaben befüllt und bearbeitet.



Programmieren Sie die Beispielwelt (Wände und Beepervorkommen) wie oben in Bild und Text beschrieben. Benutzen Sie dazu die Befehle **`World.placeNSWall(x, x, x)`** und **`World.placeBeepers(x, x, x)`**. Die Programmzeilen sollen nicht in eine extra Klasse ausgelagert werden, sondern an den Anfang des **`task`**-Bereiches gestellt werden.

Die genaue Funktionsweise der beiden Methoden entnehmen Sie bitte der KarelJ API: http://www.st.informatik.tu-darmstadt.de/pages/lectures/inf1/kareljide/api/index.html

Wir werden Ihre Abgabe natürlich mit anderen Zahlen testen, also sollte Ihr Programm allgemein und nicht nur für diese beispielhafte Vorgabe funktionieren!

Aufgabe 3: Adder

In dieser Aufgabe sollen Sie eine eigene Roboterklasse **`Adder`** schreiben, die von der Klasse **`ExtendedRobot`** erbt. Aufgabe dieses Roboters wird sein zwei Dezimalzahlen zu addieren.

Ausgangspunkt und kurzer Überblick über die Aufgabe:

- Auf (10, 10) soll eine Instanz **`markus`** der Roboterklasse **`Adder`** erzeugt werden. Der Roboter soll nach Süden schauen.
- Seine Aufgabe ist Spaltenweise die beiden Dezimalzahlen in der achten und neunten Straße zu addieren
- und das Ergebnis in die sechste Straße schreiben.
- Der Code für diese Aufgabe im **`task`**-Bereich sollte maximal **4 Zeilen** umfassen. Lagern Sie alle nicht unbedingt notwendigen Befehle in die zu schreibenden Methoden aus. Für Testläufe können Sie so mit wenigen Auskommentierungen den ganzen Aufgabenblock herausnehmen.

Aber nun Schritt für Schritt:

a) Bitte überschreiben Sie die Methode **`int numBeepers()`** aus der Klasse **`ExtendedRobot`** wie folgt: Die Methode soll die ursprüngliche Methode aufrufen und zusätzlich dafür sorgen, dass alle Beeper wieder abgelegt werden nachdem Sie gezählt wurden. Sie gibt weiterhin die Anzahl der gezählten Beeper zurück.

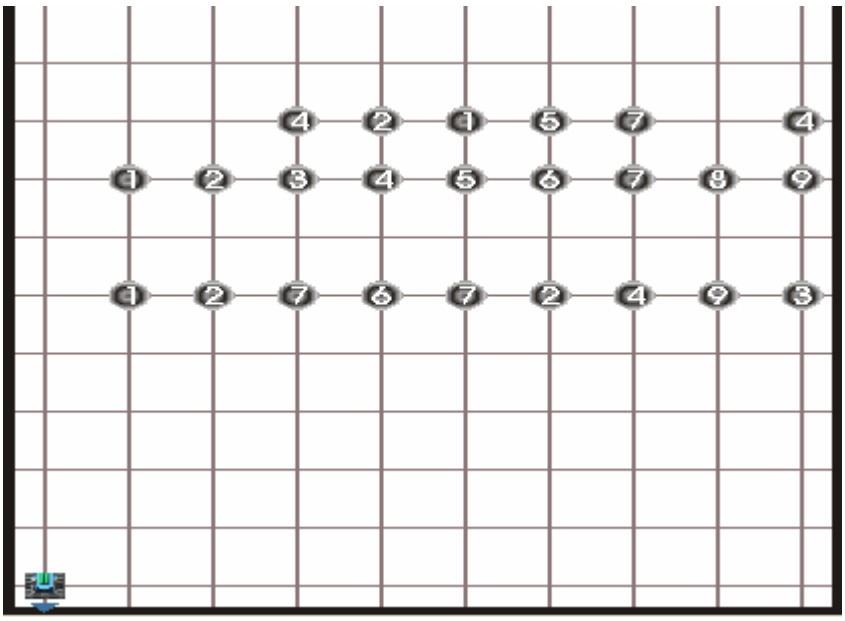
   Achtung: Es ist nicht gewünscht, dass Sie die Zeilen der ursprünglichen Methode kopieren und einfügen, sondern per Befehl die Methode aus der Mutterklasse (**`super.numBeepers()`**) aufrufen.

b) Bitte implementieren Sie die Methode **`void addDigitAndCarry()`**: Diese soll eine Dezimalstelle der beiden Zahlen und einen eventuellen Übertrag (steht auf der sechsten Straße, falls vorhanden) addieren und die Einerstelle der Summe auf die gleiche Position schreiben (im Fall zehnte Avenue auf das Feld (6, 10)). Falls ein Übertrag entsteht, wird dieser eine Avenue weiter westlich, unter die nächste zu addierenden Ziffern, geschrieben. Denken Sie daran den Übertrag mit in die Rechnung einzubeziehen

c) Bitte implementieren Sie die Methode **`add()`**. Diese soll die Methode **`addDigitAndCarry()`** dazu benutzen, die kompletten Zahlen auf der achten und neunten Straße zu addieren. Ob Sie die Positionierung für die nächste Ziffer in dieser Methode oder in **`addDigitAndCarry()`** vornehmen, bleibt Ihnen überlassen.

d) **`markus`** wird am Ende zum Ursprung bewegt und ausgeschaltet.

Am Ende dieser Aufgabe sollte ihre Welt wie folgt aussehen:



Aufgabe 4: ModifiedAdder

In dieser Aufgabe rücken wir von der Dezimalsicht ab und betrachten jede einzelne Kreuzung mit Beepern als gesonderte Zahl.

Hinweis:

- Der Code für diese Aufgabe im **`task`**-Bereich sollte maximal **5 Zeilen** umfassen. Lagern Sie alle nicht unbedingt notwendigen Befehle in die zu schreibenden Methoden aus. Für Testläufe können Sie so mit wenigen Auskommentierungen den ganzen Aufgabenblock herausnehmen.

a) Implementieren Sie eine Klasse **`ModifiedAdder`**, die von der Klasse **`Adder`** erbt.

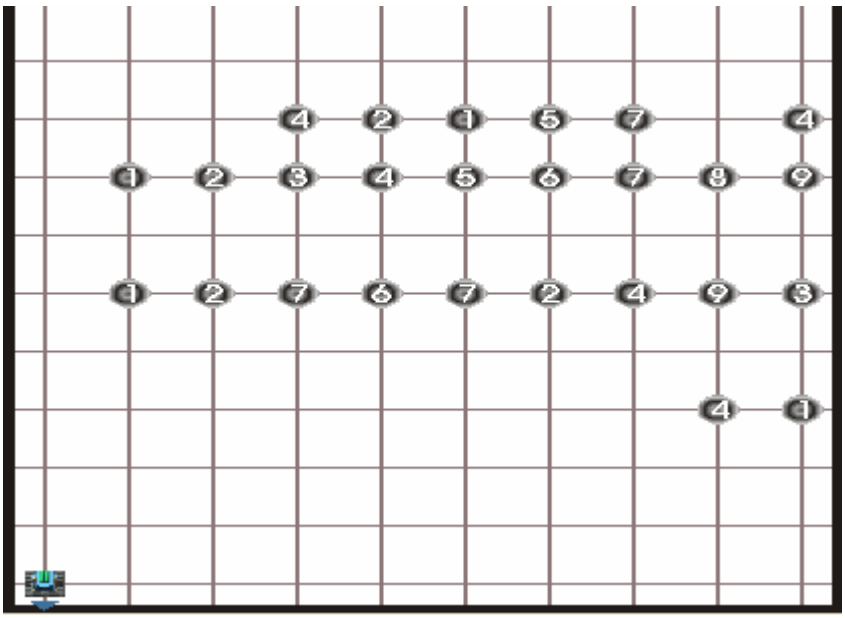
b) Bitte überschreiben Sie die Methode **`int numBeepers()`** aus der Klasse **`Adder`** wie folgt: Die Methode soll sich der ursprüngliche Methode bedienen, diese aber insofern abändern, dass die Methode nicht mehr die Anzahl der Beeper einer Kreuzung zurückgibt, sondern die Summe aller Beeper auf einer ganzen Straße (bis der Roboter auf eine Wand trifft).

c) Schreiben Sie eine Methode **`void showAsDec(int figure)`**: Die Methode bekommt eine Zahl übergeben und lässt einen nach Westen gerichteten Roboter Sie mit Hilfe von Beepern als Dezimalvariante abbilden (Das östlichste Feld entspricht der Einerstelle (10^0), das nächste Feld in Richtung Westen der Zehnerstelle (10^1) etc.).

Achtung: Sie brauchen Sich hier nur um die reine Funktionalität zu kümmern, und nicht darum, ob Fehler auftreten (nicht genügend Beeper vorhanden, Weg nicht frei etc.)

d) Erzeugen Sie einen Roboter **`katrin`** auf der Kreuzung (6, 1) mit Blickrichtung nach Osten. Lassen Sie ihn die Beeper der gesamten Straße zählen (siehe Aufgabenteil b) und geben Sie die Summe in der vierten Straße rechtsbündig als Dezimalzahl (siehe Aufgabenteil c) aus.

e) Lassen Sie den Roboter zum Ursprung laufen und schalten Sie ihn aus.



Aufgabe 5: Sorter

Dieser Aufgabe beschäftig sich mit dem Sortieren von Beeperstapeln. Aufgrund der Laufzeit betrachten wir keine ganze Straße, sondern nur 6 Kreuzungen. Bevor wir aber mit dem Sortieren anfangen können, müssen die Roboter erst einmal unsere 6 Kreuzungen vorbereiten.

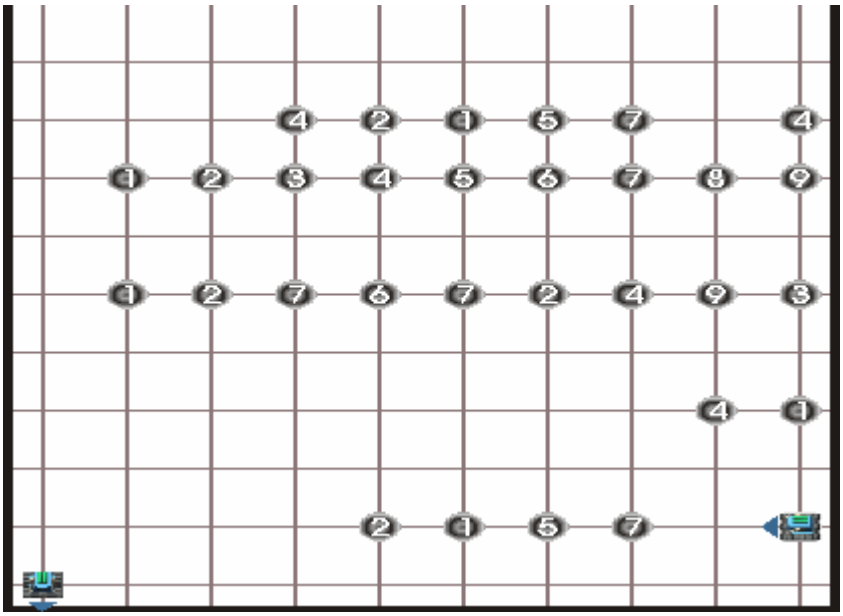
Hinweis:

- Der Code für diese Aufgabe im **`task`**-Bereich sollte maximal **7 Zeilen** umfassen. Lagern Sie alle nicht unbedingt notwendigen Befehle in die zu schreibenden Methoden aus. Für Testläufe können Sie so mit wenigen Auskommentierungen den ganzen Aufgabenblock herausnehmen.

a) Bitte implementieren Sie als erstes für diese Aufgabe eine Klasse **`Sorter`**, die von **`Adder`** erbt. Erweitern Sie die Klasse **`Sorter`** bitte nun um die Methode **`void putBeeper(int anzahl)`**, die nicht einen, sondern **`anzahl`** Beeper legt. Dabei soll natürlich die Methode **`void putBeeper()`** aus der Klasse **`Adder`** verwendet werden – überlegen Sie sich bitte wie Sie die Ursprungsmethode referenzieren müssen:
   1. **`super.putBeeper()`**;
   2. **`putBeeper()`**;

b) Für den nächsten Schritt benötigen wir zwei Roboter: **`peter`** und einen **`helfer`**. Der Roboter **`helfer`** ist nur dazu da, die Beeper auf einer Kreuzung zu zählen und **`peter`** zuzurufen, wie viele dieser ablegen muss, um auf die gleiche Anzahl an Beepern auf der Kreuzung zu kommen („kopieren“).
   1. Erstellen Sie bitte die beiden Roboter vom Typ **`Sorter`** an den Positionen (2, 5) (**`peter`**) und (9, 5) (**`helfer`**) mit der Blickrichtung nach Osten. Erzeugen Sie Peter bitte mit ausreichend vielen Beepern in seiner Tasche.
   2. Der Methodenaufruf **`peter.copy(helfer)`**, der in ihrem Hauptprogramm stehen soll, beschreibt Ihnen nun wie die Methode **`copy`** deklariert sein muss. Nun zur Funktionsweise: Solange vor den zwei Robotern keine Wand ist, soll **`helfer`** die Beeperanzahl auf seiner Kreuzung ermitteln, sie **`peter`** zurufen (vom Funktionsaufruf zurückgeben), und dieser dann genau diese Zahl an Beepern ablegen. Mittels dieses Vorgangs sollen genau 6 Kreuzungen von Straße 9 in Straße 2 kopiert werden.

3. Lassen Sie den **`helfer`** zum Ursprung wandern und schalten Sie ihn aus. **`peter`** soll sich einfach nur umdrehen und auf weitere Befehle warten.



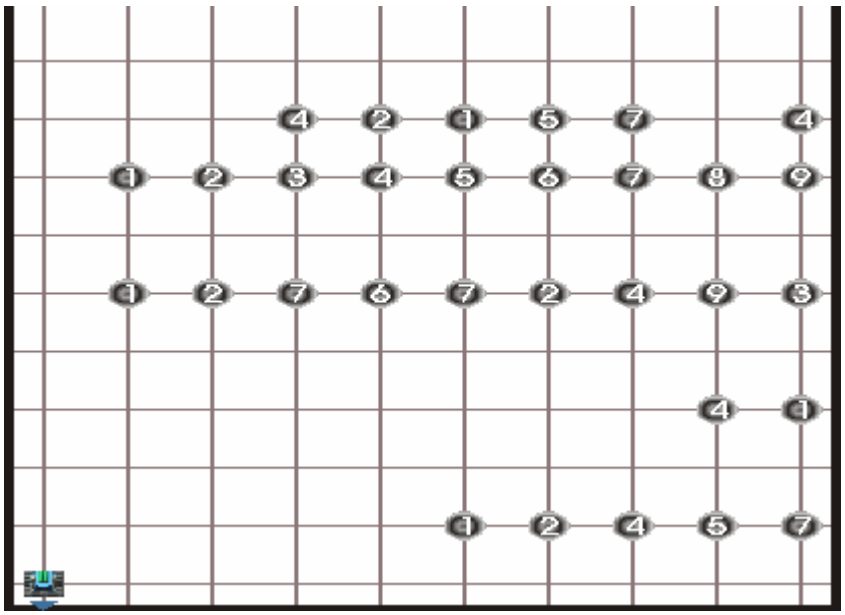
c) Bevor wir nun mit dem Sortieren beginnen können, benötigen wir noch eine unterstützende Methode. Bitte implementieren Sie die Funktion **`void tauscheBeeper(int anzahlBeeper)`**. Diese Funktion erwartet einen Parameter, der genau die Differenz an Beepern zwischen zwei Kreuzungen erwartet. **`peter`** soll bei Aufruf der Methode die Differenz an Beepern auf der aktuellen Kreuzung aufnehmen, danach einen Schritt zurückgehen, die Differenz ablegen und wieder einen Schritt nach vorne machen. (Dies entspricht dem Tauschen der beiden Beeperstapel, nur in einer effizienteren Variante!).

d) Bitte implementieren Sie den folgenden Sortieralgorithmus in die Methode **`void sort()`**. Hinweis: der Roboter Peter steht am Ende der Durchführung von Teilaufgabe b) auf der Kreuzung (2, 10) und blickt nach Westen (Ausgangspunkt der Sortierens).

Sortieralgorithmus in Pseudocode:

```
> sortiert = false
> Solange sortiert == false, tue: {
  > Sortiert = true
  > Bitte (zu sortierende Kreuzungen – 1)-mal ausführen: {
    > Falls der Wert der aktuellen Kreuzung kleiner ist als
      der darauf folgenden, tue: {
      > Sortiert = false
      > Rufe die Methode tausche_Beeper (siehe c)) auf
    }
  }
  > Der Roboter muss wieder auf der Kreuzung (2, 10) mit
    Blickrichtung Westen stehen, bevor der nächste
    Schleifendurchlauf stattfindet
}
> Bewege den Roboter zum Ursprung und schalte ihn aus
```

Am Ende der Aufgabe erhalten Sie mit dem Beispielszenario folgendes Bild:



Viel Erfolg bei der Bearbeitung!