Technische Universität Darmstadt
Fachbereich Informatik
Prof. Dr. Andreas Koch

Allgemeine Informatik 1 im WS 2007/08

Übungsblatt 8

Bearbeitungszeit: 17.12. bis 23.12.2007

Aufgabe 1: Wiederholung

a) **Methode ohne Rückgabewert:** Schreiben Sie eine Roboterklasse **`U8A1`** mit einer Methode **`void lockStep(Robot partner)`**. Beim Aufruf dieser Methode auf einem Roboter sollen sowohl dieser Roboter als auch **`partner`** einen Schritt gehen.
Der Aufruf im Task lautet also z.B. **`karel.lockStep(marvin);`**

b) **Methode mit Rückgabewert:** Ergänzen Sie die in a) geschriebene Klasse **`U8A1`** um eine Methode **`int zaphod(int a, int b)`**. Wenn die Summe von a und b kleiner als 42 ist, soll diese Summe zurückgegeben werden, sonst der Wert 42. Wenn Sie das Ergebnis der Methode **`zaphod`** in Ihrem Programm an irgendeiner Stelle benutzen wollen, reicht es **nicht**, z.B. **`karel.zaphod(23, 1337);`** zu schreiben – Sie müssen das Resultat der Methode entweder einer Variablen zuweisen oder direkt verwenden, z.B. im Befehl **`System.out.println()`**!
Testen Sie die Methoden **`lockStep`** und **`zaphod`**!

c) **return:** Betrachten Sie folgende Methoden:

```
int a(int zahl) {
  if (zahl == 1) return 2;
  return 3;
}

int b(int zahl) {
  int quadrat = zahl * zahl;
  return quadrat;
  System.out.println("Quadrat: " + quadrat);
}
```

Was gibt **`a(1)`** zurück? Und was **`a(42)`**?
Bei Aufruf der Methode **`b`** liefert die KarelJIDE die Fehlermeldung „Unreachable code“ – warum ist die Methode fehlerhaft?

Aufgabe 2: PoweredRobot

Roboterklassen können nicht nur Methoden enthalten, sondern auch Variablen, so genannte Attribute. Jeder Roboter der Klasse **`PoweredRobot`** besitzt zu Beginn 20 Energiepunkte, die in der Variablen **`int energy`** gespeichert sind:

```
class PoweredRobot extends Robot {
  int energy = 20;
  // ...
}
```

Jede Tätigkeit des Roboters kostet Energiepunkte:

- Ein Schritt nach vorne kostet 2 Energiepunkte.
- Eine Drehung nach links kostet 1 Energiepunkt.

Durch das Aufnehmen von Beepern kann der Roboter neue Energiepunkte tanken: jeder aufgenommene Beeper bringt 5 neue Energiepunkte. Hierbei ist natürlich zu beachten, dass der Roboter 5 Energiepunkte verliert, wenn er einen Beeper ablegt.
Reicht für irgendeine Tätigkeit die Anzahl der Energiepunkte nicht aus, schaltet sich der Roboter ab. Am Ende jeder Methode zeigt der Roboter seine aktuelle Energie an.

```
class PoweredRobot extends Robot {
  int energy = 20;

  void move() {
    if (energy >= 2) {
      energy -= 2; // entspricht energy = energy - 2;
      super.move();
    } else turnOff();
    System.out.println("Energie: " + energy); // Energie anzeigen
  }

  void pickBeeper() {
    super.pickBeeper();
    energy += 5; // entspricht energy = energy + 5;
    System.out.println("Energie: " + energy); // Energie anzeigen
  }

  void turnLeft() {
    // ...
  }

  void putBeeper() {
    // ...
  }
}
```

Vervollständigen Sie die Implementierung der Klasse **`PoweredRobot`** (das obige Gerüst ist auf der Übungswebseite als **uebung08-2.task** vorgegeben) und testen Sie das Verhalten von Robotern dieser Roboterklasse. Definieren Sie hierzu mit der Anweisung **`World.placeBeepers(street, avenue, beepers)`** Welten und lassen Sie dann Roboter Ihrer Klasse **`PoweredRobot`** in diesen Welten arbeiten.

a) Definieren Sie in derselben Datei eine von **`PoweredRobot`** abgeleitete Klasse **`GreedyRobot`**. Roboter dieser Klasse sollen, bevor sie eine Kreuzung verlassen, alle Beeper der Kreuzung aufnehmen und so automatisch Energiepunkte sammeln. Sie müssen dazu nur die Methode **`move()`** neu definieren.

b) Schreiben Sie (ebenfalls in derselben Datei) eine von **`Robot`** abgeleitete Klasse **`Recharger`**, die eine Methode **`void recharge(PoweredRobot pr, int beepers)`** besitzt. Diese Methode soll die Energie des Roboters **`pr`** auffüllen, indem sie **`beepers`** Beeper legt und **`pr`** dann anweist, diese aufzuheben. Achtung: das soll nur funktionieren, wenn beide Roboter (der **`Recharger`** und **`pr`**) auf derselben Kreuzung stehen (Statusabfragen **`street()`** und **`avenue()`** benutzen!). Wenn die Roboter sich nicht an der gleichen Position befinden, soll nichts passieren. Sie können davon ausgehen, dass der **`Recharger`** genügend Beeper dabei hat.

Aufgabe 3: Kreisbogen

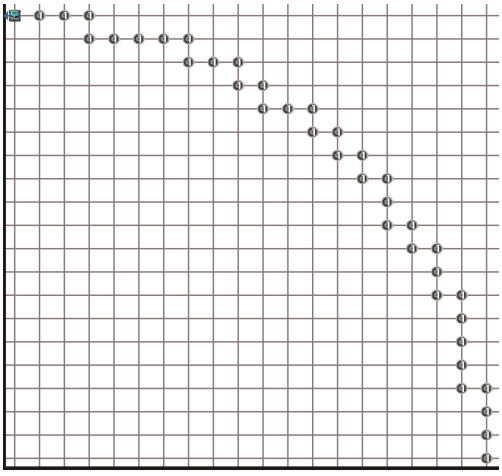
Karel ist es leid, immer nur geradeaus zu laufen. Er würde gerne, so weit das in einer Rechteckwelt möglich ist, kreisförmig laufen. Und Sie werden ihm dabei helfen…

Vervollständigen Sie die Implementierung der Klasse **`Circler`** in der Datei **uebung08-3.task** von der Webseite. Ein Roboter dieser Klasse hat ein Attribut **`int radius`**, das den Radius eines Kreisbogens darstellt. Auch eine Methode **`faceTo(direction dir)`** ist schon vorgegeben, mit deren Hilfe der Roboter in eine bestimmte Himmelsrichtung gedreht werden kann.

a) Schreiben Sie einen Konstruktor (KarelJ-Skript S. 98-102), der nur einen Parameter **`int r`** für den Radius erwartet. Zuerst muss dann mittels **`super(...);`** der Konstruktor der Überklasse **`Robot`** aufgerufen werden, damit auch tatsächlich ein Roboter erzeugt wird – und zwar auf der Kreuzung (1, r) mit Blickrichtung Norden und 999999 Beepern. Danach wird dem Attribut der Klasse der übergebene Radius zugewiesen.

b) Jetzt kommt der schwierige Teil: die Methode **`void arc()`**. Diese soll den **`Circler`** einen Viertelkreis nach links oben gehen und mit Beepern markieren lassen. Damit der Roboter sich annähernd auf einer Kreisbahn bewegt, müssen Sie vor jedem Schritt entscheiden, ob er nach Norden oder Westen geht. Berechnen Sie dazu mittels des Satz des Pythagoras die Entfernung zum imaginären Mittelpunkt (0, 0) von der Kreuzung im Norden des Roboters und der Kreuzung im Westen des Roboters (Quadratwurzel: **`Math.sqrt(...)`**). Dann berechnen Sie jeweils die Abweichung dieser Entfernungen vom gewünschten Radius (Betragsfunktion: **`Math.abs(...)`**). Speichern Sie diese vier Werte jeweils in einer Variable vom Typ **`double`**. Anschließend legt der Roboter einen Beeper, dreht sich in die Richtung, in die der Fehler kleiner war und geht einen Schritt.
Diese Schritte werden wiederholt, so lange (**`while`**!) der Roboter nicht auf der ersten Avenue angekommen ist. Abschließend wird noch ein Beeper abgelegt, um den Kreisbogen zu vollenden.

c) Testen Sie Ihre Klasse, indem Sie einen **`Circler`** namens **`karel`** erzeugen und ihn einen Kreisbogen gehen lassen. Bei einem Radius von 20 sieht das etwa so aus wie auf dem Bild rechts.



Wir wünschen Ihnen viel Spaß und ein schönes Weihnachtsfest!