Objektorientierte Programmierung

mit dem

Java Hamster-Modell

Autor: Dieter Lindenberg

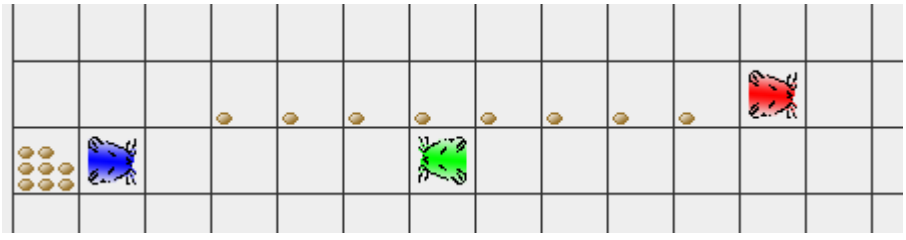
Version 2008

Inhaltsverzeichnis

Hamstererzeugung

Im Folgenden Beispielprogramm werden drei Hamster erzeugt. Mit der Hamster-Programmierumgebung lässt sich nur ein Hamster, der sog. Standardhamster festlegen. Weitere Hamster müssen während der Laufzeit erzeugt werden. Dies kann an jeder Stelle des Programms erfolgen. Es ist übrigens nicht möglich, erzeugte Hamster wieder zu löschen.

```
void main() {
  Hamster Willi = Hamster.getStandardHamster();
  Hamster  Paul = new Hamster(2,3,Hamster.OST,8);
// alternativ:        Hamster Paul = new Hamster();
//                    Paul.init(2,3,Hamster.OST, 8);
// die ersten beiden Zahlen geben die Zeile und
// Spalte an, wobei die Zelle oben links die
// Koordinaten (0;0) hat. Die letzte Zahl gibt an,
// wie viel Körner der Hamster im Maul haben soll.
// Es gibt vier sog. Konstanten:  Hamster.NORD = 0,
// Hamster.OST = 1, Hamster.SUED = 2, Hamster.WEST =3
  int zahl = 5;
  Hamster Linda = new Hamster(zahl,zahl + 1,0,8);
  if (Willi.kornDa()) Willi.nimm();
  Willi.vor();
  while(!Paul.maulLeer()) {
     Paul.gib();
     Paul.vor();
  }
  for (int i=1; i<4;i++) Linda.vor();
  Linda.linksUm();
}
```



Aufgaben

1. Ein neuer Hamster wird mit vier Körnern im Maul irgendwo in einem leeren Territorium erzeugt. Der Hamster soll in allen vier Ecken des Territoriums ein Korn ablegen. Wenn er von Ecke zu Ecke eilt, soll er
   a) links b) rechts herum laufen.
2. Drei Hamster stehen nebeneinander am linken Rand des Territoriums mit Blick nach Osten. Sie sollen zum rechten Rand des Territoriums laufen.
   a) nacheinander, b) gleichzeitig.
3. Im Territorium befinden sich vier Hamster an verschiedenen Stellen mit verschiedenen Blickrichtungen. Sie laufen alle immer von einer Wand zur anderen hin und her.
4. Die beiden Hamster Max und Moritz laufen gleichzeitig immer am Rand eines leeren Territoriums entlang. Sie starten beide oben links. Max läuft links herum, Moritz rechts herum. Löse diese Aufgabe in verschiedenen Schwierigkeitsstufen:
   a) das Territorium ist quadratisch
   b) das Territorium ist nicht quadratisch
   c) Moritz ist doppelt so schnell wie Max
5. Vier Hamster stehen nebeneinander unten links auf der Grundlinie und machen eine Polonaise um das gesamte Territorium herum. Hinweis: sobald der erste Hamster eine Ecke erreicht, sollte eine Prozedur aufgerufen werden, welche alle vier Hamster um die Ecke bringt.
6. Zwei Hamster stehen sich gegenüber. Beide Hamster haben eine unterschiedliche Anzahl von Körnern im Maul. Die beiden tauschen ihre Körner gegenseitig aus. Um kontrollieren zu können, ob das auch geklappt hat, drehen sich beide nach Norden und legen ihre jeweils neue Körneranzahl vor sich hin.
7. DerStandardhamster Willi läuft vom oberen linken zum oberen rechten Rand des ansonsten mauerlosen Territoriums, wobei er bei jedem Schritt ein Korn verliert. Rechts angekommen, ist er zu erschöpft, um die verlorenen Körner wieder einzusammeln. Deshalb erzeugt er einen Hilfshamster namens Paul, der zurückläuft, alle Körner aufsammelt und schließlich diese alle Willi übergibt.

Lösungen

Aufgabe 1b)

```
Hamster Willi = new Hamster();
// Der neue Hamster namens Willi wird hier global deklariert, weil er sonst in
// der Methode wRechtsUm() unbekannt wäre. Man könnte ihn auch hier schon
// mit Koordinaten, Blickrichtung und Körneranzahl initialisieren, leider aber
// nicht mit der Initanweisung, sondern nur so:
// Hamster Willi = new Hamster(2,5,0,4);

void main() {
 Willi.init(2,5,0,4);  // Beachte obigen Kommentar!
 while (Willi.vornFrei()) Willi.vor();
 wRechtsUm();
 while (Willi.vornFrei()) Willi.vor();
 for (int i=1;i<=4; i++)  {
   Willi.gib();
   wRechtsUm();
   while (Willi.vornFrei()) Willi.vor();
 }
}

void wRechtsUm()  {
// leider muss für jeden neuen Hamster auch die Rechtsdrehung neu definiert
// werden. Das ist lästig. Später werden wir dieses Problem allgemeiner lösen
// können!
   Willi.linksUm();    // Willi wurde oben global deklariert !
   Willi.linksUm();
   Willi.linksUm();
}
```

Aufgabe 4b)

```
Hamster Max = Hamster.getStandardHamster();
Hamster Moritz = new Hamster(0,0,2,0);

void main() {
 while (true)  {
   MaxMachtEinenSchritt();
   MoritzMachtEinenSchritt();
 }
}

void MaxMachtEinenSchritt()  {
  if (Max.vornFrei()) Max.vor();
  else {
    Max.linksUm();
    Max.linksUm();
    Max.linksUm();
    Max.vor();
  }
}

void MoritzMachtEinenSchritt()  {
  if (Moritz.vornFrei()) Moritz.vor();
  else {
    Moritz.linksUm();
    Moritz.vor();
  }
}
```

Neue Hamsterbefehle

Sei Paul der Name eines existierenden Hamsters. Dann liefern die Befehle

Paul.getReihe() einen Integerwert, der die Reihe bzw. Zeile angibt, auf der sich Paul gerade befindet.

Paul.getSpalte() einen Integerwert, der die Spalte angibt, auf der sich Paul gerade befindet.

Paul.getAnzahlKoerner() einen Integerwert, der angibt, wie viele Körner sich aktuell im Maul des Hamsters Paul befinden.

Paul.getBlickrichtung() einen Integerwert, der die Blickrichtung repräsentiert, in die der Hamster Paul aktuell schaut (0 = Norden, 1 = Osten, 2 = Süden, 3 = Westen).

Beispiel: Der Hamster Paul wird auf einer beliebigen Kachel mit beliebiger Blickrichtung erzeugt. Er läuft in Blickrichtung zur nächsten Wand. Dort angekommen, erzeugt er einen neuen Hamster, der in umgekehrter Richtung bis zur nächsten Wand läuft.

```
Hamster Paul = new Hamster(5,6,3,0);

void main() {
   while (Paul.vornFrei()) Paul.vor();

   int richtung = (Paul.getBlickrichtung() + 2) % 4;
   Hamster Willi = new Hamster(Paul.getReihe(),
                        Paul.getSpalte(),richtung,0);
   while (Willi.vornFrei()) Willi.vor();
}
```

Aufgaben

1. Der Hamster Paul wird auf einer beliebigen Kachel mit beliebiger Blickrichtung erzeugt. Er merkt sich die Koordinaten seines Erzeugungsortes, läuft geradeaus bis zur nächsten Wand, und kehrt zu seinem Erzeugungsort um. Hinweis: diese Aufgabe lässt sich auf völlig unterschiedlichen Wegen lösen. Sie soll jedoch mithilfe der Methoden *getReihe()* und *getSpalte()* gelöst werden.

2. Die drei Hamster Willi, Paul und Linda stehen an drei verschiedenen Stellen am linken Rand des ansonsten mauerlosen Territoriums. Sie laufen zum rechten Rand und sammeln dabei eventuell herumliegende Körner ein. Wer die meisten Körner gesammelt hat, ist Sieger und dreht sich vor Freude 10 mal um sich selbst. Benutze die Methode *getAnzahlKoerner()* !

3. Der Hamster steht irgendwo unten auf der Grundlinie seines Territoriums mit Blickrichtung nach Osten. Er soll einmal am Rande seines mauerlosen Territoriums herum laufen bis er wieder am Ausgangsort angelangt ist.

4. Der Hamster steht (mit beliebiger Blickrichtung) im Territorium und hat etwas den Überblick verloren. Er soll sich solange links herum drehen, bis er nach Norden schaut.

5. Der Standardhamster Paul steht oben links mit Blick nach Osten und seine Hamsterfreundin Lisa steht oben rechts mit Blick nach Westen. Das Territorium ist beliebig groß. Sie gehen abwechselnd schrittweise aufeinander zu und bleiben anschließend auf derselben Kachel stehen. Danach wird ein neuer Hamster an eben dieser Stelle erzeugt, welcher sich sofort aufmacht und Richtung Süden bis zur Wand läuft.

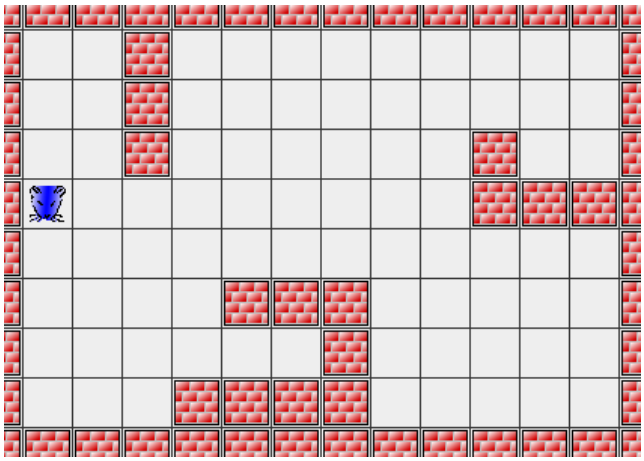
6. Der Hamster Paul steht oben rechts mit Blickrichtung nach Westen im mauerlosen Territorium, welches übrigens beliebig groß sein kann. Auf der Kachel mit den Koordinaten (3; 8) liegt ein Schatz in Form von mehreren Körnern. Paul geht zum Schatz und frisst alle Körner.

7. Der Hamster Paul steht irgendwo mit beliebiger Blickrichtung im mauerlosen Territorium und möchte seine Hamsterfreundin Lisa besuchen, die irgendwo im Territorium wohnt. Lisa bleibt in ihrer Wohnung und Paul macht sich auf den Weg.

8. Paul steht irgendwo im oberen linken Viertel des mauerlosen Territoriums mit Blick nach Osten. Lisa steht irgendwo im unteren rechten Viertel mit Blick nach Westen. Sie gehen abwechselnd schrittweise aufeinander zu und bleiben anschließend auf derselben Kachel stehen.

9. Paul und Lisa stehen irgendwo im mauerlosen Territorium mit beliebiger Blickrichtung. Sie gehen abwechselnd schrittweise aufeinander zu und bleiben anschließend auf derselben Kachel stehen.

10. Der Hamster Paul steht in einem geschlossenen, körnerlosen Raum unbekannter Größe. Rechts von ihm befindet sich eine Wand. Er soll nun solange an der Wand entlang laufen, bis er wieder seine Ausgangskachel erreicht hat.



Lösungen

Aufgabe 10

```
Hamster Paul = Hamster.getStandardHamster();

void main() {
  int xAlt = Paul.getReihe();
  int yAlt = Paul.getSpalte();
  do EinSchrittAnWandEntlang(); while(xAlt !=
        Paul.getReihe() || yAlt != Paul.getSpalte());
}
```

```
void EinSchrittAnWandEntlang()  {
  if (rechtsFrei())  {
    Paul.linksUm();
    Paul.linksUm();
    Paul.linksUm();
    Paul.vor();
  }
  else if (Paul.vornFrei()) Paul.vor();
       else {
         Paul.linksUm();
         if (Paul.vornFrei()) Paul.vor();
         else  {
           Paul.linksUm();
           Paul.vor();
         }
       }
}

boolean rechtsFrei()  {
  boolean ergebnis;
  Paul.linksUm();
  Paul.linksUm();
  Paul.linksUm();
  ergebnis = Paul.vornFrei();
  Paul.linksUm();
  return ergebnis;
}
```

Parameter vom Datentyp *Hamster*

```
Hamster Paul = Hamster.getStandardHamster();
Hamster Lisa = new Hamster(8,7,1,0);

void main() {
   tanze(Paul);
   tanze(Lisa);
}

void tanze(Hamster egal)  {
  for (int i=1; i<=8; i++) egal.linksUm();
}
```

Beim Aufruf der Methode *tanze* wird der Parameter ***egal*** mit dem entsprechenden Hamster (entweder *Paul* oder *Lisa*)initialisiert.

Aufgaben

1. Schreibe die Methode *RechtsUm(Hamster egal)* und teste sie mit mehreren Hamstern!

2. Vier Hamster befinden sich in einem Territorium an beliebigen Stellen mit beliebigen Blickrichtungen. Sie sollen immer abwechselnd einen Schritt vorwärts machen. Falls dies nicht möglich sein sollte, drehen sie sich solange a) links b) rechts herum, bis es möglich ist und machen dann den Schritt vorwärts. Schreibe dafür (u.a.) die Methode *vor(Hamster egal)*

3. Zwei Hamster stehen an unterschiedlichen Stellen am Rande des Territoriums, welches mehrere Mauern enthält. Beide Hamster sollen einmal das Territorium am Rand umrunden, bis sie wieder an ihrer Ausgangsstelle sind.

Erweiterte Hamsterklassen

Die im Folgenden erzeugten Klassen sind Unterklassen der Klasse *Hamster*. Die Klassennamen sind relativ beliebig. Objekte dieser Unterklassen kennen auch alle Befehle der Oberklasse *Hamster*.

```
class KreiselHamster extends Hamster  {
  void drehDich(int n)  {
    for (int i = 1; i <= n; i++)  linksUm();
  }

  void kehrt()  {
    drehDich(2);
  }

  void rechtsUm()  {
    drehDich(3);
  }
}


void main() {
  KreiselHamster willi = new KreiselHamster();
  willi.init(1,4,Hamster.OST,0);
// willi = new KreiselHamster(1,4,Hamster.OST,0)
// ist nicht möglich, weil der entsprechende
// Konstruktor (noch) nicht existiert.
  KreiselHamster heidi = new KreiselHamster();
  heidi.init(5,2,Hamster.OST,0);

  willi.kehrt();
  heidi.rechtsUm();
}
```

Im letzten Beispiel wurden die Klasse *KreiselHamster* und ein diese Klasse verwendendes Programm als eine einzige Datei geschrieben. Beim Kompilieren erzeugt die Hamster-Programmierumgebung allerdings dafür, dass eine eigene Klassendatei angelegt wird.

Es gibt jedoch auch die Möglichkeit, eine neue Klasse zu definieren, die für alle Programme zur Verfügung steht, welche im selben Ordner gespeichert sind:

Erstelle zunächst einen neuen Ordner namens *NeuerHamsterOrdner*, in welchem du sowohl die Klasse als auch die Anwendungsprogramme speichern wirst.
Wähle nun im Hamstereditor das Icon *Neu* und wähle den Typ *Klasse*. Schreibe die nachstehende Klassendefinition, kompiliere sie und speichere sie unter dem Namen *NeuHamster*.

```
class NeuHamster extends Hamster  {
  void drehDich(int n)  {
    for (int i = 1; i <= n; i++)  linksUm();
  }

  void kehrt()  {
    drehDich(2);
  }

  void rechtsUm()  {
    drehDich(3);
  }
}
```

Anschließend schreibst du die folgenden beiden Hauptprogramme und speicherst sie unter den Namen *Anwendung1* und *Anwendung* in dem gerade vorher angelegten Ordner namens *NeuerHamsterOrdner*.
Begründung: Wenn der Compiler auf einen neuen Klassennamen stößt, sucht er die Definition dieser Klasse in demselben Ordner, in dem auch das Anwendungsprogramm steht.

```
void main() {
   NeuHamster willi = new NeuHamster();
   willi.init(4,5,Hamster.OST,0);
   willi.kehrt();
}

void main() {
   NeuHamster heidi = new NeuHamster();
   heidi.init(1,2,Hamster.OST,0);
   heidi.drehDich(10);
}
```

Aufgaben

1. Definiere zwei neue Hamsterklassen namens *LangsamerHamster* und *SchnellerHamster*, welche beide den Befehl *lauf()* kennen. Allerdings wirkt dieser Befehl unterschiedlich in den verschiedenen Klassen. Er bringt langsame Hamster nur einen Schritt weiter und schnelle Hamster zwei Schritte weiter. Falls ein Schritt vorwärts nicht möglich ist, wird zunächst versucht, nach links auszuweichen. Falls dies auch nicht möglich sein sollte, versucht der Hamster, nach rechts zu gehen. Sollte dies wieder misslingen, geht der Hamster einen Schritt zurück.
Erzeuge nun einen langsamen Hamster namens *Schnecke* und einen schnellen Hamster namens *Speedy*. Die beiden Hamster liefern sich ein Rennen außen am Territorium entlang. *Speedy* startet oben links mit Blick nach Süden, *Schnecke* unten rechts mit Blick nach Norden. Wenn *Speedy* *Schnecke* eingeholt hat, bleibt *Schnecke* stehen und *Speedy* läuft den absolvierten Weg zurück.

2. Hamster der Klassen *LangsamerHamster* oder *SchnellerHamster* sind ziemlich dumm. Man kann ihnen in einem riesigen, fast völlig freien Territorium eine Falle bauen, so dass sie sich bei dem Befehl
`for (int i = 1; i < 200; i++) lauf();`
praktisch immer nur auf drei Kacheln hin und her bewegen. Baue ein solches Territorium und schreibe das entsprechende Programm!

Lösungen

Aufgabe 1

```
class LangsamerHamster extends Hamster  {
  void lauf()  {
    if (vornFrei()) vor();
    else  {
      linksUm();
      if (vornFrei())  vor();
      else   {
        linksUm();
        linksUm();
        if (vornFrei())  vor();
        else   {
          linksUm();
          linksUm();
          linksUm();
          vor();
        }
      }
    }
  }
}


class SchnellerHamster extends Hamster  {
  void lauf()  {
    for (int i=1; i <=2; i++)  {
       if (vornFrei()) vor();
       else  {
         linksUm();
         if (vornFrei())  vor();
         else   {
           linksUm();
           linksUm();
           if (vornFrei())  vor();
           else   {
             linksUm();
             linksUm();
```

```
            linksUm();
            vor();
          }
        }
      }
    } // Ende der For-Schleife
  }
}


LangsamerHamster Schnecke = new LangsamerHamster();
SchnellerHamster Speedy = new SchnellerHamster();

void main() {
  Schnecke.init (5,10,Hamster.NORD,0);
  Speedy.init(0,0,2,0);
  int SpeedyDoppelschritte = 0;
  while ((Schnecke.getReihe() != Speedy.getReihe()) ||
          (Schnecke.getSpalte() != Speedy.getSpalte()))   {
     Schnecke.lauf();
     Speedy.lauf();
     SpeedyDoppelschritte++;
  } // Ende der While-Schleife
  Speedy.linksUm();
  Speedy.linksUm();
  for (int i=1; i <= SpeedyDoppelschritte; i++)
     Speedy.lauf();
}
```

Klassenmethoden

Sogenannte Klassenmethoden sind spezielle Methoden, die unabhängig von der Existenz von Objekten einer Klasse aufgerufen werden können. Der Zugriff auf diese Methoden erfolgt über den Klassennamen anstatt über einen Objektnamen.

Die Klasse ***Hamster*** stellt u.a. die beiden folgenden Klassenmethoden zur Verfügung:

```
int     getAnzahlHamster()
Hamster     getStandardHamster()
```

Die letzte Methode wurde ja schon des Öfteren angewandt durch die Anweisung

```
Hamster Paul = Hamster.getStandardHamster()
```

Die Klasse ***Territorium*** stellt folgende Klassenmethoden zur Verfügung:

```
int     getAnzahlReihen()
int     getAnzahlSpalten()
boolean     mauerDa(int reihe, int spalte)
int     getAnzahlKoerner()          // Gesamtzahl im Territorium
int     getAnzahlKoerner(int reihe, int spalte)
int     getAnzahlHamster()          // Gesamtzahl im Territorium
int     getAnzahlHamster(int reihe, int spalte)
```

Aufgaben

1. Der Standardhamster soll bis zur nächsten Mauer gehen. Er darf jedoch nicht den Testbefehl *vornFrei()* benutzen. Stattdessen soll er von der Methode *mauerDa* der Klasse *Territorium* Gebrauch machen.

2. In der obersten Reihe des Territoriums befinden sich außer dem Standardhamster Paul noch mehrere weitere Hamster, deren Namen Paul aber nicht kennt. Paul soll alle anderen Hamster aufsuchen und ihnen ein Korn übergeben, d.h. auf deren Kachel ablegen. Die anderen Hamster bewegen sich übrigens nicht von der Stelle. Beachte bei der Gestaltung des Territoriums, dass der Standardhamster genügend Körner im Maul hat!

3. (etwas schwieriger) Wie Aufgabe 2, nur die anderen Hamster sind über das gesamte, leere Territorium verstreut. Zusatzbedingung: Paul soll stehen bleiben, wenn er den letzten Hamster gefunden hat.

4. Der Standardhamster Willi sitzt ruhig und unbeweglich oben links in der Ecke und schaut sich die obere Reihe des beliebig großen Territoriums an. Falls er feststellen sollte, dass er in dieser Reihe nicht alleine ist, dreht er sich vor Freude zehnmal um sich selbst.
Lösungshinweis: Untersuche in einer *for-Schleife* alle Kacheln der obersten Reihe, ob sich ein Hamster darauf befindet

5. (etwas schwieriger) Wie Aufgabe 4, aber Willi schaut sich jetzt im gesamten Territorium um.

6. Der Standardhamster Leo entdeckt im Territorium ein süßes Hamstermädchen, deren Namen er nicht kennt. Als territoriumsbekannter Casanova läuft er natürlich sofort los, um sie kennenzulernen.
a) Das Hamstermädchen wartet auf ihn.
b) (etwas schwieriger) Das Hamstermädchen, welches Leos Namen kennt, geht ihm entgegen.
c) (schwierig) Das Hamstermädchen läuft vor dem bekannten Leo weg. Trifft Leo sie trotzdem?

7. (schwierig) Irgendwo im leeren Territorium befinden sich zwei Hamster, die sich gegenseitig nicht kennen. Sie sollen aufeinander zulaufen.

Lösungen

Aufgabe 1

```
Hamster Paul = Hamster.getStandardHamster();

void main() {
  int spalte = 1;
  while (!Territorium.mauerDa(0, spalte))  {
     Paul.vor();
     spalte++;
  }
}
```

Aufgabe 2

```
Hamster Paul = Hamster.getStandardHamster();
Hamster Willi = new Hamster(0,4,3,0);
Hamster Lilli = new Hamster(0,7,3,0);
Hamster Sabine = new
     Hamster(0,Territorium.getAnzahlSpalten()-1,3,0);

void main() {
 while (Paul.vornFrei())  {
  if (Territorium.getAnzahlHamster(0,Paul.getSpalte())>1)
    Paul.gib();
  Paul.vor();
 }
 if (Territorium.getAnzahlHamster(0,Paul.getSpalte())>1)
    Paul.gib();
}
```

Aufgabe 4

```
Hamster Paul = Hamster.getStandardHamster();
// Hamster Willi = new Hamster(0,4,3,0);

void main() {
  boolean hamsterDa = false;
  for (int i = 1; i <= 9; i++)
     if (Territorium.getAnzahlHamster(0,i) > 0)
        hamsterDa = true;
  if (hamsterDa)
    for (int i = 1; i<=40; i++) Paul.linksUm();
}
```

Aufgabe 6a)

```
Hamster Leo = Hamster.getStandardHamster();
Hamster Lilli = new Hamster(2,3,3,0);

void main() {
  boolean hamsterDa = false;
  int x = 0;
  int y = 0;
  for (int reihe = 1;
    reihe <= Territorium.getAnzahlReihen()-1;reihe++)
    for (int spalte=1;
       spalte<=Territorium.getAnzahlSpalten()-1;spalte++)
       if (Territorium.getAnzahlHamster(reihe,spalte)>0) {
         x = spalte;
         y = reihe;
       }
  while (Leo.getSpalte() != x) Leo.vor();
  Leo.linksUm();
  Leo.linksUm();
  Leo.linksUm();
  while (Leo.getReihe() != y) Leo.vor();
}
```

Konstanten

Sinn und Zweck von Konstanten ist es, bestimmten Werten einen Namen zu geben. Wir haben Konstanten schon häufig genutzt: Hamster.NORD, Hamster.OST, Hamster.SUED und Hamster.WEST sind Konstanten, die den Werten 0, 1, 2 und 3 einen Namen zuordnen.

Konstanten werden als spezielle Klassenattribute definiert, indem der eigentlichen Klassenattributdefinition das Schlüsselwort ***final*** vorangestellt wird.

Beispiel:
```
class ParfumHamster extends Hamster  {
    final static int KOELNISCH_WASSER = 4711;
}
```

Java-Programmierer halten sich im Allgemeinen an die Konvention, in den Namen für Konstanten nur Großbuchstaben und den Unterstrich zu verwenden.

Aufgaben

1. Definiere eine neue Hamsterklasse namens OrientierungsHamster. Diese Klasse soll zusätzlich folgende Methoden kennen:
```
void setze Blickrichtung(int richtung)
void gotoKachel(int reihe, int spalte)
    //Voraussetzung: keine inneren Mauern
boolean linksFrei()
boolean rechtsFrei()
boolean hintenFrei()
```
Implementiere die letzten drei neuen Methoden mit Hilfe der Methoden der Klasse Territorium!

2. Ein Orientierungshamster aus der Aufgabe 1 steht irgendwo in einem quadratischen Territorium mit unbekannter Blickrichtung. Er soll alle Kacheln auf der Hauptdiagonalen (von oben links nach unten rechts) mit jeweils einem Korn belegen.

Zufallszahlen

Es gibt in Java eine Klasse *Math*, welche grundsätzlich immer zur Verfügung steht und mathematische Klassenmethoden bereitstellt. Eine davon ist die Lieferung einer reellen Zufallszahl. Reelle Zahlen sind in Java vom Typ *double*.

Die Anweisung ***double zahl = Math.random();*** liefert eine zufällige reelle Zahl aus dem Intervall [0; 1[, also einschließlich 0 und ausschließlich 1.
Möchte man eine reelle Zufallszahl aus dem Intervall [0; 15[erzeugen, so schreibt man: ***zahl = Math.random() * 15;*** Dabei muss die Variable *zahl* natürlich vorher als vom Typ *double* deklariert worden sein, z.B. durch **double zahl;**

Eine zufällige reelle Zahl aus dem Intervall [14; 18[erhält man durch die Anweisung **zahl = Math.random() * 4 + 14;**

Möchte man eine zufällige ganze Zahl haben, so ist eine Typumwandlung notwendig. Diese geschieht durch Voranstellen des gewünschten Typs in Klammern: ***ganzZahl = (int) (Math.random() * 4 + 14);***

Beachte die Klammersetzung bei der Generierung der Zufallszahl! Diese ist wegen der Prioritätenregelung *Funktion vor Punktrechnung vor Strichrechnung* notwendig. Die folgende Anweisung würde immer das Ergebnis 14 liefern. ***ganzZahl = (int) Math.random() * 4 + 14;*** Im folgenden Beispiel dreht sich (oder auch nicht) der Hamster zufällig, bevor er einen Schritt geht.

```
void main() {
  int  zahl = (int) (Math.random()*4);
  if (zahl == 0) linksUm();
  else if (zahl == 1) { linksUm(); linksUm(); }
       else  if (zahl == 2)  {
                linksUm(); linksUm(); linksUm();
              }
  vor();
}
```

Im folgenden Beispiel geht der Hamster eine zufällige Anzahl von Schritten vor (mindestens 1, höchstens 10), falls dies möglich ist:

```
void main()  {
  int zufallszahl = (int) (Math.random()*10) + 1;
  geheVor(zufallszahl);
}

void geheVor(int z)  {
  for (int j=1; j<=z; j++)  {
     if (vornFrei()) vor();
  }
}
```

Aufgaben

1. Der Hamster steht in seinem Territorium in der linken unteren Ecke mit Blick nach Osten. Er soll auf jeder Kachel der Grundlinie eine zufällige Anzahl (≤ 5) von Körnern ablegen. Er hat genügend Körner im Maul.

2. Der Hamster steht in seinem Territorium in der linken unteren Ecke mit Blick nach Osten. Er soll auf jeder Kachel der Grundlinie entweder ein oder kein Korn ablegen. Er hat genügend Körner im Maul.

3. Der Hamster steht in der Mitte seines Territoriums mit Blick nach Osten. Er soll insgesamt 100 Schritte gehen, aber vor jedem Schritt soll er sich in eine zufällige Richtung drehen.

4. Der Hamster steht irgendwo in seinem (ziemlich kleinen!) Territorium mit beliebiger Blickrichtung. Er soll solange ziel- und planlos durch das Territorium irren, bis er zufällig die Kachel mit den Koordinaten (1; 4) erreicht.

5. Gegeben sei ein Territorium mit 5 Reihen und 5 Spalten. Die fünf Hamster Max, Moritz, Lisa, Lena und Lulu sollen an zufälligen Orten mit zufälliger Blickrichtung erzeugt werden. Falls zufällig auf einer Kachel mehr als ein Hamster stehen sollte, sollen sich alle Hamster vor Freude dreimal um sich selbst drehen.

6. Gegeben sei ein Territorium mit 4 Reihen und 4 Spalten, in dem die beiden Hamster Moritz und Lili planlos umherlaufen. Falls sie sich treffen, wird ein dritter Hamster erzeugt, welcher sich in die linke obere Ecke begibt. Danach endet das Programm.

7. Gegeben sei ein Territorium mit 5 Reihen und 10 Spalten. In der obersten Reihe werden an zufälligen Stellen 7 Hamster erzeugt, die gleichzeitig wie Fallschirmspringer nach unten fallen.

8. Der Hamster Paul, der irgendwo in einem beliebig großen Territorium ohne innere Mauern steht, ist auf der Flucht. Sein Jäger ist das Standardhamster-mädchen Lili, die auf der Kachel (0, 0) erzeugt wird. Paul läuft planlos durchs Territorium. Lili bewegt sich immer auf Paul zu. Die Jagd endet, wenn die beiden sich treffen.

Lösungen

Aufgabe 8

```
Hamster Lili = Hamster.getStandardHamster();
Hamster Paul = new Hamster (5,6,1,0);

void main() {
  while ((Paul.getSpalte() != Lili.getSpalte()) ||
            (Paul.getReihe() != Lili.getReihe()))  {
    PaulVor();
    LiliVor();
  }
}
```

```
void PaulVor()  {
  int zufallszahl = (int) (Math.random()*4);
  while (Paul.getBlickrichtung() != zufallszahl)
    Paul.linksUm();
  while (! Paul.vornFrei()) Paul.linksUm();
  Paul.vor();
}


void LiliVor()  {
   if ((Paul.getSpalte() != Lili.getSpalte()) ||
                (Paul.getReihe() != Lili.getReihe()))
     {
       if (Paul.getSpalte() > Lili.getSpalte()) {
         while (Lili.getBlickrichtung() != 1)
           Lili.linksUm();
         Lili.vor();
       }
       else if (Paul.getSpalte()< Lili.getSpalte()){
              while (Lili.getBlickrichtung() != 3)
                Lili.linksUm();
              Lili.vor();
            }
            else if (Paul.getReihe()>Lili.getReihe()) {
                   while (Lili.getBlickrichtung()!=2)
                     Lili.linksUm();
                   Lili.vor();
                 }
                 else if (Paul.getReihe() <
                                        Lili.getReihe()) {
                        while (Lili.getBlickrichtung()
                           != 0) Lili.linksUm();
                        Lili.vor();
                      }
     }
}
```

Arrays

```
int[]  feldname = new int[7];
```
// feldname entspricht der Anfangsspeicheradresse, ab der 7 ganze Zahlen
// gespeichert sind.

```
int[]  feld2 = null;
```
// feld2 entspricht einem INTEGER-Feld unbekannter Größe, allerdings wurde
// keine Adresse zugewiesen bzw. der Wert null

```
boolean[] feld3;
```
// unbekannt großes Feld von Wahrheitswerten, Initialadresse null

```
boolean[] feld4 = new boolean[3*4+2];
```
// Feld enthält 14 Wahrheitswerte

```
Hamster[] feld5 = new Hamster[9];
```
// Feld enthält 9 Hamster

Die einzelnen Zellen eines Arrays sind nummeriert. Der Index fängt bei 0 an.
Beispiel: Das folgende Programm enthält ein Array mit den ersten 5 Quadratzahlen.

```
void main() {
  int[] feld = new int[7];
  for (int i=0; i <= 6; i++) feld[i]=(i+1)*(i+1);
  int summe = 0;
  for (int i=0; i <= 6; i++) summe = summe + feld[i];
}
```

1	4	9	16	25	36	49

Bei der Erzeugung eines Arrays werden die einzelnen Zellen mit den Default-Werten des entsprechenden Array-Typs initialisiert (int: 0, boolean: false, Hamster: null).

Im folgenden Programm merkt sich der Hamster, welche Zellen in der obersten Reihe seines Territoriums Körner enthalten:

```
void main() {
  Hamster Paul = Hamster.getStandardHamster();
  boolean[] feld =
         new boolean[Territorium.getAnzahlSpalten()];
  for (int i=0; i < Territorium.getAnzahlSpalten();
                                                 i++) {
    feld[i]=Paul.kornDa();
    if (i<Territorium.getAnzahlSpalten() -1) Paul.vor();
  }
}
```

Jedes Array besitzt das Attribut ***length***, mit dem man seine Größe, also die Anzahl der Zellen ermitteln kann:

```
int[] menge = new int[23];
int x = menge.length;    // liefert den Wert 23
```

Arrays können Funktionen bzw. Methoden auch als Parameter übergeben werden. Die folgende Funktion liefert den ganzzahligen Anteil des Mittelwertes eines übergebenen Arrays:

```
void main() {
  int[] A = new int[8];
  int mw = mittelwert(A);
}
```

```
int mittelwert(int[] feld) {
  int summe  = 0;
  for (int i=0; i<feld.length; i++) summe = summe +
feld[i];
  summe = summe / feld.length;
  return summe;
}
```

Aufgaben

1. Der Standardhamster läuft in der obersten Reihe von links nach rechts und sammelt dabei alle herumliegenden Körner auf. Anschließend läuft er wieder zurück und legt alle Körner an ihre Ursprungsorte zurück.

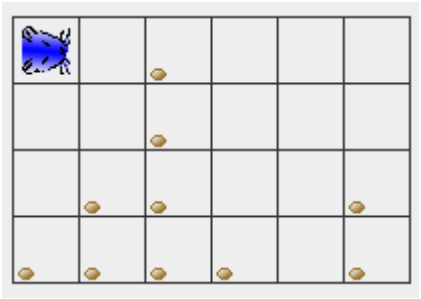
2. In der obersten Reihe liegen auf jeder Kachel einige Körner herum. Der Hamster soll diese Körnerhaufen der Größe nach sortieren. Anschließend soll links oben der kleinste und rechts oben der größte Körnerhaufen liegen.

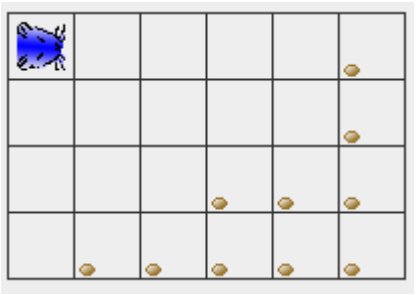
3. Gegeben sei ein Territorium mit 10 Reihen und ein Hamsterfeld durch Hamster[] tiere = new Hamster[10]; Beachte, dass dieses Feld zunächst nur Namen für Hamster enthält (z.B. tiere[4] ist ein Name). Die zugehörigen Hamster müssen alle erst noch erzeugt werden! Die Hamster sollen am linken Rand mit Blickrichtung nach Osten erzeugt werden und alle
a) gleichzeitig b) nacheinander zum rechten Rand laufen.

4. Die 10 Hamster aus Aufgabe 3 veranstalten ein Rennen zum rechten Rand. Allerdings werden sie durch herumliegende Körner aufgehalten, die sie natürlich fressen anstatt sofort weiterzugehen.
a) alle Hamster erreichen den rechten Rand
b) nur der schnellste Hamster erreicht den rechten Rand. Danach bleiben alle Hamster stehen.
c) alle Hamster erreichen den rechten Rand, und nur der Sieger vollführt anschließend ein Freudentänzchen.

5. Im Territorium werden auf zufälligen Kacheln 10 Hamster erzeugt, welche anschließend plan- und ziellos umher laufen.

6. Im Territorium liegen verschieden hohe Körnerhaufen. Der Hamster frisst alle Körner, merkt sich dabei die Größe der Haufen mithilfe eines Feldes, sortiert dieses Feld und legt anschließend die Körnerhaufen der Größe nach sortiert wieder hin.





Zweidimensionale Arrays

Die Dimension eines Arrays wird bei der Definition festgelegt, und zwar durch die Anzahl an Paaren von eckigen Klammern. Im folgenden Beispiel wird eine sog. *Matrix* mit drei Zeilen und drei Spalten erzeugt.

```
int [] []  ZweiDimFeld = new int [3] [3];
```

Im Folgenden wird der Mittelwert eines zweidimensionalen Feldes berechnet:

```
void main() {
  int [][] A = new int[3][3];
  int mw = mittelwert(A);
}

int mittelwert(int[][] feld) {
  int summe  = 0;
  for (int zeile=0; zeile<3; zeile++)
     for (int spalte=0; spalte<3; spalte++)
       summe = summe + feld[zeile][spalte];

  summe = summe / 9;
  return summe;
}
```

Bemerkung: Die Anzahl der Zeilen eines zweidimensionalen Feldes ***A*** erhält man durch den Ausdruck **`A.length`**
Die Anzahl der Spalten eines zweidimensionalen Feldes ***A*** erhält man durch den Ausdruck **`A.zeile.length`** wobei *zeile* eine gültige Zeilennummer sein muss.

Aufgaben

1. In einem quadratischen Territorium liegen überall Körner herum. Der Standardhamster soll direkt zu derjenigen Kachel gehen, welche die meisten Körner enthält und diese auffressen.
   Hinweise: Die Klasse Territorium liefert Angaben über die Körneranzahlen auf einzelnen Kacheln. Bilde das Territorium auf ein zweidimensionales Feld ab! Bestimme die größte Zahl in diesem Feld! Bewege den Hamster anschließend zu der entsprechenden Kachel!

2. Nur in der linken unteren Territoriumshälfte (unterhalb der Hauptdiagonalen) liegen vereinzelt Körner herum. Spiegle das Körnerbild an der Haupt-diagonalen!

Strings

Ein sehr wichtiger Datentyp sind Folgen von Zeichenketten, im englischen Strings genannt. Strings sind Objekte einer Klasse ***String***. Beispiel:

```
void main() {
    String satz1 = new String("Hallo");
    String satz2 = "Hallo";
    String satz3 = new String("");
}
```

Zwei aufeinander folgende Anführungszeichen `""` repräsentieren den sog. Leerstring.

Die Klasse *String* stellt u.a. folgende Methoden zur Verfügung:

public boolean equals(String anderer);
// überprüft, ob die beiden Strings die gleiche Zeichenkette enthalten.
Beispiel:
```
String s1 = "Dieter";
String s2 = "Dieter";
if (s1.equals(s2))  Paul.linksUm();
if (s1 == s2) Lili.linksUm();
```
Wichtig: im obigen Beispiel dreht sich zwar Paul, aber nicht unbedingt Lili. Das liegt daran, dass die beiden Strings zwar denselben Inhalt haben, aber unterschiedlichen Speicherbereich belegen, also unterschiedliche Variablen sind. Leider wird letzteres nicht in allen Java-Versionen gleichartig behandelt, d.h. es gibt Java-Versionen, in denen sich Lili auch dreht, und andere Versionen, in denen sich Lili nicht dreht.

public int length();
// liefert die Anzahl an Zeichen
Beispiel: `int AnzahlZeichen = s1.length();`

public String substring(int beginIndex, int endIndex);
// liefert den Teilstring zwischen den angegebenen Grenzen, ***einschließlich*** *der*
// linken und ***ausschließlich*** *der rechten Grenze. Die Indizes liegen von 0 bis*
// length() – 1

Beispiel:
```
String  s1 = "Goethe-Gymnasium";
String  s2 = s1.substring(7, 10);
// liefert: s2 = "Gym"
```

public int indexOf(String substring);
// liefert die erste Position, an welcher der übergebene Teilstring vorkommt.
// Falls er nicht vorkommt, wird -1 geliefert.

Beispiel:
```
String s1 = "Informatik am GG ist toll";
int zahl = s1.indexOf("GG");
if (zahl != -1) Paul.linksUm();
```

Man kann mehrere Strings leicht hintereinander ketten:
```
void main() {
  String s1 = new String("Hallo");
  String s2 = new String("Hallo");
  s3 = s1 + "! " + s2 + "!!!";
  // liefert s3 = "Hallo! Hallo!!!"
  String s4 = "Dieter";
}
```

Im obigen Beispiel gilt übrigens, dass s1 != s2, weil die beiden Strings unterschiedliche Speicherbereiche belegen. Allerdings liefert die Abfrage *s1.equals(s2)* den Wahrheitswert *true*.

Im Zusammenhang mit dem +-Operator werden int-Werte automatisch in Strings umgewandelt. Folgendes ist also möglich:
```
int zahl = 12;
String s = "Der Hamster besitzt " + zahl + "Körner"
```

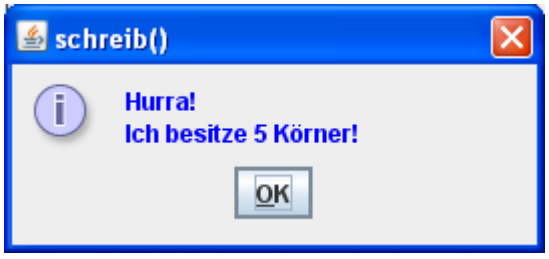
Stringausgabe

Die Klasse Hamster besitzt auch eine Ausgabemethode, welche wie folgt definiert ist:
public void schreib(String zeichenkette)

```
void main() {
  Hamster Paul = Hamster.getStandardHamster();
  String s1 = new String("Hallo");
  Paul.schreib(s1);
  String s2 = "Dieter";
  Paul.schreib(s2);
}
```

Zeilenumbrüche in der Ausgabebox werden durch das Sonderzeichen **\n** in der Zeichenkette bewirkt.

```
void main() {
  Hamster Paul = Hamster.getStandardHamster();
  Paul.schreib("Hurra!\nIch besitze "+Paul.getAnzahlKoerner() +" Körner!");
}
```




```
int   zahl = 5;
Paul.schreib(zahl);        // funktioniert nicht, weil zahl kein String ist.
Paul.schreib(""+zahl);  // funktioniert, weil Java daraus einen String bildet.
```

Aufgaben

1. Ein Hamster soll bis zur nächsten Wand laufen und dabei auf jeder Kachel alle Körner fressen. Nach dem „Abgrasen“ einer Kachel soll er auf dem Bildschirm ausgeben, wie viele Körner er gerade gesammelt hat. Zum Schluss soll er die Gesamtzahl an gesammelten Körnern ausgeben.

2. Der Standardhamster Joe soll dem Benutzer mitteilen, wie groß das Territorium ist (Anzahl der Zeilen und Spalten) und wie viele Hamster sich insgesamt im Territorium befinden.

3. Der Hamster Willi soll die Koordinaten seiner Kachel, auf der er gerade steht, ausgeben.

4. Der Standardhamster soll die ersten 10 Quadratzahlen berechnen und in einer einzigen Ausgabebox untereinander ausgeben.

5. Der Standardhamster soll die Summe der ersten 100 natürlichen Zahlen berechnen und das Ergebnis ausgeben.

6. Der Standardhamster soll den Mittelwert der ersten 10 Quadratzahlen berechnen und ausgeben.

7. Der Standardhamster soll die Fakultäten der ersten 10 natürlichen Zahlen berechnen und alle Ergebnisse in einer einzigen Ausgabebox ausgeben.

8. Die ersten beiden sog. Fibonacci-Zahlen sind 1 und 2. Die jeweils nächste Fibonacci-Zahl ist die Summe der beiden vorherigen. Die Fibonacci-Zahlen lauten also: 1, 2, 3, 5, 8, 13, …. Der Standardhamster soll die ersten 10 Fibonacci-Zahlen berechnen und in einer einzigen Ausgabebox untereinander ausgeben.

Lösungen

Aufgabe1

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int koernerAufPlatz;
  while (Paul.vornFrei())  {
    koernerAufPlatz = Territorium.getAnzahlKoerner
                          (Paul.getReihe(),Paul.getSpalte());
    while (Paul.kornDa()) Paul.nimm();
    if (koernerAufPlatz > 0) Paul.schreib("Ich habe " +
                          koernerAufPlatz + " aufgenommen.");
    Paul.vor();
  }
  koernerAufPlatz = Territorium.getAnzahlKoerner
                          (Paul.getReihe(),Paul.getSpalte());
  while (Paul.kornDa()) Paul.nimm();
  if (koernerAufPlatz > 0) Paul.schreib("Ich habe " +
                          koernerAufPlatz + " aufgenommen.");
  Paul.schreib("Insgesamt habe ich " + Paul.getAnzahlKoerner()
                                          +" gesammelt.");
}
```

Aufgabe2

```
Hamster Joe = Hamster.getStandardHamster();
Hamster Willi = new Hamster(2,3,Hamster.OST,8);
Hamster Paul = new Hamster(3,3,Hamster.OST,0);

void main() {
 Joe.schreib("Mein Territorium hat " +
     Territorium.getAnzahlReihen() + " Reihen und " +
        Territorium.getAnzahlSpalten() + " Spalten");
 Joe.schreib("Außerdem leben hier " +
        Territorium.getAnzahlHamster() + " Hamster");
}
```

Aufgabe3

```
Hamster Willi = Hamster.getStandardHamster();
void main() {
  Willi.schreib("Ich stehe auf der Kachel (Reihe " +
                    Willi.getReihe() + ";  Spalte " +
                            Willi.getSpalte() + ")");
}
```

Aufgabe4

```
Hamster Willi = Hamster.getStandardHamster();
void main() {
  String ausgabe = "10 Quadratzahlen: \n";
  for (int i = 1; i <= 10; i++)
       ausgabe = ausgabe + i*i + "\n";
  Willi.schreib(ausgabe);
}
```

Aufgabe5

```
Hamster Willi = Hamster.getStandardHamster();
void main() {
  int summe = 0;
  for (int i = 1; i <= 100; i++) summe = summe + i;
     Willi.schreib(""+summe);
}
```

Aufgabe6

```
Hamster Willi = Hamster.getStandardHamster();
void main() {
  double summe = 0;
  for (int i = 1; i <= 10; i++) summe = summe + i*i;
  double mittelwert = summe / 10;
  Willi.schreib(""+mittelwert);
}
```

Hinweis: die Variable *summe* muss auch vom Typ ***double*** sein. Ansonsten würde der Quotient *summe / 10* die Nachkommastellen ignorieren.

Aufgabe7

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int fak;
  String ausgabe = "";
  for (int i = 1; i <= 10; i++)  {
    fak = 1;
    for (int j = 1; j <= i; j++)  {
      fak = fak * j;
    }
    ausgabe = ausgabe + i + "! = " + fak + "\n";
  }
  Paul.schreib(ausgabe);
}
```

Aufgabe8

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int zahl1 = 1;
  int zahl2 = 2;
  int zahl3;
  String ausgabe = "1\n2\n";
  for (int i = 1; i <= 8; i++)  {
    zahl3 = zahl1 + zahl2;
    ausgabe = ausgabe + zahl3 + "\n";
    zahl1 = zahl2;
    zahl2 = zahl3;
  }
  Paul.schreib(ausgabe);
}
```

Eingabe

Die Klasse Hamster besitzt auch zwei Eingabemethoden, die folgendermaßen definiert sind:

```
public String liesZeichenkette(String aufforderung);
public int liesZahl(String aufforderung);
```

```
void main() {
  Hamster Paul = Hamster.getStandardHamster();
  String name = Paul.liesZeichenkette("Wie heißt du?");
  Paul.schreib("Hallo " + name);
  int n = Paul.liesZahl("Wie oft soll ich mich drehen?");
  for (int i=1; i<= 4*n; i++) Paul.linksUm();
}
```

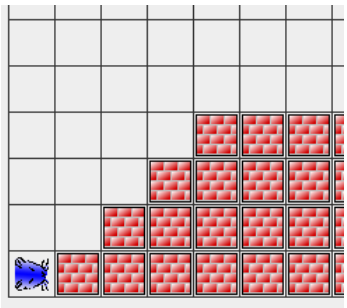
Bei der Eingabemethode *liesZahl* wird eine gültige ganze Zahl erwartet. Bei fehlerhafter Eingabe liefert diese Methode den Wert 0.

Aufgaben

1. Der Hamster wird als Rechenmaschine benutzt. Der Benutzer nennt dem Hamster eine ganze Zahl. Daraufhin antwortet der Hamster mit dem Quadrat der Zahl.

2. Der Hamster dreht sich einmal um sich selbst. Das Ganze wird solange wiederholt, wie der Benutzer auf die Frage von Paul „Noch einmal (ja / nein)?“ die Zeichenfolge „ja“ eingibt.

3. Der Hamster steht direkt vor einer Treppe, bei der jede Stufe einen Schritt hoch und einen Schritt breit ist. Er steigt hoch. Oben angekommen, teilt er dem Benutzer die Anzahl der Stufen mit.



4. Der Hamster fragt nach einem Passwort. Nur wenn das richtige Passwort *(Informatik)* eingegeben wird, dreht er sich vor Freude einmal um sich selbst. Bei falscher Eingabe erhält der Benutzer eine entsprechende Mitteilung. Wird zweimal hintereinander ein falsches Passwort eingegeben, so wird das Programm (ohne Freudendrehung) beendet.

5. Der Benutzer nennt dem Hamster eine natürliche Zahl, welche kleiner als 1000 ist. Der Hamster untersucht, ob es eine Quadratzahl ist und meldet sein Ergebnis.

6. Bei dieser Aufgabe darf mal der Benutzer und nicht der Programmierer dem Hamster Befehle geben. Dazu fordert der Hamster, der in einem beliebigen Territorium sitzt, jeweils mit „Gib mir einen Befehl!“ auf und reagiert auf die Antworten „vor“, „linksUm“, „nimm“ und „gib“ durch Ausführung der entsprechenden Hamster-Befehle. Bei der Antwort „stop“ wird das Programm beendet.
a) auf unbekannte Befehle soll der Hamster nicht reagieren.
b) unbekannte Befehle soll der Hamster mit einer entsprechenden Meldung zurückweisen.

7. Der Benutzer gibt ein nur aus Großbuchstaben bestehendes Wort ein. Der Hamster untersucht, ob dieses eingegebene Wort ein sog. *Palindrom* ist und meldet sein Ergebnis. Palindrome sind Worte, die rückwärts gelesen wieder dasselbe Wort ergeben. Beispiele: ANNA, OHO, EINNEGERMITGAZELLEZAGTIMREGENNIE

8. Der Standardhamster befindet sich oben links mit Blick nach Süden. Es gibt auf mehreren Kacheln jeweils ein Korn. Der Benutzer soll dem Hamster mitteilen, wo es Körner gibt. Dazu fragt der Hamster zuerst nach der Reihen-koordinate (beachte: die Koordinaten beginnen bei 0), geht in die entsprechende Reihe, fragt dann nach der Spaltenkoordinate, geht dorthin und frisst das Korn. Anschließend geht er in seine Ausgangsstellung zurück.

Hinweis: Die Klasse *Territorium* besitzt eine Methode, mit der man die Gesamtzahl der Körner im Territorium ermittelt.

9. Wie Aufgabe 8, nur diesmal steht der Standardhamster irgendwo im Territorium und geht auch nicht wieder in seine Ausgangsstellung zurück, sondern er begibt sich nach jeder Kornaufnahme direkt zum nächsten Korn hin.

10. Wie Aufgabe 9, nur diesmal fragt der Hamster nicht, wo er ein Korn finden kann. Denn die Klasse Territorium besitzt eine Methode, mit der man jede Kachel daraufhin überprüfen kann, ob sie ein Korn enthält. Solange es überhaupt noch Körner im Territorium gibt, nutzt der Hamster diese Methode, um selbst herauszufinden, wo noch ein Korn liegt.

11. Der Hamster hat eine unbekannte Zahl an Körnern im Maul. Er fragt den Benutzer, auf welchen Kacheln er jeweils ein Korn ablegen soll. Das macht er solange, bis er kein Korn mehr hat. Dies teilt er dann dem Benutzer auch mit.

12. Irgendwo in einem beliebigen Territorium liegt ein Korn und irgendwo steht auch der Hamster, welcher das Korn fressen möchte. Allerdings hat er völlig die Orientierung verloren. Er darf dem Benutzer jeweils zwei Fragen stellen: „Bin ich am Korn angelangt?“ und „Muss ich mich nach Norden, Osten, Süden oder Westen wenden?“. Auf die erste Frage antwortet der Benutzer mit „ja“ oder „nein“, auf die zweite mit „Nord“, „Ost“, „Süd“ oder „West“. Der Hamster macht gegebenenfalls einen Schritt oder frisst das Korn. Zum Schluss bedankt sich der Hamster.

Lösungen

Aufgabe1

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int zahl = Paul.liesZahl("bitte eine Zahl eingeben!");
  Paul.schreib("Das Quadrat der Zahl ist " + zahl*zahl);
}
```

Aufgabe2

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  String antwort;
  do  {
    drehDich(Paul);
    antwort =Paul.liesZeichenkette("nochmal? ja/nein");
  }
  while (antwort.equals("ja"));
}

void drehDich(Hamster egal)  {
  for (int i = 1; i <= 4; i++)  egal.linksUm();
}
```

Aufgabe3

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
 int anzahl = 0;
 do  {
   Paul.linksUm();
   Paul.vor();
   Paul.linksUm(); Paul.linksUm(); Paul.linksUm();
   Paul.vor();
   anzahl++;
 }
```

```
 while (! Paul.vornFrei()) ;
 Paul.schreib("es sind "+ anzahl + " Stufen");
}
```

Aufgabe4

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int anzahl = 0;
  String code = Paul.liesZeichenkette("bitte Paßwort
                                        eingeben!");
  if (!code.equals("Informatik"))  {
     anzahl++;
     code = Paul.liesZeichenkette("falsch! Bitte
                                   Paßwort eingeben!");
     if (!code.equals("Informatik"))  {
        anzahl++;
        Paul.schreib("falsch! Programm wird beendet");
    }
  }
  if (anzahl < 2) drehDich(Paul);
}

void drehDich(Hamster egal)  {
  for (int i = 1; i <= 4; i++)  egal.linksUm();
}
```

Aufgabe5

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  int zahl = Paul.liesZahl("bitte natürliche Zahl
                            kleiner 1000 eingeben!");
  boolean quadrat = false;
  for (int i=1; i<32; i++)  if (i*i == zahl) quadrat=true;
  if (quadrat == true) Paul.schreib(zahl + " ist eine
                                    Quadratzahl!");
  else Paul.schreib(zahl + " ist keine Quadratzahl!");
}
```

Aufgabe6b)

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  String befehl;
  do  {
    befehl= Paul.liesZeichenkette("Gib mir einen Befehl!");
    erlaubt = befehl.equals("vor")
              || befehl.equals("linksUm")
              || befehl.equals("nimm")
              || befehl.equals("gib")
              || befehl.equals("stop");
    if (!erlaubt) Paul.schreib("unbekannter Befehl!");
    if (befehl.equals("vor")) Paul.vor();
    else if (befehl.equals("linksUm")) Paul.linksUm();
         else if (befehl.equals("nimm")) Paul.nimm();
              else if (befehl.equals("gib")) Paul.gib();
  }
  while (! befehl.equals("stop"));
}
```

Aufgabe7

```
Hamster Paul = Hamster.getStandardHamster();
void main() {
  String wort = Paul.liesZeichenkette("Bitte
                                  Palindrom eingeben!");
  String wort2 = "";
  String buchstabe;
  for (int i = 0; i < wort.length(); i++)  {
    buchstabe =  wort.substring(i, i+1);
    wort2 = buchstabe + wort2;
  }
  if (wort.equals(wort2)) Paul.schreib("es ist ein
                                        Palindrom");
  else  Paul.schreib("es ist kein Palindrom");
}
```

Aufgabe 10

```
Hamster Paul = Hamster.getStandardHamster();

int xKorn;  // globale Variablen
int yKorn;

void main() {
  while (Territorium.getAnzahlKoerner() > 0)  {
    int reihenZahl = Territorium.getAnzahlReihen();
    int spaltenZahl = Territorium.getAnzahlSpalten();
    for (int reihe=0; reihe < reihenZahl; reihe++)
       for (int spalte=0; spalte<spaltenZahl; spalte++)
         if (Territorium.getAnzahlKoerner(reihe,
                                          spalte) > 0)   {
           xKorn = spalte;
           yKorn = reihe;
         }
    geheZumKornUndNimmEs();
  }
}

void geheZumKornUndNimmEs()  {
  if (yKorn>Paul.getReihe())  dreheNachSued();
  else  if (yKorn<Paul.getReihe())  dreheNachNord();
  while (yKorn != Paul.getReihe()) Paul.vor();

  if (xKorn>Paul.getSpalte())   dreheNachOst();
  else  if (xKorn<Paul.getSpalte())  dreheNachWest();
  while (xKorn != Paul.getSpalte()) Paul.vor();

  Paul.nimm();
}



void dreheNachNord()  {
  while (Paul.getBlickrichtung()!= 0) Paul.linksUm();
```

```
}

void dreheNachOst()  {
  while (Paul.getBlickrichtung()!= 1) Paul.linksUm();
}

void dreheNachSued()  {
  while (Paul.getBlickrichtung()!= 2) Paul.linksUm();
}

void dreheNachWest()  {
  while (Paul.getBlickrichtung()!= 3) Paul.linksUm();
}
```