Teil

Objektorientierte Programmierung

Unterrichtseinheit 20

(Hamster-) Objekte

Dr. Dietrich Boles

Gliederung

- Imperatives Java-Hamster-Modell
- Objektorientiertes Java-Hamster-Model
- Hamster-Objekte
- Standard-Hamster
- Hamster-Befehle
- Objektorientierte Hamster-Programme
- Neue Hamster-Befehle
- Bezug zur Objektorientierten Programmierung

Imperatives Java-Hamster-Modell

Imperatives Hamster-Modell (Charakteristika):

- einzelner Hamster → Standard-Hamster
- Hamster ist namenlos
- Hamster ist immer da
- 4 Grundbefehle und 3 Testbefehle
- 4 Eigenschaften (Attribute) (→ Zustand):
  - Reihe der Kachel
  - Spalte der Kachel
  - Blickrichtung
  - Anzahl Körner im Maul
- Hamster-Programm: Aufruf von Funktionen (Befehlen)

Objektorientiertes Java-Hamster-Modell (1)

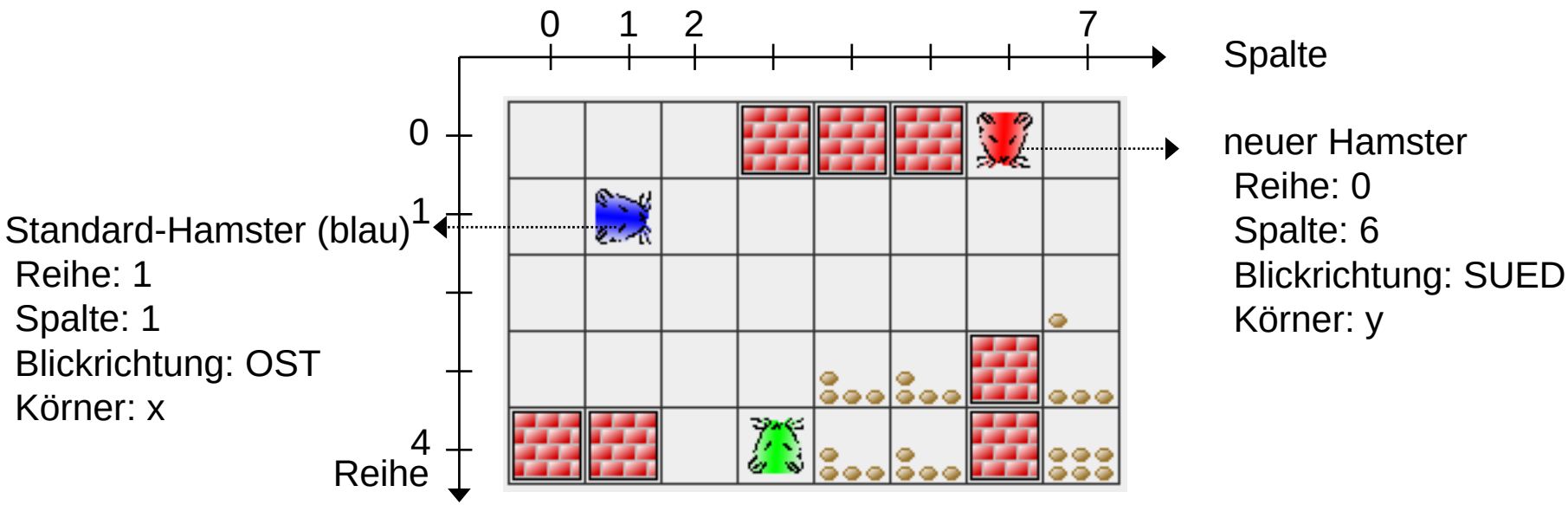
Objektorientiertes Hamster-Modell (Charakteristika):

- Standard-Hamster existiert weiterhin
- Erzeugung weiterer Hamster möglich
- Jeder Hamster kennt die 4 Grundbefehle und 3 Testbefehle
- Es kommen einige neue Befehle hinzu
- Auf einer Kachel können mehrere Hamster stehen
- Jeder Hamster besitzt die 4 Attribute
- Jeder Hamster hat einen (oder mehrere) Namen
- Wenn ein Hamster einen Fehler macht, sterben alle
  (→ Programmabbruch)
- OO-Hamster-Programm: Aufruf von Funktionen für Hamster

Objektorientiertes Java-Hamster-Modell (2)

Objektorientiertes Hamster-Modell (Voraussetzungen):

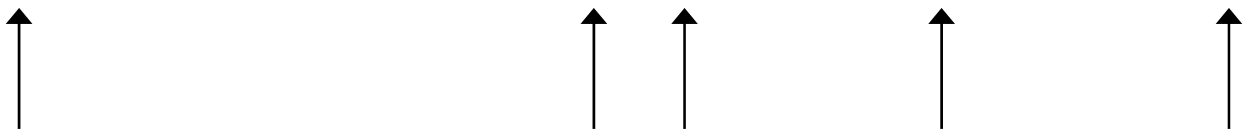
- Zuordnung von Namen zu Hamstern
- Anweisung zur Erzeugung von Hamstern
- Notation, welcher Hamster einen bestimmten Befehl ausführen soll
- Koordinatensystem für das Territorium



Hamster-Objekte

Erzeugung neuer Hamster:

```
Hamster paul = new Hamster(0, 6, Hamster.SUED, 8);
```



Name (Bezeichner)

Reihe (`int`)

Spalte (`int`)

Körneranzahl (`int`)

`Blickrichtung (int)`
`Hamster.NORD (= 0)`
`Hamster.OST (= 1)`
`Hamster.SUED (= 2)`
`Hamster.WEST (= 3)`

Mögliche Laufzeitfehler:

- ungültige oder mit Mauern besetzte Kachel
- ungültige Blickrichtung
- negative Körneranzahl

Standard-Hamster

Zuordnung eines Namens an den Standard-Hamster:

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

Name (Bezeichner)

Befehle

Aufruf von Hamster-Befehlen:

```
<name>.<befehl>
```

Beispiele:

```
Hamster paul = Hamster.getStandardHamster();
if (paul.vornFrei()) paul.vor();

Hamster willi = new Hamster(0, 2, Hamster.OST, 2);
while (!willi.maulLeer()) willi.gib();
```

OO-Hamster-Programme

Objektorientiertes Hamster-Programm:

```
void main() {
  Hamster paul = Hamster.getStandardHamster();
  while (paul.vornFrei()) {
    paul.vor();
    if (paul.kornDa()) {
      paul.nimm();
    }
  }
}
```

Demo

Neue Befehle (1)

Zusätzliche Hamster-Befehle:

- `int getReihe()`
- `int getSpalte()`
- `int getBlickrichtung()`
- `int getAnzahlKoerner()`

- `void schreib(String nachricht)`
- `String liesZeichenkette(String aufforderung)`
- `int liesZahl(String aufforderung)`

Neue Befehle (2)

```
void main() {
  Hamster paul = Hamster.getStandardHamster();
  Hamster willi =
    new Hamster(paul.getReihe() + 1, paul.getSpalte(),
                Hamster.OST, 0);
  int schritte = paul.liesZahl("Anzahl an Schritten?");
  while (schritte>0 && paul.vornFrei() && willi.vornFrei()) {
        paul.vor();
        willi.vor();
        schritte--;
  }
  willi.schreib("Fertig!");
}
```

Demo

Bezug zur Objektorientierten Programmierung

- Hamster sind Objekte einer (vordefinierten) Klasse `Hamster`
- Eine **Klasse** ist quasi ein Bauplan für gleichartige Dinge (**Objekte**)
- Von einer Klasse können Objekte erzeugt werden
- Eine Klasse definiert (intern) Variablen (**Attribute)**, die jedes Objekt besitzt und die seinen Zustand repräsentieren
- Eine Klasse definiert Funktionen (**Methoden**), die für Objekte der Klasse aufgerufen werden können
- Der Methodenaufruf erfolgt via der Punktnotation über Namen (**Objektvariablen**), die einem Objekt zugeordnet werden können
- Eine Klasse definiert einen neuen **Typ**
- OO-Programme: Aufruf von Methoden für/von Objekten