Praktische Informatik I
TutorIn: Berghöfer/Senger

WS 2012/13
Gruppe: 8

Beate Ruffer
Mohamadreza Khostevan
Leopold Schulz-Hanke

Übungsblatt 8

Aufgabenlösung
Abgabe: 16.12.2012

Aufgabe 1 Alles super hier (20%)

In unserem Spiel sind 2 Klassen implementiert dessen Instanzen Aufgehoben werden können: Hammer und Scissor. Sie besaßen schon die Superklasse `Collectable` seit einigen Abgaben. Auch die Klassen der Hindernisse: Bush, Stone und Comb erbten schon von der Superklasse `Obstacle`. Da wir aber in vorherigen Aufgaben mit Hilfe vom *instanceof*-Operator auf ein passendes Inventar prüfen sollten, war es nicht möglich das passende Inventar als Attribut zu hinterlegen und die Methode in der Superklasse die Prüfung zu überlassen. Nun haben wir eine Objektattribut `isBeatenBy` vom Typ Class eingeführt, mit dessen Hilfe der Vergleich durchgeführt wird. Somit können wir die prüfende Methode in der Superklasse Obstacle verschieben. Dadurch haben wir 3-Fach duplizierten Code in nur eine Methode zusammengeführt.

Listing 1: Klasse *Obstacle*

```
3  /**
4   * Oberklasse für alle Hindernisse. Diese Klasse vereint alle Hindernisse,
5   * so dass man leichter auf ein Hindernis prüfen kann.
6   *
7   * @author Beate Ruffer (Bea), Mohamadreza Khostevan (Amir), Leopold Schulz-
     Hanke (Leo)
8   * @version 2.0.0
9   *
10  */
11 public class Obstacle extends Actor
12 {
13     /**
14      * Sound Datei, die Abgespielt wird, wenn dieses Hindernisse überwunden
         wurde.
15      */
16     public String beatenSound;
17
18     /**
19      * Definiert durch Objekte welcher Klasse das Hindernis überwunden werden
          kann
20      */
21     protected Class isBeatenBy;
22
23     /**
24      * Konstruktor von Obstacle wird von der Subklasse aufgerufen.
25      * @param isBeatenBy Die Klasse des Objekts, welches das Hindernis ü
         berwinden kann.
26      */
27     public Obstacle(Class isBeatenBy){
28         this.isBeatenBy = isBeatenBy;
29     }
30
31     /**
```

```
32      * Prüft ob das sich im Inventar befindende Objekt eine Instanz von
         isBeatenBy ist. Falls ja, wird das Hindernis von dem Objekt
33      * im Inventar geschlagen und die Heldin kann passieren.
34      *
35      * @param collectable Das zu prüfende Objekt.
36      * @return true, wenn Hindernis überwunden ist.
37      */
38     public boolean isBeaten(Actor collectable)
39     {
40         return collectable.getClass() == isBeatenBy ;
41     }
42 }
```

Nun müssen die Klassen der Hindernisse, nur noch das passende Inventar (über den Konstruktor der Superklasse) und den Sound setzen.

Listing 2: Konstruktor von *Stone*

```
18 public Stone(){
19         super(Hammer.class);
20         beatenSound = "explosion.wav";
21     }
```

Tests. Da nun mal jedes kleine Refactoring Regression-Bugs hervorbringen kann, haben wir erneut getestet. Wir haben einen Hammer in unser Inventar aufgenommen, konnten wie erwartet keinen Bush oder Comb passieren. Nur der Stone konnte überwunden werden (der richtige Sound wurde auch abgespielt). Mit der Schere im Inventar konnten wir den Stone nicht überwinden, dafür aber Comb und Bush (ebenfalls mit den richtigen Sounds).

Aufgabe 2 Massenkarambolage (30%)

1. Methode

Listing 3: *getCollidingObject(Class cls)*-Methode

```
25 /**
26      * Liefert das Objekt der Klasse cls mit dem eine Kolliosion besteht
         .
27      * @param cls die Klasse, nach dessen Kollision geprüft wird
28      * @return das Objekt, mit dem Kollidiert wurde
29      */
30     public Actor getCollidingObject(Class cls){
31         return getOneIntersectingObject(cls);
32     }
```

2. Methode

Listing 4: *collidesWith(Class cls)*-Methode

```
25 /**
26      * Prüft ob gerade eine Kollision besteht
27      * @param cls die Klasse, nach dessen Kollision geprüft wird
28      * @return ob gerade eine Kollision mit einer Instanz der Klasse cls
          besteht
29      */
```

```
30      public boolean collidesWith(Class cls){
31          return getCollidingObject(cls) != null;
32      }
```

3. Methode

Listing 5: *newCollisionWith(Class cls)*-Methode

```
25 /**
26       * Prüft, ob gerade eine neue Kollision mit einer Instanz
27       * der Klasse cls begonnen hat.
28       * @param cls die Klasse, nach dessen Kollision geprüft wird
29       * @return ob gerade eine neue Kollision mit einer Instanz der
          Klasse cls begonnen hat
30       */
31      public boolean newCollisionWith(Class cls) {
32          Crashable obj =  null;
33          if ( collidesWith(cls) ){
34              if (collisions.get(cls) == null || collisions.get(cls) !=
                 obj ){
35                  collisions.put(cls, obj);
36                  return true;
37              }
38          }
39          return false;
40      }
```

Erneut musste einiges an Refactoring gemacht werden. Alle Klassen, mit dessen Instanzen die Biene nun kollidieren kann, erben nun von `Crashable`. Jede Subklasse von Crashable hat nun auch eine Methode `handleCrash(Bee bee)`, mit der sie auf (von der Biene registrierten) Kollisionen reagieren kann. Registriert die Biene nun eine Kollision, ruft sie den Crash Hanlder des Objektes auf, mit dem sie kollidiert. Dabei übergibt sie sich selbst als Paramter (für eventuelle Zugriffe auf das Inventar oder Scoreboard der Biene).

Aufgabe 3 Richtig kollidieren (50%)

1.

2.

3.