Prof. Dr. Stefan Brass
Dipl.-Inform. Annett Thüring
Dipl.-Inform. Steffen Schiele
Dipl.-Inform. Andreas Bienert

Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik

Wintersemester 2014/15

Objektorientierte Programmierung: Hausaufgabenblatt 12

Abgabe: 05.02.2015, 12:00 (Lösung im Internet)

Hausaufgabe 12: (4 Theoriepunkte)

Das Spiel aus Hausaufgabe 11 soll jetzt noch etwas weiter entwickelt werden, und zwar soll es um Gegenstände erweitert werden.

- Die meisten Gegenstände sind Schätze, die nur die Punktzahl des Spielers erweitern.
- Einige Gegenstände sind Schlüssel, die es ermöglichen, ein Feld zu betreten, das man sonst nicht betreten könnte. Das kann z.B. eine Tür sein, die man nur mit dem Schlüssel öffnen kann. Oder eine Mauer, die man nur mit einer Leiter überwinden kann. Oder man braucht eine Taucherausrüstung, um eine überflutete Höhle zu durchschwimmen. Es ist auch möglich, dass man das Feld zwar betreten kann, aber Schaden erleidet, wenn man den Gegenstand nicht hat (z.B. eine Rüstung etc. gegen Dornen). Es gibt also viele Möglichkeiten, wie man diesen Mechanismus nutzen kann. Für diese Hausaufgabe verlangt ist aber nur eine einfache Tür.
- Wir wollen dann noch eine extra-starke Lampe vorsehen, mit der man zwei Kästchen weit statt nur ein Kästchen sehen kann. Allgemein könnte die Wahrnehmung auf unterschiedliche Arten verändert werden. Z.B. könnte der Spieler unsichtbar oder blind werden, oder schon von weit weg die Position des Ausgangs oder der Schätze magisch erkennen.

Um das Spiel einfach zu gestalten, sammelt der Spieler Gegenstände automatisch auf, wenn er das Feld betritt, auf dem sie liegen. Wir sehen keine Gewichtsbeschränkung vor und auch keine Möglichkeit, die aufgenommenen Gegenstände wieder abzulegen.

Das einzige Benutzer-Kommando, das hinzukommt, ist "`i`" (für "inventory"), was die Besitztümer des Spielers auflistet (alle Gegenstände, die er bei sich trägt).

a) Schreiben Sie eine Klasse `Item` (für Gegenstände) mit folgender Schnittstelle:

- einem Konstruktor mit den Parametern
  - `String name` (Bezeichnung des Gegenstandes, z.B. für die Inventar-Liste des Spielers),
  - `String takeMsg` (Text, der beim Entdecken/Aufnehmen des Gegenstands ausgedruckt wird),
  - `int value` (Wert des Gegenstandes in Goldstücken), und
  - `boolean extraLight` (der Gegenstand gibt die erweiterte Sicht).
- Einer Methode `getName()`, die die Bezeichnung des Gegenstands liefert.

- Einer Methode `getTakeMsg()`, die den Entdeckungstext liefert.
- Einer Methode `getValue()`, die den Wert des Gegenstands liefert.
- Einer Methode `givesExtraLight()`, die den booleschen Wert für das "weite Sehen" liefert.

b) Erweitern Sie die Klasse `Player` um folgende Methoden:

- `void take(Item item)`:
  Der Gegenstand soll in der Inventar-Liste des Spielers gespeichert werden (und zwar am Ende angefügt werden).
- `int numItems()`:
  Diese Methode soll die aktuelle Anzahl Gegenstände in der Inventar-Liste liefern (d.h. die Länge der Liste).
- `Item getItem(int i)`:
  Diese Methode soll das `i`-te Item der Inventar-Liste liefern. Wie üblich, beginnt die Zählung bei `0`. Wenn `i` außerhalb des Bereiches ist, soll eine Exception des Typs `IndexOutOfBoundsException` erzeugt werden.
- `boolean has(Item item)`:
  Diese Methode liefert `true` genau dann, wenn der Spieler in seiner Inventar-Liste den Gegenstand `item` hat. Das kann z.B. verwendet werden, um zu prüfen, ob er den von einer Tür verlangten Schlüssel hat.
- `int gold()`:
  Diese Methode soll die Summe der `getValue()`-Werte der Gegenstände in der Inventarliste des Spielers liefern.
- `boolean hasExtraLight()`:
  Dies liefert `true`, wenn die Inventar-Liste des Spielers einen Gegenstand enthält, für den `givesExtraLight() true` ist. Wenn er keinen solchen Gegenstand besitzt, soll `false` geliefert werden.

Benutzen Sie zur Implementierung der Inventar-Liste die Klasse `ArrayList<Item>`:

`http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html`

c) Erweitern Sie die abstrakte Klasse "`Field`" um die Möglichkeit, einen Gegenstand auf dem Feld abzulegen. Zur Vereinfachung soll es nicht mehrere Gegenstände auf einem Feld geben. Es sind also die folgenden Methoden zu implementieren:

- `boolean canStoreItem()`:
  Kann die Software auf diesem Feld einen Gegenstand ablegen? Dies soll für Objekte der Subklasse `Path` wahr sein, für alle anderen falsch. Wenn man einen Gegenstand in der Wand ablegen würde, könnte ihn der Spieler ja nicht erreichen.
- `void storeItem(Item item)`:
  Legt den Gegenstand auf dem Feld ab. Es ist garantiert, dass dies nur aufgerufen

wird, wenn `canStoreItem()` wahr ist (der Feldtyp also passt), und `hasItem()` falsch ist (auf dem Feld also noch kein Gegenstand liegt). Wenn Sie wollen, können Sie in diesen Fehlerfällen passende Exceptions erzeugen (das wäre die saubere, "defensive" Lösung). Zur Vereinfachung können sich aber auch auf die Garantie verlassen.

- `boolean hasItem()`:
  Liefert `true` genau dann, wenn auf diesem Feld ein Gegenstand liegt. Diese Methode wird u.a. beim Zeichnen der Karte benötigt.
- `Item fetchItem()`:
  Dies liefert den Gegenstand auf diesem Feld bzw. `null`, falls kein Gegenstand auf dem Feld liegt. Das Feld soll anschließend leer sein (d.h. der Gegenstand wird fortgenommen).

Falls Sie Hausaufgabe 11 nicht gemacht haben, können Sie eine Lösung mit den Klassen `Field` und `Map` hier herunterladen:

`http://www.informatik.uni-halle.de/~brass/oop14/homework/Map.java`

d) Sorgen Sie dafür, dass man bei der Subklasse `Path` von `Field` mit `putItem(item)` wirklich einen Gegenstand ablegen kann.

e) Definieren Sie eine weitere Unterklasse "`Door`" der Klasse "`Field`". Während bei den bisherigen Unterklassen "`Wall`" und "`Path`" alle Objekte die gleichen Daten hatten, sollen die "`Door`"-Objekte individualisiert werden, so dass man verschiedene Türen mit verschiedenen Schlüsseln haben kann. Deswegen hat der Konstruktor folgende Parameter:

- `Item key`: Schlüssel, der zu dieser Tür passt.
- `String lockedMsg`: Rückgabewert von "`cantEnter(Player)`", falls der Spieler nicht den Schlüssel hat. Wenn er den Schlüssel hat, muss `cantEnter(Player)` `null` liefern (d.h. Spieler kann Feld betreten). Ein Beispiel für diesen Text wäre: "Da ist eine massive Eisentür, die fest verschlossen ist."
- `String unlockMsg`: Wenn der Spieler das erste Mal das Feld betritt, soll die Methode `enter(Player)` diesen Text liefern. Er könnte z.B. lauten: "Du benutzt den goldenen Schlüssel, um die Tür zu öffnen."
- `String passMsg`: Wenn der Spieler das Feld noch einmal betritt, nachdem die Tür schon geöffnet wurde, soll dieser Text ausgegeben werden, z.B.: "Du gehst durch die offene Eisentür."

Demgemäß müssen Sie natürlich auch die übrigen Methoden der Klasse überschreiben. Die Methode `isWall()` soll `false` liefern, die Methode `symbol()` soll das Zeichen "`+`" liefern.

f) Erweitern Sie die Klasse "`Map`" um eine Methode zur Erzeugung von Türen. Man muss hier ja die Daten für den Konstruktor übergeben.

- `void makeDoor(int x, int y, Item key, String lockedMsg, String unlockMsg, String passMsg)`:
  Dies soll das Feld mit den Koordinaten `(x,y)` auf ein neu erzeugtes Tür-Objekt mit den gegebenen Daten setzen. Ansonsten die Methode wie `makeField(x,y,t)` aus der letzten Hausaufgabe funktionieren.

Ein Rahmenprogramm, das die Karte anzeigt und die Bewegung des Spielers auf der Karte übernimmt, ist schon vorgegeben:

`http://www.informatik.uni-halle.de/~brass/oop14/homework/Maze2.java`

Wegen Beschränkungen der Übungsplattform schreiben Sie bitte Ihre Klassen unten in diese Datei. Wenn Sie diese Datei compiliert haben, können Sie das Spiel ausführen mit

```
java MazeGUI
```

Natürlich ist das Spiel im Vergleich zu den Vorbildern noch immer sehr einfach. Wenn Sie wollen, überlegen Sie sich, wie Sie zufällig eine Karte generieren können, so dass der "Dungeon" jedesmal neu ist. Natürlich könnte man auch Monster, Waffen, Rüstungen vorsehen, sowie Läden, um zu handeln, und natürlich mehr als einen "Level".

Übungsaufgabe 12A: (ohne Abgabe)

Bitte bearbeiten Sie auch diese Übungsaufgabe noch, aber geben Sie sie nicht ab. Es wird eine Lösung auf die Webseite der Vorlesung gestellt.

Was sind die Fehler oder Warnungen in dieser Klasse?

```
class GenType<T> {
        private static T s;

        private T attr;
        private T[] arr;

        GenType(T[] a) {
                this.arr = a;
        }

        public T m(T a) {
                T l = null;
                if(a instanceof Integer) {
                        System.out.println("Integer!");
                }
                this.attr = a;
        }

        public T[] f() {
                T[] a = new T[3];
                for(i = 0; i < 3; i++)
                        a[i] = new T();
                return a;
        }

        public void g(Object o) {
                this.attr = (T) o;
        }
}
```