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 11

Abgabe: 29.01.2015, 12:00 Übung: 02./03.02.2015

Hausaufgabe 11: (4 Theoriepunkte)

Ziel dieser Hausaufgabe ist ein kleines Rollenspiel des Rogue/Hack/Nethack/Larn-Typs. Sie brauchen allerdings nur einen (kleinen) Teil davon zu entwickeln.

In diesen Spielen sieht man eine Karte eines unterirdischen Höhlensystems von oben — allerdings nur so weit, wie man den "Dungeon" bisher erkundet hat. Die Position des Spielers ist mit "`@`" markiert. Da man keine besonders starke Lampe hat, kann man in unserer Variante nur die direkt angrenzenden Felder sehen. Man erinnert sich an alle Felder, die man schon einmal gesehen hat. So entfaltet sich nach und nach die gesamte Karte. In unserer Variante sind die Wände schwarz — bzw. dunkelgrau, wenn das Licht darauf fällt. Der begehbare Boden ist hellgrau bzw. weiß. Ziel ist es, den Ausgang "`<`" zu finden.

Die Steuerung der Spielfigur funktioniert mit den Cursor-Tasten oder alternativ den Buchstaben "h" (links), "j" (unten), "k" (hoch), "l" (rechts) oder auch den Buchstaben "a" (links), "s" (unten), "w" (oben), "d" (rechts).

a) Definieren Sie eine abstrakte Klasse "`Field`" für Positionen (Kästchen) auf dem Spielfeld mit folgenden Methoden:

- `boolean isKnown()`:
  Diese Methode soll `true` liefern, wenn der Spieler das Spielfeld schon einmal gesehen hat. Sonst `false`. Nach Erzeugung des Objektes wird `false` geliefert. Dies ändert sich erst durch Aufruf der folgenden Methode.
- `void makeKnown()`:
  Nach Aufruf dieser Methode soll `isKnown() true` liefern.
- `abstract String cantLeave(Player p)`:
  Diese Methode wird aufgerufen, wenn der Spieler das Spielfeld verlassen will. Wenn Sie etwas anderes als `null` liefert, geht das nicht. Der gelieferte String sollte erklären, warum es nicht geht. Z.B. könnte der Spieler an einem Dornbusch festhängen, und bei jedem Zug klappt es nur mit einer gewissen Wahrscheinlichkeit, sich wegzubewegen. Sie brauchen solche Fallen nicht zu programmieren — das Beispiel soll nur erklären, was der Zweck der Methode ist. Was genau passiert, hängt also vom Typ des Spielfeldes ab. Deswegen ist die Methode für allgemeine Spielfelder erstmal "`abstract`". Der Spieler wird als Parameter übergeben, weil ja eventuell Eigenschaften des Spielers (z.B., eine Rüstung zu tragen) relevant sein könnten für die Entscheidung. Allerdings ist die bisherige Klasse `Player` noch sehr einfach und enthält solche Eigenschaften nicht.

- `abstract String cantEnter(Player p)`:
  Wenn ein Spielfeld betreten werden soll, wird mit dieser Methode entscheiden, ob das möglich ist. Für Wände liefert sie z.B. "`Da ist eine Wand.`" Für Felder, die betreten werden können, liefert sie `null`.
- `String leave(Player p)`:
  Nachdem feststeht, dass der Zug ausgeführt werden kann (beide obige Methoden haben `null` geliefert), wird diese Methode aufgerufen. Meistens tut sie nichts und liefert die `null`-Referenz. So soll auch die Methode in dieser Klasse `Feld` definiert sein. Unterklassen haben aber die Möglichkeit, die Methode umzudefinieren. Sie könnte dann dem Spieler aber etwas Gutes oder Böses tun, und sollte eine Erklärung liefern (die von der Benutzerschnittstelle dann ausgedruckt wird).
- `String enter(Player p)`:
  Nach Aufruf von `leave(p)` für das alte Spielfeld wird `enter(p)` für das neue Spielfeld aufgerufen. Fallen könnten dem Spieler Gesundheitspunkte abziehen (z.B. durch Aufruf von `p.changeHP(-1)`). Auch dann sollte natürlich eine Erklärung als Text geliefert werden. Eine Teleportation ist auch möglich z.B. mit `p.setXY(5,7)`. Man gibt hierzu die Koordinaten des Spielfeldes an. Solche Spezialfelder müssen in dieser Aufgabe aber nicht entwickelt werden. Im Normalfall macht die Methode einfach nichts und liefert `null`. Hinterlegen Sie diese Implementierung in der abstrakten Oberklasse `Feld`, dann haben Sie bei den hier verlangten Unterklassen nichts mehr zu tun.
- `abstract boolean isWall()`:
  Diese Methode entscheidet über die Farbe, die für das Spielfeld verwendet wird: Falls sie `true` liefert, wird das Spielfeld dunkelgrau bzw. schwarz gezeichnet. Wenn sie `false` liefert (keine Wand), wird das Spielfeld weiß bzw. hellgrau gezeichnet.
- `char symbol()`:
  Um Spezialfelder unterscheiden zu können, kann diese Methode ein Zeichen liefert, das in das Feld gezeichnet wird. Die Version in der abstrakten Klasse `Field` soll ein Leerzeichen liefern. Dann wird das Feld einfach nur als Quadrat in der entsprechenden Farbe gezeichnet.

b) Definieren Sie nun eine (nicht abstrakte) Subklasse "`Wall`" für Wände (nicht betretbare Felder). Demgemäß müssen Sie die abstrakten Methoden folgendermaßen implementieren:

- `cantLeave` soll die Zeichenkette "`Du bist eingemauert.`" liefern. In unserem Spiel kommt das nicht vor, da man ein Wandfeld ja gerade nicht betreten kann. Vielleicht sorgen Sie so aber für missglückte Teleportationen vor.
- `cantEnter` soll die Zeichenkette "`Da ist eine Wand.`" liefern.

- `isWall` liefert `true`.

c) Definieren Sie nun noch eine (nicht abstrakte) Subklasse “`Path`” für Wege (betretbare Felder). Demgemäß müssen Sie die abstrakten Methoden folgendermaßen implementieren:

- `cantLeave` liefert `null`.
- `cantEnter` liefert `null`.
- `isWall` liefert `false`.

d) Definieren Sie außerdem eine Klasse `Map` für die Karte. Dies ist im wesentlichen eine Matrix von Objekten der Klasse `Field`. Implementieren Sie die folgenden Methoden:

- Einen Konstruktor mit zwei Parametern `sizeX` und `sizeY` für die Anzahl Felder (Kästchen) in X- und Y-Richtung. Beide Parameter haben natürlich den Typ `int`. Nach Aufruf des Konstruktors sollen alle Felder auf die Null-Referenz initialisiert sein.
- `int getSizeX()`:
  Liefert die Anzahl Felder in X-Richtung (also den Wert, der dem Konstruktor übergeben wurde).
- `int getSizeY()`:
  Entsprechend für die Y-Richtung.
- `void makeField(int x, int y, char type)`:
  Dies soll das Feld mit den Koordinaten `(x,y)` auf ein neu erzeugtes Objekt setzen. Falls als `type` das Zeichen `'#'` übergeben wird, soll das Objekt der Klasse `Wall` angehören. Falls als `type` eines der Zeichen `'.'`, `'@'`, oder `'<'` übergeben wird, soll das Objekt der Klasse `Path` angehören. Es steht Ihnen frei, noch weitere Unterklassen von `Field` zu definieren, und dann hier entsprechend für andere Typ-Codes Felder dieser Unterklassen zu erzeugen.

  Ihre Methode muss prüfen, ob die Koordinaten gültig sind, also z.B. $\texttt{x} \geq 0$ und $\texttt{x} < \texttt{getSizeX()}$. Ist das nicht der Fall, soll eine `IllegalArgumentException` ausgelöst werden mit dem erklärenden Text (“`message`”) “`X-Wert ungueltig:`” und dann den entsprechenden Wert. Die gleiche Art von Exception soll erzeugt werden, wenn der `type`-Wert ungültig ist. In diesem Fall soll die `message` der Exception sein: “`Unbekannter Feld-Typ:`” (auch mit dem `type`-Wert).
- `Field getField(int x, int y)`:
  Dies soll das Feld an der entsprechenden Position in der Matrix liefern. Wieder sind Exceptions des Typs `IllegalArgumentException` zu erzeugen, wenn die Koordinaten ungültig sind.

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/Maze.java`

Ihre Klassen schreiben Sie bitte nicht in diese Datei, sondern in eine Datei “`Map.java`” (alle vier Klassen in diese Datei). Nur diese Datei geben Sie ab. Wenn Sie das Rahmenprogramm und Ihre Datei compiliert haben, können Sie das Spiel ausführen mit

```
java MazeGUI
```

Natürlich ist das Spiel im Vergleich zu den Vorbildern noch sehr einfach. Wir haben noch keine Monster, Waffen, Schätze, Zauber. Und auch nur einen einzigen Level (in den Vorbildern gibt es viele “Stockwerke” im Dungeon). Bessere Programme erzeugen auch jedes Mal eine neue Karte. Aber ein Anfang ist gemacht. Wenn Sie Spass daran haben, können Sie das Programm beliebige modifizieren und erweitern.

Falls Ihnen die zu implementierenden Methoden noch nicht ganz klar sind, könnte es helfen, ihre Verwendung in folgendem Programmstück anzuschauen:

```
// Ausfuehrung des Spielzugs (Bewegung):
void go(int stepX, int stepY) {
        // Ausgabefeld loeschen:
        this.output.clear();

        // Laeuft das Spiel noch?
        if(this.gameWon) {
                println("Das Spiel ist gewonnen. " +
                        "Bitte Beenden oder Neustart.");
                return;
        }
        if(this.gameLost) {
                println("Das Spiel ist verloren. " +
                        "Bitte Beenden oder Neustart.");
                return;
        }

        // Jetzt zaehlt es als Zug:
        this.numMoves++;

        // Aktuelle Position des Spielers:
        int playerX = this.player.getX();
        int playerY = this.player.getY();
        Field currField = this.map.getField(playerX, playerY);

        // Kann man das aktuelle Feld verlassen?
        String cantLeave = currField.cantLeave(this.player);
        if(cantLeave != null) {
                println(cantLeave);
                return;
        }
```

```java

    // Grenzen des Spielfeldes pruefen:
    int nextX = playerX + stepX;
    int nextY = playerY + stepY;
    if(nextX < 0 || nextX >= this.sizeX ||
            nextY < 0 || nextY >= sizeY) {
        // Karte sollte eigentlich aussen Mauer haben.
        println("Da␣ist␣nur␣g\u00e4hnende␣Leere␣" +
                "-␣das␣Nichts");
        return;
    }

    // Kann man das gewuenschte Feld betreten?
    Field nextField = this.map.getField(nextX, nextY);
    String cantEnter = nextField.cantEnter(this.player);
    if(cantEnter != null) {
        println(cantEnter);
        return;
    }

    // Nun Aktion durchfuehren:
    String leaveMessage = currField.leave(this.player);
    if(leaveMessage != null)
        println(leaveMessage);
    this.player.setXY(nextX, nextY);
    String enterMessage = nextField.enter(this.player);
    if(enterMessage != null)
        println(enterMessage);

    // Lebt der Spieler noch?
    if(this.player.getHP() <= 0) {
        println("Leider␣sind␣die␣Gesundheitspunkte␣" +
                "verbraucht.␣Game␣over!");
        this.gameLost = true;
        return;
    }

    // Spieler wurde ggf. teleportiert:
    playerX = this.player.getX();
    playerY = this.player.getY();

    // Ist der Spieler im Ziel?
    if(playerX == this.goalX && playerY == this.goalY) {
        println("Herzlichen␣Gl\u00FCckwunsch!␣" +
            "Sie␣haben␣den␣Ausgang␣gefunden.");
        this.gameWon = true;
    }
}
```


Übungsaufgabe 11A: (ohne Abgabe)

Diese Aufgabe wird in der Übung besprochen. Ggf. müssen Sie Ihre Lösung vorführen.

Was gibt das folgende Programm aus?

```java
class ExceptionTest {
    private static java.util.Scanner s;

    public static void main(String[] args) {
        System.out.print("A");
        try {
            int n = s.nextInt();
        } catch(Exception e) {
            System.out.println(e);
        }
        System.out.print("B");
        try {
            h();
        } catch(Exception e) {
            System.out.println(e);
        }
    }

    static void h() {
        System.out.print("C");
        try {
            g();
            System.out.print("D");
        } catch(IndexOutOfBoundsException e) {
            System.out.print("E");
        } catch(RuntimeException e) {
            System.out.print("F");
        }
        System.out.print("G");
    }

    static void g() {
        System.out.print("H");
        f();
        System.out.print("I");
    }

    static void f() {
        System.out.print("J");
        int[] a = new int[3];
        a[3] = 27;
        System.out.print("K");
    }
}
```