ZORK

**Ein einfaches Adventure-Game für die Stufe EF,
entwickelt von U. Helmich, inspiriert durch viele bekannte Spiele.**

Zielsetzung in Stichpunkten

Ein Held soll durch einen Dungeon laufen, der hauptsächlich aus Bäumen und schmalen Wegen besteht. Auf den Wegen lauern Monster, mit denen der Held kämpfen muss. Aber auch wertvolle Gegenstände liegen auf den Wegen, die der Held aufsammeln kann. Diese Gegenstände verbessern die Eigenschaften des Helden, so dass er größere Chancen hat, die Kämpfe mit den Monstern zu bestehen.

Ziel des Spiels ist es, lebend einen Zielpunkt zu erreichen, und zwar in möglichst kurzer Zeit und mit möglichst vielen Lebenspunkten.

Weiterentwicklung

Wenn das Spiel so weit programmiert ist wie oben beschrieben, kann es von jeder Gruppe nach eigenem Geschmack weiterentwickelt werden. Hier werden keine Vorgaben gemacht, außer natürlich, dass die Weiterentwicklung auf den gemeinsam erarbeiteten Versionen beruht.

Inhalt

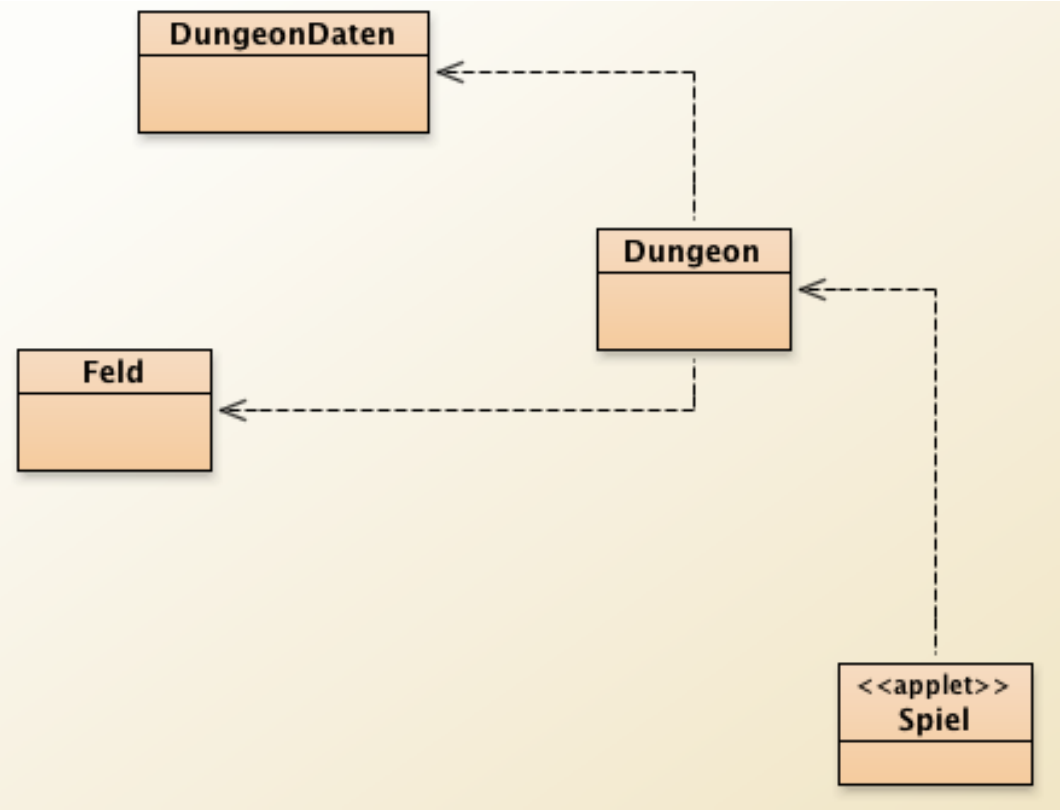
Erste Überlegungen

Welche Klassen benötigt die erste Version des Spiels?

Die wichtigste Klasse ist **Dungeon**, welche die Hauptarbeit übernimmt. **Dungeon** greift auf zwei untergeordnete Klassen zu, nämlich **DungeonDaten** und **Feld**. In **DungeonDaten** sind - wie der Name schon verrät - die eigentlichen Daten des Dungeons gespeichert, zum Beispiel welche Wege wo verlaufen sowie Start- und Zielpunkt des Helden. In **Feld** sind die Eigenschaften eines einzelnen Feldes des Spiels gespeichert, also Typ (Wald, Weg, Start, Ziel, Geheimtür etc.). Das Ganze wird schließlich von einem Java-Applet **Spiel** gezeichnet. In **Spiel** werden später auch die Bedienelemente des Programms untergebracht, also Buttons wie in Folge 6 (Roboter) oder eine Tastatursteuerung (neu).

Klassendiagramme

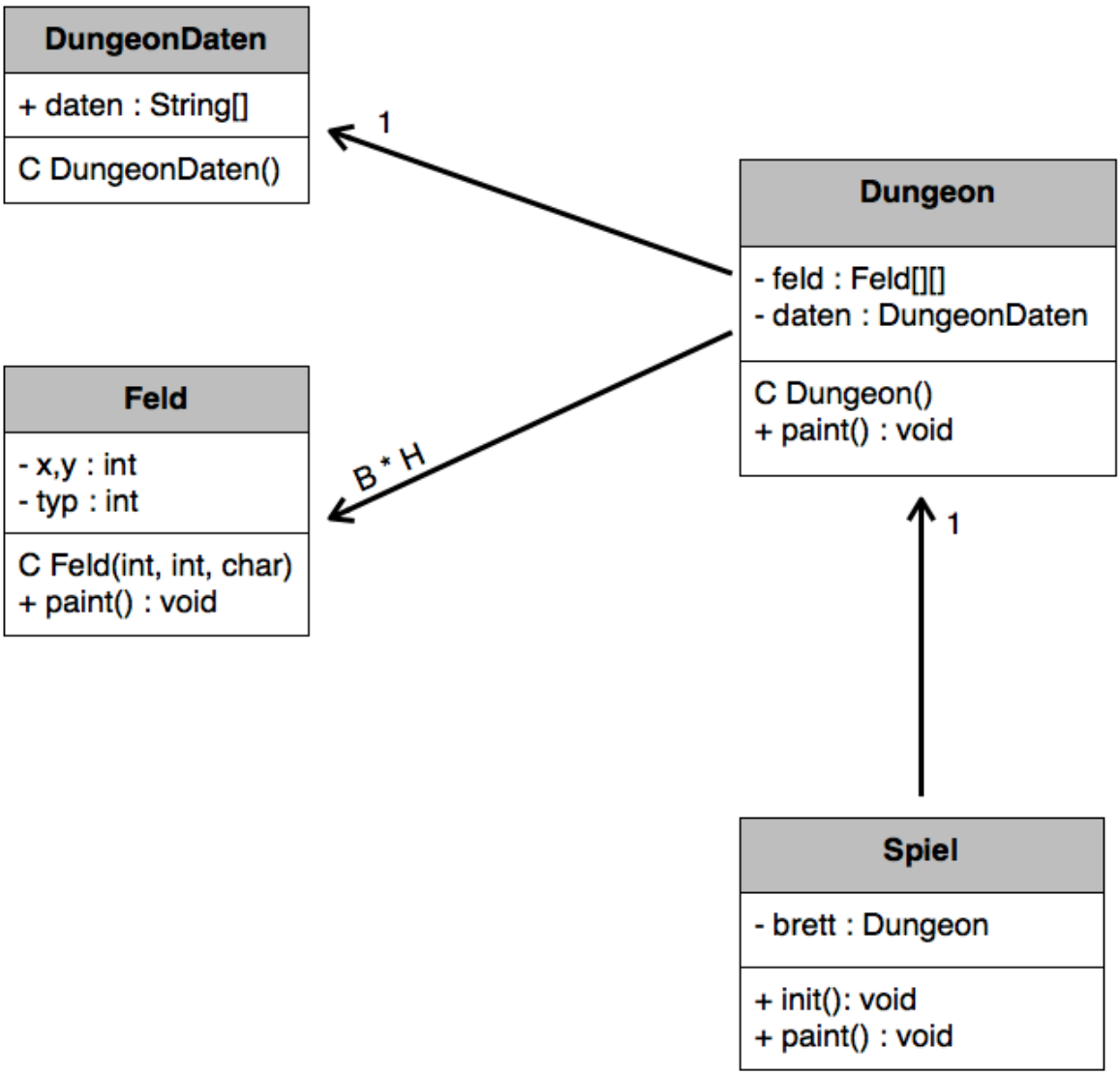
Graphische Darstellung der Klassen in BlueJ



Hier sieht man schön, wie das Applet, also die Klasse **Spiel**, auf die Klasse **Dungeon** zurückgreift. **Dungeon** wiederum hat Objekte der Klassen **Feld** und **DungeonDaten**.

Darstellung als UML-Klassendiagramm

In der Sprache UML (Unified modeling language), die zur Beschreibung komplexer Zusammenhänge (wie zum Beispiel Beziehungen zwischen Klassen) dient, sieht das Ganze in etwa so aus (UML lässt einem ziemlich viele Freiheiten bei der graphischen Darstellung).



Diese Graphik sieht im Prinzip so aus wie das BlueJ-Klassendiagramm. In der UML-Darstellung sind allerdings mehr Details zu erkennen. Für jede Klasse werden die wichtigsten Attribute und die wichtigsten Methoden angezeigt, außerdem steht an den Pfeilen die Anzahl der Objekte, die von einer Klasse benötigt werden.

So hat die Klasse **Dungeon** genau ein Objekt der Klasse **DungeonDaten**, und die Klasse **Spiel** wiederum hat genau ein Objekt der Klasse **Dungeon**. Die Klasse **Dungeon** hat aber viele Objekte der Klasse **Feld**, nämlich B * H (= Breite * Höhe).

Für ein Schachbrett wären das 8 * 8 Felder, also 64. Unser Dungeon ist sicherlich wesentlich größer.

In der Modellierungssprache UML wird eine Klasse als dreigeteilter Kasten dargestellt. In das oberste "Fach" schreibt man den Klassennamen. In das zweite "Fach" die wichtigsten Attribute und in das dritte "Fach" die wesentichen Methoden. Unwichtige Attribute oder Methoden, die gerade keine Rolle spielen, kann man einfach weglassen.

Ein Minuszeichen vor einem Attribut oder einer Methode ist als "private" zu lesen, ein Pluszeichen als "public". Das "C" steht für "Constructor".

Benötigt eine Methode Parameter, so können diese angegeben werden, allerdings nicht die Namen der Parameter, sondern nur die Datentypen, zum Beispiel "int, int, char". Man kann die Parameter aber auch weglassen. Bei den paint()-Methoden fehlt zum Beispiel der **Graphics**-Parameter, weil dies bei einer paint()-Methode selbstverständlich ist. Man hätte diesen Parameter aber genauso gut auch hinschreiben können, UML ist da sehr offen.

Die Syntax ist hier etwas anders als in Java, schließlich ist UML unabhängig von einer Programmiersprache und muss für alle Programmiersprachen gelten.

Java-Deklaration von zwei int-Attribute:

```
private int x,y;
```

UML-Deklaration dieser beiden Attribute:

```
- x,y : int
```

Hinweis für die Abiturienten (3. Fach)
Kenntnisse von UML werden im Zentralabitur verlangt!

Aufgaben

Aufgabe 1

Bringen Sie die erste Version des ZORK-Spiels zum Laufen!

Aufgabe 2

Analysieren Sie die Quelltexte der ersten Version und geben Sie dann an, auf welche Weise eigentlich die Dungeon-Daten in die Klasse **Dungeon** eingelesen werden.

Experten-Aufgabe

Man könnte die Dungeon-Daten auch aus einer **Textdatei** einlesen. "Bauen" Sie die Klasse **DungeonDaten** entsprechend um. An der Schnittstelle der Klasse (öffentliche Methoden etc.) darf sich allerdings nichts ändern, so dass die neue Version von **DungeonDaten** unverändert von den anderen Klassen benutzt werden kann.

Bei einer Experten-Aufgabe wird Ihnen übrigens nicht vom Lehrer geholfen; Sie müssen damit alleine zurecht kommen!

Aufgabe 3

Erläutern Sie, warum der Feld-Typ dem Konstruktor der Klasse **Feld** als char übergeben wird und nicht als int.

Aufgabe 4

Erläutern Sie folgenden Befehl in dem Konstruktor der Klasse **Dungeon**:

```
feld[x][y] = new Feld(x,y,daten.daten[y].charAt(x));
```

Wie würde sich dieser Befehl ändern, wenn das Attribut **daten** in der Klasse **DungeonDaten** nicht **public** wäre, sondern **private**?