Objektorientierte Programmierung WS 2013/2014 - Aufgabenblatt 4

Prof. Dr. U. Kastens
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn

Ausgabe: 26.11.2013

Aufgabe 1 (Das Composite Pattern)

Auf Folie 208 der Vorlesung wird das Design Pattern "Composite" vorgestellt, mit dem zusammengesetzte rekursive Strukturen modelliert werden können. Dieses Entwurfsmuster soll für eine Datenstruktur für Dateisysteme angewandt werden:

- Ein **Dateisystemeintrag** ist entweder eine **Datei** oder ein **Ordner**. Er hat einen Namen.
- Ein **Ordner** kann beliebig viele Dateisystemeinträge enthalten.

Schreiben Sie die Klassen `Datei`, `Ordner` und `DateiSystemEintrag` nach dem Composite-Muster. Zum Testen Ihrer Klassen dient das Hauptprogramm `VerzeichnisBaum`:

```
import java.io.File;

public class VerzeichnisBaum
{
   private static DateiSystemEintrag durchlaufen (File f)
   {
      if (f.isDirectory())
      {
         // es ist ein Ordner
         Ordner res = new Ordner(f.getName());
         // der Ordnerinhalt als Array von Strings
         String[] ordnerinhalt = f.list();

         // rekursives Durchlaufen des Ordnerinhaltes
         for (int i = 0; i < ordnerinhalt.length; i++)
            res.hinzufuegen(durchlaufen(new File(f.getPath() + "/"
                                                 + ordnerinhalt [i])));

         return res;
      }
      else
      {
         // es ist eine gewoehnliche Datei
         Datei res = new Datei(f.getName());
         return res;
      }
   }

   public static void main (String argv [])
   {
      // Die Wurzel fuer unseren Verzeichnisbaum
      String wurzel = argv [0];
      // Durchlaufen des gesamten Verzeichnisbaums
      DateiSystemEintrag e = durchlaufen(new File(wurzel));

      // Ausgabe des Verzeichnisbaums
      e.druckeVerzeichnisBaum(""); // Der String-Parameter ist die Einrueckung
   }
}
```

Es liest einen Verzeichnisbaum eines Verzeichnisses, dessen Name auf der Kommandozeile übergeben wird, und produziert eine formatierte Auflistung des Inhaltes. Für den beigefügten `TestOrdner` liefert des Programm folgende Ausgabe:

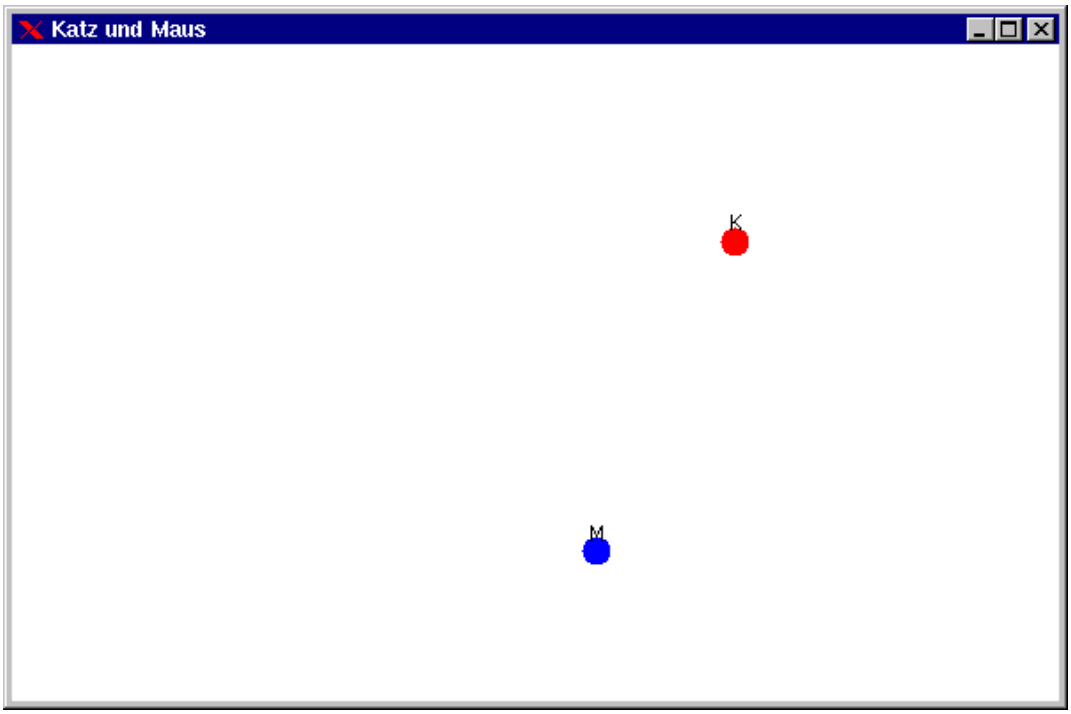
```
TestOrdner
   Staedte
      Bielefeld
      Paderborn
   Laender
      Belgien
      Niederlande
      Luxemburg
   Fluesse
      Amerika
         Orinoko
         Mississippi
      Europa
         Saar
```

Die Quelldatei des Hauptprogramms und das TestOrdner-Verzeichnis findet sich in

- `blatt4/verzbaum`

Aufgabe 2 (Rollen)

Ein Beispiel für die Implementierung von Rollen durch Delegation an Rollenobjekte (siehe Folie 215) ist das folgende Spiel: Auf einer leeren Spielfläche bewegen sich zwei Bälle. Der blaue Ball spielt die Rolle der Maus, die von dem roten Ball, der die Rolle der Katze spielt, gejagt wird. Die Maus wird dabei vom Benutzer mit den Pfeiltasten gesteuert, die Katze wird vom Rechner gespielt.



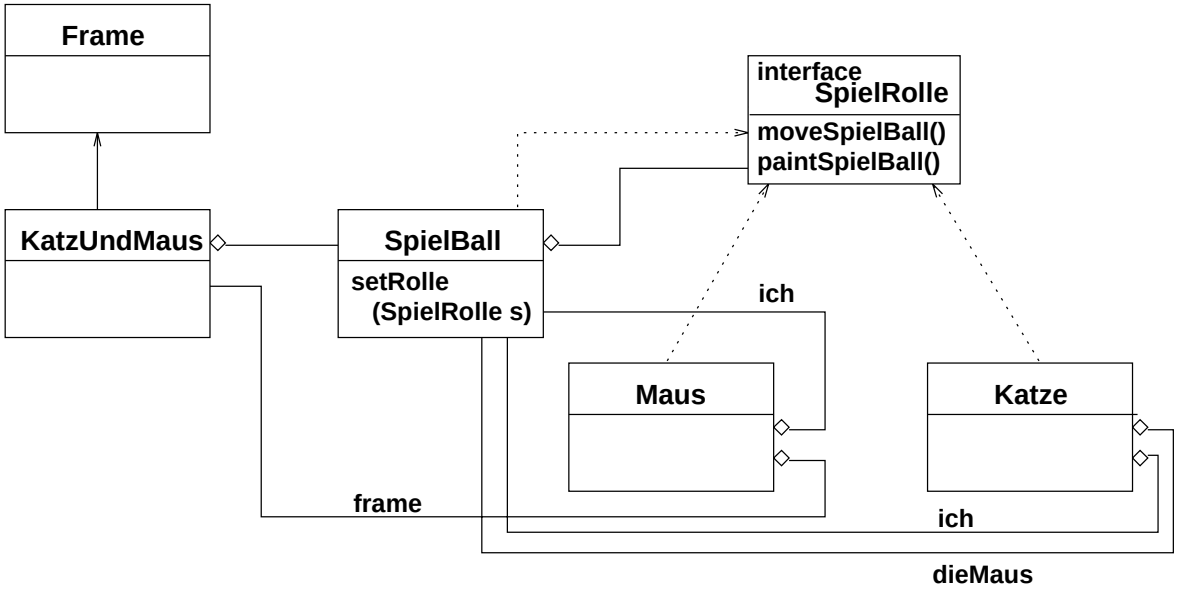
Die Quellen des KatzUndMaus-Spiels finden Sie in `blatt4/KatzUndMaus`

a) Geben Sie für die beiden Aufrufe

```
ball1.moveSpielBall();
ball2.moveSpielBall();
```

aus der Klasse `KatzUndMaus` jeweils die Aufrufkette an. Eine Aufrufkette ist eine Folge von Methodenaufrufen, die von einem Aufruf ausgelöst werden. In unserem Fall betrachten wir nur Methoden aus den 6 Klassen unserer Anwendung und keine Bibliotheksklassen.

b) Die vorliegende Ausprägung des Delegations-Schemas von Folie 215 sieht wie folgt aus:



Zeigen Sie, dass sich dieses Schema auch für dynamisch wechselnde Rollen eignet. Nachdem die Katze die Maus vollständig gefressen hat (d.h. die beiden Bälle haben die gleiche Position) sollen die Rollen getauscht werden. Der Spielball mit der Katzenrolle wird zur Maus und umgekehrt. Die so mutierten Bälle kehren zu ihrer Anfangsposition zurück und das Spiel beginnt mit der neuen Rollenverteilung von vorne.

Aufgabe 3 (Spezialisierung)

Wir betrachten eine Spezialisierung der Bibliotheksklasse `TreeMap`:

```
import java.util.TreeMap;

public class WordCount extends TreeMap<String, Integer> {
     public void countMe(String s) {
         // ergänzen
     }

     public void stat() {
         System.out.print(size() + " distinct words: ");
         System.out.println(this);
     }

     public static void main(String[] args) {
           WordCount wl = new WordCount();

           for (String a : args) {
                wl.countMe(a.toUpperCase());
           }
           wl.stat();
        }
}
```

`WordCount` erhält eine Folge von Strings auf der Kommandozeile und soll zählen, wie viele verschiedene Wörter wie oft vorkommen.

Beispiel:

```
java WordCount To be or not to be
```

liefert

```
4 distinct words: {BE=2, NOT=1, OR=1, TO=2}
```

a) Ergänzen Sie die fehlende Methode `countMe` entsprechend.
b) "Has-a statt Is-a": Transformieren Sie das Erben von `TreeMap` in eine `TreeMap`-Komponente in `WordCount`.