Persistenz

Eine Lösung des Persistenzproblems, also der Sicherung des OO Zustandes des Systems, liefert JAVA in den Klassenbibliotheken mit. Bruce Eckel beschreibt sie in seinem Buch[1] u.a. am Beispiel eines (sehr!) einfachen CAD – Systems.
Man gibt den zu sichernden Objekten im Kopf der Klassendefinition einfach die Erweiterung **`implements Serializable`** mit. Serializable ist also ein interface. Dabei ist es nicht einmal notwendig, irgendeine Methode zu implementieren.

Sichern und Wiederherstellen der Objekte

Das Sichern und Wiederherstellen der Objekte erfolgt dann mit den entsprechenden Ein – Ausgabe – Klassen, die man auch unabhängig vom speziellen Persistenzproblem nutzen müsste. Dabei ist der entscheidende Punkt, dass sie auch Klassen zum Schreiben und Lesen von Objekten enthalten. Sie heißen ObjectOutputStream – zum Schreiben – und ObjectInputStream zum Lesen. In der Javadoc heißt es :

`An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.`
`An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.`

Bemerkenswert daran ist die Ergänzung im ersten Satz , denn sie bedeutet, dass automatisch auch die abhängigen Objekte mit gesichert werden und daher auch wiederhergestellt werden können[2].

Beteiligte Klassen müssen bekannt sein

Weiterhin bemerkenswert an dem Mechanismus ist auch, dass eine Sicherung auch über das Netz erfolgen kann. Für eine Wiederherstellung des Zustandes ist dabei nur wichtig, dass die beteiligten Klassen in der wiederherstellenden Umgebung bekannt sind.
Da dies ein ernstes Problem ist, verlangt Java, dass der dabei entstehende Fehler **`ClassNotFoundException`** abgefangen wird oder als abzufangen gekennzeichnet wird, indem man im Kopf der Klasse **`throws ClassNotFoundException`** angibt. In unserer BlueJ – Umgebung fängt das System die Exceptions ab, so dass wir ein Projekt auch ohne ausdrückliche Behandlung der Exceptions übersetzen und starten können.
Vergleichbare Probleme ergeben die IOException und die FileNotFoundException. Letztere sollte man tunlichst selbst abfangen.

File

Wie der Zugriff auf Dateien unter Java geregelt ist, beschreibt Bruce Eckel in seinem Buch ausführlich und geht dabei auch auf die Probleme u.a. der Namensgebung der Klassen ein. Das mag man sich durchlesen, mir scheint nur wichtig zu sein, dass den ObjectOutputStream bzw. ObjectInputStream – Objekten im Konstruktor jeweils mit **`new FileOutputStream(<`***`dateiname`***`>)`** / **`new FileInputStream(<`***`dateiname`***`>)`** entsprechende Objekte übergeben werden müssen.

---

1 Bruce Eckel: „Thinking In Java“, Prentice Hall. CAD – System ab S. 633

2 Dies ist anders bei den weiteren Möglichkeiten Persistenz herzustellen. Mit Externalizable lässt sich die Sicherung über die beiden zu implementierenden Methoden writeExternal() und readExternal() kontrollieren.
Eine weitere Möglichkeit der Kontrolle hat man auch bei Serializable, wenn man einzelne Attribute, etwa ein gespeichertes Passwort, von der Sicherung ausschließen will. Man kennzeichnet das Attribut dann als transient.

Die Datei

Liest man die Datei mit den gesicherten Daten mit einem einfachen Editor, erkennt man die Prinzipien wieder, die man selbst auch bei der Sicherung verwenden würde. Klassennamen und Attributnamen sind zu finden und irgendwo verstecken sich die Werte.

Programm

Für das Auslösen der Sicherung bzw. der Wiederherstellung ist natürlich eine eigene Klasse[1] notwendig.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.EOFException;

public class Moeblierung{

    ArrayList liste;

    // Konstruktor für Objekte der Klasse Moeblierung
    public Moeblierung()
    { liste = new ArrayList(); }

    /**
     * sichert und löscht die Darstellung.
     */
    public void sichernUndLoeschen()throws IOException{
        sichern();
        loescheAlle();
        liste = new ArrayList();
    }

    /**
     * holt und stellt alle dar.
     */
    public void holenUndZeigen()
           throws ClassNotFoundException, IOException{
        holen();
        zeigeAlle();
    }

    /**
     * sichern
     */
    private void sichern()
    throws IOException{
        try{
            ObjectOutputStream ausgabe= new ObjectOutputStream(
```

einfacher mit import java.io.*; aber nicht klarer.

Exceptions müssen an den Aufrufer weiter gereicht werden

1 Hier zeigt sich wieder ein Nachteil des Entwurfes mit der Klasse Leinwand, die selbst die Controllerfunktion für die Shapes hält. Will man eine akzeptable Möbelanwendung realisieren, braucht man die Controllerfunktion für die Möbelobjekte in einer weiteren Klasse. Ein redesign mit einer neuen Grafik ist fast zwingend.

```
                new FileOutputStream("ausgabe.moe"));
            for (Iterator it= liste.iterator();it.hasNext();){
                ausgabe.writeObject(it.next());
            }
            ausgabe.close();
        } catch (FileNotFoundException e){
                System.out.println("Datei existiert nicht!"); };
    }

    /**
     * holen
     */
    private void holen()
    throws ClassNotFoundException, IOException{
        try{
            Object o=null;
            liste = new ArrayList();
            ObjectInputStream eingabe= new ObjectInputStream(
                    new FileInputStream("ausgabe.moe"));
            boolean fertig = false;
            while (!fertig){
                try{
                    o = eingabe.readObject();
                } catch (EOFException eofe) {fertig=true;};
                if (fertig)
                eingabe.close();
                else liste.add(o);
            }
        } catch (FileNotFoundException e){
                    System.out.println("Datei existiert nicht!"); };
    }

    /**
     * erzeugt einen Stuhl und zeigt ihn.
     */
    public void erzeugeStuhl(){
        Stuhl stuhl=new Stuhl();
        liste.add(stuhl);
        stuhl.zeige();
    }

    /**
     * erzeugt einen Tisch und zeigt ihn.
     */
    public void erzeugeTisch(){
        Tisch tisch=new Tisch();
        liste.add(tisch);
        tisch.zeige();
    }

    /**
     * zeigt die gesamte Möbelliste an.
     */
    private void zeigeAlle(){
        Moebel moebel;
        for (Iterator it= liste.iterator();it.hasNext();){
            moebel= (Moebel)it.next();
            moebel.zeige();
        }
```

```
    }

    /**
     * löscht die dargestellte Möbelliste.
     */
    private void loescheAlle(){
        Moebel moebel;
        for (Iterator it= liste.iterator();it.hasNext();){
            moebel= (Moebel)it.next();
            moebel.verberge();
        }
    }
}
```