Persistente Objekte
— Der Elchtest für ein Java-Programm –

Zusammenfassung:
Ein Java-Programm besteht aus Objekten. Ein Objekt hat einen passiven Teil, genannt Attribute, Variablen oder Slots und einen aktivierbaren Teil, genannt Methoden oder Funktionen.

Ein solches Objekt ist persistent (≈ dauerhaft), wenn es unabhängig vom Ort und der Lebensdauer seines erzeugenden Java-Programms existiert. Diese Art von Persistenz versprechen objekt-orientierte Datenbankmanagementsysteme (OODBMS).

Am OODBMS-Beispiel POET[1] werden die tatsächlich realisierbaren Leistungen „provokativ" hinterfragt. Wie beim berühmt berüchtigten Elchtest, keiner wollte es glauben, aber er fiel trotzdem um, so wird in Form eines Beispiels gezeigt, daß gar keine Objekte (passiver + aktiver Teil) „gemanaged" werden. Unstrittig, fahren kann man mit dem Testling, nur eben überraschend anders!

Inhaltsverzeichnis

[1] POET Software GmbH, D-22359 Hamburg, Version 5.1, `http://www.poet.de`.

1 Objektwelt: Buch ⟶ Autor ⟶▷ Person

Im objekt-orientierten Paradigma[2] bleibt ein „Objekt“ der realen (oder erdachten) Welt stets erhalten. Es ist über die verschiedenen Abstraktionsebenen[3] leicht verfolgbar. Das gewachsene Verständnis über die Objekte der realen Welt soll die Durchschaubarkeit des Modells gewährleisten.

Als Beispielobjekte dienen hier Bücher, die stets eine natürliche Person als Autor haben. Aus der Sicht der Basismaschine (≈ Java-Programm) umfaßt ein solches Buchobjekt einen passiven und einen aktiv(ierbar)en Teil. Der passive Teil enthält beispielsweise ein Attribute wie den Buchtitel oder das Erscheinungsjahr des Buches. Der aktiv(ierbar)e Teil wird durch „Anfragen“ (Nachrichten), die man dem Objekt schicken kann und die von diesem beantwortet werden, gebildet. Eine „Anfrage“ entspricht der Applikation einer Methode im Kontext des jeweiligen Objektes. Man kann daher das Buch `meineFibel` fragen: Wie alt bist Du? Das entspricht der Applikation der Methode `ermittleAlter()` beim Buchobjekt `meineFibel`. Formuliert in Java-Notation:

```
meineFibel.emittleAlter();
```

Für die folgenden Betrachtungen halten wir fest:

Objekt ≡ passiver Teil + aktivierbarer Teil

mit:		
	passiver Teil	≡ Attribute
	aktivierbarer Teil	≡ Methoden

1.1 Definition einer Objekt(spiel)welt

Der Testling für den „Elchtest[4]“ ist eine sehr einfache *Java-Application* zur Verwaltung von Büchern. In dieser Objekt(spiel)welt wird ein Buch mit folgenden Attributen[5] beschrieben: Buch

- `titel`

[2] Ein Paradigma ist ein von der wissenschaftlichen Fachwelt als Grundlage der weiteren Arbeiten anerkanntes Erklärungsmodell, eine forschungsleitende Theorie.

[3] von der Benutzermaschine (≈ Anforderungen) zur Basismaschine (≈ Java-Programm)

[4] Ein von der Daimler AG, Stuttgart produziertes Automobil mit der Bezeichnung A-Klasse fiel beim Durchfahren einer Slalomstrecke um. Die Slalomstrecke sollte das Ausweichen eines Elches, der plötzlich die Straße überquert, realitätsnah abbilden.

[5] Die Attribute sind stets mit den Bezeichnern entsprechend dem Java-Quellcode genannt.

- isbn
- erscheinungsJahr
- hauptAutor
- schlagWoerter
- alter
- verlagsKurzName

Der Wert für das Attribut `hauptAutor` ist ein Objekt der Klasse `Autor`. Ein Autor selbst hat hier vereinfacht nur folgende Attribute: Autor

- themen
- orgKurzName

Beim Testling ist ein Autor stets eine natürliche Person. Sie ist hier mit den folgenden Attributen beschrieben: Person

- zuName
- vorNamen

Zugriffe auf diese Attribute sind nur über eigene Zugriffsmethoden, den sogenannten „`get`-Methoden“ (Projektsprache: Getter) zugelassen. Auch die Werte der Attribute sind nur über entsprechende Zustandsmodifikatoren möglich. Es sind die sogenannten „`set`-Methoden“ (Projektsprache: Setter). Die Attribute, Methoden und Beziehungen zwischen den Klassen `Buch`, `Autor` und `Person` zeigt die Abbildung[6] 1 auf Seite 4. Dabei wird die UML-Notation[7] (*Unified Modeling Language*) genutzt. UML

Die Assoziation zwischen der Klasse `Buch` und der Klasse `Autor` ist daher als einfacher Pfeil (⟶) dargestellt. In Java abgebildet wird diese Assoziation über die Klassenangabe `Autor` für das Attribut `hauptAutor`. Die Vererbungsbeziehung zwischen der Klasse `Person` und Autor ist als Pfeil mit Dreiecksspitze (—▷) dargestellt. Dabei zeigt die Spitze in Richtung der „Elternklasse“. Im folgenden ist der Java-Code für die Klassen `Buch` (1.1.1 auf Seite 6), `Autor` (1.1.2 auf Seite 8) und `Person` (1.1.3 auf Seite 8) aufgelistet.[8] ⟶ —▷

[6] UML Version 1.0 — Bild erzeugt mittels Werkzeug Innovator Version 6.0 der Softwarefirma MID GmbH, Nürnberg. Einen Gesamtüberblick über diese Objekt(spiel)welt im Werkzeug Innovator gibt Abbildung 2 auf Seite 5.

[7] Näheres zur Modelierung mit UML im Java-Kontext siehe beispielsweise [Bo98] .

[8] Die Zeilennummern sind nicht Bestandteil des Java-Quellcodes. Sie dienen hier nur zur leichteren Bezugnahme beim Erläutern.

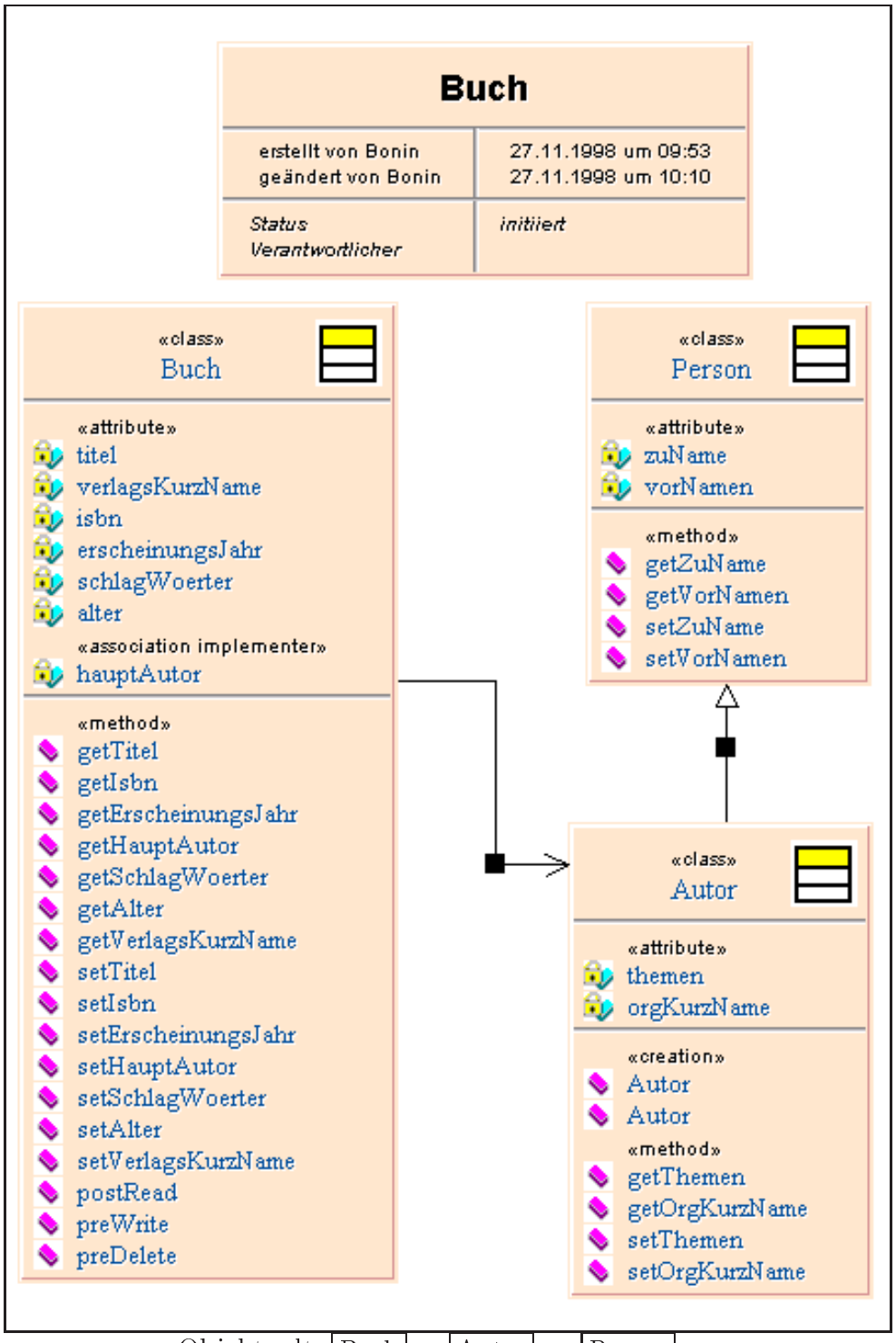


Abbildung 1: Klassendiagramm in *Unified Modeling Language* (UML-Notation)

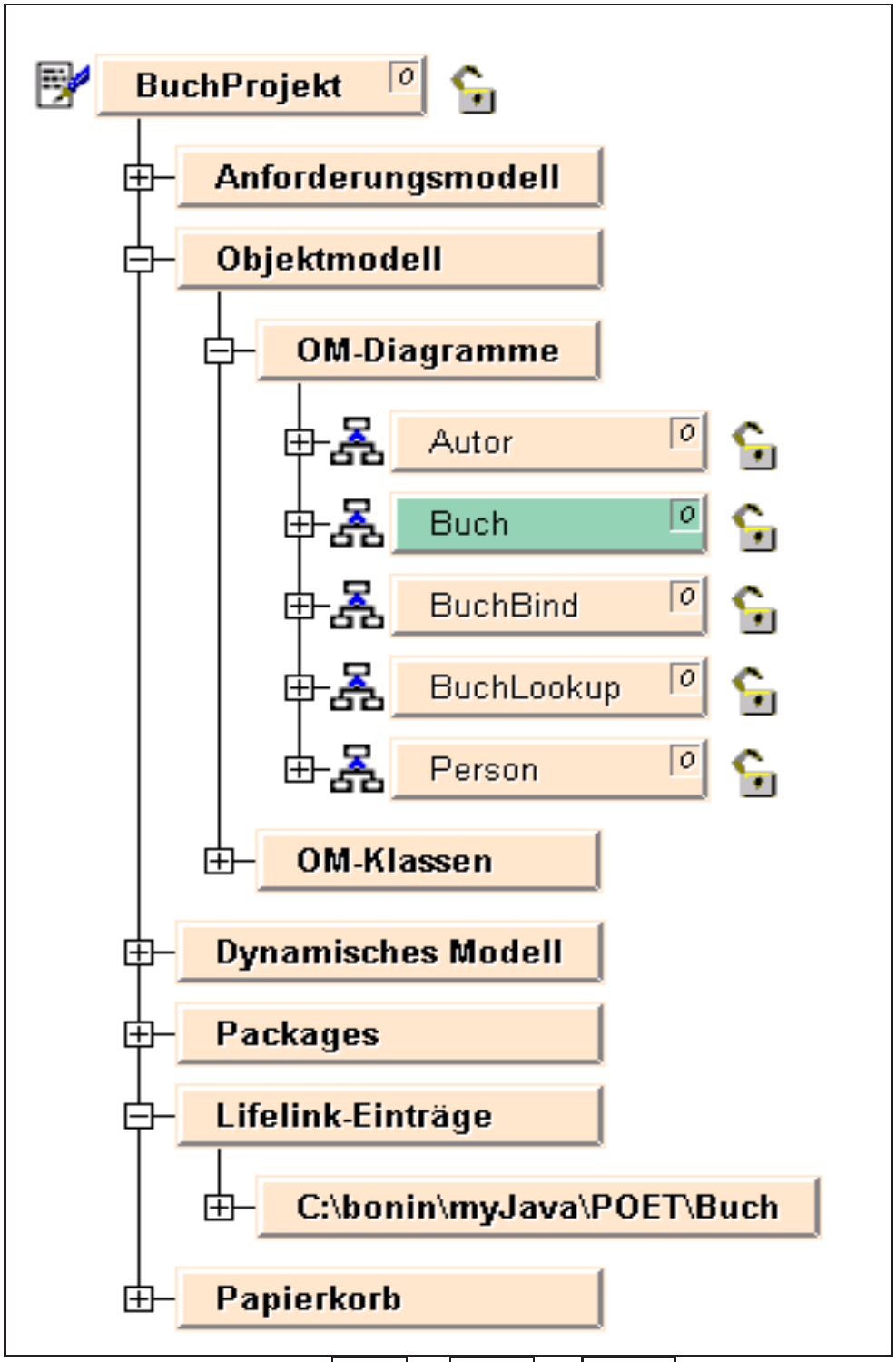


Abbildung 2: Überblick über das Buchprojekt

1.1.1 Java-Code für die Klasse `Buch`

Ein konkretes Buch, also eine Instanz der Klasse `Buch`, hat ein Attribut `alter`. Das Alter des Buches erhöht sich laufend. Es ist daher nicht zweckmäßig dieses Attribut als ein Bestandteil des persistenten Objektes zu erklären. Vielmehr bietet sich an, daß aktuelle Alter beim Zugriff auf das persistente Objekt jeweils zu berechnen. Dazu dienen Methoden des Interfaces `Constraints`. Dieses Interface bewirkt, daß folgende Methoden automatisch appliziert werden:

- `postRead()`
- `preWrite()`
- `preDelete()`

Hier wird die Methode `postRead()` zur Altersberechnung genutzt. Das Attribut `alter` ist mit dem Modifikator `transient` gekennzeichnet, damit es nicht zu den persistenten Variablen zählt.

`transient`

```
 1  /** Einfaches POET-Uebungsbeispiel
 2      @author Bonin 10-Jun-1998
 3      @version 1.0
 4  */
 5  import COM.POET.odmg.*;
 6  import java.util.*;
 7
 8  public class Buch implements Constraints {
 9      private String titel;
10      private String isbn;
11      private int erscheinungsJahr;
12      private Autor hauptAutor;
13      private String schlagWoerter;
14      private transient int alter;
15      private String verlagsKurzName;
16
17      // Zugriffsmethoden (Getter)
18
19      public String getTitel() {
20          return titel;
21      }
22      public String getIsbn() {
23          return isbn;
24      }
25      public int getErscheinungsJahr() {
26          return erscheinungsJahr;
27      }
28      public Autor getHauptAutor() {
```

```
29          return hauptAutor;
30      }
31      public String getSchlagWoerter() {
32          return schlagWoerter;
33      }
34      public int getAlter() {
35          return alter;
36      }
37      public String getVerlagsKurzName() {
38          return verlagsKurzName;
39      }
40
41      // Zustandsmodifikatoren (Setter)
42      public void setTitel(String titel) {
43          this.titel = titel;
44      }
45      public void setIsbn(String isbn) {
46          this.isbn = isbn;
47      }
48      public void setErscheinungsJahr(int erscheinungsJahr) {
49          this.erscheinungsJahr = erscheinungsJahr;
50      }
51      public void setHauptAutor(Autor hauptAutor) {
52          this.hauptAutor = hauptAutor;
53      }
54      public void setSchlagWoerter(String schlagWoerter) {
55          this.schlagWoerter = schlagWoerter;
56      }
57      public void setAlter(int alter) {
58          this.alter = alter;
59      }
60      public void setVerlagsKurzName(String verlagsKurzName) {
61          this.verlagsKurzName = verlagsKurzName;
62      }
63
64      // Methode zur Objektrekonstruktion (automatisch appliziert!)
65      // (Erfuellung des Interfaces:  Constraints)
66      public void postRead() {
67          int heute = new Date().getYear();
68          this.setAlter(heute - this.getErscheinungsJahr());
69      }
70      public void preWrite() {
71          System.out.println("preWrite-Methode appliziert!");
72      }
73      public void preDelete() {
74          System.out.println("preDelete-Methode appliziert!");
75      }
76  }
```

1.1.2 Java-Code für die Klasse `Autor`

Da ein Objekt der Klasse `Autor` die Eigenschaften der Klasse `Person` aufweisen soll, ist eine Vererbungsbeziehung zwischen `Autor` und `Person` zweckmäßig. In Java wird diese Vererbung wie folgt notiert: `public class Autor extends Person {...}`

extends

```
 1  /** Einfaches POET-Uebungsbeispiel
 2      @author Bonin 10-Jun-1998
 3      @version 1.0
 4  */
 5  import COM.POET.odmg.*;
 6  import java.util.*;
 7
 8  public class Autor extends Person {
 9      private String themen;
10      private String orgKurzName;
11
12      // Konstruktoren
13      public Autor() {};
14      public Autor(String name){
15          this();
16          this.setZuName(name);
17      }
18
19      // Zugriffsmethoden (Getter)
20      public String getThemen() {
21          return themen;
22      }
23      public String getOrgKurzName() {
24          return orgKurzName;
25      }
26
27      // Zustandsmodifikatoren (Setter)
28      public void setThemen(String themen) {
29          this.themen = themen;
30      }
31      public void setOrgKurzName(String orgKurzName) {
32          this.orgKurzName = orgKurzName;
33      }
34  }
```

1.1.3 Java-Code für die Klasse `Person`

Die Klasse `Person` stellt (nur) die beiden Attribute `zuName` und `vorNamen` bereit.

```
1  /** Einfaches POET-Uebungsbeispiel
2      @author Bonin 10-Jun-1998
3      @version 1.0
4  */
5  import COM.POET.odmg.*;
6  import java.util.*;
7
8  public class Person {
9      private String zuName;
10     private String vorNamen;
11
12     // Zugriffsmethoden (Getter)
13     public String getZuName() {
14         return zuName;
15     }
16     public String getVorNamen() {
17         return vorNamen;
18     }
19
20     // Zustandsmodifikatoren (Setter)
21     public void setZuName(String zuName) {
22         this.zuName = zuName;
23     }
24     public void setVorNamen(String vorNamen) {
25         this.vorNamen = vorNamen;
26     }
27 }
```

1.1.4 Konfigurationsdatei für das OODBMS

In der Konfigurationsdatei `ptjavac.opt` sind folgende Werte anzugeben:

- Name des Datenbankservers, hier: `oodbserver`
- Name der Datenbank, hier: `BuchDB`
- Name des Datenbankschemata, hier: `BuchDict`

Zusätzlich sind die persistenten Klassen zu nennen, hier:

- `Buch`
- `Autor`
- `Person`

Der POET-Server bewirtschaftet die Verweise und Nutzdaten entweder jeweils in einer Datei (auf Betriebssystemebene) oder in zwei

Dateien. Beim Zweidateien-Fall ist die Anwortzeit durch Zugriffsoptimierung (auf Betriebsystemebene) kürzer. Hier ist daher diese Option gewählt. Sie wird angegeben durch den Wert `onefile=false`.

```
1  /*
2    ptjavac.opt
3    Konfiguration fuer Buch-Beispiel
4   */
5
6  [schemata\dict]
7  name=BuchDict
8  server="oodbserver"
9  onefile=false
10
11 [databases\base]
12 name=BuchDB
13 schema=dict
14 location=same
15 onefile=false
16
17 [classes\Buch]
18 persistent=true
19 schema=dict
20
21 [classes\Autor]
22 persistent=true
23 schema=dict
24
25 [classes\Person]
26 persistent=true
27 schema=dict
```

1.1.5 Protokoll der Compilierung

Die Datenbank für die Objekt(spiel)welt wird in einer Client-Server-Architektur eingerichtet. Der POET-Server mit der Bezeichnung `oodbserver` läuft auf dem Intel-Rechner mit der `IP 193.174.33.20` unter dem Betriebssystem Windows-NT. Die Compilierung erfolgt auf einem anderen Rechner im Netz. Dieser Intel-Rechner hat die `IP 193.174.33.99`. Es wird ebenfalls Windows NT als Betriebssystem verwendet. Die Abbildung 3 auf Seite 11 zeigt Einstellungen für die Datenbank `BuchDB` auf dem POET-Server `oodbserver`.

```
1  C:\bonin\myJava\POET\Buch>ptjavac *.java
2  ptjavac *.java
3  POET Java! Preprocessor Version 1.50.09
4  Copyright (C) 1996-98 POET Software Corporation
5  POET Java! Schema Creation Version 1.50.09
```

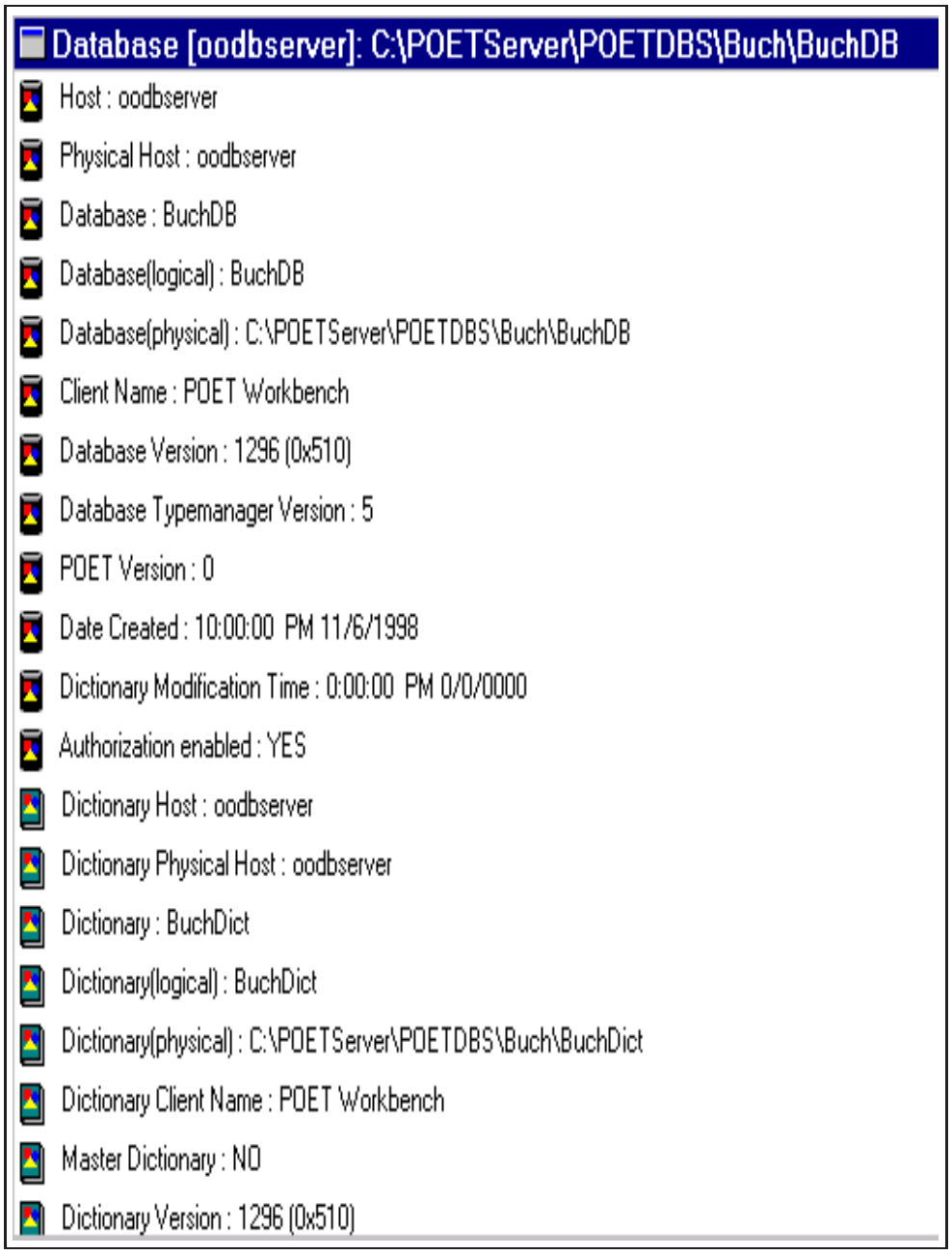


Abbildung 3: POET-Darstellung der Datenbank `BuchDB`

```
 6  Copyright (C) 1997 POET Software Corporation
 7  Registered: Person
 8  Registered: Buch
 9  Registered: Autor
10  Update database: BuchDB => Done
11  .
12  .
13  .
14  Verzeichnis von C:\bonin\myJava\POET\Buch
15
16  25.11.98  10:04         <DIR>          .
17  25.11.98  10:04         <DIR>          ..
18  25.11.98  10:04                  1.575 Autor.class
19  25.11.98  10:04                  4.302 Buch.class
20  25.11.98  10:04                  2.726 Person.class
21  25.11.98  10:04                  1.795 _pt_metaAutor.class
22  25.11.98  10:04                  1.986 _pt_metaBuch.class
23  25.11.98  10:04                  1.816 _pt_metaPerson.class
24  .
25  .
26  .
```

Das obige Protokoll zeigt, daß zusätzlich zu einer „persistenten“ Klasse stets eine Klasse mit dem Präfix `_pt_meta` vom POET-Precompiler `ptjavac` generiert wurde. Die generierte Klasse `_pt_metaBuch` stellt für die Klasse `Buch` den Zugriff zum `oodbserver` bereit. Sie ist eine Unterklasse von:

ODMG

`COM.POET.odmg.imp._pt_metaObject`

und importiert POET-Klassen nach dem Konzept der *Object Data Management Group* (ODMG)[9]:

`COM.POET.odmg.imp.PersistentObject`
`COM.POET.odmg.imp.ObjectController`

Exemplarisch wird hier der generierte Java-Code für die Klasse `_pt_metaBuch` gezeigt:

```
 1  import COM.POET.odmg.imp.PersistentObject;
 2  import COM.POET.odmg.imp.ObjectController;
 3  public class _pt_metaBuch extends COM.POET.odmg.imp._pt_metaObject
 4  {
 5      private static _pt_metaBuch Buchinstance;
 6      public static int BuchId = 0;
 7      public static COM.POET.odmg.imp._pt_metaObject get() {
 8  return Buchinstance==null ? new _pt_metaBuch() : Buchinstance;
 9      }
10      public _pt_metaBuch() {
11      }
```

[9] ODMG ≡ vormals: Object Database Management Group, Näheres siehe `http://www.odmg.org/`

```
12      protected void register() {
13          Buchinstance = this;
14          }
15      public PersistentObject create(ObjectController controller)
16      {
17          try {
18      return new Buch(controller);
19  } catch ( Exception exc) {
20          throw new COM.POET.odmg.ODMGRuntimeException(
21  exc.getClass() +":  " + exc.getMessage());
22  }
23      }
24      public Object[] createArray(int size) {
25          return new Buch[size];
26      }
27      public void assignClassId(int classId) {
28          BuchId = classId;
29      }
30      public String getClassName() {
31          return "Buch";
32      }
33      public int getNumberOfSlots() {
34          return super.getNumberOfSlots()+6;
35      }
36      public String getSlotType(int i) {
37          switch (i-super.getNumberOfSlots()) {
38              case 0:
39                  return "java.lang.String";
40              case 1:
41                  return "java.lang.String";
42              case 2:
43                  return "int";
44              case 3:
45                  return "Autor";
46              case 4:
47                  return "java.lang.String";
48              case 5:
49                  return "java.lang.String";
50              default:
51                  return super.getSlotType(i);
52          }
53      }
54      public String getSlotName(int i) {
55          switch (i-super.getNumberOfSlots()) {
56              case 0:
57                  return "titel";
58              case 1:
59                  return "isbn";
60              case 2:
61                  return "erscheinungsJahr";
62              case 3:
63                  return "hauptAutor";
64              case 4:
65                  return "schlagWoerter";
```

```
66              case 5:
67                  return "verlagsKurzName";
68              default:
69                  return super.getSlotName(i);
70          }
71      }
72  }
```

1.2 Nutzung der Objekt(spiel)welt

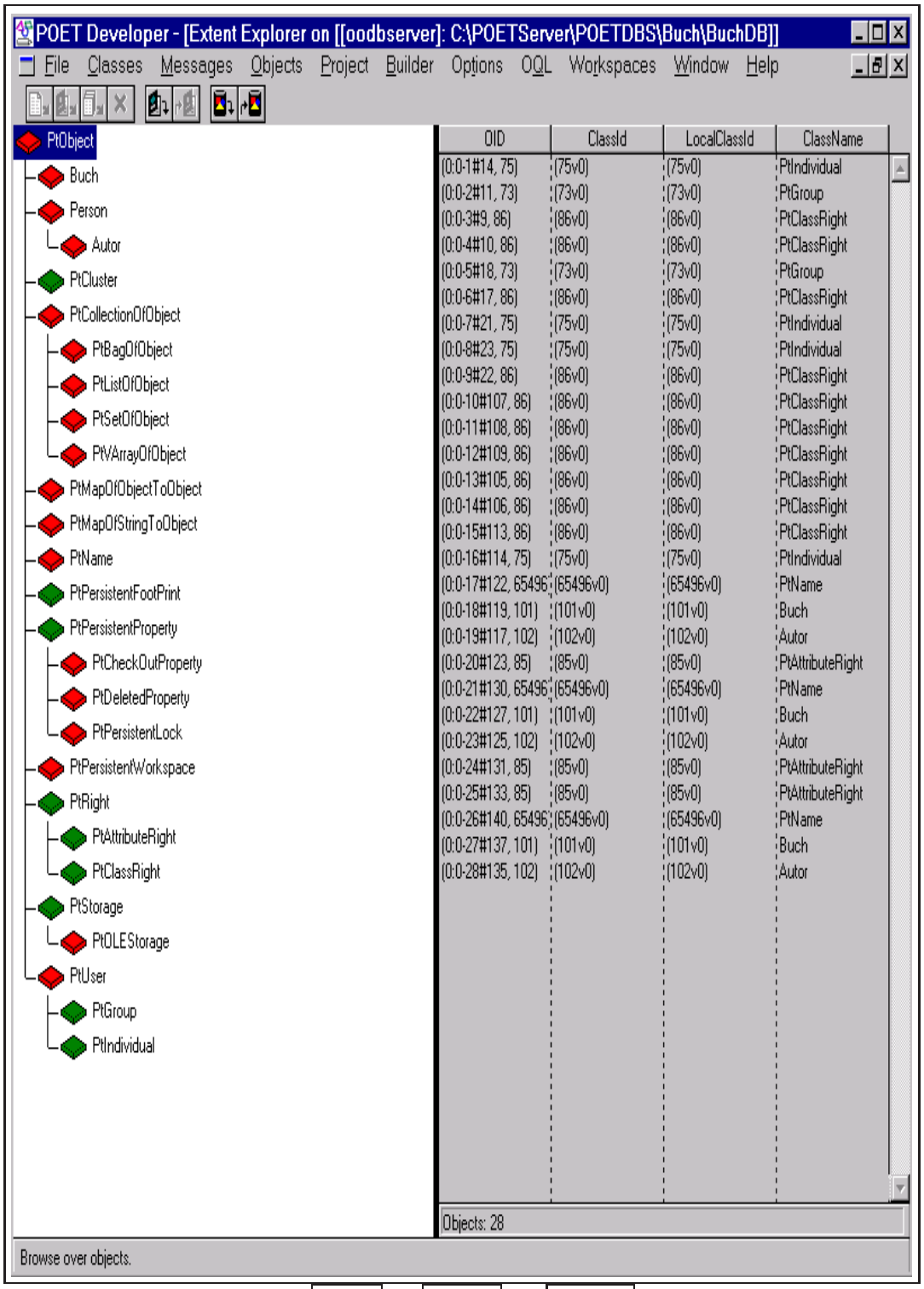
Die Nutzung eine objekt-orientierten Datenbank basiert — wie bei Datenbanken üblich — auf einem Transaktionskonzept. Eine Transaktion ist daher mit einer Datenbank zu verknüpfen. Hier ist die Datenbank `myDB` ein Objekt der Klasse `Database`. Sie wird mit der Transaktion über den Konstruktor `Transaction(myDB)` verknüpft, wie der folgende Javacodeausschnitt zeigt:

```
Database myDB = Database.open("poet://oodbserver/BuchDB",
      Database.openReadWrite);
Transaction myT = new Transaction(myDB);
myT.begin();
    Buch myBuch = new Buch();
    myDB.bind(myBuch, "PKS01");
myT.commit();
```

Den vollständigen Javacode für das simple „Einspeicherungsprogramm“ `BuchBind.java` zeigt Abschnitt 1.2.2 auf Seite 16. Das simple „Selektionsprogramm“ `BuchLookup.java` steht in Abschnitt 1.2.3 auf Seite 17. Die Anwendung dieser Programme demonstriert das folgende Protokoll einer Session.

1.2.1 Protokoll einer Session

```
 1  C:\bonin\myJava\POET\Buch>java BuchBind
 2  java BuchBind
 3  Zuname des Autors: Bonin
 4  Alter des Buches : 7
 5  PKS01 gibt es schon!
 6
 7  C:\bonin\myJava\POET\Buch>java BuchLookup
 8  java BuchLookup
 9  Zuname des Autors: Bonin
10  Alter des Buches : 7
11
12  ...
```



Objektwelt: Buch ⟶ Autor —▷ Person

Abbildung 4: POET-Developer-Darstellung der Datenbank `BuchDB`

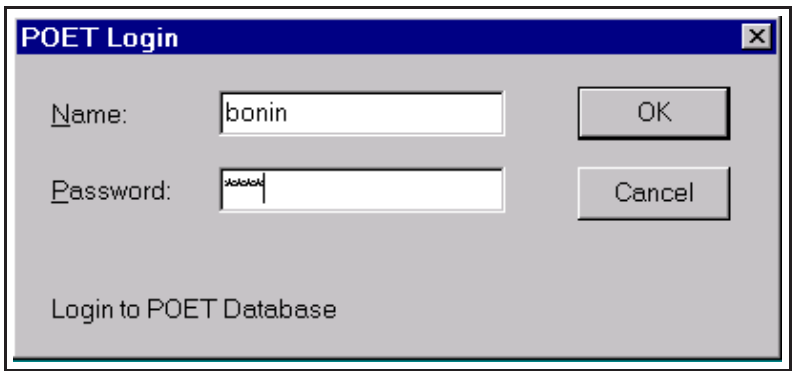


Abbildung 5: POET-Login

1.2.2 Buchobjekt dem OODBMS übergeben

Eine wesentliche Leistung eines OODBMS besteht darin, das referenzierte Objekte automatisch mit gespeichert werden. In unserem Beispiel wird mit dem einfachen Konstrukt:

```
myDB.bind(myBuch, "PKS01");
```

automatisch auch eine Objekt der Klasse `Autor` und ein Objekt der Klasse `Person` gespeichert. Die Referenz zu diesen Objekten wird durch das Konstrukt:

```
myBuch.setHauptAutor(new Autor("Bonin"));
```

aufgebaut.

```
 1  /** Einfaches Beispiel fuer POET
 2      @author Bonin 10-Jun-1998
 3      @version 1.0
 4  */
 5  import COM.POET.odmg.*;
 6
 7  public class BuchBind {
 8      public static void main(String[] argv)
 9          throws ODMGException {
10          Database myDB = Database.open(
11                                      "poet://oodbserver/BuchDB",
12                                      Database.openReadWrite);
13          Transaction myT = new Transaction(myDB);
14          myT.begin();
15          try {
16              Buch myBuch = new Buch();
17              myBuch.setTitel(" Softwarekonstruktion mit LISP");
18              myBuch.setIsbn("3-11-011786-X");
19              myBuch.setErscheinungsJahr(91);
```

```
20
21              myBuch.setHauptAutor(new Autor("Bonin"));
22
23              myBuch.setSchlagWoerter("Arbeitstechniken, Qualitaet");
24              myBuch.postRead();
25              myBuch.setVerlagsKurzName("WalterDeGruyter");
26              System.out.println("Zuname des Autors: " +
27                                 myBuch.getHauptAutor().getZuName() +
28                                 "\nAlter des Buches : " +
29                                 myBuch.getAlter());
30              myDB.bind(myBuch, "PKS01");
31          }
32          catch (ObjectNameNotUniqueException exc) {
33              System.out.println("PKS01 gibt es schon!");
34          }
35          catch (ODMGRuntimeException exc) {
36              myT.abort();
37              throw exc;
38          }
39          myT.commit();
40          myDB.close();
41      }
42  }
```

1.2.3 Buchobjekt aus OODMS zurückgewinnen

Das dem OODBMS übergebene Buch kann über den String `PKS01` wiedergewonnen werden. Damit das richtige Buch wiedergewonnen wird, muß dieser String in der gesamten Datenbank `BuchDB` eindeutig sein. Die persistenten Daten, die das Buch bilden, müssen erst mittels `cast`-Operation aus ihrer allgemeinen Struktur `Object` in die Struktur der Klasse `Buch` überführt werden:

Casting

```
Buch myBuch = (Buch) myDB.lookup("PKS01");
```

```
1   /** Einfaches Beispiel fuer POET
2       @author Bonin 10-Jun-1998
3       @version 1.0
4   */
5   import COM.POET.odmg.*;
6
7   public class BuchLookup {
8       public static void main(String[] argv) throws ODMGException {
9           Database myDB = Database.open(
10                                      "poet://oodbserver/BuchDB",
11                                      Database.openReadWrite);
12          Transaction myT = new Transaction(myDB);
```

```
13          myT.begin();
14          try {
15              Buch myBuch = (Buch) myDB.lookup("PKS01");
16              System.out.println("Zuname des Autors: " +
17                                 myBuch.getHauptAutor().getZuName() +
18                                 "\nAlter des Buches : " +
19                                 myBuch.getAlter());
20          }
21          catch (ODMGRuntimeException exc) {
22              myT.abort();
23              throw exc;
24          }
25          myT.commit();
26          myDB.close();
27      }
28  }
```

2 Der Elchtest — Leistungsdefizit: OO-Rekonstruktion statt OO-Management

Die kurze POET-Probefahrt mit der Objekt(spiel)welt macht folgende Fakten deutlich:

1. „halbes Objekt“
   Im OODBMS ist der aktiv(ierbar)e Teil des Objektes nicht gespeichert!

2. allgemeine Objektstruktur
   Im OODBMS ist der passive Teil des Objektes in einer allgemeinen Objektstruktur gespeichert. Die eigentliche Klassenstruktur ist erst durch eine `cast`-Operation auf der Client-Seite wieder herzustellen.

3. keine Objekt-Identität
   Das OODBMS benötigt einen eindeutigen String um das „Objekt“ zu finden.

Das OODBMS verwaltet nicht Objekte, sondern verwaltet Bytes aus denen ein Objekt mit den selben Eigenschaften des ursprünglichen Objektes wieder rekonstruiert werden kann. Die Objekt-Identität wird mit Hilfe eines eindeutigen String (Namensraum = `database`) simmuliert.

3 Fazit & Ausblick

Bei komplexen Objektbeziehungen ist ein heutiges OODBMS hilfreich. nützlich!
Man muß sich kaum um die referenzierten Objekte kümmern. Geht es jedoch um den Zugriff auf persistente Objekte von vielen Rechner im Netz, dann müssen stets die betroffenen Klassen vor Ort verfügbar sein. Ein handhabbares Objektmanagement im Netz muß sicherlich zukünftig die Klassen-Verteilung stärker integrieren.

Auch die Frage der Objekt-Identität bedarf einer Verbesserung. Ein Beispiel dazu wäre eine Klasse `Set` im mathematischen Sinne einer Menge. Bei ihr kann das Objekt nicht zweimal vorkommen. In unserer Objekt(spiel)welt braucht nur der Zuordnungsstring `PKS01` geändert werden und POET „speichert das Buch“ nochmal.

4 Quellen

Literatur

[Bo98] Hinrich Bonin; Der Java-Coach, FINAL, 8. Jahrgang Heft 1, Oktober 1998, ISSN 0939-8821, (CD-ROM und `http://as.fh-luenburg.de`)

[FINAL] Fachhochschule Nordostniedersachsen, INformatik, Arbeitsbereichte, Lüneburg, ISSN 0939-8821, beziehbar über FHNON, Volgershall 1, D-21339 Lüneburg, Fax: xx49/4131/677149.

Index