OOD Zusammenfassung

René Bernhardsgrütter, 22.02.2012

1 JDBC

Prepared Statements verwenden.

Treiber laden:

```
Class.forName("com.mysql.jdbc.Driver");
```

Mit Datenbank verbinden:

```
String url = "jdbc:mysql://localhost:3306/";
String dbName = "ood_test";
String driver = "com.mysql.jdbc.Driver";
String userName = "root";
String password = "";
Class.forName(driver).newInstance();
Connection conn = DriverManager.getConnection(url+dbName,userName,password);
```

Man sollte PreparedStatements verwenden, da diese die Möglichkeit zu SQL Injection verringern:

```
int wert = 500;
String sql_statement="SELECT * FROM meinetabelle WHERE x = ?"; // ? ist Platzhalter
PreparedStatement st = db.prepareStatement(sql_statement);
st.setInt(1, wert);
ResultSet rs = st.executeQuery();
while (rs.next()) {
       System.out.print("Spalte 1 ergab ");
       System.out.println(rs.getString(1));
}
rs.close(); // ResultSet sollte nach der Verwendung geschlossen werden
st.close(); // Statement sollte nach der Verwendung ebenfalls geschlossen werden
```

ResultSet: mit rs.next() kann man prüfen, ob eine weitere Zeile vorhanden ist. Dies muss man immer machen, bevor man die Zeile ausliest (NullPointerException…).

DDL: Data Definition Language
DML: Data Manipulation Language

Auslesen von Daten mit: st.executeQuery() => gibt ein ResultSet zurück.
Einfügen von Daten mit preparedStatement.executeUpdate(). Ps.execute() braucht man nur selten.

1.1 Transaktionen

Eine Transaktion ist eine Folge von SQL-Abfragen. Nach dem Absetzen können diese ausgeführt (committed) bzw. zurückgenommen (rolled back) werden.

- **A**tomic: Das DBMS verhält sich dabei gegenüber dem Benutzer so, als ob die Transaktion eine einzelne elementare Operation wäre, die nicht von anderen Operationen unterbrochen werden kann.
- **C**onsistent: heißt, dass eine Sequenz von Daten-Operationen nach Beendigung einen konsistenten Datenzustand hinterlässt.
- **I**ndependet: verhindert, dass sich nebenläufig in Ausführung befindliche Daten-Operationen gegenseitig beeinflussen.
- **D**urable: Das Ergebnis einer Daten-Operation ist dauerhaft.

Die Ausführung ist einfach und wird „manuell“ gemacht:

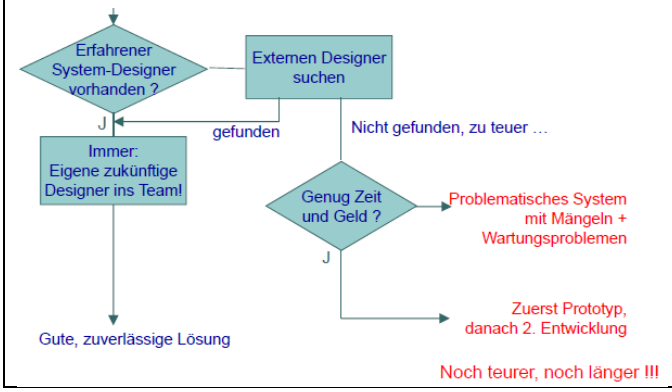
```
Connection con =
```

```
DriverManager.getConnection(...);
con.setAutoCommit(false);
Statement s = con.createStatement();
s.executeUpdate("... SQL statement 1 ... ");
s.executeUpdate("... SQL statement 2 ... ");
s.executeUpdate("... SQL statement 3 ... ");
con.commit(); // transaction (3 statements) is committed
```

2 Objektorientierte Analyse OOA, Objektorientiertes Design OOD

Eine Methodik oder ein Hilfsmittel zum Darstellen eines Software-Systems als Basis für eine erfolgreiche Realisierung. = Planung

Wozu:
• Aufgabe begreifen (formulieren!)
• System-Struktur darstellen
• Kommunikation im Team
• Arbeitsteilung ermöglichen
• Dokumentation für Pflege/
• Weiterentwicklung
	Wann:
	• „Am Anfang“ des Projekts
	• Fortwährend / Schubweise
	• Je nach Projekt-Vorgehen:
	  ○ Wasserfall
	  ○ eXtreme Programming

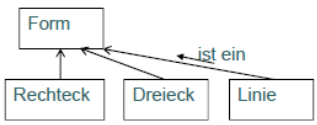
Wer: • „der Beste/Erfahrenste von innen / von aussen (Spirale) • mit Lehrlingen 	Wie: • Aufgabenstellung formulieren • Klassen + Use Cases formulieren • GUI-Prototyp • Voraussehbar → Planbare Entwicklung • Nicht voraussehbar → (wiss.) Forschung • Kommunikation mit Kunde und Team

Einen guten System-Design kann man nur machen, wenn man schon weiss, wies geht, und schon weiss, worauf es ankommt.

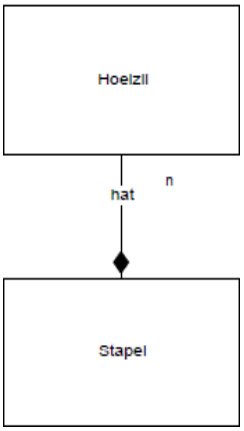
2.1 Beziehungen zwischen Klassen

Es gibt drei Beziehungsarten zwischen Klassen:

1. Ist
2. Hat
3. Braucht



Ist-Ein-Beziehung wird durch Vererbung beziehungsweise Ableitung implementiert.

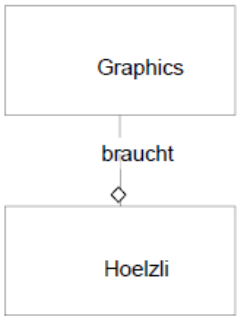


Hat-n-Beziehung (oder **Besteht-aus-Beziehung**) wird dadurch implementiert, dass in einer Klasse n Instanzen einer anderen Klasse erzeugt werden (Objekte, mit new).

Lesen vom Beispiel link: Ein Objekt des Typs Stapel besteht aus (oder: hat / ist Teil von) n Objekten des Typs Hoelzli.

Dabei ist die Klasse Stabel die **Besitzerklasse**.

Es wird die **schwarz gefüllte Raute bei der Besitzerklasse** als Kennzeichnung verwendet.



Bracht-Beziehung wird dann implementiert, wenn die eine Klasse Funktionalitäten einer anderen braucht. Beispiel: Math, Random, BCrypt, Graphics.

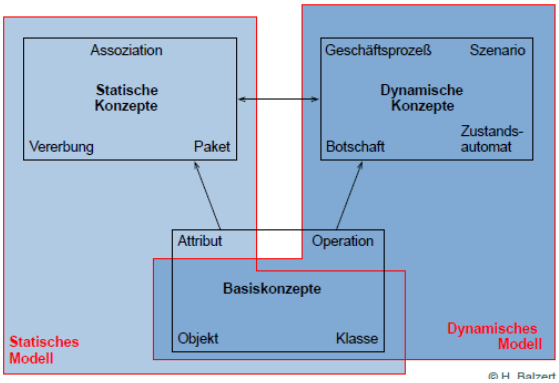
Dies wird durch die **leere Raute bei der „Braucher"-Klasse** dargestellt.

Man lies: Hoelzli braucht Graphics.

2.2 Statisches und dynamisches Modell

Das **statische Modell** beschreibt:

- die **Klassen** des Fachkonzepts
- die **Assoziationen** (Beziehungen) zwischen den Klassen
- die **Vererbungsstrukturen**
- die **Attribute** der Klassen (Daten des
- Systems)
- die Bildung von **Paketen** (Teilsysteme)



Das **dynamische Modell** zeigt Funktionsabläufe:

- **Geschäftsprozesse** (Use Cases) beschreiben die durchzuführenden Aufgaben auf einem sehr hohen Abstraktionsniveau
- **Szenarios** zeigen, wie Objekte miteinander kommunizieren, um eine bestimmte Aufgabe zu erledigen
- **Zustandsautomaten** beschreiben die Reaktionen eines System(teil)s auf verschiedene Ereignisse (=Botschaften)

2.3 GUI-Prototyp

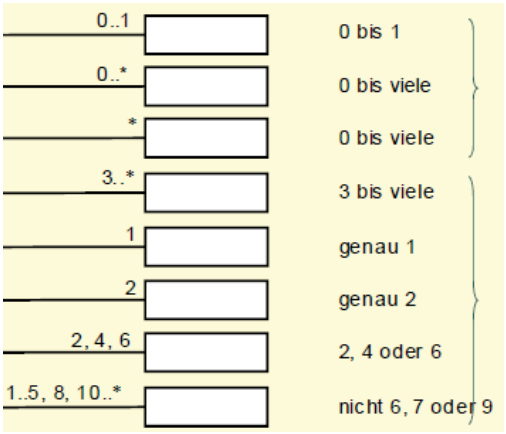
Der GUI-Prototyp stellt alle Eigenschaften des OOD/A Modells auf dem GUI dar. Die Logik funktioniert aber noch nicht.

3 UML und statische Konzepte

Eine Assoziation bedeutet allgemein eine Verbindung zwischen zwei Klassen/Objekten, oder noch einfacher, dass „die zwei Objekte miteinander etwas zu tun haben".

- Eine Assoziation zwischen Objekten derselben Klasse (z.B. eine Kette von Objekten) nennt man reflexiv
- Assoziationen werden (in der Systemanalyse) grundsätzlich als bidirektional betrachtet; in der Implementation sind sie jedoch gerichtet
- Es gibt 1:1 (binäre) und 1:n (höherwertige) Assoziationen (Die Assoziation im vorherigen Beispiel war 1:*, also höherwertig)

3.1 Kardinalität



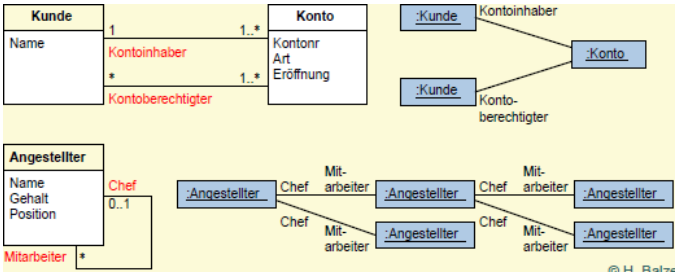
Kann-Assoziation

Muss-Assoziation

3.2 Assoziation

Man beschreibt meistens nur eine Richtung der Assoziation.

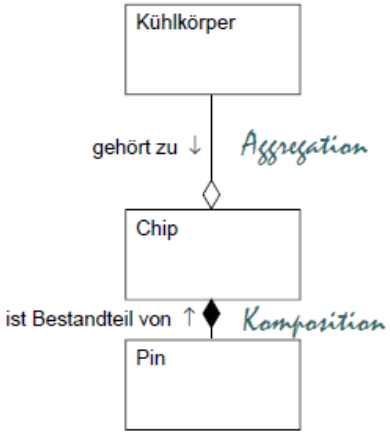
Name darf fehlen, wenn die Bedeutung der Assoziation offensichtlich ist. Andernfalls kann man den Namen zur zugehörigen Seite schreiben, beispielsweise bei dem Kunden „Kontoinhaber“.



Der Rollenname ist obligatorisch, wenn zwischen zwei Klassen mehr als eine Assoziation besteht oder wenn es sich um eine reflexive Assoziation handelt.

3.2.1 Arten von Assoziationen

- **Einfache Assoziation** (ordinary association)
  → haben irgend etwas zu tun miteinander)
- **Aggregation** (*aggregation*)
  → Das Eine / die Einen gehören zu dem/den Andern
- **Komposition** (*composite aggregation*)
  → das Eine ist Bestandteil des Andern



Die genaue Zuweisung kann manchmal unklar sein:

3.3 Vererbung

Vererben heisst, alle Attribute und Operationen von der Oberklasse zu übernehmen
Polymorphismus heisst, gewisse Operationen der Oberklasse mit eigenen zu überschreiben

3.4 Pakete

Ein Paket definiert einen Namensraum für alle enthaltenen Klassen oder Pakete.

UML-/Java-Notation:

Allgemein:	Paket::Klasse
UML-Struktur:	Paket1::Paket11::Paket111::Klasse
Java-Struktur:	paket1.paket11.paket111.Klasse

Auf jede Klasse in jedem Paket kann verwiesen werden.

4 UML und dynamische Konzepte

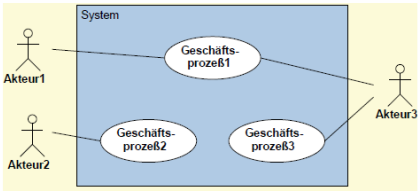
4.1 Use Cases

Geschäftsprozesse und Use Cases Modellieren die Abläufe im Gesamtsystem.

Als Use Cases werden dabei Geschäftsinterne Prozesse angesehen, die nötig sind, um die Wünsche des Kunden zu befriedigen (Beispiel bei einem Sportgeschäft: Verkauf > Auftragsbearbeitung > Laber > Spedition: Dies ist die Prozesskette).

Im Programm sind es eine oder mehrere Funktionen, die ausgeführt werden müssen, um ein Problem zu lösen. Diese kann man so bestimmen, indem man die **Tätigkeiten** im Pflichtenheft herausfiltert. Anschliessend muss man feststellen, wer welche Tätigkeiten ausführen darf => alle verschiedenen Use Cases.

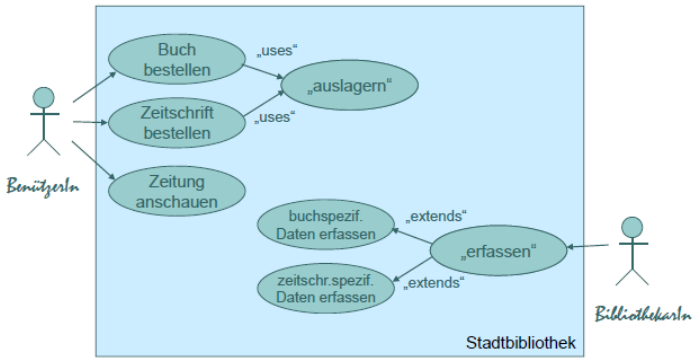
Die Geschäftsprozesse werden in Ovalen dargestellt. Die Akteure werden mit den Use Cases verbunden.



4.1.1 Beziehungen unter Use Cases

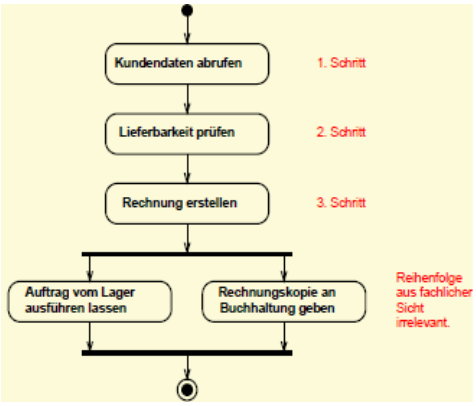
Es gibt auch **extends- und uses-Beziehungen**! Diese sind wie bei den Klassen.

Beispiel mit einer Stadtbibliothek:



4.1.2 Zerlegung jedes Use Cases

Jeder Use Case wird in einzelne, überschaubare Schritte zerlegt. Dabei finden sich in der Regel sehr viele gemeinsame Schritte.



4.2 Botschaften / messages

Eine Botschaft (message, Nachricht) ist die Aufforderung eines Senders (client) an einen Empfänger (server, supplier) eine Dienstleistung zu erbringen. Empfänger interpretiert diese Botschaft und führt eine Operation (gleichen Namens) aus. Sender der Botschaft weiß nicht, wie die entsprechende Operation ausgeführt wird.

4.3 Szenarien

Szenarien sind die nächste Detaillierungsstufe der Use Cases bzw. Aktivitätsdiagramme. Sie werden wahlweise in Form von Sequenz- oder Kommunikationsdiagrammen dargestellt. Es wurden Sequenzdiagramme und Kommunikationsdiagramme übernommen.

- Sequenzdiagramme zeigen besser die zeitliche Abfolge
- Kommunikationsdiagramme zeigen Klassenabhängigkeiten besser

// TODO Sequenzdiagramme und Kommunikationsdiagramme eintragen (das, was wir brauchen)

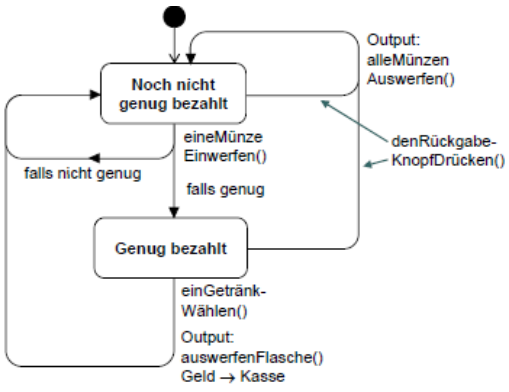
4.4 Zustandsautomat

Die Ausführung von Operationen ändert nicht nur interne Daten, sondern auch das Verhalten des Objekts:

- Die Ausleihe einer Bibliothek verhält sich anders, je nachdem ob das gewünschte Buch da ist oder nicht
- Das Erfassen einer neuen Buchung dagegen verändert das Verhalten des Buchhaltungssystems nicht

Beispiel an einem Getränkeautomat:

Regeln:



- Am Automaten kostet eine Flasche 2.30 Fr. Als erstes muss man das Geld einwerfen.
- Dann muss man die gewünschte Getränkesorte durch einen Knopf wählen.
- Der Automat wirft jetzt die Flasche aus.
- Der Automat hat immer alle Getränkesorten vorrätig.
- Der Automat gibt kein Rückgeld.
- Wird der Rückgabeknopf gedrückt, so wird alles bisher eingeworfene Geld zurückgegeben, und der Automat kehrt in den Ausgangszustand zurück.