1 ENTWURFSPHASE

Wiederholung: Welche sind die typischen Phasen in der Entwicklung eines Softwareprojekts?
[?[2]]

Def.: Entwurf ist Strukturierung des Systems aus Entwicklersicht:

- Zerlegung in Komponenten, ***Bsp***.: [?[3]]

- ihre Anordnung in Hierarchie
- ihre Spezifikation (WAS leisten sie?)
- Festlegung der Schnittstellen zwischen den Komponenten (Export, Import)

Dualität WAS vs. WIE

WIE arbeitet eine Komponente ("white box"-Sicht) wird erklärt durch [?[4]]

Bsp.: WIE arbeitet ein Telefon?

[?[5]]

Zum Analysemodell hinzukommende Aspekte

- Objektverwaltung
- (Zwischen-)Speicherung
- Effizienz

1.1 Objektverwaltung

ist [?[6]]

→ [Balzert99], p. 24

dazu folgende externe Operationen:

[Wiederholung:

intern?	[?[7]]
extern?	

]

- `erfassen()`: Geprüftes Erfassen eines neuen Objektes mit persistentem Abspeichern und Folgeoperationen
- `ändern()`: Geprüftes Ändern eines schon bestehenden persistenten Objekts mit Folgeoperationen
- `löschen()`: Geprüftes Löschen eines schon bestehenden persistenten Objekts mit Folgeoperationen
- `suchen()`: Geprüftes Suchen eines/mehrerer schon bestehender persistenter Objekte, statt Balzert: `erstelleListe()`

Im Entwurf müssen deren benötigte Varianten mit ihrer Signatur (Parametrierung) festgelegt werden, es sollten jedoch nur Objektoperationen verwendet werden, da Klassenoperationen in Java nur Stiefkinder sind (können z.B. nicht **`abstract`** sein). ***Bsp***.:

```
io_kunde.löschen()                vs.
lgApplication.löschen(io_kunde)   vs.
lgApplication.löschen(i_kundeId)
```

Varianten von `suchen()`

Von der Operation `suchen()` müssen im Entwurf alle benötigten Varianten festgelegt werden, ***Bsp***.:

```
lgApplication.kundeSuchen(i_kundeId: String): LgKunde
                                          throws LgKundeUnbekanntExc

lgApplication.kundeSuchen(i_telefonNr: String): LgKunde
                                          throws TelefonnrUnbekanntExc

lgApplication.kundenSuchen(i_namensanfang: String): LgKunde_List
```

1.2 Drei-Schichten-Architektur

→ [Balzert99], Kap. 10, p. 370 ff.

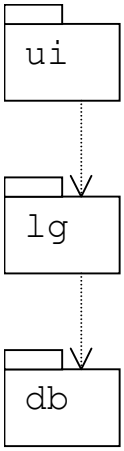
Grundlegende Entwurfsentscheidungen

- Programmiersprache: {Java, C++, C#, VB, PHP, Skriptsprachen, ...}
- UI-System: {Java/Swing, JSP, Windows, Tcl/Tk, ...}
- DB-System: {SQL, OO-DB, ...}

Entwurfsprinzipien für Komponenten

- Minimale Schnittstellen
- Austauschbarkeit ⇒ Zusammenhängendes in einer Komponente, Schnittstelle als Interface
- Testbarkeit ⇒ möglichst als API (Application Programming Interface: Unterprogrammschnittstelle)
- Wiederverwendbarkeit

Ergebnis: Strenge 3-Schichten-Architektur



User Interface: Reine Darstellung, Ereignisbehandlung, I/O;
ohne Verarbeitung

Applikationslogik (Fachkonzept) als API: Buchungen;
ohne UI-, DB-Aspekte, SQL

Datenbankzugriffe als API: Reines Speichern/Suchen; ohne Verarbeitung

Regeln

- `lg`-Schicht exportiert nur geprüfte Operationen, garantiert die Einhaltung der Geschäftsregeln.

- `ui`-Schicht darf nicht auf `db`-Schicht zugreifen, damit die Prüfungen nicht umgangen werden können.
- `ui`-Schicht sollte die Prüfungen durch `lg`-Schicht nicht duplizieren (Bsp. Plausi-Prüfungen).
- `lg`-Schicht exportiert an `ui`-Schicht nur `lg`-Objekte, für deren Erzeugung aus elementaren Daten exportiert sie spezielle, geprüfte Konstruktoroperationen, ***Bsp***.:

```
class LgDatum {
   static LgDatum fromNumeric(String i_string){…} //jjjjmmtt
   static LgDatum fromIso    (String i_string){…} //jjjj-mm-tt
   ...
}
```

1.2.1 Vom Analysemodell zum Entwurf

Aus einer in der Analyse gefundenen Klasse (***Bsp***. Konto) werden im Entwurf Klassen in allen drei Schichten.

- In der **Oberflächenschicht** ist typischerweise je Dialog eine Klasse nötig (***Bsp***. `UiKonto`, `UiKontoÜberweisung`), bei Java Server Pages (JSP) zusätzlich zur eigentlichen JSP. Außerdem Klassen für wiederkehrende Elemente von Dialogen (***Bsp***. `UiKontoList`, `UiButton`).
- In die **Logikschicht** wird die Fachklasse aus der Analyse mit allen ihren Operationen übernommen (***Bsp***. `LgKonto`), um die Objektverwaltungsoperationen sowie bei Bedarf ergänzt oder zwecks Schichtentkopplung bei top-down-Entwicklung (→ Kapitel 1.3.4) in Interface und Implementierung aufgespalten. Jede Klassenoperation (***Bsp***. `erfassen`, `suchen`) sollte dabei durch Verlagerung in ihre Ausgangsklasse zu einer Objektoperation umgewandelt werden.
- In der **Datenbankschicht** haben wir für Sprachen mit Reflection wie Java eine generische Klasse `DbObject`, die für eine beliebige Lg-Klasse deren Attribute persistent machen kann. Außerdem benötigen wir gemäß [Siedersleben] ein Interface `Pool`, das den operativen Zugriff auf den Persistenzmanager kanalisiert.

In Sprachen ohne Reflection wie C++ kann man `DbObject` nicht generisch lösen, sondern muß je Fachklasse eine Db-Klasse schreiben/generieren, die die Attribute der OOA-Klasse und die Operationen `dbInsert()`, `dbUpdate()` und `dbDelete()` enthält. Auch in Java wird dieser Ansatz häufiger gewählt („Entity Bean“), obwohl ich ihn für zu aufwändig halte.

Daneben gibt es Hilfsklassen folgender Arten:

- Zur Zwischenspeicherung von jeweils einer Menge von Objekten einer Fachklasse, ***Bsp*** `LgKonto_List`.
- “Value-Klassen” ohne Persistenz zur Aufnahme von konzeptionell primitiven Werten, ***Bsp***. `LgDatum`.

Siehe dazu das folgende Bild.