Was die OOP-Tutorials verschweigen

oder: Wie man objektorientiert denkt

Christian Rehn aka R2C2

Delphi-Treff

Delphi-Tage 2010

Vorstellung

- Christian Rehn aka R2C2
- Informatikstudent an der TU Kaiserslautern
- Moderator und Redakteur bei Delphi-Treff
- `http://www.christian-rehn.de`

Überblick

Einführung

Ziele der Objektorientierung

- Änderbarkeit
- Wiederverwendbarkeit
- Generizität
- Komplexitätsreduzierung
- Wartbarkeit
- Flexibilität
- Robustheit
- Verständlichkeit
- Redundanzfreiheit
- Austauschbarkeit
- Testbarkeit
- ...

Wichtig

Die OO verspricht viel. Die Umsetzung liegt aber immer noch beim Entwickler.

Die OOP als Werkzeug

- „In C we had to code our own bugs. In C++ we can inherit them.“
- Selbiges gilt auch für Pascal und ObjectPascal.
- OOP bietet mehr Möglichkeiten und somit auch mehr Möglichkeiten, Fehler zu machen.

Die Realität

Die OOP ist keine Wunderwaffe, sondern ein mächtiges Werkzeug in den Händen derer, die wissen, wie man damit umgeht.

Die OOP als Werkzeug

- „In C we had to code our own bugs. In C++ we can inherit them.“
- Selbiges gilt auch für Pascal und ObjectPascal.
- OOP bietet mehr Möglichkeiten und somit auch mehr Möglichkeiten, Fehler zu machen.

Die Realität

Die OOP ist keine Wunderwaffe, sondern ein mächtiges Werkzeug in den Händen derer, die wissen, wie man damit umgeht.

OOP und Objektorientierung

OOP

- Klassen und Instanzen
- Vererbung, Überschreiben, etc.

Objektorientierung

- Generelles Programmierparadigma $\Rightarrow$ Denkweise
- Objekte, die über Nachrichten miteinander kommunizieren
- OOA, OOD, OOP

OOP und Objektorientierung

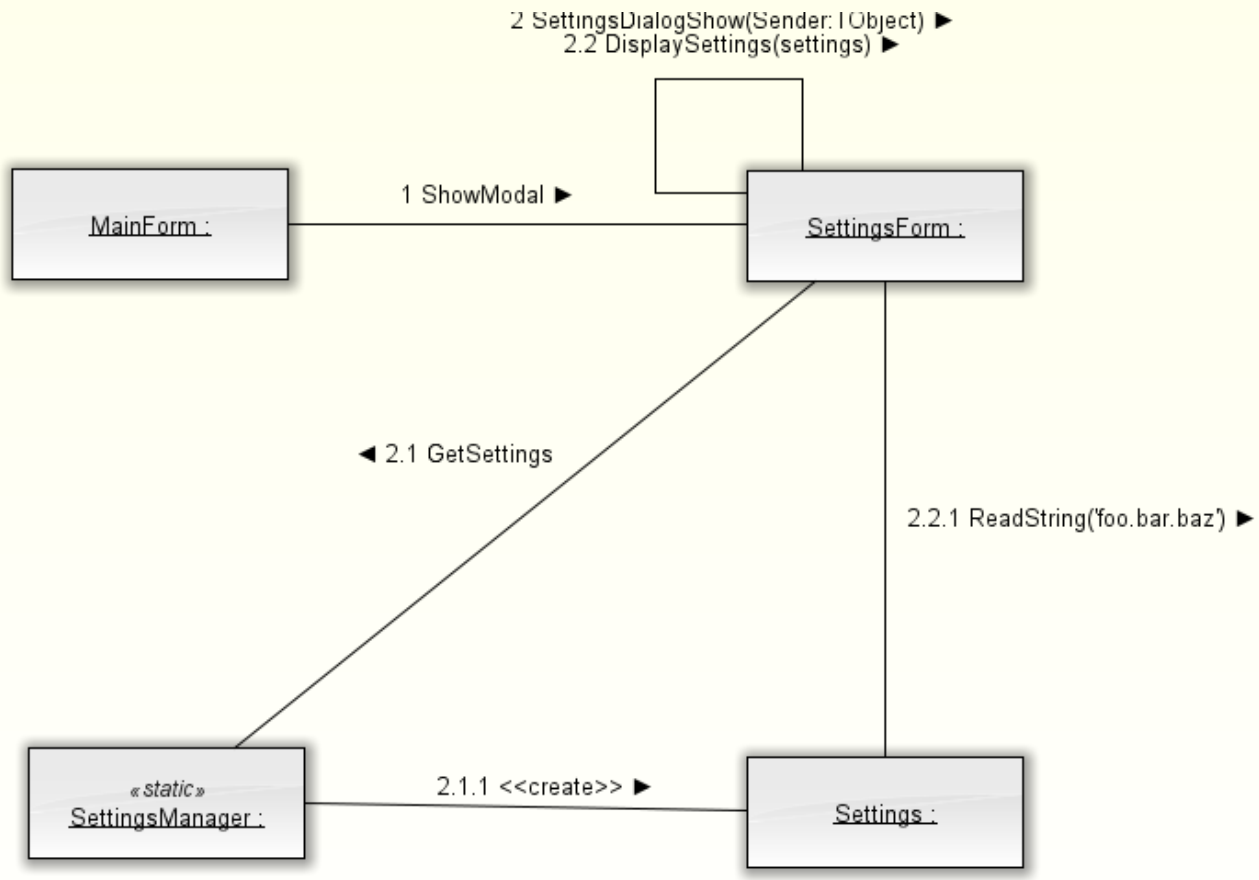
OOP

- Klassen und Instanzen
- Vererbung, Überschreiben, etc.

Objektorientierung

- Generelles Programmierparadigma $\Rightarrow$ Denkweise
- Objekte, die über Nachrichten miteinander kommunizieren
- OOA, OOD, OOP

Objekte kommunizieren



Eine objektorientierte Denkweise

Meine Sichtweise

Objektorientierte Softwareentwicklung ist das ständige Ausbalancieren objektorientierter Prinzipien („Daumenregeln“).

(Meine) oberste Daumenregel

Wenn du meinst, eine Regel brechen zu müssen, tu es. Wundere dich aber nicht über die Konsequenzen.

Eine objektorientierte Denkweise

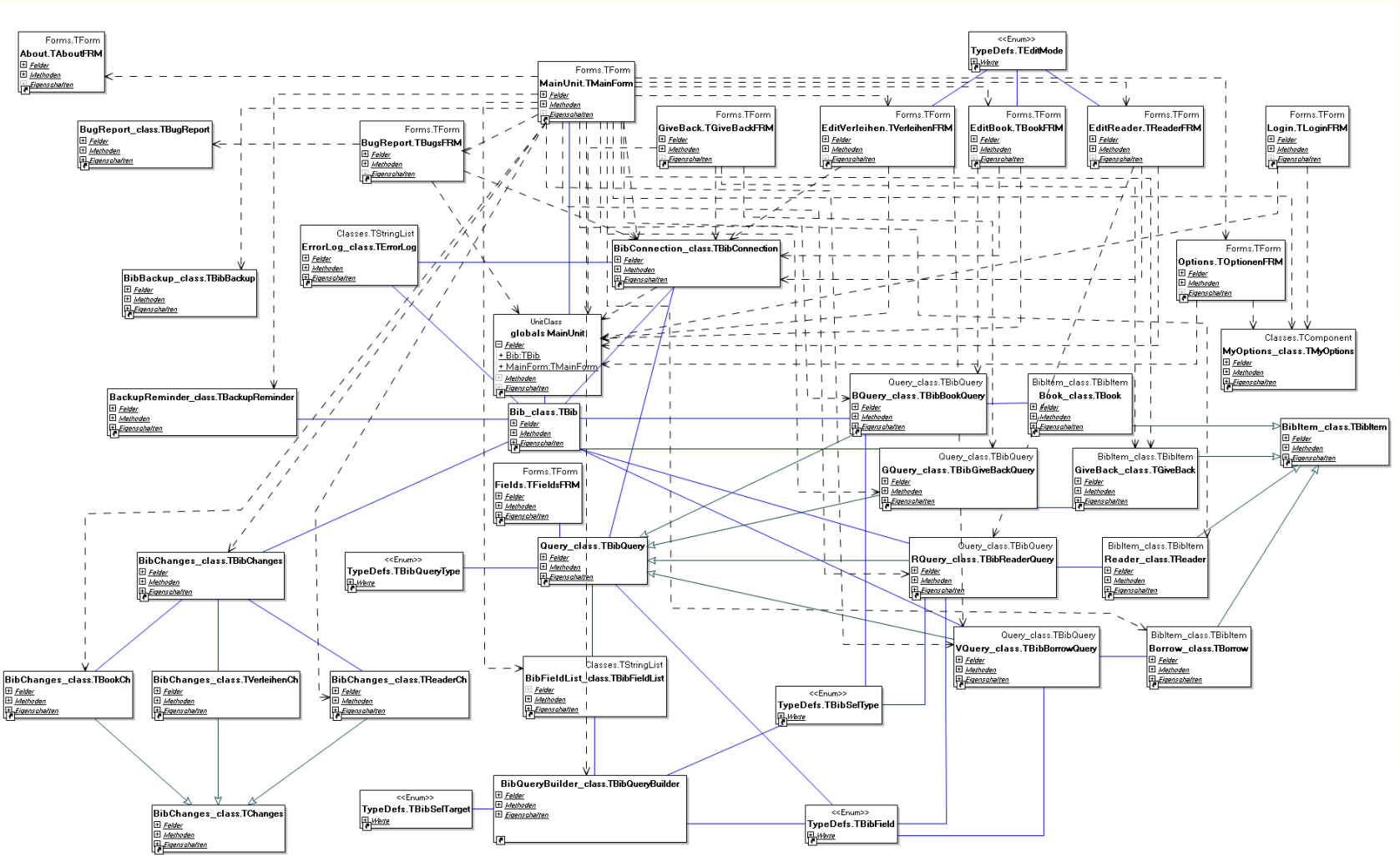
Meine Sichtweise

Objektorientierte Softwareentwicklung ist das ständige Ausbalancieren objektorientierter Prinzipien („Daumenregeln“).

(Meine) oberste Daumenregel

Wenn du meinst, eine Regel brechen zu müssen, tu es. Wundere dich aber nicht über die Konsequenzen.

Ein Negativbeispiel – Mein erstes „OOP“-Programm



Modularisierung

Kapselung und Information Hiding

Objekte sind wie Abwasserleitungen: Ich will, dass sie zuverlässig funktionieren; mit den Details will ich aber nichts zu tun haben.

Natürliche und künstliche Klassen

- OOA: Object Oriented Analysis
  - Fachliche Modellierung der Problemstellung
  - Führt zu „natürlichen“ Klassen
  - Beispiele: TBook, TReader, ...
- OOD: Object Oriented Design
  - Technische Ausgestaltung
  - Technische Aspekte auf natürliche Klassen verteilen
  - Wenn nötig, „künstliche“ Klassen ergänzen

Faustregel

Zuerst einmal vergessen, dass das Problem auch gelöst werden muss.

Natürliche und künstliche Klassen

- OOA: Object Oriented Analysis
  - Fachliche Modellierung der Problemstellung
  - Führt zu „natürlichen“ Klassen
  - Beispiele: TBook, TReader, ...
- OOD: Object Oriented Design
  - Technische Ausgestaltung
  - Technische Aspekte auf natürliche Klassen verteilen
  - Wenn nötig, „künstliche“ Klassen ergänzen

Faustregel

Zuerst einmal vergessen, dass das Problem auch gelöst werden muss.

Natürliche und künstliche Klassen

- OOA: Object Oriented Analysis
  - Fachliche Modellierung der Problemstellung
  - Führt zu „natürlichen“ Klassen
  - Beispiele: TBook, TReader, ...
- OOD: Object Oriented Design
  - Technische Ausgestaltung
  - Technische Aspekte auf natürliche Klassen verteilen
  - Wenn nötig, „künstliche“ Klassen ergänzen

Faustregel

Zuerst einmal vergessen, dass das Problem auch gelöst werden muss.

Pseudo-OOP

- Die Verwendung von Klassen macht einen Entwurf noch nicht objektorientiert
- Häufiges Problem:
  - Übermäßige Verwendung von künstlichen Klassen
  - Alarmsignal: Viele Klassennamen enden auf -er, -or, -Handler, -Manager, ...
- Grund: prozedurales Denken
  - „Ich brauch jetzt etwas, das XY macht.“

Nicht „Wie?“, sondern „Wer?“

Prozedurale Denkweise

Die Lösung des Problems (also das „Wie“) schrittweise beschreiben und Gleichartiges in Prozeduren und Funktionen auslagern.

Objektorientierte Denkweise

Aufgaben auf einzelne Klassen verteilen und Arbeit delegieren. Also „Wer“ (welche Klasse) macht was?

CRC-Karten

- Karteikarten
- Benennen
  - Klasse (Class)
  - Verantwortlichkeiten (Responsibilities)
  - Beziehungen zu anderen Klassen (Collaborations)

TXMLSettings	
Superclass: TSettings	
Subclasses:	
Laden Speichern Lesen von Werten Schreiben von Werten XML-Schema	TSettingsManager TStringList

Aufgaben der Objekte

- Single Responsibility Principle (SRP)
- Separation of Concerns (SoC)
- Information Expert
- Don't repeat yourself! (DRY)

$\Rightarrow$ Eine Klasse sollte genau eine klar definierte Aufgabe haben.

Tell, don't ask! oder: Do It Myself

Prozedural

EVA-Prinzip: Eingabe (Daten holen), Verarbeitung, Ausgabe (Daten schreiben)

Objektorientiert

Objekte entscheiden selbst über ihren Zustand; andere Objekte können Anweisungen erteilen, ändern aber nicht selbst den Zustand

Tell, don't ask! – Ein Beispiel (1/2)

Prozedural

```
if not book.IsLent then
begin
  book.IsLent := true;
  book.LentTo := user;
  book.DueDate := IncDays(Now, 14);
end;
```

Nicht ganz objektorientiert

```
if not book.IsLent then
begin
  book.LendTo(user, IncDays(Now, 14));
end;
```

Tell, don't ask! – Ein Beispiel (1/2)

Prozedural

```
if not book.IsLent then
begin
  book.IsLent := true;
  book.LentTo := user;
  book.DueDate := IncDays(Now, 14);
end;
```

Nicht ganz objektorientiert

```
if not book.IsLent then
begin
  book.LendTo(user, IncDays(Now, 14));
end;
```

Tell, don't ask! – Ein Beispiel (2/2)

Objektorientiert

```
book.LendTo(user, Now); // alles Weitere in LendTo()
```

- Das Buch weiß selbst, wann es zurückgegeben werden muss
- Das Buch fängt selbst den Fall, dass es bereits ausgeliehen ist, ab
  - Wirft ggf. Exception

Beispiel: Datentransformation (1/2)

Prozedural

- Unterschiedliche Arten von Listen für unterschiedliche Daten
- Prozeduren arbeiten auf diesen Daten; füllen Listen, etc.

Prozedural mit Klassen

- Eine Klasse, die die Datentransformation kapselt

Beispiel: Datentransformation (2/2)

OO – einfach

- Verschiedene Klassen für verschiedene Arbeitsschritte

OO – fortgeschritten

- „Pipe and Filter“-Pattern
- Verarbeitungsschritte geschehen in Filter-Klassen
- Pipe-Klassen bilden die Kommunikationswege

Kommunikation

Bindung und Kopplung

Bindung

Bindung (engl. *cohesion*) ist ein Maß für den inneren Zusammenhalt eines Moduls (einer Klasse). Die Bindung sollte immer möglichst hoch sein. $\Rightarrow$ SRP

Kopplung

Kopplung (engl. *coupling*) ist ein Maß für die Abhängigkeiten zwischen den Modulen. Die Kopplung sollte möglichst lose sein, d. h. es sollten möglichst wenige Abhängigkeiten bestehen.

Bindung und Kopplung

Bindung

Bindung (engl. *cohesion*) ist ein Maß für den inneren Zusammenhalt eines Moduls (einer Klasse). Die Bindung sollte immer möglichst hoch sein. $\Rightarrow$ SRP

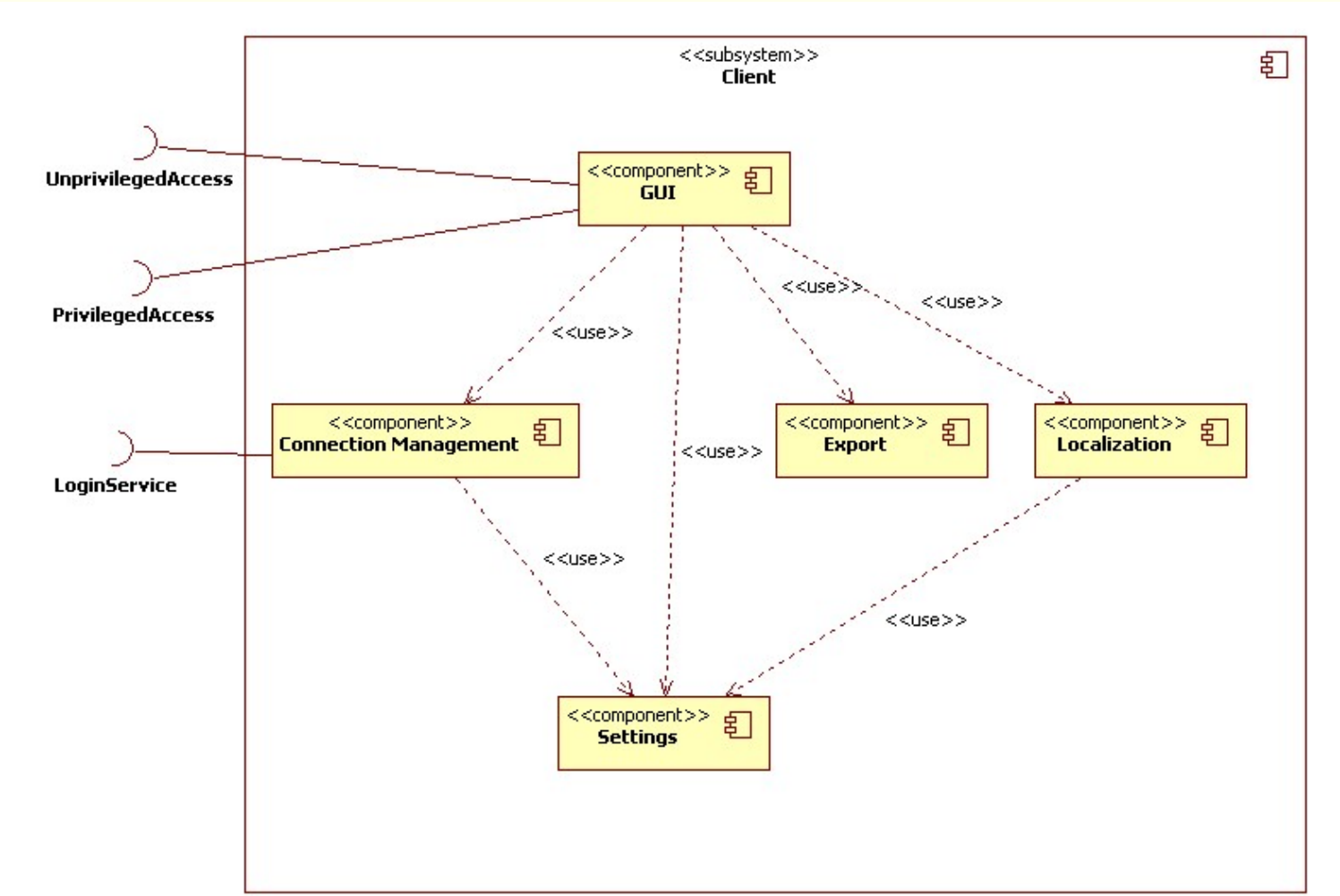
Kopplung

Kopplung (engl. *coupling*) ist ein Maß für die Abhängigkeiten zwischen den Modulen. Die Kopplung sollte möglichst lose sein, d. h. es sollten möglichst wenige Abhängigkeiten bestehen.

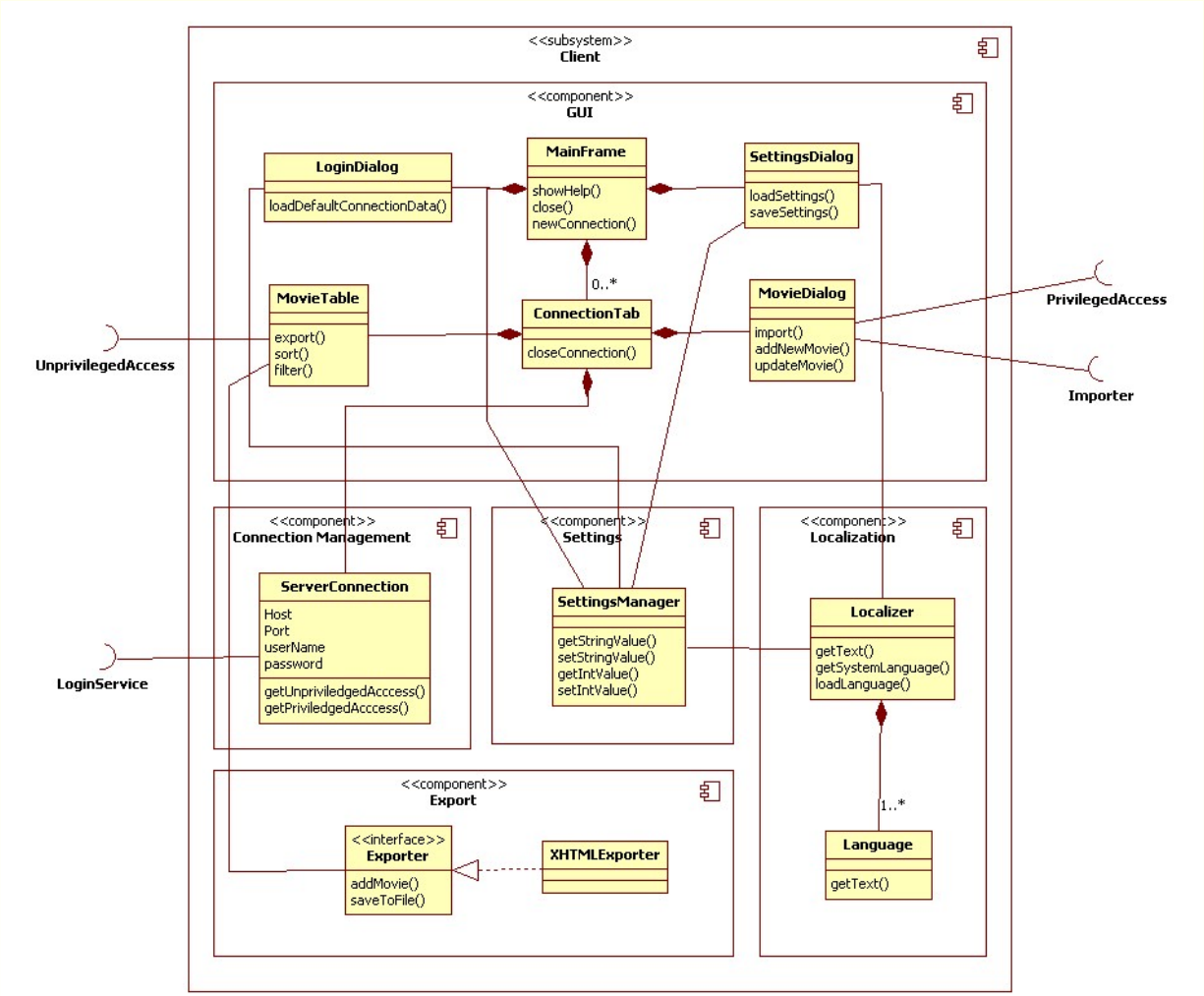
Navigierbarkeit

- Gute Entwürfe schränken die Kommunikation ein
- Wer darf mit wem reden?
- Zirkulare Beziehungen vermeiden
- Abstraktionsschichten
  - Obere Schichten benutzen niedrigere
  - Niemals umgekehrt

Navigierbarkeit - Ein reales Beispiel (1/2)



Navigierbarkeit - Ein reales Beispiel (2/2)



Law of Demeter: „Don't talk to stragers!“

Eine Methode sollte nur „befreundete“ Methoden aufrufen:

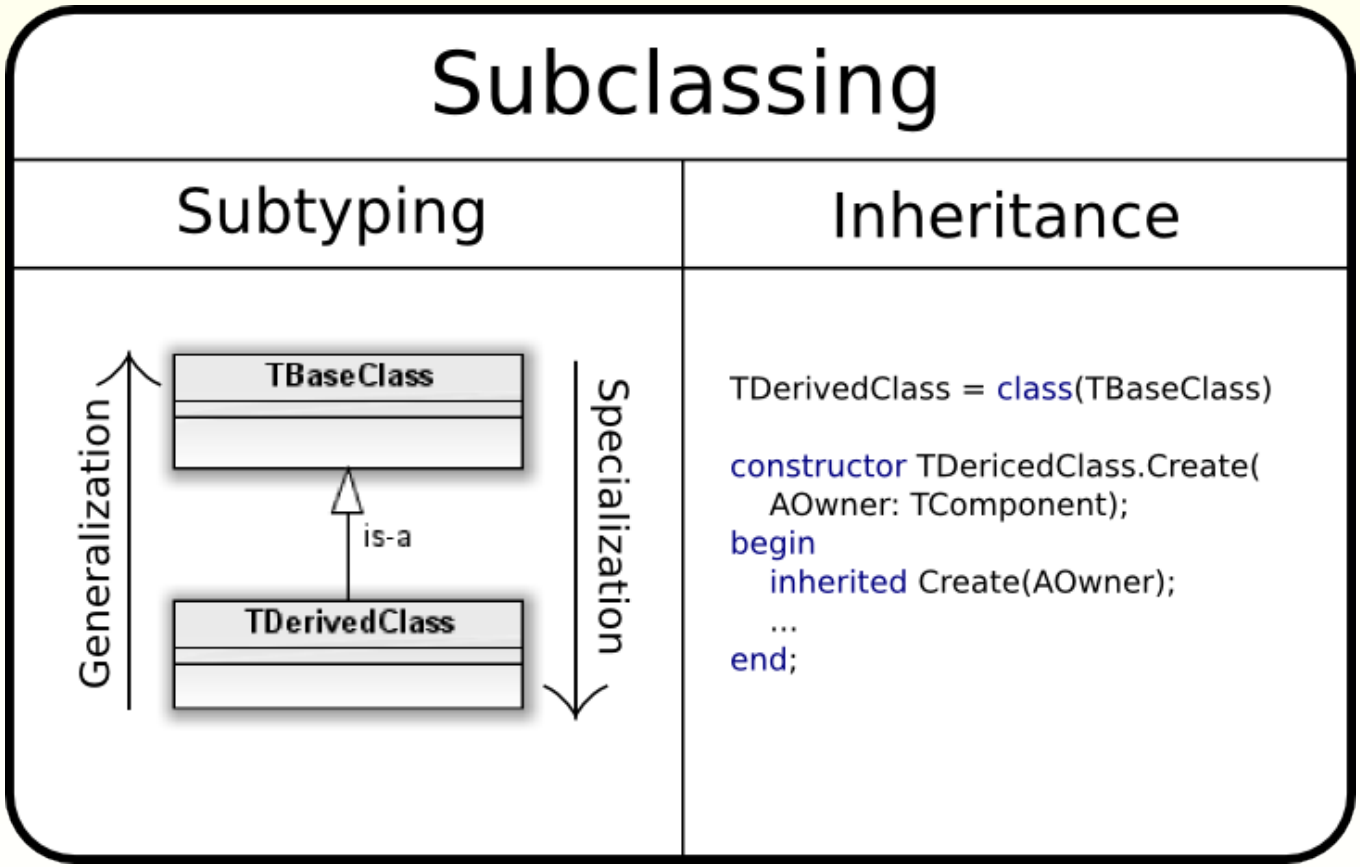
```
TSomeClass = class
  ...
  FField: TFoobar;
  procedure OtherMethod(AParam: Integer);
  ...
end;

procedure TSomeClass.SomeMethod(AParameter: TFoobar);
var
  foobar, obj: TFoobar;
begin
  OtherMethod(42);          // andere Methoden des selben Objekts

  AParameter.DoSomething; // Methoden der Parameter

  FFlield.DoSomething;      // Methoden der Felder

  foobar := TFoobar.Create;
  obj := foobar.DoSomething; // Methoden von selbst erstellten Objekten

//  obj.DoSomething;        // *nicht* Methoden von Rueckgabewerten
//  foobar.foo.bar.baz;     // *keine* Aufrufketten
end;
```

Vererbung

- Ziel der OO ist Wiederverwendbarkeit
- Wiederverwendbarkeit wird durch Vererbung erreicht
- Von verwandten Klassen gemeinsam genutzter Code liegt in den Basisklassen
- Klingt logisch, oder?

Was Vererbung wirklich ist

- Schnittstellenvererbung vs. Implementierungsvererbung

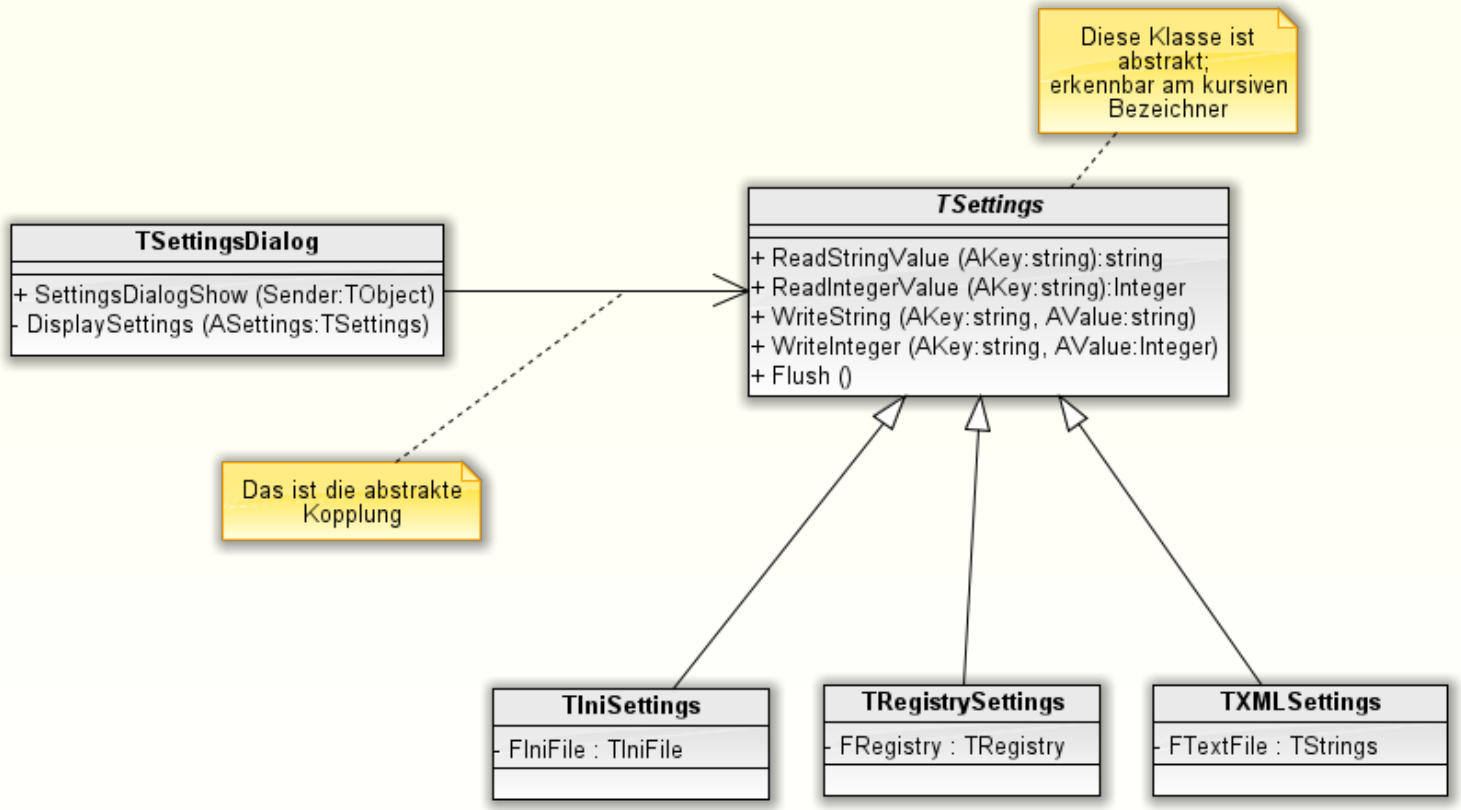


Liskovsches Substitutionsprinzip

- Wenn ein Stück Code Objekte einer bestimmten Klasse erwartet, sollte dieses auch Objekte von Subklassen verarbeiten können.
- $\Rightarrow$ Subklassen sollten sich erwartungsgemäß wie ihre Basisklassen verhalten

Polymorphie und abstrakte Kopplungen

- „If Statement Considered Harmful“
  - Warnsignal: is-Operator



Zusammenfassung

- OOP ist nur ein Teil der OO
- Objektorientierte Programme bestehen aus miteinander kommunizierenden Objekten

- Es gibt natürliche und künstliche Klassen
- Eine Klasse sollte genau eine Aufgabe haben

- Starke Bindung, lose Kopplung
- Vererbung ist nur ein Nebenprodukt von Generalisierung/Spezialisierung

Fragen?

Anhang

Dependency Injection

Selbst erzeugen

```
constructor TFoo.Create;
begin
  inherited Create;
  FBar := TBar.Create;
end;
```

Dependency Injection

```
constructor TFoo.Create(ABar: TBar); // Alternativ auch ueber Property
begin
  inherited Create;
  FBar := ABar;
end;
```

- Dadurch kann von außen eine Subklasse übergeben werden

LSP – Beispiel

- Mengen und Multimengen
- Eine Multimenge ist eine Menge, die Elemente mehrfach enthalten kann
  - Problem bei Remove(): Die Annahme, dass das Element nach dem Entfernen nicht mehr enthalten ist stimmt nicht mehr
- Eine Menge ist eine Multimenge, die keine Elemente Doppelt enthalten kann
  - Problem bei Add(): Die Annahme, dass sich Count erhöht, stimmt nicht mehr
- Was ist die Vereinigungsmenge, Schnittmenge, etc. einer Menge mit einer Multimenge?

$\Rightarrow$ Mengen und Multimengen nicht voneinander ableiten, sondern gemeinsame Basisklasse nutzen.