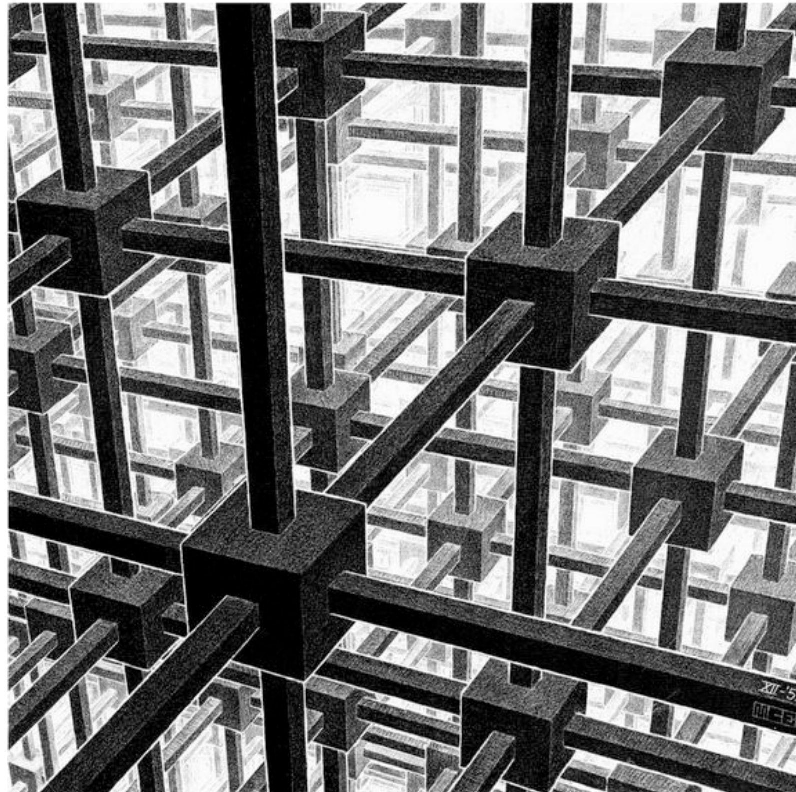
ywesee
intellectual capital connected

Facharbeit Mike Walder



Erweiterung für das ODBA Framework

Februar 2005

Inhaltsverzeichnis

1. UMFELD UND ABLAUF ... 3

1.1. Aufgabenstellung ... 3
1.1.1. Detailierte Aufgabenstellung ... 3
1.1.2. Die konkrete User-Story ... 3
1.2. Vorkenntnisse ... 4
1.3. Vorarbeiten ... 4
1.4. Firmenstandards ... 4
1.4.1. Filesystem ... 4
1.4.2. File-Namen ... 5
1.4.3. Code-Formatierung ... 5
1.4.4. Namensgebung ... 6
1.4.5. eXtreme-Programming Grundsätze (die Wichtigsten) ... 7
1.5. Zeitplan ... 8
1.6. Arbeitsjournal ... 9
1.7. Einführung in die ODBA ... 13
1.7.1. Anmerkung zu den Klassendiagrammen ... 13
1.7.2. Was ist die ODBA? ... 13
1.7.3. ER-Diagramm der ODBA Datenbank ... 14
1.7.4. Simplifiziertes Klassendiagramm der ODBA ... 14
1.7.5. Serialisierung eines verknüpften Persistable Objektes ... 14
1.8. ODBA vs. Prevalyer ... 16

2. PROJEKT ... 16

2.1. Anmerkung zu extremeProgramming ... 16
2.2. Planung und Verfeinerung des Auftrages ... 16
2.2.1. Meine Erweiterung ... 16
2.2.2. Klassendiagramm ... 16
2.2.3. Was muss beachtet werden? ... 17
2.3. Konzept (Spikes) ... 17
2.3.1. Auslesen der Klassenvariablen - 1. Spike ... 18
2.3.2. Auslesen der Klassenvariablen - 2. Spike ... 19
2.3.3. Wiederherstellung von Klassenvariablen ... 21
2.4. Realisation ... 21
2.4.1. Die Demoapplikation ... 21
2.4.2. Einbau meiner Erweiterung in die ODBA ... 22
2.4.3. Einbau meiner Erweiterung in die oddb.org Applikation ... 22
2.4.4. Das Migrationsscript ... 24
2.5. Tests ... 25
2.5.1. oddb.org Applikation ... 25
2.6. Anwendung, Erweiterung und Limitation meiner Lösung ... 25
2.6.1. Anwendung ... 25
2.6.2. Limitation ... 26
2.6.3. Erweiterung ... 26

ANHANG ... 26

2.7. Glossar ... 26
2.8. Literaturverzeichnis ... 28
2.9. Code Anhang ... 29
2.9.1. test_class_var_arsenal.rb ... 29
2.9.2. class_var_arsenal.rb ... 32
2.9.3. test_class_var_container.rb ... 33
2.9.4. class_var_container.rb ... 34
2.9.5. demo_model.rb ... 35

1. Umfeld und Ablauf

1.1. Aufgabenstellung

Die ODBA-Library ist ein inhouse entwickelter Objekt-Cache für die objektorientierte Skriptsprache Ruby. Ihr Hauptziel ist es, objektorientierte Applikationen transparent persistieren zu können. oddb.org ist eine Webapplikation, die diverse Daten in der Applikationsdomain Gesundheitswesen sammelt und veröffentlicht. oddb.org wurde ursprünglich nach dem Prevalenz-Prinzip (www.prevayler.org) entwickelt. Da bei diesem Ansatz Klassenvariablen nicht persistiert werden, mussten z.B. intern vergebene Ids als Aggregatwerte über alle Instanzen initialisiert werden. Bei der Migration von oddb.org zur ODBA-Library taucht nun das Problem auf, dass zu keinem garantierten Zeitpunkt alle vorhandenen Instanzen einer Klasse tatsächlich instantiiert sind. Aggregatfunktionen über alle Instanzen sind somit nicht mehr möglich.

1.1.1. Detailierte Aufgabenstellung

Entwickle einen Mechanismus innerhalb der ODBA-Library, mittels dem über beliebig viele (zur Demonstration und Erfüllung der Aufgabe genügen drei) konsekutive Instanzen einer Applikation sequentielle Ids vergeben werden können.
Statische (Klassen-) Variablen sind im vorliegenden Fall nicht persistent, d.h. darin gespeicherte Werte gehen bei einem Neustart der Applikation verloren. Gehe ausserdem davon aus, dass der Extent einer Klasse (also die Sammlung aller aktiven Instanzen dieser Klasse) nicht verfügbar ist, dass also zu einem Zeitpunkt 't' nicht alle aktiven Instanzen einer Klasse geladen sind.

1.1.2. Die konkrete User-Story

Programmierer Ümit arbeitet an der Oddb.org-Plattform. Er stellt fest, dass seit der Migration zum ODBA-Backend gespeicherte User-Feedbacks überschrieben werden und findet heraus, dass bei jedem Neustart der Id-Zähler für die Feedbacks wieder bei 0 beginnt. Ümit braucht einen Mechanismus, der ihm ermöglicht, diesen Bug zu reparieren.

Finde zwei verschiedene Lösungsansätze, vergleiche sie falls nötig mit Hilfe von Spikes und entscheide dich für eine der Lösungen. Implementiere die gewählte Lösung nach "Test driven Design"-Regeln. Schreibe also vor jedem Programmier-Schritt einen Unit-Test. Überprüfe, dass der Test _nicht_ besteht (also einen Fall testet, der noch nicht korrekt implementiert ist), und mache dich dann an die Implementation, bis der Test besteht.
Dokumentiere Integrations- und Acceptance-Tests unabhängig der Unit-Tests. Die Dokumentation ist bei eXtreme-Programming im Code inhärent - achte deshalb darauf, die Namensvergabe für Klassen, Variablen und Methoden so zu gestalten, dass der Code für (Weiter-)Entwickler

selbsterklärend wird, dass also anhand des Namens einer Methode darauf geschlossen werden kann, wofür die Methode verantwortlich ist.

Integriere die neue Version der ODBA-Library in die oddb.org-Applikation und zeige, dass Feedbacks wieder korrekt gespeichert werden können.

1.2. Vorkenntnisse

Mitarbeit bei Design und Entwicklung der ODBA-Library. Ruby als hauptsächliche Programmiersprache, die während des gesamten Praktikumsjahrs verwendet wurde. Intensive Auseinandersetzung mit dem objektorientierten Programmieren (Patterns, objektorientierte Systeme persisiteren).

1.3. Vorarbeiten

Sowohl die ODBA-Library als auch die oddb.org Applikation wurden während des Praktikumsjahrs entwickelt bzw. weiterentwickelt und werden auf dem Produktionsserver eingesetzt.

1.4. Firmenstandards

ywesee -- intellectual capital connected

Coding-Standards
Version 1.0, 15.02.2005.

1.4.1. Filesystem

ywesee orientiert sich bezüglich Filesystem am Ruby-Standard:

Projekt-name	Pfad	Bemerkungen
	/bin/executable-files	(executables können auch ruby-files sein)
	/ext/require-pfad/c-extension-files	
	/lib/require-pfad/library-files	(alle Ruby-Files enden mit '.rb')
	/test/test-files	(alle Test-Files beginnen mit 'test_')

Zusätzlich möglich:

Projekt-name	Pfad	Bemerkungen
	/src/code-files	(Applikations-Code, der nicht als Library verwendet werden kann)
	/src/subsystem-name/code-files	(bei grösseren Projekten)
	/test/'test_' + subsystem-name/test-files	(bei grösseren Projekten, Files heissen gleich wie Code-Files)
	/doc/	(Document-Root bei Webprojekten)
	resources	(statische Web-Resources, z.B. CSS-Files, Images etc.)
	/etc/konfigurations-files	

Begründung:

- Standard-Compliance macht es z.B. einfacher ein Installations-Script zu erstellen.

1.4.2. File-Namen

Grundsätzlich gilt:

- 1 File pro implementierter Klasse (mehrere Klassen pro File nur in Ausnahmefällen, z.B. für Subklassen mit geringer Behaviordifferenz)
- File.name = Class.name.downcase + .rb
- TestFile.name = 'test_' + File.name

Ausnahme:
Bei grösseren Projekten, bei denen der Code in Subsystem-Directories abgelegt ist, beginnen Subsystem-Test-Directories mit 'test_'. Dann gilt TestFile.name = File.name

Begründung:

- Ruby erzwingt keinerlei Beschränkungen für die Nomenklatur. Trotzdem ist es für das Verständnis des Code-Lesers sinnvoll, schon im Filesystem eine Verbindung zwischen Code und Tests zu sehen.
- Um die require-Pfade eindeutig zu halten, haben Test-Files oder Subsystem-Test-Directories das Präfix 'test_'

1.4.3. Code-Formatierung

- Klassen: CamelCase:
  `Bsp: class CodeStandardPolice`
- Konstanten: UPPERCASE, underscores für Worttrennung::
  `Bsp: DOCUMENT_AUTHOR = 'hwyss'`
- Methoden und Variablen: lowercase, underscores für Worttrennung
  ```
  Bsp: def example_method()
  Bsp: my_var = 10
  ```
- Konditions-Ausdrücke, Methoden-Argumente (Definition und Aufruf) immer in Klammern:
  ```
  Bsp: if(foo == bar)
  Bsp: def my_method(arg1, arg2)
  Bsp: my_object.my_method('foo', 'bar')
  ```
- Abstand vor und nach Operators:
  ```
  Bsp: 1 + 2
  Bsp: rule_number = 15
  ```
- Abstand nach Komma:
  `Bsp: my_object.my_method('foo', 'bar', 'baz')`
- Block-Definitionen: Geschweifte Klammern (Abstand vorher), Abstand vor und nach der Block-Argument-Definition:
  `Bsp: hash.each { |key, value| printf("%i => %s", key, value) }`

- Mehrzeilige Block-Definitionen: Einrückung um ein Tab, Block-Argumente auf der Ursprungszeile, abschliessende geschweifte Klammer auf neuer Zeile und auf ursprünglicher Höhe.
```
Bsp: hash.each { |key, value|
        printf("%i => %s", key, value)
 }
```
- HashTable-Literals: Wie Block-Definitionen, bei mehrzeiligen Hash-Literals Pfeil-Syntax und Werte horizontal aligniert
```
Bsp: table = {
          1234      => 'simple',
          385636213 => 'a little more complicated',
     }
```
- Lange Code-Zeilen werden umgebrochen: Immer Backslash am Ende der Zeile, auch wenn vom Syntax nicht gefordert. Folgezeilen um ein Tab eingerückt.
- Bei Konditions-Ausdrücken werden logische Operatoren auf die neue Zeile genommen.
```
Bsp: @a_long_variable_name.and_a_very_long_method_name(with, many, \
          arguments, to, boot)
Bsp: if(my_condition_variable == MY_CONDITION_CONSTANT \
          && my_condition_method(my_condition_variable))
```

Begründung:

- Lesbarkeit in erster Linie eine Frage der Konsistenz und erst in zweiter Linie einer Frage der Formatierung. Die Regeln sind so gewählt, dass das Auge genügend Whitespace zur Orientierung erhält und der Editor keine zusätzlichen Zeilenumbrüche einfügen muss.

1.4.4. Namensgebung

- Getter-Methoden heissen gleich wie die Instanzvariable auf die sie sich beziehen:
```
Bsp: class Foo
       def initialize
          @bar = 23
       end
       def bar
          @bar
       end
     end
```

(Dies ist ein Beispiel. Im Production Code wird für ein reines Auslesen die 'attr_reader' Klassenmethode verwendet.)

- Setter-Methoden heissen gleich wie die Instanzvariable, auf die sie sich beziehen, mit dem Suffix '=':
```
Bsp: class Foo
       def bar=(bar_value)
          @bar = bar_value
       end
     end
```

(Dies ist ein Beispiel. Im Production Code wird für ein reines Zuweisen die 'attr_writer' Klassenmethode verwendet.)

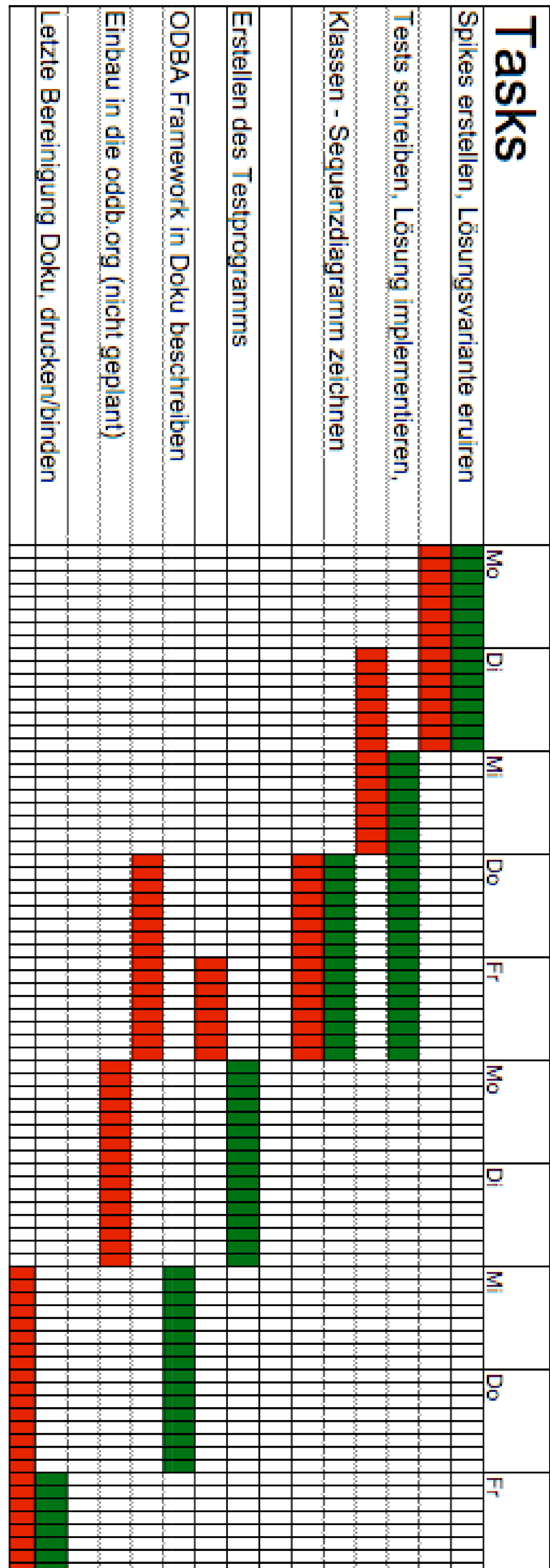
- Keine Namenswiederholung. Eine Klasse 'Person' hat eine Getter-Methode 'name' und nicht 'person_name'.
- Descriptive Names: Namen von Klassen, Methoden und Variablen beschreiben die Funktion/Responsibility des Codes in Englisch.

1.4.5. eXtreme-Programming Grundsätze (die Wichtigsten)

- Für jede neue Funktionalität wird ein Unit-Test (und gegebenenfalls ein Integrations-Test) erstellt, bevor mit der Implementation begonnen wird.
- Do the simplest thing that could possibly work. (Die Antizipation von Kundenbedürfnissen durch den Entwickler und sogar durch den Kunden selbst führt meistens in die falsche Richtung).
- Refactor mercilessly: Code-Duplikation eliminieren, unpräzise Namensgebungen ändern.

1.5. Zeitplan

Anmerkung: Das Arbeitsjournal und die Dokumentation wurden täglich nachgeführt.



1.6. Arbeitsjournal

14. Februar 2005

Ausgeführte Arbeiten
Heute habe ich mich mit dem Erstellen der Spikes beschäftigt. Ich analysierte 2 verschiedene Lösungsansätze. Beim Ersten wurde mir relativ schnell klar, dass diese Lösung nicht zum gewünschten Ziel führen wird. Der Zweite war erfolgsversprechend und erfüllte meine Anforderungen. Konkret ging es darum herauszufinden, in welcher Klasse oder in welchem Mixin eine Klassenvariable definiert war.

Erreichte Ziele
Ich habe mich für einen Lösungsansatz entschieden. Erstes Gespräch mit dem Fachexperten.

Aufgetretene Probleme
Weil Mixins nicht in der direkten Klassenhierarchie sind, wurden die Klassenvariablen dieser Mixins zum Teil einer falschen Klasse zugeordnet.

Vergleich mit Zeitplan
Die Spikes habe ich schneller als geplant fertiggestellt. Ich werde aber bei später auftretenden Problemen vielleicht weitere Spikes erstellen müssen.

15. Februar 2005

Ausgeführte Arbeiten
Ich habe heute ein paar Tests geschrieben. Einen weiteren Spike für das Setzen von Klassenvariablen habe ich auch erstellt. Mit der Dokumentation habe ich ebenfalls begonnen. Die Überlegungen in den Spikes habe ich vermerkt. Ich habe mit der Implementation begonnen.

Erreichte Ziele
Problemstellung mit Spikes gelöst. Konnte mit dem Implementieren beginnen.

Aufgetretene Probleme
Deadlock in der ODBA wegen Datenbanktransaktion beim Speichern des `ClassVarArsenals`. Ich habe das `ClassVarArsenal` an einem anderen Ort gespeichert.

Vergleich mit Zeitplan
Im Moment bin ich dem Zeitplan voraus, weil ich schon mit dem Implementieren des richtigen Codes begonnen habe.

16. Februar 2005

Ausgeführte Arbeiten
Heute habe ich meine Klassen ausprogrammiert. Dabei habe ich bemerkt, dass unser Framework unter gewissen Umständen Collections fehlerhaft abspeichert. Ich habe das Problem mit Hannes besprochen, wir konnten es relativ schnell lösen. Ein paar kleine Umstellungen im `CacheServer` waren nötig.

Erreichte Ziele
ODBA Code Implementiert

Aufgetretene Probleme
Ich habe einen Fehler im Framework gefunden, zuerst dachte ich, dass das Problem durch eine Fehlüberlegung in meinem Code verursacht wurde. Anfänglich wurden nicht alle Objekte der ClassVarArsenal Klasse persistent gespeichtert, ich musste mehere `odba_store` Aufrufe einfügen.

Vergleich mit Zeitplan
Ich bin einen Tag voraus, da ich mit der Implementation schon fertig bin.

17. Februar 2005

Ausgeführte Arbeiten
Heute habe ich meine UML Kenntnisse aufgefrischt. Ich habe ein Klassendiagramm meiner Abschlussarbeit und der ODBA erstellt. Ein Sequenzdiagramm eines Speichervorgangs habe ich ebenfalls gezeichnet. Ein paar Begriffe habe ich im Glossar beschrieben.

Erreichte Ziele
Klassendiagramme erstellt

Aufgetretene Probleme
X-Server auf Linux strapaziert meine Nerven, er stürzt oft ab.

Vergleich mit Zeitplan
Ich bin immer noch ein bisschen voraus, die Diagramme benötigen aber mehr Zeit als angenommen.

18. Februar 2005

Ausgeführte Arbeiten
Ich habe mein Demoprogramm mit einer Konsoleneingabe erweitert und den Code überarbeitet. An der Doku habe ich auch weitergearbeitet (Diagramme, Glossar).

Erreichte Ziele
Demoprogram läuft.

Aufgetretene Probleme
Keine.

Vergleich mit Zeitplan
Ich bin einen Tag voraus. Ich werde voraussichtlich meine Erweiterung noch in die oddb.org einbauen

21. Februar 2005

Ausgeführte Arbeiten
Heute habe ich meine Erweiterung in die oddb.org eingebaut. Eigentlich war es nicht schwierig. Für die Migration habe ich ein Script geschrieben, welches die Anzahl Objekte einer Klasse zusammenzählt und diesen Wert als Klassenvariable speichert.
Mit Hannes habe ich eine kurze Code-Review durchgeführt. Er war der Meinung, dass die Klassenvariablen nicht auch beim Speichern eines Objektes mitgespeichert werden sollen, da sonst dieser Vorgang nicht transparent sei. Man muss nun eine Klasse explizit speichern, damit ihre Klassenvariablen persistent werden.

Erreichte Ziele
Einbau in die oddb.org fertiggestellt.

Aufgetretene Probleme
Legacycode im Persistence Mixin hat die Klassenvariablen falsch zugeordnet (dem Mixin selber anstatt der Klasse, in der das Mixin eingefügt ist). Diesen Teil habe ich umgeschrieben.

Vergleich mit Zeitplan
Durch den Einbau in die oddb.rog habe ich ein bisschen mehr Zeit benötigt. Ich liege im Zeitplan, ich bin noch etwa ½ Tag voraus.

22. Februar 2005

Ausgeführte Arbeiten
Heute habe ich mich der Dokumentation gewidmet. Ich habe den Einbau in die oddb.org beschrieben, sowie meine Acceptance Tests, welche ich manuell ausgeführt habe.

Erreichte Ziele
Doku erweitert, Acceptancetest durchgeführt und dokumentiert.

Aufgetretene Probleme
Word-Terror mit Excel-Tabelle.

Vergleich mit Zeitplan

Ich liege jetzt im Zeitplan, das Dokumentieren benötigt mehr Zeit als angenommen.

23. Februar 2005

Ausgeführte Arbeiten
Ich habe die Diagramme überarbeitet und in der Dokumentation den Abschnitt *Anwendung, Erweiterung und Limitation meiner Lösung* hinzugefügt

Erreichte Ziele
Doku erweitert, Überlegungen zur Erweiterung meiner Lösung notiert.

Aufgetretene Probleme
Keine.

Vergleich mit Zeitplan
Ich bin im Zeitplan.

24. Februar 2005

Ausgeführte Arbeiten
Gestern Abend habe ich nochmals meine Dokumentation durchgelesen. Ich habe noch diverse Rechtschreibefehler korrigiert. Die Arbeit meines Mitpraktikanten habe ich ebenfalls nocht durchgelesen und korrigiert. Im Glossar habe ich noch ein Diagramm eines Mixins gezeichnet.

Erreichte Ziele
Dokumentation angepasst.

Aufgetretene Probleme
Keine.

Vergleich mit Zeitplan
Ich bin im Zeitplan

25. Februar 2005

Ausgeführte Arbeiten
Ich habe noch ein Diagramm einer Klassenhierarchie gezeichnet.

Erreichte Ziele
Dokumentation fertig.

Aufgetretene Probleme
Keine.

Vergleich mit Zeitplan
Ich bin im Zeitplan.

1.7. Einführung in die ODBA

1.7.1. Anmerkung zu den Klassendiagrammen

In den folgenden Klassendiagrammen sind alle Instanzvariablen public. Ruby erlaubt Zugriffe auf Instanzvariablen nur via Methoden. Jede Variable, welche von aussen zugänglich sein muss, wird in der Klassendefinition mit einem `attr_accessor` vermerkt. Dies spart das Ausprogrammieren von Setter- und Gettermethoden. Die Entkapsulierung wird somit immer gewährleistet.

Beispiel:

```
class Foo
   attr_accessor :bar_var
end
```

Ist gleich wie:

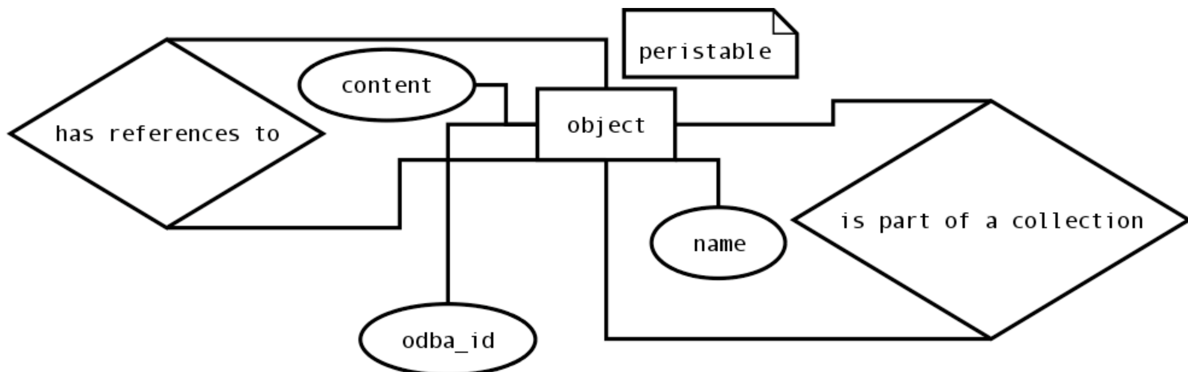
```
class Foo
   def bar_var
     @bar_var
   end
   def bar_var= (val)
     @bar_var = val
   end
end
```

Da mein Projekt eine Erweiterung für die ODBA ist, stelle ich nachfolgend kurz das ODBA Framework vor.

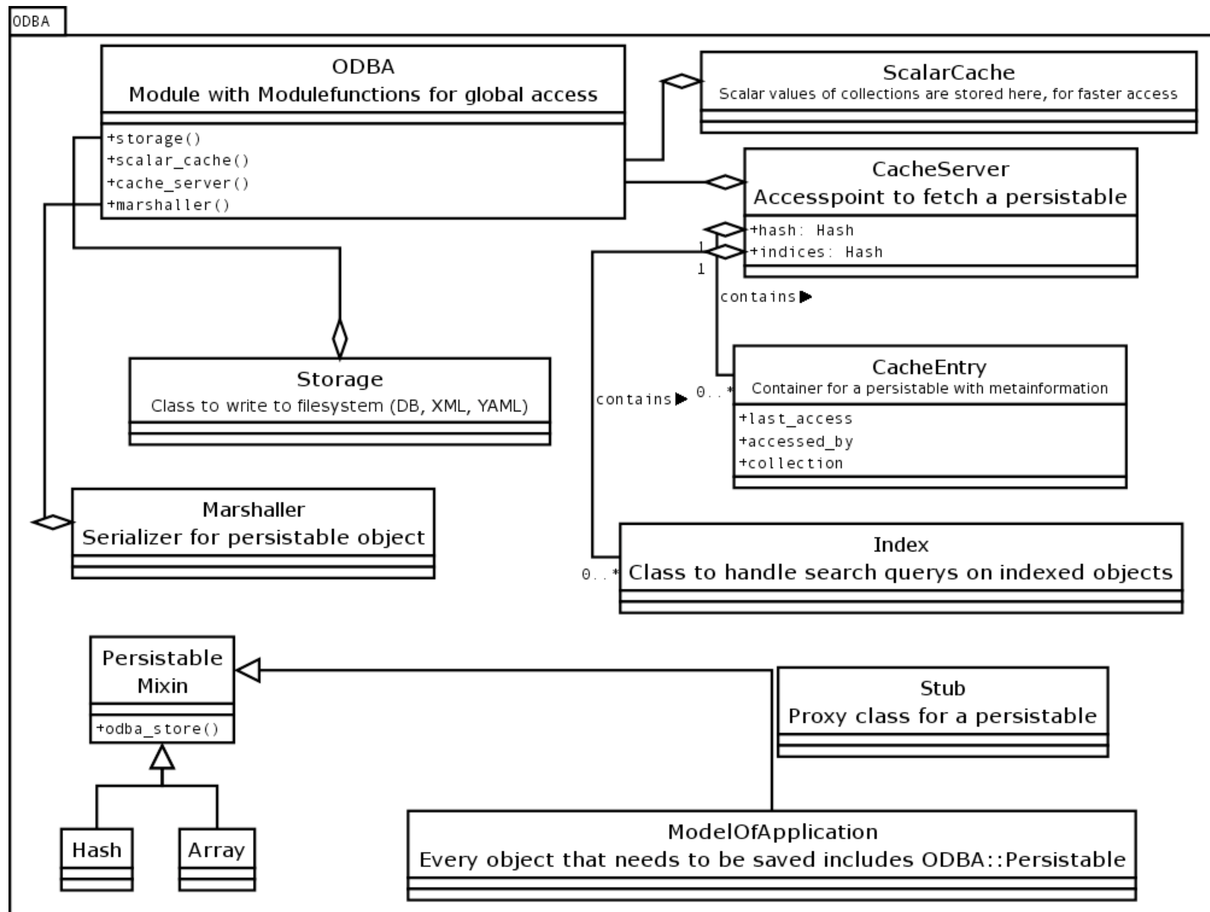
1.7.2. Was ist die ODBA?

Die ODBA ist ein Object-Cache für Applikationen, welche in Ruby geschrieben sind. Im Gegensatz zu einem DB-Abstraktionslayer (z.B. Torque) werden die Daten aber nicht in ein relationales DB-Schema gespeichert. Die Objekte werden zwar in eine DB gespeichert, doch existieren nur drei Tabellen `object`, `collection` und `object_connection`. Theoretisch wäre es auch möglich, die serialisierten Objekte in YAML oder XML abzuspeichern.

1.7.3. ER-Diagramm der ODBA Datenbank



1.7.4. Simplifiziertes Klassendiagramm der ODBA

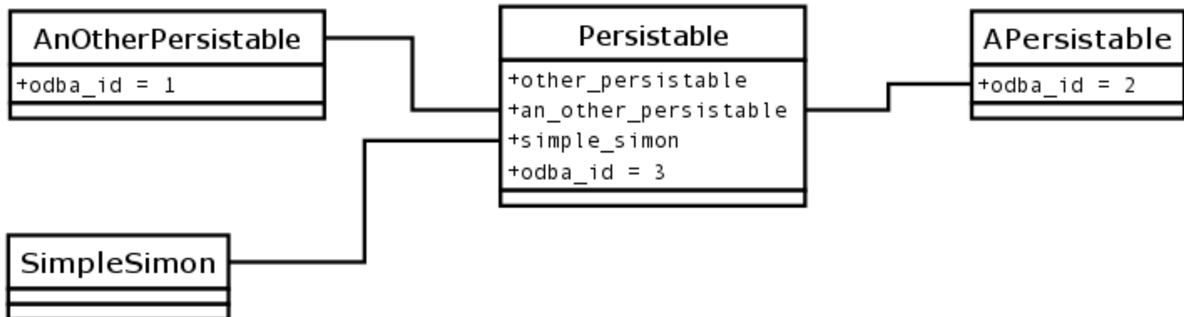


1.7.5. Serialisierung eines verknüpften Persistable Objektes

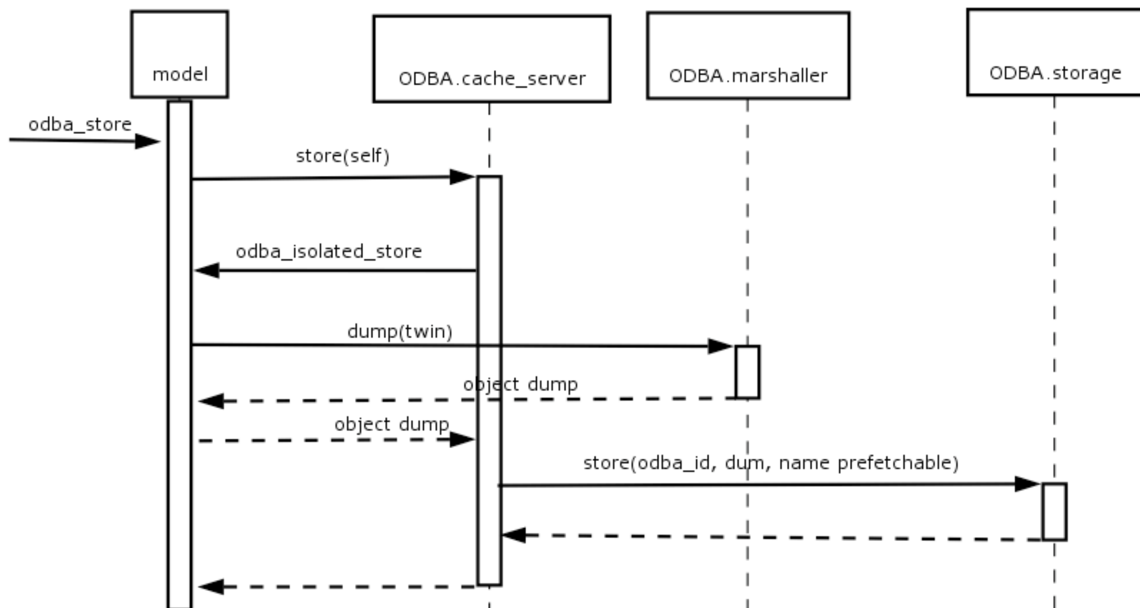
Das zu serialisierende Objekt wird als erstes dupliziert, dabei werden nur die Instanzvariablen, welche auf persistente Objekte zeigen durch Stubs ersetzt, alle anderen Instanzvariablen werden serialisiert. Allfällige Instanzvariablen, die auf Kollektionen (Arrays, Hashes) zeigen, werden auseinandergenommen, die einzelnen Elemente (Key, Value) werden serialisiert oder durch Stubs ersetzt und in die `object_collection` Tabelle

abgelegt. Die leere Kollektion wird in der `object` Tabelle gespeichert. Beim Laden wird das Objekt mit Stubs in den Speicher geholt. Sobald ein Aufruf auf ein Stub erfolgt, lädt dieser das richtige Objekt nach.

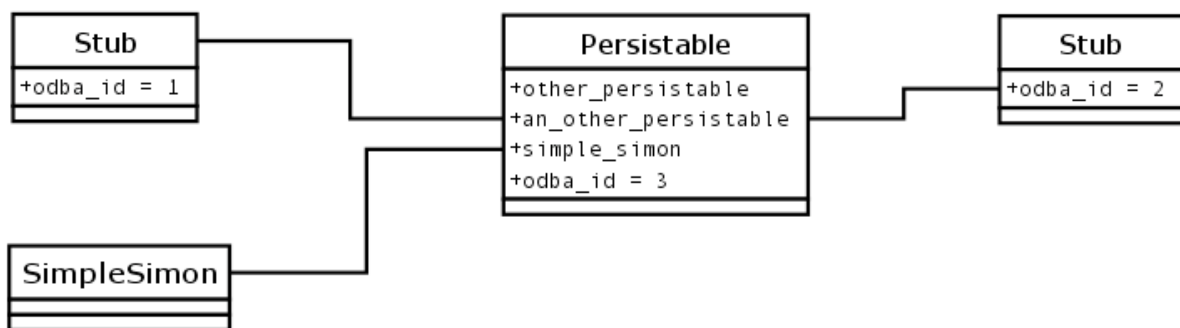
Zustand eines Objektes zur Laufzeit der Applikation:



Erhält das Persistable die odba_store Nachricht, passiert folgender Ablauf:



Zustand des Objektes, wie es in der Datenbank abgelegt wird:



Wie man hier sieht, wurden die zwei persitenten Objekte (Apersistable und AnotherPersistable) abgetrennt und durch Stubs ersetzt. Das Objekt der Klasse SimpleSimon wurde mit dem Objekt der Klasse Persistable mitserialisiert.

1.8. ODBA vs. Prevalyer

Im Gegensatz zum Prevayler Ansatz sind in der ODBA nicht immer alle Objekte im Arbeitsspeicher. Objekte, die für eine gewisse Zeitspanne nicht benötigt werden, entfernt die ODBA wieder aus dem Speicher und ersetzt sie durch Stubs. Durch die Migration von Prevayler zum ODBA Framework konnten wir so ein paar hundert MB Arbeitsspeicher freimachen.

2. Projekt

2.1. Anmerkung zu extremeProgramming

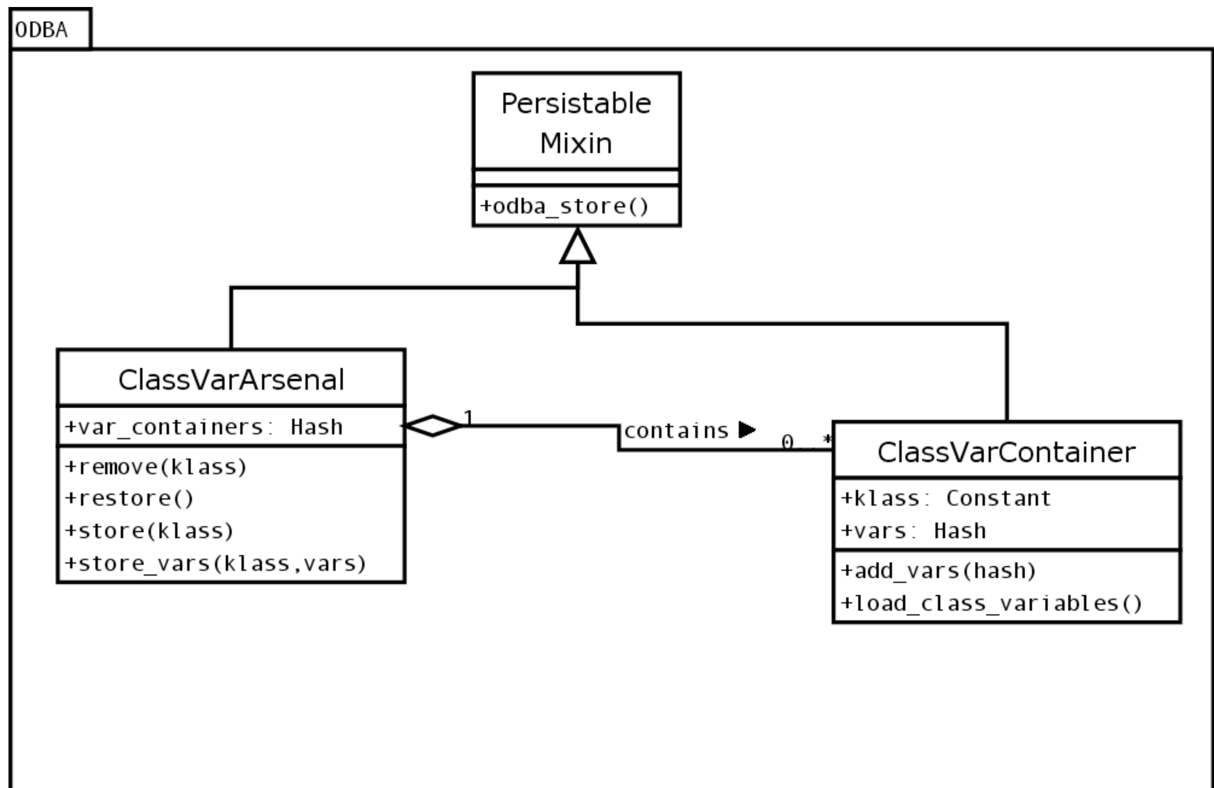
Mein Projekt wir nach dem extremeProgramming Ansatz durchgeführt. Bei dieser Vorgehensweise schreibt man von Anfang an Code. Die Dokumentation wird auf ein Minimum reduziert. Da extremeProgramming meistens mit Test-First kombiniert wird, ist ein Grossteil der Dokumentation eines Systems in den Unittests enthalten.

2.2. Planung und Verfeinerung des Auftrages

2.2.1. Meine Erweiterung

Meine Erweiterung soll Klassenvariablen persistent abspeichern. Als Datenstruktur verwende ich einen Hash, welcher als Schlüssel den Klassennamen und als Wert die Klassenvariablen enthält. Die Klassenvariablen werden im `ClassVarContainer` entkapsuliert.

2.2.2. Klassendiagramm

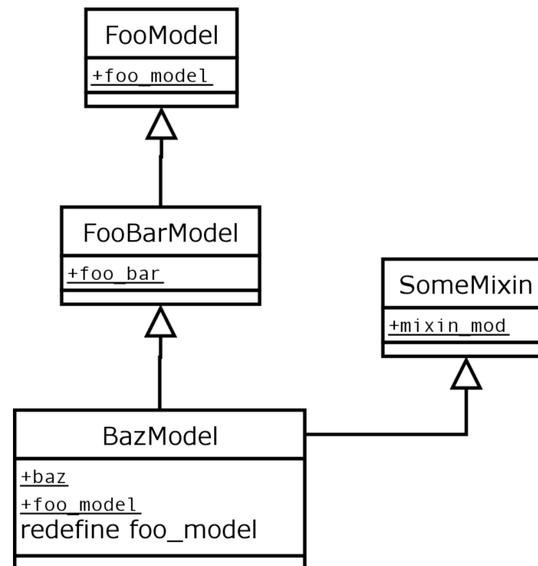


Im ODBA Framework wird das `class_var_arsenal` Objekt eine Instanzvariable des `cache_servers` sein.

2.2.3. Was muss beachtet werden?

Beim Speichern der Klassenvariablen muss darauf geachtet werden, dass die Klassenvariablen in derjenigen Klasse gespeichert werden, in der sie auch definiert sind. Die Hierarchie muss ebenso berücksichtigt werden wie Mixins.

Beispiel einer Klassenhierarchie:



Die `foo_model` Variable darf also nur unter dem FooModel abgelegt werden, obwohl sie in der BazModel Klasse neu definiert wurde.

2.3. Konzept (Spikes)

Um den Lösungsansatz meines Problems kurz zu skizzieren, habe ich ein paar Spikes erstellt.

Ich wollte wissen, wie ich in einer Hierarchie herausfinde, in welcher Klasse eine Klassenvariable definiert ist. Wichtig ist, dass jede Klassenvariable nur ein Mal für alle Klassen, die sie benutzen, abgespeichert wird.

Für die Spikes habe ich ein paar Klassen und Module erstellt (Foo, Bar, Baz, Persistable etc.). Sie würden in einem richtigen System die Model Klassen, welche persistent sind, repräsentieren. Die Klasse `ClassVarArsenal` ist zuständig für das Speichern der Klassenvariablen

Ich möchte hier ausdrücklich darauf hinweisen, dass ich `eval` in den folgenden Spikes bewusst verwendet habe, denn grundsätzlich gilt: *eval is evil*. Leider lässt sich das Setzen und Lesen von Klassenvariablen in Ruby 1.8 aber nur so lösen. Ruby 1.9 bietet setter und getter Methoden für Klassenvariablen (`class_variable_set, class_variable_get`). Die Version 1.9 ist aber noch experimental und wird nicht für eine produktive Umgebung empfohlen. Ruby bietet verschiedene `eval` Funktionen (`eval,`

class_eval, instance_eval, module_eval), die den Code Scopebezogen ausführen. Eval zum Beispiel führt den Code auf der Ebene des globalen Objectspace aus, class_eval hingegen nur für die Klasse, in der die Methode aufgerufen wird. Ich habe darauf geachtet, dass ich jeweils auf den Scope bezogen, die restriktivste eval Funktion verwende.

2.3.1. Auslesen der Klassenvariablen – 1. Spike

```
module Persistable
  def odba_class_values
    c_vars = Hash.new
    self::class::class_variables.each { |c_var|
      c_vars[c_var.to_s] = self::class::class_eval("#{c_var}")
    }
    c_vars
  end
end

class ClassVarArsenal
  attr_accessor :class_vars
  def initialize
    @class_vars = {}
  end
  def store(obj)
    vars = obj.odba_class_values
    @class_vars[obj::class] = vars
  end
end

module Models
  class Foo
    include Persistable
    @@class_var = "class_var"
    def initialize
    end
  end
  class Bar < Foo
    @@class_var = "new Val"
    @@bar_var = "bar val"
    def initialize
    end
  end
end

c_cache = ClassVarArsenal.new
foo = Models::Foo.new
c_cache.store(foo)
bar = Models::Bar.new
c_cache.store(bar)
#produces duplicate class vars, which is wrong
puts c_cache.class_vars.inspect
```

Der Output sieht wie folgt aus:

```
{Models::Bar=>{"@@class_var"=>"new Val", "@@bar_var"=>"bar val"},
Models::Foo=>{"@@class_var"=>"new Val"}}
```

Wie man sieht, ist diese Datenstruktur redundant. Die Klassenvariable @@class_var wird zwei Mal abgespeichert. In der Klasse Foo sowie in der Subklasse Bar. In einem objektorientierten System ist die Klassenvariable aber nur ein Mal für die ganze Hierarchie vorhanden.

2.3.2. Auslesen der Klassenvariablen – 2. Spike

```
module Persistable
  @@some_odba_class_persistable = "persistable"
end

class ClassVarArsenal
  attr_accessor :class_vars
  def initialize
    @class_vars = {}
  end
  def store(obj)
    class_hierarchy = obj::class::ancestors
    class_hierarchy.each_with_index { |klass, idx|
      puts "*****Inspecting #{klass}*****"
      bloodline_vars = klass.class_variables
      if(bloodline_vars.empty?)
        puts "no more vars in legacy, terminating iteration.."
        break
      end
      puts "all class vars in bloodline"
      puts bloodline_vars
      puts "class vars for this class only"
      if(parent = class_hierarchy[idx+1])
        class_vars = bloodline_vars - parent.class_variables
        extract_vars(klass, class_vars)
      end
      puts "***FINISHED CLASS #{klass}***"
    }
  end
  def extract_vars(klass, class_vars)
    class_vars.each { |c_var|
      @class_vars[klass]  = class_vars
    }
  end
end

module Models
  class Foo
    include Persistable
    @@foo_class_var = "foo class var"
  end
  class Bar < Foo
    @@foo_class_var = "overwrite foo value"
    @@bar_class_var = "bar class var"
    @@bar_class_var2 = "bar2"
  end
end

c_arsenal = ClassVarArsenal.new
bar = Models::Bar.new
foo = Models::Foo.new
c_arsenal.store(foo)
c_arsenal.store(bar)
puts "content of ClassVarArsenal"
puts c_arsenal.class_vars.inspect
```

Der Output sieht wie folgt aus:

```
*****Inspecting Models::Foo*****
all class vars in bloodline
@@foo_class_var
@@some_odba_class_persistable
class vars for this class only
***FINISHED CLASS Models::Foo***
*****Inspecting Persistable*****
all class vars in bloodline
@@some_odba_class_persistable
class vars for this class only
***FINISHED CLASS Persistable***
*****Inspecting Object*****
no more vars in legacy, terminating iteration..
*****Inspecting Models::Bar*****
all class vars in bloodline
@@bar_class_var
@@bar_class_var2
@@foo_class_var
@@some_odba_class_persistable
class vars for this class only
***FINISHED CLASS Models::Bar***
*****Inspecting Models::Foo*****
all class vars in bloodline
@@foo_class_var
@@some_odba_class_persistable
class vars for this class only
***FINISHED CLASS Models::Foo***
*****Inspecting Persistable*****
all class vars in bloodline
@@some_odba_class_persistable
class vars for this class only
***FINISHED CLASS Persistable***
*****Inspecting Object*****
no more vars in legacy, terminating iteration..
content of ClassVarArsenal
{Models::Bar=>["@@bar_class_var", "@@bar_class_var2"],
Persistable=>["@@some_odba_class_persistable"],
Models::Foo=>["@@foo_class_var"]}
```

Wie zu erkennen ist, ist jede Klassenvariable nur ein Mal vorhanden und zwar in der Klasse, in der sie ursprünglich definiert wurde. Den Ansatz dieses Spikes werde ich für meine Lösung verwenden.

2.3.3. Wiederherstellung von Klassenvariablen

Diesen Spike habe ich für die Wiederherstellung der Klassenvariablen erstellt.

```
module Baz
  class Bar
  end
end
class ClassVarDef
  attr_accessor :name, :value
end
module Foo
end
puts "class vars of Bar"
puts Baz::Bar.class_variables.inspect
c_def = ClassVarDef.new
c_def.name = "@@foo_bar"
c_def.value = "foo bar Var"
class_vars = {
  Baz::Bar => c_def
}
klass = Baz::Bar
var = class_vars[Baz::Bar].name
value = class_vars[Baz::Bar].value
code = ("#{var} = \"#{value}\"")
klass.class_eval(code)
puts "class var of Bar after restoration"
puts Baz::Bar.class_variables.inspect
```

Output:

```
class vars of Bar
[]
class var of Bar after restoration
["@@foo_bar"]
```

Wie man hier sieht, wird die Klassenvariable der Klasse wieder hinzugefügt. Ich habe wiederum `class_eval` benutzt nicht `eval`, damit nicht unter Umständen der ganze Objectspace verändert werden könnte.

2.4. Realisation

2.4.1. Die Demoapplikation

Um meine Lösung in einem kleinen Rahmen zu testen, habe ich eine Demoapplikation mit Kommandozeileneingabe geschrieben. Der Code befindet sich im Anhang (demo_model.rb). Beim „Spielen“ mit dieser Applikation habe ich einen Fehler im ODBA Framework gefunden. Eine Optimierung des Speicherns für Collections, die vor meiner Facharbeit geschrieben wurde, funktionierte beim ersten Abspeichern der Collection falsch.

2.4.2. Einbau meiner Erweiterung in die ODBA

Meine Implementation erfolgte auf Basis des zweiten Spikes. Ich musste den Algorithmus zum Auslesen der Klassenvariablen für Mixins noch anpassen. Der Einbau in das ODBA Framework erfolgte in der Klasse `CacheServer`. Ich habe hier eine `store_class` und `class_var_arsenal` Methode hinzugefügt. Grundsätzlich soll der Benutzer der ODBA nur mit dem `CacheServer` kommunizieren. Ich habe deshalb folgende Facade Methoden hinzugefügt: `restore_class_variables` und `remove_class_variables(AClass)`.

Code Ausschnitte:

```
def remove_class_variables(klass)
      self.class_var_arsenal.remove(klass)
end

def restore_class_variables
      self.class_var_arsenal.restore
end

def store_direct(object)
      if(object.is_a?(Module))
        store_class_vars(object)
      else
        store_object(object)
      end
end

def class_var_arsenal
      @class_var_arsenal ||= fetch_named('__class_var_arsenal__', self){
        ClassVarArsenal.new
      }
end
```

2.4.3. Einbau meiner Erweiterung in die oddb.org Applikation

Änderungen musste ich vor allem im `ODDB:Persistence` Mixin vornehmen. Jedes Model erhält ein `ODDB::Persistence` als Mixin. Im `ODDB::Persistence` Mixin ist wiederum das `ODBA::Persistable` Mixin eingefügt. Dieser Aufbau ist legacy-bedingt, damit wir die Applikation und Daten nahtlos vom Prevayler-System migrieren konnten.

Das `ODDB::Persistence` Mixin erweitert jede Modelklasse durch eine Methode `next_oid`. Diese Methode inkrementiert die Klassenvariable `@@oid` und weist den Wert der Instanzvariable `@oid` zu. Die Methode next_oid wurde in der ursprünglichen Applikation mit `instance_eval` hinzugefügt. Dies hatte die Wirkung, dass die Klassenvariable `@@oid` im Persistence Mixin definiert wurde und nicht in der entsprechenden Model Klasse. Dieses Verhalten ist falsch, weil der Scope des Zählers für die jeweilige Klasse und nicht für alle Models gelten soll. Mit `class_eval` verhält sich die Methode richtig.

Ausschnitt des Codes im `ODDB::Peristence` Mixin:

```
def set_oid
      self.class.class_eval <<-EOS
unless(self.class.respond_to?(:next_oid))
        class << self
          def next_oid
            @@oid ||= 0
            @@oid = @@oid.next
            puts self
            puts @@oid
            ODBA.cache_server.store(self)
          end
        end
      EOS
      @oid ||= self.class.next_oid
 end
```

Beim Starten der Applikation habe ich noch folgende Zeile hinzugefügt:

```
(oddbapp.rb)
def initialize
      ODBA.cache_server.prefetch
      ODBA.cache_server.restore_class_variables
      @system = ODBA.cache_server.fetch_named('oddbapp', self) {
         OddbPrevalence.new
      }
      puts "init system"
      @system.init
      puts "setup drb-delegation"
      super(@system)
      puts "reset"
      reset()
      puts "system initialized"
end
```

Dieser Aufruf bewirkt, dass die Klassenvariablen und ihre Werte in die Klassendefinitionen geladen werden.

2.4.4. Das Migrationsscript

Folgendes Script liest alle Objekte aus der Datenbank und nimmt die höchste oid einer Instanz und speichert diese dann als Klassenvariable der betreffenden Klasse ab. Dieser Schritt ist nötig, da wir als zwischenzeitliche Lösung die `odba_id` benutzt haben und die Daten konsistent übernommen werden sollen. Dieses Script muss vor dem Start der Applikation mit meiner Erweiterung ausgeführt werden.

```
max = ODBA.storage.max_id
$stdout.flush
puts <<-EOS
#### Staring OID rebuild ####
EOS
class_vars = {}
1.upto(max) { |id|
  if(dump = ODBA.storage.restore(id))
    obj = ODBA.marshaller.load(dump)
    if(obj.respond_to?(:oid) && obj.oid && \
      class_vars[obj.class].to_i < obj.oid)
        puts obj.class
        class_vars[obj.class] = obj.oid
    end
  end
  print " " * 8
  print "\b" * 8
  print "#{sprintf('%7.3f', id.to_f/max.to_f*100.0)}%"
  print "\b" * 8
  $stdout.flush
}
puts "storing class vars"
class_vars.each { |name, val|
  vars = {
    '@@oid' => val
  }
  ODBA.cache_server.class_var_arsenal.store_vars(name, vars)
}
puts "storing Arsenal"
ODBA.cache_server.class_var_arsenal.odba_store
puts "contents"
puts ODBA.cache_server.class_var_arsenal.inspect
```

2.5. Tests

Beim extremeProgramming Ansatz wird schon vor dem eigentlichen Programm Testcode geschrieben. Die Testklassen befinden sich im Code Anhang. Hier habe ich einige manuelle Acceptancetests durchgeführt.

2.5.1. oddb.org Applikation

Folgende Tests sind erfolgreich durchgeführt worden.

Testfall	**Ausgeführte Tätigkeit**
Sind alle Klassenvariablen nach einem Neustart in den Klassen definiert?	Admin-Tool Eingabe: `Feedback.class_variables`
Aufnahme eines Feedbacks, wurde die Klassevariable inkrementiert?	Via Webseite ein Feedback gepostet, dann Output in der Konsole betrachtet.
Neustart der Applikation nach Aufnahme des Feedbacks, Überprüfung der Klassenvariable, wurde sie persistent gespeichert?	Applikation neu gestartet, Feedback gepostet. Output in der Konsole angeschaut.
Sind keine Klassenvariablen im Persistence Mixin definiert? Wichtig, weil sonst das Iterieren über die Klassenhierarchie fehlerhaft wäre (wurde zuvor schon in den Unittests ausgiebig getestet).	Admin Tool Eingabe: `Persistence.class_variables`

2.6. Anwendung, Erweiterung und Limitation meiner Lösung

2.6.1. Anwendung

- Möchte man in einer anderen Applikation, die auf der ODBA basiert Klassenvariablen persistieren, so muss man folgende Zeile im Code aufrufen:
  ```
  ODBA.cache_server.store(AClass)
  ```
- Beim Neustart lädt man die Klassenvariablen folgendermassen:
  ```
  ODBA.cache_server.restore_class_variables
  ```
- Möchte man die Klassenvariablen einer Klasse entfernen, muss Folgendes eingegeben werden:
  ```
  ODBA.cache_server.remove_class_variables(Aclass)
  ```

2.6.2. Limitation

Laut Aufgabenstellung muss meine Lösung lediglich skalare Datentypen persistent speichern können (*sequenzielle ids*). Im Moment ist es nicht möglich, andere persistente Objekte als Klassenvariablen zu speichern, es würde zu Objektduplikationen kommen, da die Werte der Variablen nicht durch Stubs ersetzt, sondern direkt serialisiert werden. Beim Laden der Variable wird ein `inspect` auf den Wert aufgerufen. Dies ist nötig, weil es sonst bei einem String mit Code oder Sonderzeichen zu Syntaxfehlern kommen würde.

2.6.3. Erweiterung

Die Klassenvariablen werden im Moment via `class_eval` gesetzt. Sobald Ruby 1.9 in einer stabilen Version vorliegt, würde ich vorschlagen, diesen Teil des Codes mit `class_var_set` und `class_var_get` zu ersetzen. Somit wäre es dann auch möglich, dass Klassenvariablen auf andere persistente Objekte zeigen und man diese so abspeichern kann.

Anhang

2.7. Glossar

Admin-Tool
Bash Eingabe, in der man direkt Ruby Code eingeben kann, um die oddb.org Applikation zu bearbeiten.

Cache Server
Die `CacheServer` Klasse ist ein Delegator auf einen Hash und enthält das Singleton Mixin. Verwaltet das Speichern und Laden von persistenten Objekten, führt Indices nach.

Prevayler
Mit Hilfe des Command-Patterns und Serialisierung werden Objekte direkt gespeichert. Das Command-Pattern verhilft ausserdem zu einer sauberen Struktur, z.B. Trennung zwischen den Applikationsschichten nach MVC (Model-View-Controller). Transaktionen sind damit automatisch und transparent inklusive. Man benötigt keine Datenbank.

- Vorteile:
  Keine DB nötig, es wird alles mit Programmiersprache erledigt, schnell, da alles im RAM abläuft, transparente und saubere Entwicklung
- Nachteile:
  Die Grenze liegt beim RAM. Beim Starten der Anwendung müssen alle Objekte neu in das RAM eingelesen werden. Das kann lange dauern.

ODBA
Framework, welches Objekte persistent abspeichert.

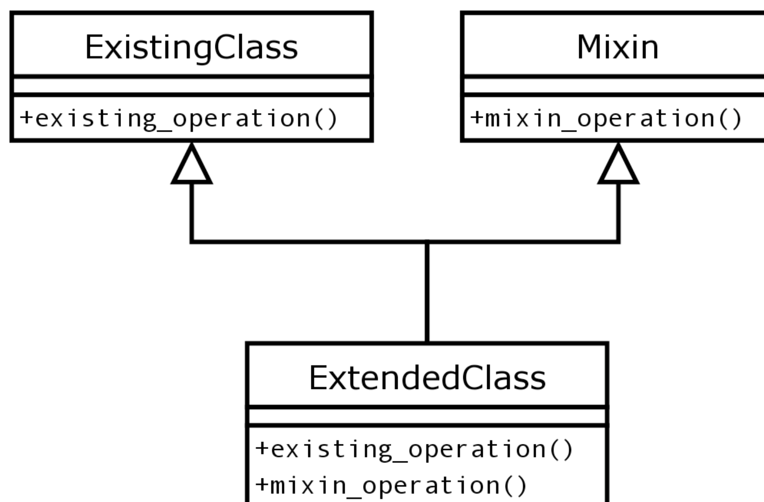
odba_id
Eindeutige Nummer eines persistenten Objektes, wird vom ODBA Framework vergeben.

Persistentes Objekt
Ein Objekt, welches nach einem Neustart der Applikation noch vorhanden ist. Durch Einfügen des `ODBA::Persistable` Moduls, kann eine Klasse diese Funktionalität erhalten. In der oddb.org Aplikation sind z.B. alle Models persistent.

Legacycode
Code aus dem alten System, der aus Gründen der Kompatibilität zu anderen Systemen übernommen werden muss.

Marshal
Der Marshaller in Ruby erlaubt es, Objekte als Bytestream abzuspeichern (Serialisierung).

Mixin
Modul, welches einer Klasse zusätzliche Funktionalität verleiht. Ruby bietet die multiple Vererbung in Form von Mixins an.



Ruby
Objektorientierte Scriptsprache.

Spike
Eine sehr kleine Applikation, die in der Regel erstellt wird, um mit einer Technologie vertraut zu werden oder die Effizienz von verschiedenen Lösungsansätzen zu vergleichen. Spikes werden normalerweise nach Gebrauch gelöscht, sollen aber im vorliegenden Prüfungsfall zu Dokumentationszwecken behalten werden.

Stub
Eine Implementation des Proxy Patterns. Da im ODBA Framework nicht alle Objekte zur Laufzeit im Speicher sind, verwenden wir Stubs als Platzhalter und Zugriffskontrolle für persistente Objekte.

YAML
Yet another markup language. Maschinenlesbare Serialisierungssprache – ähnlich wie XML.

2.8. Literaturverzeichnis

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:
  Design Patterns: Elements of reusable object oriented software
  Addison Wesley Publishing Company 1994, ISBN 0-201-63361-2

- Bernd Oestereich
  Objektorientierte Softwareentwicklung
  Analyse und Design mit der Unified Modeling Language
  Oldenbourg Verlag 4. Auflage

- www.ruby-doc.org
  Ruby 1.8 Referenz

2.9. Code Anhang

Folgende Klassen habe ich von Grund auf geschrieben. Der Einbau in die ODBA ist im Kapitel *Einbau meiner Erweiterung in die oddb.org Applikation* beschrieben.

2.9.1. test_class_var_arsenal.rb

```
#!/usr/bin/env ruby
# TestClassVarArsenal -- odba -- 14.02.2005 -- mwalder@ywesee.com

$: << File.expand_path('../lib/', File.dirname(__FILE__))

require 'test/unit'
require 'odba'

class CountingStub
   def initialize
       @called = {}
   end
   def called?(meth)
       @called[meth]
   end
   def method_missing(meth, *args, &block)
       @called[meth] = @called[meth].to_i.next
   end
   def __define(meth, result)
       instance_variable_set("@#{meth}", result)
       eval <<-EOS
           def #{meth}(*args)
               @#{meth}
           end
       EOS
   end
end
module OtherModule
   @@other_module = "other module"
end
module TestClassPersistent
   include OtherModule
   @@class_persistent = "test class persistent"
end
module Void
end
module OtherPerstistent
   @@other_persistent = "other persistent"
end
class TopClass
   def odba_store
   end
   @@top_class = 'top class'
end
class Baz < TopClass
   @@baz_class_var = "baz class var"
end
class Foo < Baz
   include TestClassPersistent
   include OtherPerstistent
   include Void
```

```
   @@foo_class_var = "foo class var"
end
class Bat
end
module BatModule
end
module TestPersistence
       def set_oid
           self.class.class_eval <<-EOS
unless(self.class.respond_to?(:next_oid))
               class << self
                   def next_oid
                       @@oid ||= 0
                       @@oid = @@oid.next
                   end
               end
           EOS
           @oid ||= self.class.next_oid
       end
end
class FooFeedback
   include TestPersistence
       @@foo_var = 'foo'
   def initialize
   end
end
module ODBA
   class TestClassVarArsenal < Test::Unit::TestCase
       def setup
           ODBA.storage = CountingStub.new
           ODBA.cache_server = CountingStub.new
           @class_var_arsenal = ODBA::ClassVarArsenal.new
       end
       def test_store_vars
           vars = {
               '@@baz_var' => 'baz var value'
           }
           @class_var_arsenal.store_vars(Baz, vars)
           class_var_container = @class_var_arsenal.var_containers[Baz]
           assert_instance_of(ODBA::ClassVarContainer, class_var_container)
       end
       def test_store
           @class_var_arsenal.store(Foo)
           arsenal = @class_var_arsenal
           #puts @class_var_arsenal.var_containers.inspect
           foo = {
               "@@foo_class_var" => "foo class var"
           }
           assert_equal(foo, arsenal.var_containers[Foo].vars)
           baz = {
               "@@baz_class_var" => "baz class var"
           }
           assert_equal(baz, arsenal.var_containers[Baz].vars)
           other_persistent = {
               "@@other_persistent" => "other persistent"
           }
           assert_equal(other_persistent,
arsenal.var_containers[OtherPerstistent].vars)
           test_class_persistent = {
               "@@class_persistent" => "test class persistent"
           }
           assert_equal(test_class_persistent,
arsenal.var_containers[TestClassPersistent].vars)
```

```
			other_module = {
				"@@other_module" => "other module"
			}
			assert_equal(other_module,
arsenal.var_containers[OtherModule].vars)
		top_class = {
			'@@top_class' => 'top class'
		}
		assert_equal(top_class, arsenal.var_containers[TopClass].vars)
		end
		def test_restore_vars_class
			c_container = ODBA::ClassVarContainer.new(Bat)
			c_container.vars = {
				"@@bat_var" => 'value of bat var'
			}
			assert_equal([],Bat.class_variables)
			@class_var_arsenal.var_containers[Bat] = c_container
			@class_var_arsenal.restore
			assert_equal(['@@bat_var'], Bat.class_variables)
		end
		def test_restore_vars_module
			c_container = ODBA::ClassVarContainer.new(BatModule)
			c_container.vars = {
				"@@bat_module_var" => "value of bat module"
			}
			assert_equal([], BatModule.class_variables)
			@class_var_arsenal.var_containers[BatModule] = c_container
			@class_var_arsenal.restore
			assert_equal(['@@bat_module_var'], BatModule.class_variables)
		end
		def test_store_class_vars2
			@class_var_arsenal.store(FooFeedback)
			expected = {
				'@@foo_var' => 'foo'
			}
			assert_equal(expected,
@class_var_arsenal.var_containers[FooFeedback].vars)
		end
		def test_store_class_vars3
			foo_feed = FooFeedback.new
			foo_feed.set_oid
			@class_var_arsenal.store(FooFeedback)
			expected = {
				'@@foo_var' => 'foo',
				'@@oid'		=> 1
			}
			expected2 = {
			}
			assert_equal(expected,
@class_var_arsenal.var_containers[FooFeedback].vars)
			assert_equal(1, @class_var_arsenal.var_containers.size)
		end
	end
end
```

2.9.2. class_var_arsenal.rb

```
#!/usr/bin/env ruby
# ClassVarArsenal -- odba -- 14.02.2005 -- mwalder@ywesee.com

module ODBA
   class ClassVarArsenal
       include Persistable
       attr_reader :var_containers
       def initialize
           @var_containers = {}
       end
       def store_vars(klass, vars)
           if(!class_var_container = \
               @var_containers.fetch(klass, nil))
               class_var_container = ClassVarContainer.new(klass)
           end
           class_var_container.add_vars(vars)
           class_var_container.odba_store
           @var_containers.store(klass, class_var_container)
           @var_containers.odba_store
       end
       def store(klass)
           ancestors = klass::ancestors
           ancestors.each_with_index { |const, idx|
               bloodline_vars = const.class_variables
               id_next = idx+1
               parent_classes = ancestors[id_next..-1]
               remove_vars = []
               parent_classes.each { |parent|
                   #mixins are not in hierarchy so we musst collect
                   #multiple mixins and subtract them
                   if(!parent.is_a?(Class))
                       remove_vars.concat(parent.class_variables)
                   else
                       break
                   end
                   id_next = id_next+ 1
               }
               if(ancestors[id_next])
                   remove_vars.concat(ancestors[id_next].class_variables)
               end
               class_vars = bloodline_vars - remove_vars
               vars_and_values = Hash.new
               class_vars.each { |c_var|
                   vars_and_values[c_var] = const::class_eval("#{c_var}")
               }
               #classes with no class_variables musn't be stored
               if(!vars_and_values.empty?)
                   store_vars(const, vars_and_values)
                   #ensure perstistence of arsenal
                   self.odba_store
               end
           }
       end
       def remove(klass)
           @var_containers.delete(klass)
           @var_containers.odba_store
       end
       def restore
           @var_containers.values.each { |c_container|
```

```
                c_container.load_class_variables
            }
        end
    end
end
```

2.9.3. test_class_var_container.rb

```
#!/usr/bin/env ruby
# TestClassVarContainer -- odba -- 17.02.2005 -- mwalder@ywesee.com

$: << File.expand_path('../lib/', File.dirname(__FILE__))

require 'test/unit'
require 'odba'
=begin
module ODBA
    module Persistable
        attr_accessor :odba_stored
        def odba_store
            @odba_stored = @odba_stored.to_i + 1
        end
    end
end
=end
class Hash
    attr_accessor :odba_stored
    def odba_store
            @odba_stored = @odba_stored.to_i + 1
    end
end
module Arsenal
    class Foo
        def odba_store
        end
        def Foo.get_var1
            @@c_var1
        end
        def Foo.get_var2
            @@c_var2
        end
    end
end
module ODBA
    class TestClassVarContainer < Test::Unit::TestCase
        def setup
            @class_container = ODBA::ClassVarContainer.new(Arsenal::Foo)
        end
        def test_add_vars
            vars = {
                '@@klass_var' => 'val'
            }
            assert_equal(nil, @class_container.vars.odba_stored)
            @class_container.add_vars(vars)
            assert_equal(vars, @class_container.vars)
            assert_equal(1, @class_container.vars.odba_stored)
        end
        def test_eval_code
            vars = {
                '@@c_var1' => 'val1',
                '@@c_var2' => 'val2'
            }
            assert_equal([], Arsenal::Foo.class_variables)
```

```
            @class_container.add_vars(vars)
            @class_container.load_class_variables
            assert_equal('val1', Arsenal::Foo.get_var1)
            assert_equal('val2', Arsenal::Foo.get_var2)
        end
    end
end
```

2.9.4. class_var_container.rb

```
#!/usr/bin/env ruby
# ClassVarArsenal -- odba -- 16.02.2005 -- mwalder@ywesee.com
module ODBA
    class ClassVarContainer
        include Persistable
        attr_accessor :klass
        attr_accessor :vars
        def initialize(klass)
            @klass = klass
            @vars = {}
        end
        def add_vars(hash)
            @vars.merge!(hash)
            @vars.odba_store
        end
        def load_class_variables
            eval_code = ""
            @vars.each { |name, value|
                #works only with scalar values!!!
                eval_value = value.inspect
                eval_code << "#{name} = #{eval_value};"
            }
            @klass.class_eval(eval_code)
        end
    end
end
```

2.9.5. demo_model.rb

```
require 'odba'
require 'db_connection'
require 'readline'
include Readline
module FaDemo
   module Mixin
       @@class_mod = ""
       def Mixin.set_module_var(var)
           @@class_mod = var
       end
       def Mixin.get_module_var
           @@class_mod
       end
   end
   class Model
       include ODBA::Persistable
       include Mixin
       @@class_var = ""
       def Model.set_class_var(val)
           @@class_var = val
       end
       def Model.get_class_var
           @@class_var
       end
   end
   class OtherModel
       include ODBA::Persistable
   end
end
ODBA.cache_server.class_var_arsenal.restore
fa_demo = ODBA.cache_server.fetch_named('demoModel', nil) {
   FaDemo::Model.new
}
puts "class var values of FaDemo::Model"
puts FaDemo::Model.get_class_var
puts "class var values of FaDemo::Mixin"
puts FaDemo::Mixin.get_module_var
puts "please enter a string, exit to leave, store to save values"
promt = "DemoProgram>"
while (line = readline(promt, true))
   line.strip!
   if (line == 'exit')
       puts "byebye"
       break
   end
   puts eval(line)
end
```