Rückwärtssalto

Kodras, Tiryaki

2015

Acer

4AHIT

02.01.2015

Inhalt

Aufgabenstellung 4

Designüberlegung 5

Implementierung/Durchführung 7

Testbericht 9

GitHub-Link 10

# Aufgabenstellung

## JDBC: RÜCKWÄRTSSALTO

Erstelle ein Java-Programm, dass Connection-Parameter und einen Datenbanknamen auf der Kommandozeile entgegennimmt und die Struktur der Datenbank als EER-Diagramm und Relationenmodell ausgibt (in Dateien geeigneten Formats, also z.B. PNG für das EER und TXT für das RM)

Verwende dazu u.A. das ResultSetMetaData-Interface, das Methoden zur Bestimmung von Metadaten zur Verfügung stellt.

Zum Zeichnen des EER-Diagramms kann eine beliebige Technik eingesetzt werden für die Java-Bibliotheken zur Verfügung stehen: Swing, HTML5, eine WebAPI, ... . Externe Programme dürfen nur soweit verwendet werden, als sich diese plattformunabhängig auf gleiche Weise ohne Aufwand (sowohl technisch als auch lizenzrechtlich!) einfach nutzen lassen. (also z.B. ein Visio-File generieren ist nicht ok, SVG ist ok, da für alle Plattformen geeignete Werkzeuge zur Verfügung stehen)

Recherchiere dafür im Internet nach geeigneten Werkzeugen.

Die Extraktion der Metadaten aus der DB muss mit Java und JDBC erfolgen.

Im EER müssen zumindest vorhanden sein:

* korrekte Syntax nach Chen, MinMax oder IDEFIX
* alle Tabellen der Datenbank als Entitäten
* alle Datenfelder der Tabellen als Attribute
* Primärschlüssel der Datenbanken entsprechend gekennzeichnet
* Beziehungen zwischen den Tabellen inklusive Kardinalitäten soweit durch Fremdschlüssel nachvollziehbar. Sind mehrere Interpretationen möglich, so ist nur ein (beliebiger) Fall umzusetzen: 1:n, 1:n schwach, 1:1
* Kardinalitäten

Fortgeschritten (auch einzelne Punkte davon für Bonuspunkte umsetzbar)

* Zusatzattribute wie UNIQUE oder NOT NULL werden beim Attributnamen dazugeschrieben, sofern diese nicht schon durch eine andere Darstellung ableitbar sind (1:1 resultiert ja in einem UNIQUE)
* optimierte Beziehungen z.B. zwei schwache Beziehungen zu einer m:n zusammenfassen (ev. mit Attributen)
* Erkennung von Sub/Supertyp-Beziehungen

# Designüberlegung

Aufwand

## Schätzung & Aufteilung

**Geschätzter Aufwand:** 5h \* 2 Personen = 10 Stunden

|  |  |
| --- | --- |
| **Aufgabe** | **Name** |
| Designüberlegung, UML-Klassendiagramm | Kodras, Tiryaki |
| Connection | Kodras |
|  |  |
|  |  |
|  |  |
| Testen | Kodras, Tiryaki |
| Protokoll fertigstellen | Kodras, Tiryaki |

## Reeller Aufwand

|  |  |  |
| --- | --- | --- |
| **Aufgabe** | **Name** | **Zeit (min)** |
|  |  |  |

**Aufwand Kodras:** 265 Minuten: 4 Stunden 25 Minuten

**Aufwand Tiryaki:** 500 Minuten: 8 Stunden 20 Minuten

**Gesamtaufwand:** 765 Minuten: 12 Stunden 45 Minuten

# Implementierung/Durchführung

Folgende Klassen/Packages/Funktionalitäten wurden implementiert:

## Package kodrasTiryaki

### Main

* Initialisiert einen neuen Controller
* Beinhält main-Methode
* Überprüft die Kommandozeilenargumente

### Controller

* Initialisiert ein neues Display zur Darstellung, Networkcontroller (Verbindungsaufbau) und Actionlistener
* Sendet Eingaben aus der GUI an den Networkcontroller, indem ein neues Message-Objekt erstellt wird, welches je nach gewählten Optionen dekoriert wird.

## Package kodrasTiryaki.connection

### Networkcontrollable

* Initialisiert einen neun Client und falls nicht vorhanden auch einen neuen Server (Verbindungen über Sockets)
* Sendet Nachrichten an den Server
* Empfänger Nachrichten vom Server und stellt sie am Display dar
* Schließt die Verbindung

### Networkcontroller

* Implementiert das Interface Networkcontrollable

### Client

* Sendet Nachrichten an den Server
* Empfängt Nachrichten vom Server (nebenläufig)

### ChatClient

* Implementierung von Client

### Server

* Akzeptiert neue Clients (nebenläufig)
* Empfängt Nachrichten der Clients in je einem ClientReceiver-Thread pro Client.
* Sendet empfangene Nachrichten an alle registrierte Clients

### ChatServer

* Implementierung von Server

### ClientReceiver

* Empfängt Nachrichten eines Clients (nebenläufig)

## Package kodrasTiryaki.message

### Message

* Stellt eine Nachricht dar und enthält deren Inhalt
* Enthält eine Abstrakte Methode process zur Verarbeitung des Nachrichteninhalts

### ChatMessage

* Erbt von Message
* Implementiert die Verarbeitung

### Modifier

* Erbt von Message
* Enthält eine weitere Message

### UpperCase

* Erbt von Modifier
* Verarbeitet den Inhalt der Nachricht, indem die Buchstaben groß geschrieben werden

### DoubleCharacter

* Erbt von Modifier
* Verarbeitet den Inhalt der Nachricht, indem die Buchstaben verdoppelt werden

### Censorship

* Erbt von Modifier
* Verarbeitet den Inhalt der Nachricht, indem „Bad Words“ zensiert werden

## Package kodrasTiryaki.gui

### Displayable

* Updatet das Display & den User-Input
* Gibt Auskunft über vom User ausgewählte Optionen sowie den Userinput

### ChatWindow

* Implementierung von Displayable

### ChatActionListener

* Implementierung eines Action- & Window-Listeners

# Testbericht

# GitHub-Link