|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |
| TOPAZ MasterRindApp  Technische Dokumentation |
|  |

Inhaltsverzeichnis

TOPAZ MasterRindApp 1

Einleitung 1

Benutzte Entwurfsmuster 2

Datenmodel 7

Datenbankstruktur 10

Fehlerbehandlung 12

Testen 13

Quellcodedokumentation 14

# TOPAZ MasterRindApp

### Einleitung

Ziel dieser Dokumentation ist, einen Einblick in Entwicklung der MasterRindApp zugeben und die Einarbeitung in das Projekt zu erleichtern.

Auf den nachfolgenden Seiten werden die Entwurfsmuster, die bei der Erstellung benutzt wurden, beschrieben. Des Weiteren werden die wichtigsten Klassen des Datenmodels besprochen und es wird die Datenbankstruktur erklärt. Es werden auch die Methoden zur Fehlerbehandlung und zum Testen vorgestellt.

Diese Dokumentation ist nicht vollständig und wird im Laufe des Projektes weiter wachsen.

### Benutzte Entwurfsmuster

Entwurfsmuster sind bewährte Lösungsschablonen, die für die wiederkehrenden Probleme in der Softwarearchitektur- und Entwicklung benutzt werden. In diesem Abschnitt werden die Entwurfsmuster, die bei der Erstellung benutzt wurden, einzeln erklärt.

#### Model View Controller (MVC)

Model View Controller ist ein Muster zur Strukturierung von Softwareentwicklung. Ziel dieses Musters ist es, einen flexiblen Programmentwurf zu ermöglichen, bei dem die Änderungen einfach durchgeführt werden können. Hierbei wird der Code in drei Einheiten aufgeteilt:

* Datenmodel (model)
* Präsentation (view)
* Programmsteuerung (controller)

Jede dieser Einheiten wird unabhängig von einander entwickelt und kann einfach ersetzt werden, ohne dabei die Projektstruktur zu zerstören.

Das Titaniumframework bietet eine vordefinierte MVC-Struktur an. Diese Struktur wird für die Erstellung der MasterRindApp benutzt. Die Abbildung 1 zeigt die Projektstruktur der MasterRindApp. Wie man aus der Abbildung gut erkennen kann, sind in dem Projekt unter anderen die Ordner controllers, models und views enthalten. In dem Ordner „views“ werden die XML-Dateien abgelegt, die die Struktur der GUI bilden und in dem Ordner „styles“ befinden sich die TSS-Dateien, die das Aussehen der GUI bestimmen. Zusammen bilden diese beiden Ordner die Präsentationseinheit der MasterRindApp. Der Ordner „models“ enthält die JavaScript-Dateien, in denen die Daten abgebildet werden und in dem Ordner „controllers“ wird die Programmsteuerung implementiert. Zur welchen Einheit der einzelne Ordner gehört, lässt sich aus dem Namen leicht ableiten.

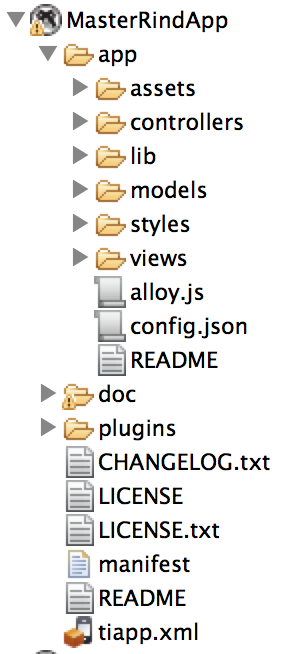


Abbildung : Projektstruktur im Titanium

#### Abstrakte Fabrikmethode

Abstrakte Fabrikmethode ist ein Objekterzeugungsmuster, dabei werden gleichartige Objekte in einer extra Klasse erzeugt. Diese Klasse bildet die Schnittstelle für die anderen Klassen des Systems. Dadurch werde die Klassen von einander entkoppelt, was die Erweiterung der Architektur erleichtert. Abbildung 2 zeigt ein verallgemeinertes UML-Diagramm einer Abstrakten Fabrik.

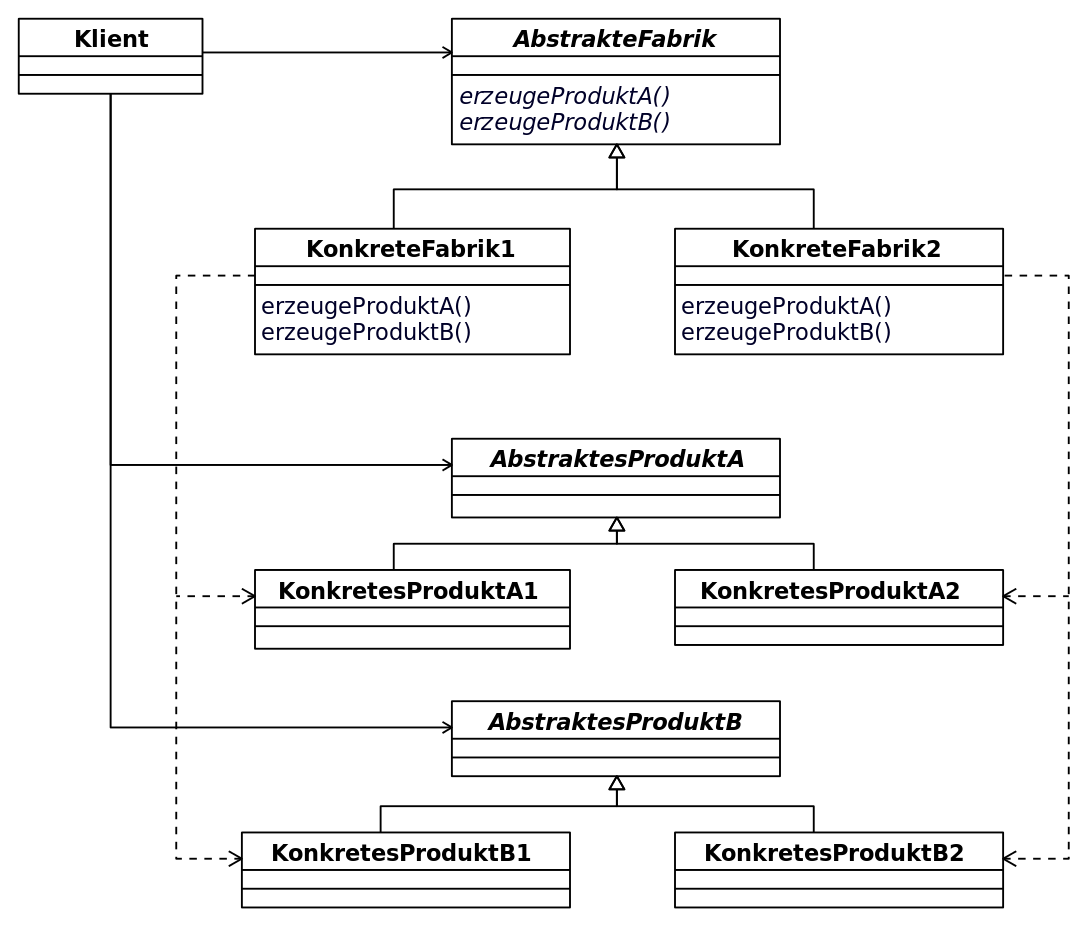


Abbildung : Abstrakte Fabrik

Abbildung 3 zeigt ein UML-Diagramm, wo die Abstrakte Fabrik auf die Klassen der MasterRindApp angewendet wurde. Die einzelnen Klassen werden im Abschnitt „Datenmodel“ beschrieben. Deshalb werden sie hier nicht weiter erklärt.

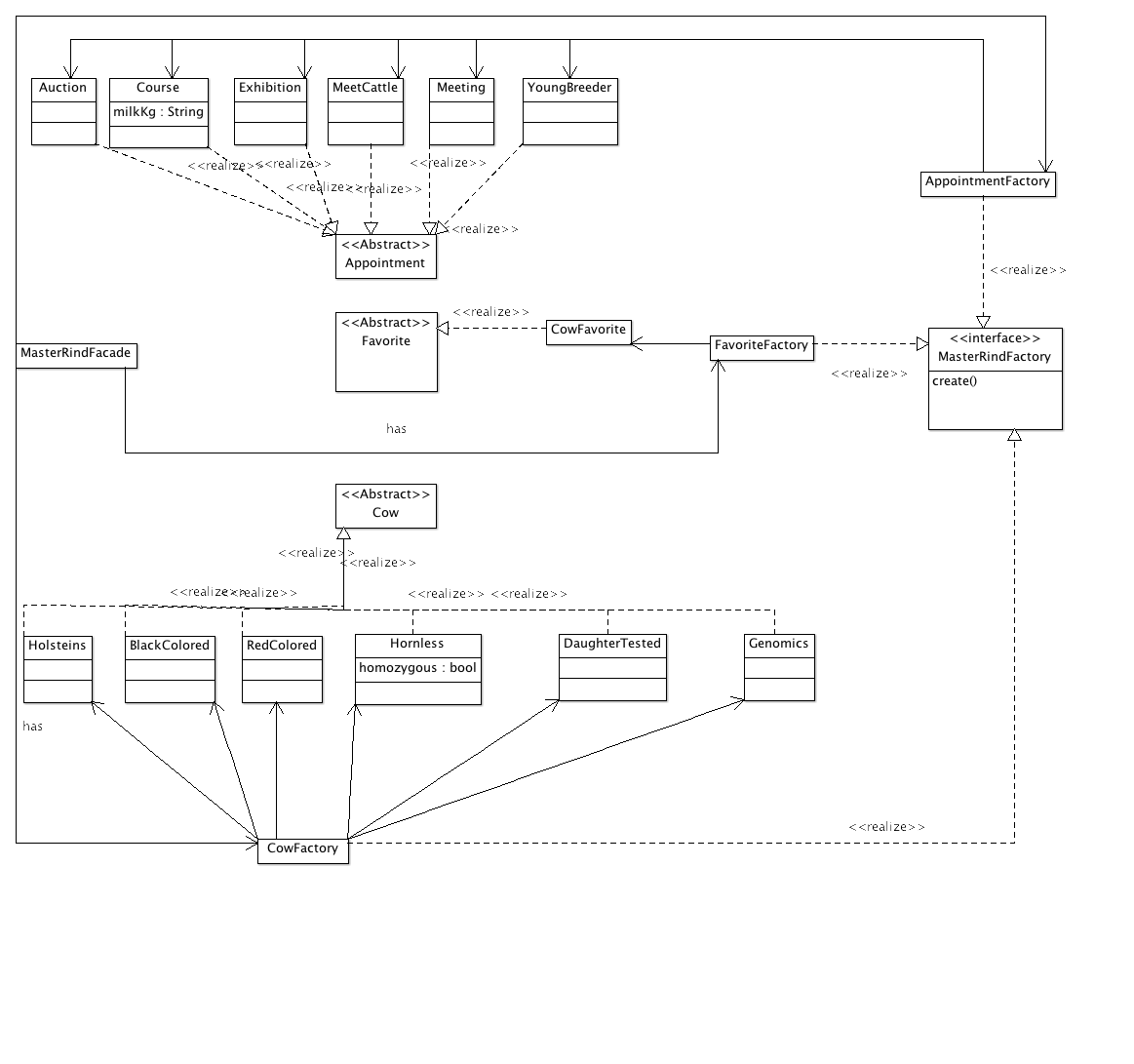


Abbildung : Abstrakte Fabrik für MasterRindApp

#### Fassade

Fassade ist ein Entwurfsmuster, der die Struktur der Klassen definiert. Er bietet eine Schnittstelle mit ausgewählten Methoden für die Klassen des Subsystems. Die Abbildung 4 zeigt ein verallgemeinertes UML-Diagramm des Fassade Entwurf Musters und Abbildung 3 wurde Fassade für die Klassen der MasterRindApp angewendet. Die Klasse MasterRindFacade fungiert als Schnittstelle für Objekte die sie aufrufen. In der MasterRindApp werden es vorüberwiegend die Kontroller Klassen sein.

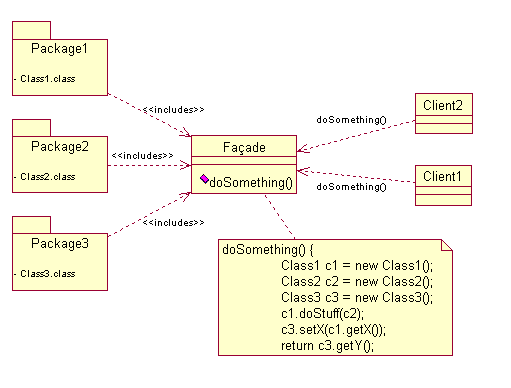
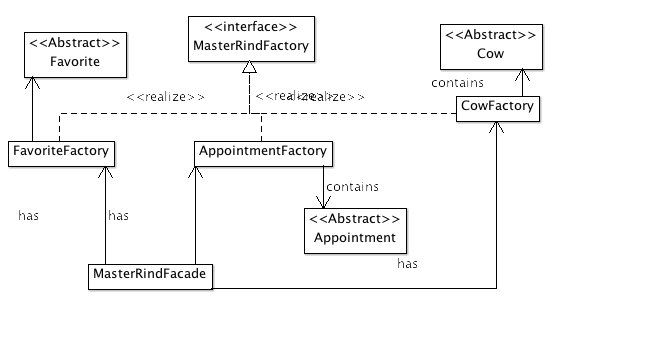


Abbildung : Fassade

### Datenmodel

In diesem Abschnitt wird das Datenmodel der MasterRindApp vorgestellt. Das Datenmodel ist eine Menge von Klassen und deren Beziehungen unter einander. Es bildet den Kern der Architektur der MasterRindApp und sollte deshalb gut durchdacht sein. Abbildung 4 zeigt das Datenmodel für die MasterRindApp. Einfachheitshalber werden in der Abbildung 4 nur die Basisklassen angezeigt. Das komplette Datenmodel befindet sich im Anhang. Auf den nächsten Seiten werden die Klassen des Datenmodels näher erläutert.



#### Klassenbeschreibung

##### Cow

Cow.js ist die Basisklasse die, die Kuh abbildet. Sie enthält alle für die MasterRindApp relevanten Eigenschaften der Kuh. Von dieser Klasse werden alle weiteren Kühe z.B. Holsteins abgeleitet.

##### Favorite

Favorite.js ist die Basisklasse für die Favoriten. Alle anderen Favoritenklassen werden von dieser Klasse abgeleitet.

##### Appointment

Appointment.js ist die Basisklasse für die Termine. Sie enthält die Firmen und deren Kontakte. Alle anderen Terminklassen werden von dieser Klasse abgeleitet.

##### MasterRindFactory

MasterRindFactory.js ist die Basisklasse für alle anderen FactoryKlassen. Diese Klasse ist ein Teil der Abstrakten Fabrikmethode. Sie enthält eine abstrakte Klasse „create“.

##### MasterRindFacade

MsterRindFacade.js fungiert als Schnittstelle für die Kontroller Klasse der MasterRindApp. Sie ist ein Teil des Fassade Entwurf Musters.

#### Namespacebildung

Um dem Quellcode eine Struktur zu geben, werden die Namensräume benutzt. Da JavaScript keine Namensräume kennt, muss eine Hilfsfunktion erstellt werden, die diese für uns erstellt. Diese Funktion heißt namespace(name) und befindet sich im der Datei „alloy.js“. Abbildung 4 zeigt die Funktion zur Erstellung der Namensräume. TOPAZ ist das Wurzelelement des Namensraumes und bilden somit den Hauptnamensraum. Um einen Namensraum zu erstellen, schreibt man den folgenden Code in die Datei: TOPAZ.namespace(Name des Namensraumes). Um z.B. einen Namensraum für die MasterRindFacade zu erstellen, schreibt man in die MasterRindFacade.js folgenden Code: TOPAZ.namespace.(TOPAZ.masterrind.impl.MasterRindFacade).

Abbildung : Funktion zur Namensraumerstellung

var TOPAZ = TOPAZ || {};

TOPAZ.namespace = function(name) {

var parts = name.split('.'), parent = TOPAZ, i;

if (parts[0] === "TOPAZ") {

parts = parts.slice(1);

}

pl = parts.length;

for ( i = 0; i < parts.length; i++) {

if ( typeof parent[parts[i]] === 'undefined') {

parent[parts[i]] = {};

}

parent = parent[parts[i]];

}

return parent;

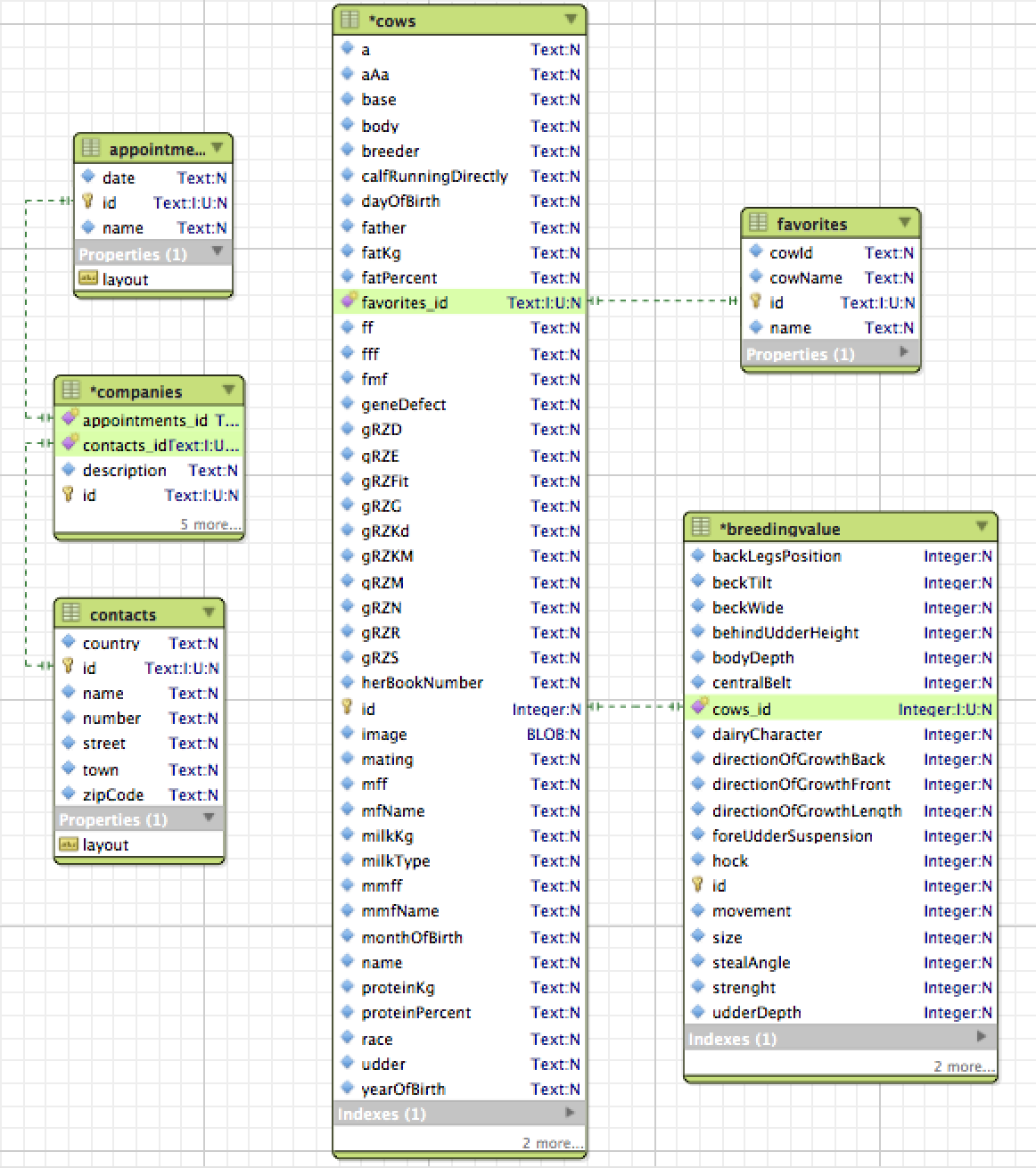
};

### Datenbankstruktur

In diesem Abschnitt wird die Struktur der Datenbank für die MasterRindApp erklärt. Die Datenbank ist eine SQLite-Datenbank. Sie heißt „masterrind“ und enthält folgende Tabellen:

* appointments
* breedingvalues
* companies
* contacts
* cows
* favorites

In der Abbildung 5 kann man die Struktur der Datenbank mit den entsprechenden Tabellen und deren Beziehungen untereinander sehen.



### Fehlerbehandlung

Damit man die Fehler rechtzeitig erkennt und auf diese reagieren kann, werden in diesem Abschnitt einigen Methoden zur Fehlerbehandlung vorgestellt.

#### Manuelle Fehlerbehandlung

Wenn ein Fehler im Code auftritt, der nicht sofort behoben werden kann, dann sollte dieser Fehler durch einen Kommentar kenntlich gemacht werden. Der Kommentar sollte entweder direkt über der Zeile stehen, wo der Fehler aufgetreten ist oder in der Zeile über der Methode, in der er aufgetreten ist. Der Kommentar sollte wie folgt aussehen:

/\*\*

\* Fehler: Fehlermeldung

\* Ursache: wenn bekannt, beschreiben

\* Lösung: wenn bekannt, beschreiben

\*/

Wenn der Fehler korrigiert wurde, dann wird der Kommentar ausgeschnitten und in die Datei „ERROR\_REPORT.txt“ reinkopiert. Diese Datei befindet sich im Verzeichnis MasterRindApp/app/tests/.

#### Ausnahmen (exceptions)

### Testen

Zum Testen des Codes werden Unit Tests und Integrationstests verwenden. In diesem Abschnitt wird erklärt, was diese Tests sind und wie sie durchgeführt werden.

#### Unit Test

Unit Test ist ein Test, der nur einen kleinen Stück Codes testet. In den meisten Fällen handelt es sich um eine Methode oder Funktion. Zum Testen wird das von „Google JS Test Framework“ verwendet.

##### Testen mit Google JS Test

Bsp.:

#### Integrationstest

Beim Integrationstest handelt es sich um einen Test des kompletten Programms oder eines Teils des Programms. In der Regel werden Integrationstests von Menschen durchgeführt, die einen vorgegebenen Ablauf (z.B. durch Anklicken bestimmter Buttons) durchgehen.

### Quellcodedokumentation

Zum guten Stil des Programmierens gehört ein gut dokumentierter Code. Dies erleichtert die Arbeit in einem Team und spart die Einarbeitungszeit. Dieser Abschnitt beschreibt, wie der Code für die MasterRindApp dokumentiert werden soll.

Vor jeder Klasse sollte ein Kommentarblock stehen, der folgendes Aussehen hat:

Bsp.:

/\*\*

\* Stellt Methoden für die Kontrollerklasse bereit. Kann als eine

\* Schnittstelle angesehen werden.

\*

\* @namespace TOPAZ.masterrind.impl

\* @class MasterRindFacade

\*/

Vor jeder Methode sollte ein Kommentarblock stehen, der folgendes Aussehen hat:

Bsp.:

/\*\*

\* Erstellt eine Liste mit Terminobjekten.

\*

\* @method getAppointments

\* @param {String} type

\* @return {Array} appointments

\*/

Aus diesen Kommentaren lässt sich mit dem Tool „YUIdoc“ leicht eine Codedokumentation im HTML-Format erstellen. Nach der Installation des Tools, tippt man folgendes: „*yuidoc –o Ausgabeverzeichnis VerzeichniMitQuellcode“* in die Konsole und die Dokumentation wird automatisch erstellt.

Beispiel für MasterRindApp:

*„yuidoc –o /Users/mbp02/Documents/TOPAZ/Implementation/Prototypes/MasterRindApp/doc /Users/mbp02/Documents/TOPAZ/Implementation/Prototypes/MasterRindApp/app*“

Anhang

MasterRindApp: Datenmodel.

MasterRindApp: Datenmodel vereinfacht.

MasterRindApp: Fabrikmethode.