

TwoNote

Felix Kibellus

11.8.2014

Inhaltsverzeichnis

I. Dokumentation	4
1. Programmbeschreibung	5
1.1. Was soll TwoNote leisten	5
1.2. Wie soll TwoNote arbeiten	5
2. Entwicklungsdokumentation	6
2.1. Projektplanung und Entwicklung	6
2.2. Verwendete Programme	6
2.3. Projektdesign	6
2.3.1. Modell View Controller	7
2.3.2. Kapselung der Komplexität	7
2.4. Weiterführende Entwicklung	7
3. Programmdokumentation	9
3.1. Quellcodedokumentation	9
3.2. Methodendokumentation	9
3.3. Javadoc	9
3.4. Datenbankdokumentation	82
3.4.1. ON DELETE CASCADE	82
4. Benutzerdokumentation	83
4.1. Installationsdokumentation	83
4.2. Benutzung	83
II. Quellcode	84
5. dataManagement	85
5.1. IDatabase	85
5.2. Tree	87
5.3. Serializer	88
5.4. MySQLDatabase	89
5.5. Model	101
6. dataProcessing	108
6.1. Controller	108

7. GUI	111
7.1. GUI	111
8. pageData	112
8.1. ActionType	112
8.2. Content	112
8.3. ContentInstruction	114
8.4. ContentType	115
8.5. DirectoryAlreadyExistsException	115
8.6. DirectoryDoesNotExistsException	116
8.7. DirectoryType	117
8.8. Page	118
8.9. PageInformation	120
8.10. TextBox	122
8.11. Startpage	124
9. Tools	127
9.1. TextEditTool	127
9.2. Tool	132
9.3. ToolEndsType	133
9.4. ToolReadyHandler	133

Teil I.

Dokumentation

1. Programmbeschreibung

1.1. Was soll TwoNote leisten

Bei TwoNote handelt es sich um ein primitives Programm zum Erstellen von Notizen. Um die Komplexität des Projekts zu verringern, soll es vorerst nur möglich sein Text anzulegen und zu modifizieren. Eine Notiz wird nur auf einer Seite gespeichert. Notizen müssen in einem Kapitel und Kapitel in einem Buch zusammengefasst werden. Es soll möglich sein, neue Verzeichnisse anzulegen und diese zu löschen. Die Daten werden mittels Datenbank gespeichert und verwaltet. Als Benutzerschnittstelle soll ein GUI geschrieben werden.

1.2. Wie soll TwoNote arbeiten

Eine Notiz soll als Klasse gekapselt werden. Die Seiteninhalte werden innerhalb einer Seite gespeichert und sollen von einer abstrakten Oberklasse Content erben. Die Klasse Content speichert bereits die wichtigsten Attribute einer Notiz (Position und Größe). Die von Content abgeleitete Klasse TextBox stellt dann die konkrete Textnotiz dar und speichert zusätzlich den Text, die Schriftgröße und die Schriftfarbe. Eine Seite soll aus der Datenbank geladen werden können. Danach soll sich jedes Seitenelement selbst zeichnen können. Es sollen Werkzeuge implementiert werden, welche die Seitenelemente bearbeiten können. Eine solche bearbeitete Seite soll dann wieder in der Datenbank gespeichert werden. Es soll eine Klassen Tree geben, welche den Verzeichnisbaum repräsentiert, sodass dieser dem Nutzer angezeigt werden kann. Durch Klicken auf diesen Verzeichnisbaum soll laden und löschen von Seiten möglich sein.

2. Entwicklungsdokumentation

2.1. Projektplanung und Entwicklung

Das Projekt wurde größtenteils vom 3. bis zum 5. März geplant. In dieser Zeit wurde das grobe Programmdesign festgelegt und bereits die wichtigsten Klassen im UML skizziert. Entwickelt wurde dann in der Zeit vom 5. März bis zum 7. März und in der Zeit vom 4. bis zum 8. August. In der weiterführenden Entwicklung wurde zuerst die Datenbank entworfen. Danach wurde in einem Bottom-Up Entwurfsverfahren zuerst das Datenmodell, später die Datenverarbeitung und zuletzt die Benutzerschnittstelle erarbeitet. Vor jeder größeren Programmiereinheit wurde eine kurze Planungsphase vorgenommen, in welcher Überlegungen angestellt wurden, was die neuen Klassen leisten müssen. Nach jeder Programmiereinheit wurde geschaut, ob jede Klasse die nötige Funktionalität erfüllt. Eine grobe Dokumentation fand bereits während der Implementierung statt. Im Entwicklungsprozess wurde die bei der Planung verwendete Bottom-Up Arbeitsweise größtenteils weitergeführt. Jedoch musste des Öfteren, besonders in einer Optimierungsphase das Design angepasst oder überarbeitet werden. So wurde beispielsweise das Datenmodell noch einmal grundsätzlich überarbeitet um Programmlogik und Datenbankbindung grundsätzlich voneinander zu trennen.

2.2. Verwendete Programme

Entwickelt wurde größtenteils mit Eclipse. Da Eclipse jedoch kaum benutzerfreundliche Bearbeitung einer GUI unterstützt, ist die View mit NetBeans entwickelt worden. Die Versionsverwaltung wurde zu Beginn mit bazaar vorgenommen. Da es später vonnöten war auf mehreren Rechnern zu entwickeln wurde zu Git gewechselt. Für die Datenbank ist MySql verwendet worden. Zur Administration der Datenbank wurde ein Webinterface mit phpMyAdmin benutzt. Die Dokumentation wurde mittels Javadoc und \LaTeX erstellt. Zur Verwaltung der Designideen und größeren Entwürfen wurde OneNote verwendet. Zum Erstellen der UML-Klassendiagramme ist ObjectAid benutzt worden.

2.3. Projektdesign

Beim Programmdesign wurden mehrere Entwurfsmuster verwendet. Die Verwendung des MVC-Patterns wurde bereits in den ersten Entwurf mit aufgenommen. Einige kleinere Muster stellten sich erst während des Entwicklungsprozesses als hilfreich heraus und wurden nachträglich mit aufgenommen.

2.3.1. Modell View Controller

Das Programm ist grundlegend nach dem MVC-Pattern designed worden. Die Datenhaltung wird vollständig von den Klassen Model, MySQLDataBase, und Serializer implementiert. Alle Klassen zur Datenhaltung befinden sich im Paket dataManagement. Hier wurde besonders Wert darauf gelegt, möglichst viel Komplexität zu kapseln, sodass ein komfortabler Zugriff von außen auf das Datenmodell möglich ist, ohne sich mit genaueren Implementierungen beschäftigen zu müssen. Der Controller befindet sich im Paket dataProcessing und verwaltet die aktuell geöffnete Sitzung. Unüblicherweise befindet sich relativ wenig Verarbeitungslogik im Controller, da innerhalb des Models bereits eine Überprüfung des Datenbankzugriffs erfolgt und in der View primitive Aufgaben bereits erledigt werden. Daher ist das Design hier nicht sehr sauber und muss ggf. nochmal überarbeitet werden. Der Controller implementiert nur die Aufgaben, welche die View nicht ohne größeren Aufwand direkt ans Modell weiterleiten kann. Primitive Anweisungen wie das Erstellen eines Buches werden direkt von der View an das Modell weitergeleitet.

2.3.2. Kapselung der Komplexität

Zielsetzung des Entwurfs war es möglichst viel Komplexität zu kapseln und über eine klein gehaltene Schnittstelle in der weiterführenden Programmierung zu verbergen. Dieses Konzept kam besonders beim Design der Page zum Tragen. Das Laden einer Page ist von außen nur durch einen einzigen Methodenaufruf möglich indem Buch, Kapitel und Seitenname als Parameter angegeben werden. Intern wird dann zuerst die id des Buches aus der Datenbank abgefragt, mit dieser id und dem Namen des Kapitels die id des Kapitels bestimmt und schließlich mit dieser id und dem Namen der Seite die id der Seite ermittelt. Danach wird die Seite aus der Datenbank geladen. Nun wird jeder Seiteninhalt aus der Datenbank geladen, welcher zur passenden Seite gehört. Dieser wird dann deserialisiert und der Seite hinzugefügt. Des Weiteren wird überprüft ob Seite, Kapitel und Buch überhaupt existieren. Falls nicht, wird die passende Exception geworfen. Es wird eine vollständig geladene Seite zurückgegeben, die vom Controller bequem geladen werden kann ohne sich um die Details der Implementierung zu kümmern.

2.4. Weiterführende Entwicklung

Da es bisher nur möglich ist Textnotizen zu erstellen und somit nur primitive Notizen erstellt werden können wäre es denkbar weiteren Seiteninhalt zu implementieren. Denkbar wären beispielsweise Kreise, Linien, Pfeile oder andere Formen. Des Weiteren könnte das Importieren von Bildern oder pdfs ermöglicht werden, um bereits bestehende Dateien in TwoNote verarbeiten zu können. Das Design der View müsste nochmals grundlegend überarbeitet werden, da es bisher nicht möglich ist die von NetBeans erstellte GUI nachträglich zu ändern. Die Aufgaben von View und Controller sind noch nicht klar genug getrennt und benötigen ebenfalls eine Überarbeitung. Da TwoNote eine korrekt angelegte Datenbank benötigt, um Fehlerfrei arbeiten zu können sollte ein

Installer geschrieben werden, welcher die Datenbank einrichtet und den Treiber installiert, falls dieser nicht vorhanden ist. Es ist noch nicht möglich Seiten umzubenennen oder zu verschieben. Für eine Benutzerfreundliche Handhabung sollte dies dringend noch implementiert werden.

3. Programmdokumentation

3.1. Quellcodedokumentation

Die Quellcodedokumentation wurde vollständig per Inlinedokumentation vorgenommen.

3.2. Methodendokumentation

Als Methodendokumentation dienen die von Javadoc erstellten HTML-Files.

3.3. Javadoc

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dataManagement

Interface IDataBase

All Known Implementing Classes:

[MySQLDatabase](#)

public interface **IDataBase**

Dieses Interface stellt die Schnittstelle zur Datenbank her. Alle Methoden verwenden keinerlei Logik, sondern übersetzen lediglich eine elementare Programmanweisung in die Sprache des DMBS und leiten diese an das DMBS zur Ausführung weiter. Die Überprüfung der korrekten Arbeitsweise dieser Methoden ist somit nicht Aufgabe der Klassen die dieses Interface implementieren, sondern muss von den verwendenden Klassen geleistet werden. Dieses Interface führt somit die nötige Abstraktion ein um verschiedene Datenbanken zu verwenden oder den Umstieg auf eine andere Datenbank zu erleichtern.

Method Summary

Methods

Modifier and Type	Method and Description
void	createBook (java.lang.String bookName) Diese Methode legt ein neues Buch in der Tabelle "Books" an
void	createChapter (java.lang.String chapterName, int bookID) Diese Methode legt ein neues Kapitel in der Tabelle "Chapter" an
void	createContent (byte[] contentBytes, int contentNumber, int pageID) Diese Methode legt einen neuen Seiteninhalt in der Tabelle "Content" an
void	createPage (PageInformation pageInfo, int chapterID) Diese Methode legt eine neue Seite in der Tabelle "Pages" an
void	deleteBook (int bookID) Diese Methode löscht ein Buch aus der Tabelle "Books"
void	deleteChapter (int chapterID) Diese Methode löscht ein Kapitel aus der Tabelle "Chapter"
void	deleteContent (int contentID) Diese Methode löscht einen Seiteninhalt aus der Tabelle "Content"
void	deletePage (int pageID) Diese Methode löscht eine Seite aus der Tabelle "Pages"
boolean	existsBook (java.lang.String bookName) Diese Methode gibt zurück, ob es in der Tabelle "Books" ein Buch mit dem Namen "bookName" gibt

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

boolean	existsChapter (java.lang.String chapterName, int bookID) Diese Methode gibt zurück, ob es in der Tabelle "Chapter" ein Kapitel mit dem Namen "chapterName" gibt
boolean	existsContent (int contentNumber, int pageID) Diese Methode gibt zurück, ob es in der Tabelle "Content" ein Eintrag mit der Nummer "contentNumber" gibt
boolean	existsPage (java.lang.String pageName, int chapterID) Diese Methode gibt zurück, ob es in der Tabelle "Pages" eine Seite mit dem Namen "pageName" gibt
java.lang.String[]	getAllBooks () Diese Methode liefert den Namen aller Bücher welche in der Datenbank angelegt sind
java.lang.String[]	getBookChildren (int bookID) Diese Methode liefert zur id eines Buches alle Kapitel welche zu diesem Buch gehören
int	getBookID (java.lang.String bookName) Diese Methode liefert zu dem Namen eines Buches die entsprechende id in der Datenbank
java.lang.String[]	getChapterChildren (int chapterID) Diese Methode liefert zur id eines Kapitels alle Seiten welche zu diesem Kapitel gehören
int	getChapterID (java.lang.String chapterName, int bookID) Diese Methode liefert zu dem Namen eines Kapitels die entsprechende id in der Datenbank
int	getContentID (int contentNumber, int pageID) Diese Methode liefert zu der Nummer eines Seiteninhalts die entsprechende id in der Datenbank
int[]	getPageChildren (int pageID) Diese Methode liefert zur id einer Seite alle Inhalte welche zu dieser Seite gehören
int	getPageID (java.lang.String pageName, int chapterID) Diese Methode liefert zu dem Namen einer Seite die entsprechende id in der Datenbank
byte[]	loadContent (int contentID) Diese Methode lädt einen Seiteninhalt aus der Tabelle "Content"
PageInformation	loadPageInformation (int pageID) Diese Methode lädt die Seiteninformation aus der Tabelle "Pages"
void	saveContent (byte[] contentBytes, int contentID) Diese Methode speichert einen Seiteninhalt in der Tabelle "Content"
void	savePage (PageInformation pageInfo, int chapterID, int pageID) Diese Methode speichert eine Seite in der Tabelle "Pages"

Method Detail

createBook

```
void createBook(java.lang.String bookName)
    throws java.sql.SQLException
```

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Diese Methode legt ein neues Buch in der Tabelle "Books" an

Parameters:

bookName - der Name des anzulegenden Buches

Throws:

java.sql.SQLException - Datenbankfehler

createChapter

```
void createChapter(java.lang.String chapterName,  
                  int bookID)  
    throws java.sql.SQLException
```

Diese Methode legt ein neues Kapitel in der Tabelle "Chapter" an

Parameters:

chapterName - der Name des anzulegenden Kapitels

bookID - die id des Buches zu dem das anzulegende Kapitel gehören soll

Throws:

java.sql.SQLException - Datenbankfehler

createPage

```
void createPage(PageInformation pageInfo,  
               int chapterID)  
    throws java.sql.SQLException
```

Diese Methode legt eine neue Seite in der Tabelle "Pages" an

Parameters:

pageInfo - informationen zur Seite

chapterID - die id des Kapitels zu dem die anzulegende Seite gehören soll

Throws:

java.sql.SQLException - Datenbankfehler

createContent

```
void createContent(byte[] contentBytes,  
                  int contentNumber,  
                  int pageID)  
    throws java.sql.SQLException
```

Diese Methode legt einen neuen Seiteninhalt in der Tabelle "Content" an

Parameters:

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

contentBytes - das serialisierte Objekt
contentNumber - die nummer des Seiteninhalts innerhalb seiner Seite
pageID - die id der Seite zu welcher der Inhalt gehören soll

Throws:

java.sql.SQLException - Datenbankfehler

deleteBook

```
void deleteBook(int bookID)
    throws java.sql.SQLException
```

Diese Methode löscht ein Buch aus der Tabelle "Books"

Parameters:

bookID - Die id des Buches, welches gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

deleteChapter

```
void deleteChapter(int chapterID)
    throws java.sql.SQLException
```

Diese Methode löscht ein Kapitel aus der Tabelle "Chapter"

Parameters:

chapterID - Die id des Kapitels, welches gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

deletePage

```
void deletePage(int pageID)
    throws java.sql.SQLException
```

Diese Methode löscht eine Seite aus der Tabelle "Pages"

Parameters:

pageID - Die id der Seite, welche gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

deleteContent

```
void deleteContent(int contentID)
    throws java.sql.SQLException
```

Diese Methode löscht einen Seiteninhalt aus der Tabelle "Content"

Parameters:

contentID - Die id des Seiteninhaltes, welcher gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

saveContent

```
void saveContent(byte[] contentBytes,
    int contentID)
    throws java.sql.SQLException
```

Diese Methode speichert einen Seiteninhalt in der Tabelle "Content"

Parameters:

contentBytes - das serialisierte Objekt

contentID - die innerhalb der Tabelle "Content"

Throws:

java.sql.SQLException - Datenbankfehler

savePage

```
void savePage(PageInformation pageInfo,
    int chapterID,
    int pageID)
    throws java.sql.SQLException
```

Diese Methode speichert eine Seite in der Tabelle "Pages"

Parameters:

pageInfo - die Information zur Seite

chapterID - das Kapitel zu welchem die Seite gehört

pageID - die id der Seite innerhalb der Tabelle "Pages"

Throws:

java.sql.SQLException - Datenbankfehler

loadContent

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

```
byte[] loadContent(int contentID)
        throws java.sql.SQLException
```

Diese Methode lädt einen Seiteninhalt aus der Tabelle "Content"

Parameters:

contentID - die id des angefragten Seiteninhalts

Returns:

das serialisierte Objekt zur angefragten id

Throws:

java.sql.SQLException - Datenbankfehler

loadPageInformation

```
PageInformation loadPageInformation(int pageID)
        throws java.sql.SQLException
```

Diese Methode lädt die Seiteninformation aus der Tabelle "Pages"

Parameters:

pageID - die id der angefragten Seite

Returns:

die Seiteninformation zur angefragten id

Throws:

java.sql.SQLException - Datenbankfehler

existsBook

```
boolean existsBook(java.lang.String bookName)
        throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Books" ein Buch mit dem Namen "bookName" gibt

Parameters:

bookName - der Name des Buchs

Returns:

zum angefragten Buch existiert ein Eintrag in der Datenbank

Throws:

java.sql.SQLException - Datenbankfehler

existsChapter

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

```
boolean existsChapter(java.lang.String chapterName,  
                      int bookID)  
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Chapter" ein Kapitel mit dem Namen "chapterName" gibt

Parameters:

chapterName - der Name des Kapitels
bookID - die id des Buches zu welchem das Kapitel gehört

Returns:

zum angefragten Kapitel existiert ein Eintrag in der Datenbank

Throws:

java.sql.SQLException - Datenbankfehler

existsPage

```
boolean existsPage(java.lang.String pageName,  
                   int chapterID)  
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Pages" eine Seite mit dem Namen "pageName" gibt

Parameters:

pageName - der Name der Seite
chapterID - die id des Kapitels zu welchem die Seite gehört

Returns:

zur angefragten Seite existiert ein Eintrag in der Datenbank

Throws:

java.sql.SQLException - Datenbankfehler

existsContent

```
boolean existsContent(int contentNumber,  
                      int pageID)  
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Content" ein Eintrag mit der Nummer "contentNumber" gibt

Parameters:

contentNumber - die Nummer des Seiteninhalts
pageID - die id der Seite zu welcher der Inhalt gehört

Returns:

zum angefragten Seiteninhalt existiert ein Eintrag in der Datenbank

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Throws:

java.sql.SQLException - Datenbankfehler

getBookID

```
int getBookID(java.lang.String bookName)
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen eines Buches die entsprechende id in der Datenbank

Parameters:

bookName - Der Name des angefragten Buches

Returns:

die id des angefragten Buches

Throws:

java.sql.SQLException - Datenbankfehler

getChapterID

```
int getChapterID(java.lang.String chapterName,
    int bookID)
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen eines Kapitels die entsprechende id in der Datenbank

Parameters:

chapterName - Der Name des angefragten Kapitels

bookID - die id des Buches zu welchem das Kapitel gehört

Returns:

die id des angefragten Kapitels

Throws:

java.sql.SQLException - Datenbankfehler

getPageID

```
int getPageID(java.lang.String pageName,
    int chapterID)
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen einer Seite die entsprechende id in der Datenbank

Parameters:

pageName - Der Name der angefragten Seite

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

chapterID - die id des Kapitels zu welchem die Seite gehört

Returns:

die id der angefragten Seite

Throws:

java.sql.SQLException - Datenbankfehler

getContentID

```
int getContentID(int contentNumber,  
                int pageID)  
    throws java.sql.SQLException
```

Diese Methode liefert zu der Nummer eines Seiteinhalts die entsprechende id in der Datenbank

Parameters:

contentNumber - Die Nummer des angefragten Seiteninhalts

pageID - die id der Seite zu welchem dir Inhalt gehört

Returns:

die id ders angefragten Seiteinhalts

Throws:

java.sql.SQLException - Datenbankfehler

getBookChildren

```
java.lang.String[] getBookChildren(int bookID)  
    throws java.sql.SQLException
```

Diese Methode liefert zur id eines Buches alle Kapitel welche zu diesem Buch gehören

Parameters:

bookID - Die id des angefragten Buches

Returns:

der Name aller Kapitel, welche zum angefragten Buch gehören

Throws:

java.sql.SQLException - Datenbankfehler

getChapterChildren

```
java.lang.String[] getChapterChildren(int chapterID)  
    throws java.sql.SQLException
```

3.3. JAVADOC

IDataBase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Diese Methode liefert zur id eines Kapitels alle Seiten welche zu diesem Kapitel gehören

Parameters:

chapterID - Die id des angefragten Kapitels

Returns:

der Name aller Seiten, welche zum angefragten Kapitel gehören

Throws:

java.sql.SQLException - Datenbankfehler

getPageChildren

```
int[] getPageChildren(int pageID)
    throws java.sql.SQLException
```

Diese Methode liefert zur id einer Seite alle Inhalte welche zu dieser Seite gehören

Parameters:

pageID - Die id der angefragten Seite

Returns:

der Name aller Inhalte, welche zur angefragten Seite gehören

Throws:

java.sql.SQLException - Datenbankfehler

getAllBooks

```
java.lang.String[] getAllBooks()
    throws java.sql.SQLException
```

Diese Methode liefert den Namen aller Bücher welche in der Datenbank angelegt sind

Returns:

ein String[] mit den Namen aller angelegten Büchern

Throws:

java.sql.SQLException - Datenbankfehler

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dataManagement

Class Model

java.lang.Object
dataManagement.Model

```
public class Model
extends java.lang.Object
```

Die Klasse Model dient der Kapselung der Datenhaltung und ist Teil des MVC-Design. Das Model verwaltet die Schnittstelle zur Datenbank und implementiert die nötige Verarbeitungslogik im Umgang mit der Datenbank. Dazu überprüft es die Existenz von Verzeichnissen und wirft die passenden Exceptions, falls es zu Konflikten in der Verarbeitung kommt. Das Model kann aus dem Pfad eines Elementes die id in der Datenbank bestimmen und diese zur weiteren Verarbeitung nutzen. Zielsetzung dieser Klasse ist ein komfortabler Zugriff auf die Daten, welcher lediglich über Pfadnamen operiert. Die Komplexität der Datenbank soll somit gekapselt werden und vom Controller aus nicht mehr ersichtlich sein. Die Schnittstelle zum Controller soll minimal gehalten sein. Das Model serialisiert und deserialisiert den Seiteninhalt, welcher in der Datenbank abgelegt werden soll.

Constructor Summary

Constructors

Constructor and Description

Model()

Instanziert die Attribute zum Serializer und db Pfad und Logininformationen zur Datenbank sind übergangsweise "hard coded"
--

Method Summary

Methods

Modifier and Type	Method and Description
void	createBook (java.lang.String bookName) Überprüft ob das zu erstellende Buch bereits existiert, falls nicht leitet diese Methode den Auftrag ein Buch zu erstellen an die Datenbankschnittstelle weiter
void	createChapter (java.lang.String chapterName, java.lang.String bookName) Die id des Buches in welchem das Kapitel gespeichert werden soll wird geladen Überprüft ob das zu erstellende Kapitel bereits existiert, falls nicht leitet diese Methode den Auftrag ein Kapitel zu erstellen an die Datenbankschnittstelle weiter

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Page	createPage (java.lang.String pageName, java.lang.String chapterName, java.lang.String bookName) Die id des Kapitels in welchem die Seite gespeichert werden soll wird geladen Überprüft ob die zu erstellende Seite bereits existiert, falls nicht leitet diese Methode den Auftrag die Seite zu erstellen an die Datenbankschnittstelle weiter Die Attribute der Seite werden mit Defaultwerten gefüllt
void	deleteBook (java.lang.String bookName) Diese Methode lädt die id des zu löschenden Buches.
void	deleteChapter (java.lang.String chapterName, java.lang.String bookName) Diese Methode lädt die id des zu löschenden Kapitels.
void	deletePage (java.lang.String pageName, java.lang.String chapterName, java.lang.String bookName) Diese Methode lädt die id der zu löschenden Seite.
Tree	getTree () Der Verzeichnisbaum wird aus der Datenbank abgerufen und als Objekt der Klasse Tree zurückgegeben.
Page	loadPage (java.lang.String pageName, java.lang.String chapterName, java.lang.String bookName) Es wird überprüft ob der angegebene Pfad gültig ist.
void	savePage (Page page) Diese Methode speichert eine Seite.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Model

```
public Model()  
    throws java.lang.ClassNotFoundException,  
           java.sql.SQLException
```

Instanziert die Attribute zum Serializer und db Pfad und Logininformationen zur Datenbank sind
übergangsweise "hard coded"

Throws:

java.lang.ClassNotFoundException
java.sql.SQLException

Method Detail

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

createBook

```
public void createBook(java.lang.String bookName)
    throws DirectoryAlreadyExistsException,
           java.sql.SQLException
```

Überprüft ob das zu erstellende Buch bereits existiert, falls nicht leitet diese Methode den Auftrag ein Buch zu erstellen an die Datenbankschnittstelle weiter

Parameters:

bookName - der Name des Buches

Throws:

DirectoryAlreadyExistsException - das Buch existiert bereits

java.sql.SQLException - Datenbankfehler

createChapter

```
public void createChapter(java.lang.String chapterName,
    java.lang.String bookName)
    throws DirectoryAlreadyExistsException,
           java.sql.SQLException,
           DirectoryDoesNotExistException
```

Die id des Buches in welchem das Kapitel gespeichert werden soll wird geladen Überprüft ob das zu erstellende Kapitel bereits existiert, falls nicht leitet diese Methode den Auftrag ein Kapitel zu erstellen an die Datenbankschnittstelle weiter

Parameters:

bookName - der Name des Buches

chapterName - der Name des Kapitels

Throws:

DirectoryDoesNotExistException - das Buch existiert nicht

DirectoryAlreadyExistsException - das Kapitel existiert bereits

java.sql.SQLException - Datenbankfehler

createPage

```
public Page createPage(java.lang.String pageName,
    java.lang.String chapterName,
    java.lang.String bookName)
    throws DirectoryAlreadyExistsException,
           java.sql.SQLException,
           DirectoryDoesNotExistException
```

Die id des Kapitels in welchem die Seite gespeichert werden soll wird geladen Überprüft ob die zu erstellende Seite bereits existiert, falls nicht leitet diese Methode den Auftrag die Seite zu erstellen an die

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Datenbankschnittstelle weiter Die Attribute der Seite werden mit Defaultwerten gefüllt

Parameters:

bookName - der Name des Buches
chapterName - der Name des Kapitels
pageName - der Name der Seite

Throws:

`DirectoryDoesNotExistException` - das Buch oder Kapitel existiert nicht
`DirectoryAlreadyExistsException` - die Seite existiert bereits
`java.sql.SQLException` - Datenbankfehler

deleteBook

```
public void deleteBook(java.lang.String bookName)
    throws java.sql.SQLException,
           DirectoryDoesNotExistException
```

Diese Methode lädt die id des zu löschenden Buches. Falls das Buch nicht existiert wird eine Exception geworfen. Falls das Buch existiert wird es gelöscht. Alle Unterverzeichnisse dieses Buches werden durch das dmbs mittels ON DELETE CASCADE gelöscht.

Parameters:

bookName - Der Name des Buches

Throws:

`java.sql.SQLException` - Datenbankfehler
`DirectoryDoesNotExistException` - das Buch existiert nicht

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

deleteChapter

```
public void deleteChapter(java.lang.String chapterName,  
                           java.lang.String bookName)  
    throws java.sql.SQLException,  
           DirectoryDoesNotExistException
```

Diese Methode lädt die id des zu löschenden Kapitels. Falls das Buch oder Kapitel nicht existiert wird eine Exception geworfen. Falls das Kapitel existiert wird es gelöscht. Alle Unterverzeichnisse dieses Kapitels werden durch das dmbs mittels ON DELETE CASCADE gelöscht.

Parameters:

bookName - Der Name des Buches

chapterName - Der Name des Kapitels

Throws:

java.sql.SQLException - Datenbankfehler

DirectoryDoesNotExistException - das Buch oder Kapitel existiert nicht

deletePage

```
public void deletePage(java.lang.String pageName,  
                        java.lang.String chapterName,  
                        java.lang.String bookName)  
    throws java.sql.SQLException,  
           DirectoryDoesNotExistException
```

Diese Methode lädt die id der zu löschenden Seite. Falls das Buch, Kapitel oder Seite nicht existiert wird eine Exception geworfen. Falls die Seite existiert wird sie gelöscht. Alle Unterverzeichnisse dieser Seite werden durch das dmbs mittels ON DELETE CASCADE gelöscht.

Parameters:

bookName - Der Name des Buches

chapterName - Der Name des Kapitels

pageName - Der Name der Seite

Throws:

java.sql.SQLException - Datenbankfehler

DirectoryDoesNotExistException - das Buch oder Kapitel existiert nicht

savePage

```
public void savePage(Page page)  
    throws DirectoryDoesNotExistException,  
           java.sql.SQLException,  
           java.io.IOException
```

Diese Methode speichert eine Seite. Alle Pfadangaben liegen in der PageInformation. Die PageInformation

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

beinhalten ebenfalls alle Informationen zum Speichern der Seite. Die Schnittstelle zum Controller besteht nur aus dieser Methode, das Speichern der Seiteninformation wird bewusst gekapselt und in dieser Methode ebenfalls ausgeführt. Jede Seite speichert alle ungespeicherten Änderungen, welche an ihr vorgenommen werden. In dieser Methode wird über die Liste der Änderungen iteriert und abhängig von der ContentInstruction der Seiteninhalt serialisiert und erstellt, gespeichert oder gelöscht.

Parameters:

page - die zu speichernde Seite

Throws:

`DirectoryDoesNotExistException` - die Seite oder ein Seiteninhalt existiert nicht

`java.sql.SQLException` - Datenbankfehler

`java.io.IOException` - Fehler beim Serialisieren

loadPage

```
public Page loadPage(java.lang.String pageName,  
    java.lang.String chapterName,  
    java.lang.String bookName)  
    throws java.sql.SQLException,  
        DirectoryDoesNotExistException,  
        java.lang.ClassNotFoundException,  
        java.io.IOException
```

Es wird überprüft ob der angegebene Pfad gültig ist. Das Laden einer Seite besteht aus zwei Schritten. Zuerst wird die PageInformation aus der Datenbank abgerufen und ein Objekt der Klasse Page erstellt. Danach wird der Seiteninhalt zu dieser Seite geladen, deserialisiert und der Seite zugefügt.

Parameters:

pageName - der Name der Seite

chapterName - der Name des Kapitels

bookName - der Name des Buches#

Returns:

die angefragte Seite

Throws:

`java.sql.SQLException` - Datenbankfehler

`DirectoryDoesNotExistException` - der Pfad zur Seite ist falsch

`java.lang.ClassNotFoundException` - Fehler beim deserialisieren

`java.io.IOException` - Fehler beim deserialisieren

getTree

```
public Tree getTree()  
    throws java.sql.SQLException
```

3.3. JAVADOC

Model

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Der Verzeichnisbaum wird aus der Datenbank abgerufen und als Objekt der Klasse Tree zurückgegeben. Dazu wird iterativ mittels 3 Schleifen immer zuerst das obere Level des Baumes erstellt, und dann zu jedem Element dieses Levels die Kindbäume geladen

Returns:

der Verzeichnisbaum

Throws:

java.sql.SQLException - Datenbankfehler

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

Content

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

pageData

Class Content

java.lang.Object
pageData.Content

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

TextBox

```
public abstract class Content
extends java.lang.Object
implements java.io.Serializable
```

Von dieser abstrakten Klasse können Seiteninhalte abgeleitet werden. Gespeichert wird ein Seitenelement indem es serialisiert und in der Datenbank gespeichert wird. Es ist darauf zu achten nur serialisierbare Attribute zu verwenden.

See Also:

[Serialized Form](#)

Constructor Summary

Constructors

Constructor and Description

Content (int x, int y, int width, int height, ContentType contentType)
--

Konstruktor muss aus Unterklasse aufgerufen werden.

Method Summary

Methods

Modifier and Type	Method and Description
abstract java.awt.Component	draw (javax.swing.JPanel panel) Der Seiteninhalt wird angewiesen sich auf das übergebene JPanel zu zeichnen.
ContentType	getContentType ()
int	getHeight ()

3.3. JAVADOC

Content

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

int	<code>getNumber()</code>
int	<code>getWidth()</code>
int	<code>getX()</code>
int	<code>getY()</code>
void	<code>setContentType(ContentType contentType)</code>
void	<code>setHeight(int height)</code>
void	<code>setNumber(int number)</code>
void	<code>setWidth(int width)</code>
void	<code>setX(int x)</code>
void	<code>setY(int y)</code>

Methods inherited from class `java.lang.Object`

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Constructor Detail

Content

```
public Content(int x,
               int y,
               int width,
               int height,
               ContentType contentType)
```

Konstruktor muss aus Unterklasse aufgerufen werden. Es werden alle Attribute mit den übergebenen Parametern initialisiert.

Parameters:

x - die x-Koordinate des Seiteninhalts

y - die y-Koordinate des Seiteninhalts

width - die Breite des Seiteninhalts

height - die Höhe des Seiteninhalts

contentType - der Typ der abgeleiteten Klasse zum sicheren Upcast

Method Detail

3.3. JAVADOC

Content

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

getNumber

```
public int getNumber()
```

getX

```
public int getX()
```

setX

```
public void setX(int x)
```

getY

```
public int getY()
```

setY

```
public void setY(int y)
```

getWidth

```
public int getWidth()
```

setWidth

```
public void setWidth(int width)
```

getHeight

```
public int getHeight()
```

setHeight

```
public void setHeight(int height)
```

3.3. JAVADOC

Content

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

setNumber

```
public void setNumber(int number)
```

getContentType

```
public ContentType getContentType()
```

setContentType

```
public void setContentType(ContentType contentType)
```

draw

```
public abstract java.awt.Component draw(javax.swing.JPanel panel)
```

Der Seiteninhalt wird angewiesen sich auf das übergebene JPanel zu zeichnen.

Parameters:

panel - das JPanel auf welches gezeichnet werden soll

Returns:

die gezeichnete Komponente, kann verwendet werden um listener zu initialisieren

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

ContentInstruction

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

pageData

Class ContentInstruction

java.lang.Object
pageData.ContentInstruction

```
public class ContentInstruction
extends java.lang.Object
```

Diese Klasse speichert eine in der Datenbank noch nicht eingetragene Änderung innerhalb einer Seite. Pro Änderung muss eine ContentInstruction angelegt werden. Jede ContentInstruction speichert die Nummer des betroffenen Contents und die Aktion welche vorgenommen werden muss. Aktion kann löschen, speichern, und erstellen sein

Constructor Summary

Constructors

Constructor and Description

<code>ContentInstruction(int contentNumber, ActionType actionType)</code> Es wird ein Objekt der Klasse ContentInstruction erstellt.

Method Summary

Methods

Modifier and Type	Method and Description
ActionType	getActionType()
int	getContentNumber()
void	setActionType(ActionType actionType)
void	setContentNumber(int contentNumber)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

3.3. JAVADOC

ContentInstruction

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Constructor Detail

ContentInstruction

```
public ContentInstruction(int contentNumber,  
                          ActionType actionType)
```

Es wird ein Objekt der Klasse ContentInstruction erstellt. Die Attribute werden anhand der Parameter belegt.

Parameters:

contentNumber - Nummer des betroffenen Seiteninhalts
actionType - löschen, speichern oder erstellen

Method Detail

getContentNumber

```
public int getContentNumber()
```

setContentNumber

```
public void setContentNumber(int contentNumber)
```

getActionType

```
public ActionType getActionType()
```

setActionType

```
public void setActionType(ActionType actionType)
```

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

3.3. JAVADOC

Controller

file:///home/felilein/git/TwoNote/TwoNote/doc/dat...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dataProcessing

Class Controller

java.lang.Object
dataProcessing.Controller

```
public class Controller
extends java.lang.Object
```

Die Klasse Controller dient der Kapselung der Datenverwaltung und ist Teil des MVC-Design. Der Controller speichert und verwaltet die aktuell geöffnete Sitzung des Programms. Mittels des Controllers kann eine Seite geladen und modifiziert werden. Alle komplexeren Aufgaben, welcher nicht ohne Verarbeitungslogik von View zu Model weitergeleitet werden können werden vom Controller verarbeitet.

Constructor Summary

Constructors

Constructor and Description

Controller(Model model)

Es wird eine Instanz eines Controllers erstellt und der Verzeichnisbaum aktualisiert
--

Method Summary

Methods

Modifier and Type	Method and Description
Content	createContent (ContentType type, int x, int y) Auf der Aktuell geöffneten Seite wird ein Seiteninhalt erstellt, und somit auf der aktuellen Seite als ungespeicherte Änderung vermerkt.
void	deleteContent (Content c) Ein übergebener Seiteninhalt wird auf der aktuellen Seite gelöscht.
Content[]	getContent () Es wird der Inhalt der aktuell geöffneten Seite als Content[] zurückgegeben
PageInformation	getCurrentPageInformation () Es werden die Seiteninformationen zur aktuellen Seite zurückgegeben.
Tree	getDataTree () Der aktuell geladene Verzeichnisbaum wird zurückgegeben.

3.3. JAVADOC

Controller

file:///home/felilein/git/TwoNote/TwoNote/doc/dat...

void	loadPage (java.lang.String pageName, java.lang.String chapterName, java.lang.String bookName) Es zu einem vorgegebenen Pfad eine Seite aus dem Model geladen und als aktuelle Seite gespeichert.
void	refreshTree () Der Verzeichnisbaum wird aktualisiert.
void	saveContent (Content c) Ein übergebener Seiteninhalt wird auf der aktuellen Seite gespeichert.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Controller

```
public Controller(Model model)
    throws java.sql.SQLException
```

Es wird eine Instanz eines Controllers erstellt und der Verzeichnisbaum aktualisiert

Throws:

java.sql.SQLException - Datenbankfehler

Method Detail

loadPage

```
public void loadPage(java.lang.String pageName,
    java.lang.String chapterName,
    java.lang.String bookName)
    throws DirectoryDoesNotExistsException,
        java.sql.SQLException,
        java.io.IOException,
        java.lang.ClassNotFoundException
```

Es zu einem vorgegebenen Pfad eine Seite aus dem Model geladen und als aktuelle Seite gespeichert. Sollte die angeforderte Seite Startseite sein, wird eine Instanz der Startseite erstellt und in currentPage gespeichert. Die geladene Seite ist direkt nicht von der View aus ansprechbar, sondern es müssen die Inhalte oder die PageInformation über die entsprechenden Methoden abgefragt werden.

Parameters:

pageName - der Name der Seite

chapterName - der Name des Kapitels

3.3. JAVADOC

Controller

file:///home/felilein/git/TwoNote/TwoNote/doc/dat...

bookName - der Name des Buches

Throws:

`DirectoryDoesNotExistsException` - Pfad nicht korrekt

`java.sql.SQLException` - Datenbankfehler

`java.lang.ClassNotFoundException` - Fehler beim deserialisieren

`java.io.IOException`

getContent

```
public Content[] getContent()  
    throws java.lang.NullPointerException
```

Es wird der Inhalt der aktuell geöffneten Seite als Content[] zurückgegeben

Returns:

der Inhalt der aktuell geöffneten Seite

Throws:

`java.lang.NullPointerException` - es ist aktuell keine Seite geladen

createContent

```
public Content createContent(ContentType type,  
    int x,  
    int y)
```

Auf der Aktuell geöffneten Seite wird ein Seiteninhalt erstellt, und somit auf der aktuellen Seite als ungespeicherte Änderung vermerkt.

Parameters:

type - der Typ des Seiteninhalts

x - Die x-Koordinate an welcher der Inhalt erstellt werden soll

y - Die y-Koordinate an welcher der Inhalt erstellt werden soll

Returns:

der erstellte Seiteninhalt

saveContent

```
public void saveContent(Content c)  
    throws DirectoryDoesNotExistsException,  
    java.sql.SQLException,  
    java.io.IOException
```

Ein übergebener Seiteninhalt wird auf der aktuellen Seite gespeichert. Sollte die aktuell geöffnete Seite nicht

3.3. JAVADOC

Controller

file:///home/felilein/git/TwoNote/TwoNote/doc/dat...

die Startseite sein wird diese gespeichert.

Parameters:

c - der Seiteninhalt

Throws:

`DirectoryDoesNotExistException` - der Pfad der Seite ist falsch

`java.sql.SQLException` - Datenbankfehler

`java.io.IOException` - Fehler beim Serialisieren

deleteContent

```
public void deleteContent(Content c)
    throws DirectoryDoesNotExistException,
           java.sql.SQLException,
           java.io.IOException
```

Ein übergebener Seiteninhalt wird auf der aktuellen Seite gelöscht. Sollte die aktuell geöffnete Seite nicht die Startseite sein wird diese gespeichert.

Parameters:

c - der Seiteninhalt

Throws:

`DirectoryDoesNotExistException` - der Pfad der Seite ist falsch

`java.sql.SQLException` - Datenbankfehler

`java.io.IOException` - Fehler beim Serialisieren

refreshTree

```
public void refreshTree()
    throws java.sql.SQLException
```

Der Verzeichnisbaum wird aktualisiert. Diese Methode wird bei instanziierten des Controllers aufgerufen, und muss danach von der View aufgerufen werden, falls die Anzeige des Verzeichnisbaums im GUI aktualisiert werden soll. Der vom Model geladene Verzeichnisbaum wird in `dataTree` gespeichert und kann über die Methode `getDataTree()` abgefragt werden.

Throws:

`java.sql.SQLException` - Datenbankfehler

getDataTree

```
public Tree getDataTree()
```

Der aktuell geladene Verzeichnisbaum wird zurückgegeben. Gegebenfalls muss dieser vorher mittels der Methode `refreshTree()` aktualisiert werden.

3.3. JAVADOC

Controller

file:///home/felilein/git/TwoNote/TwoNote/doc/dat...

Returns:

der Verzeichnisbaum

getCurrentPageInformation

```
public PageInformation getCurrentPageInformation()
```

Es werden die Seiteninformationen zur aktuellen Seite zurückgegeben.

Returns:

die Seiteninformationen

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested | Field | Constr | Method](#) [Detail: Field | Constr | Method](#)

pageData

Class DirectoryAlreadyExistsException

java.lang.Object
 java.lang.Throwable
 java.lang.Exception
 java.lang.RuntimeException
 pageData.DirectoryAlreadyExistsException

All Implemented Interfaces:

java.io.Serializable

```
public class DirectoryAlreadyExistsException
extends java.lang.RuntimeException
```

Diese Exception dient der Information über eine fehlerhaften Zugriff auf die Datenbank. Soll ein Verzeichnis angelegt werden, welches bereits existiert wird diese Exception geworfen. Es wird gespeichert, ob es sich um ein Buch, ein Kapitel, eine Seite oder einen Seiteninhalt handelt. Des Weiteren wird gespeichert, bei welcher Aktion die Ausnahme aufgetreten ist. Im Normalfall ist dies create.

See Also:

[Serialized Form](#)

Constructor Summary

Constructors

Constructor and Description

DirectoryAlreadyExistsException(DirectoryType dirType, ActionType actionType, java.lang.String directoryTitle)

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

DirectoryAlreadyExistsException(DirectoryType dirType, ActionType actionType, java.lang.String directoryTitle, java.lang.String msg)

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

Method Summary

Methods

Modifier and Type	Method and Description
ActionType	getActionType()

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

java.lang.String	getDirectoryTitle()
DirectoryType	getDirType()
void	setActionType(ActionType actionType)
void	setDirectoryTitle(java.lang.String directoryTitle)
void	setDirType(DirectoryType dirType)

Methods inherited from class java.lang.Throwable

addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DirectoryAlreadyExistsException

```
public DirectoryAlreadyExistsException(DirectoryType dirType,
                                       ActionType actionType,
                                       java.lang.String directoryTitle)
```

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

Parameters:

- dirType - der Typ des problematischen Verzeichnisses
- actionType - die Art der Aktion bei welcher der Fehler aufgetreten ist
- directoryTitle - der Titel des Verzeichnisses

DirectoryAlreadyExistsException

```
public DirectoryAlreadyExistsException(DirectoryType dirType,
                                       ActionType actionType,
                                       java.lang.String directoryTitle,
                                       java.lang.String msg)
```

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt. Hier kann zusätzlich eine Fehlermeldung übergeben werden.

Parameters:

- dirType - der Typ des problematischen Verzeichnisses
- actionType - die Art der Aktion bei welcher der Fehler aufgetreten ist
- directoryTitle - der Titel des Verzeichnisses

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

msg - die Fehlermeldung

Method Detail

getDirType

```
public DirectoryType getDirType()
```

setDirType

```
public void setDirType(DirectoryType dirType)
```

getDirectoryTitle

```
public java.lang.String getDirectoryTitle()
```

setDirectoryTitle

```
public void setDirectoryTitle(java.lang.String directoryTitle)
```

getActionType

```
public ActionType getActionType()
```

setActionType

```
public void setActionType(ActionType actionType)
```

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested | Field | Constr | Method](#) [Detail: Field | Constr | Method](#)

pageData

Class DirectoryAlreadyExistsException

java.lang.Object
 java.lang.Throwable
 java.lang.Exception
 java.lang.RuntimeException
 pageData.DirectoryAlreadyExistsException

All Implemented Interfaces:

java.io.Serializable

```
public class DirectoryAlreadyExistsException
extends java.lang.RuntimeException
```

Diese Exception dient der Information über eine fehlerhaften Zugriff auf die Datenbank. Soll ein Verzeichnis angelegt werden, welches bereits existiert wird diese Exception geworfen. Es wird gespeichert, ob es sich um ein Buch, ein Kapitel, eine Seite oder einen Seiteninhalt handelt. Des Weiteren wird gespeichert, bei welcher Aktion die Ausnahme aufgetreten ist. Im Normalfall ist dies create.

See Also:

[Serialized Form](#)

Constructor Summary

Constructors

Constructor and Description

DirectoryAlreadyExistsException(DirectoryType dirType, ActionType actionType, java.lang.String directoryTitle)

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

DirectoryAlreadyExistsException(DirectoryType dirType, ActionType actionType, java.lang.String directoryTitle, java.lang.String msg)

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

Method Summary

Methods

Modifier and Type	Method and Description
ActionType	getActionType()

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

java.lang.String	getDirectoryTitle()
DirectoryType	getDirType()
void	setActionType(ActionType actionType)
void	setDirectoryTitle(java.lang.String directoryTitle)
void	setDirType(DirectoryType dirType)

Methods inherited from class java.lang.Throwable

addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DirectoryAlreadyExistsException

```
public DirectoryAlreadyExistsException(DirectoryType dirType,
                                       ActionType actionType,
                                       java.lang.String directoryTitle)
```

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt.

Parameters:

- dirType - der Typ des problematischen Verzeichnisses
- actionType - die Art der Aktion bei welcher der Fehler aufgetreten ist
- directoryTitle - der Titel des Verzeichnisses

DirectoryAlreadyExistsException

```
public DirectoryAlreadyExistsException(DirectoryType dirType,
                                       ActionType actionType,
                                       java.lang.String directoryTitle,
                                       java.lang.String msg)
```

Es wird ein Objekt der Klasse DirectoryAlreadyExistsException erzeugt. Hier kann zusätzlich eine Fehlermeldung übergeben werden.

Parameters:

- dirType - der Typ des problematischen Verzeichnisses
- actionType - die Art der Aktion bei welcher der Fehler aufgetreten ist
- directoryTitle - der Titel des Verzeichnisses

3.3. JAVADOC

DirectoryAlreadyExistsException

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

msg - die Fehlermeldung

Method Detail

getDirType

```
public DirectoryType getDirType()
```

setDirType

```
public void setDirType(DirectoryType dirType)
```

getDirectoryTitle

```
public java.lang.String getDirectoryTitle()
```

setDirectoryTitle

```
public void setDirectoryTitle(java.lang.String directoryTitle)
```

getActionType

```
public ActionType getActionType()
```

setActionType

```
public void setActionType(ActionType actionType)
```

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dataManagement

Class MySqlDatabase

java.lang.Object
dataManagement.MySqlDatabase

All Implemented Interfaces:

IDataBase

```
public class MySqlDatabase
    extends java.lang.Object
    implements IDataBase
```

Dies ist die konkrete Implementierung der Schnittstelle zur Datenbank. Dazu implementiert diese Klasse das Interface IDataBase, welches eine abstrakte Definition der Schnittstelle von Java zu einer beliebigen Datenbank vorgibt. Bei der konkret in dieser Klasse verwendeten Datenbank handelt es sich um MySql

Constructor Summary

Constructors

Constructor and Description

MySqlDatabase (java.lang.String url, java.lang.String name, java.lang.String pw)

Der Konstruktor lädt den jdbc Treiber und richtet danach eine Datenbankverbindung mittels der übergebenen Daten ein, indem er das Attribut con instanziiert.

Method Summary

Methods

Modifier and Type	Method and Description
void	createBook (java.lang.String bookName) Diese Methode legt ein neues Buch in der Tabelle "Books" an
void	createChapter (java.lang.String chapterName, int bookID) Diese Methode legt ein neues Kapitel in der Tabelle "Chapter" an
void	createContent (byte[] contentBytes, int contentNumber, int pageID) Diese Methode legt einen neuen Seiteninhalt in der Tabelle "Content" an
void	createPage (PageInformation pageInfo, int chapterID) Diese Methode legt eine neue Seite in der Tabelle "Pages" an

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

void	deleteBook (int bookID) Diese Methode löscht ein Buch aus der Tabelle "Books"
void	deleteChapter (int chapterID) Diese Methode löscht ein Kapitel aus der Tabelle "Chapter"
void	deleteContent (int contentID) Diese Methode löscht einen Seiteninhalt aus der Tabelle "Content"
void	deletePage (int pageID) Diese Methode löscht eine Seite aus der Tabelle "Pages"
boolean	existsBook (java.lang.String bookName) Diese Methode gibt zurück, ob es in der Tabelle "Books" ein Buch mit dem Namen "bookName" gibt
boolean	existsChapter (java.lang.String chapterName, int bookID) Diese Methode gibt zurück, ob es in der Tabelle "Chapter" ein Kapitel mit dem Namen "chapterName" gibt
boolean	existsContent (int contentNumber, int pageID) Diese Methode gibt zurück, ob es in der Tabelle "Content" ein Eintrag mit der Nummer "contentNumber" gibt
boolean	existsPage (java.lang.String pageName, int chapterID) Diese Methode gibt zurück, ob es in der Tabelle "Pages" eine Seite mit dem Namen "pageName" gibt
java.lang.String[]	getAllBooks () Diese Methode liefert den Namen aller Bücher welche in der Datenbank angelegt sind
java.lang.String[]	getBookChildren (int bookID) Diese Methode liefert zur id eines Buches alle Kapitel welche zu diesem Buch gehören
int	getBookID (java.lang.String bookName) Diese Methode liefert zu dem Namen eines Buches die entsprechende id in der Datenbank
java.lang.String[]	getChapterChildren (int chapterID) Diese Methode liefert zur id eines Kapitels alle Seiten welche zu diesem Kapitel gehören
int	getChapterID (java.lang.String chapterName, int bookID) Diese Methode liefert zu dem Namen eines Kapitels die entsprechende id in der Datenbank
int	getContentID (int ContentNumber, int pageID) Diese Methode liefert zu der Nummer eines Seiteninhalts die entsprechende id in der Datenbank
int[]	getPageChildren (int pageID) Diese Methode liefert zur id einer Seite alle Inhalte welche zu dieser Seite gehören
int	getPageID (java.lang.String pageName, int chapterID) Diese Methode liefert zu dem Namen einer Seite die entsprechende id in der Datenbank
byte[]	loadContent (int contentID) Diese Methode lädt einen Seiteninhalt aus der Tabelle "Content"
PageInformation	loadPageInformation (int pageID) Diese Methode lädt die Seiteninformation aus der Tabelle "Pages"
void	saveContent (byte[] contentBytes, int contentID) Diese Methode speichert einen Seiteninhalt in der Tabelle "Content"

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

void	<code>savePage(PageInformation pageInfo, int chapterID, int pageID)</code> Diese Methode speichert eine Seite in der Tabelle "Pages"
------	---

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

MySqlDatabase

```
public MySqlDatabase(java.lang.String url,  
                    java.lang.String name,  
                    java.lang.String pw)  
    throws java.lang.ClassNotFoundException,  
           java.sql.SQLException
```

Der Konstruktor lädt den jdbc Treiber und richtet danach eine Datenbankverbindung mittels der übergebenen Daten ein, indem er das Attribut con instanziiert.

Parameters:

url - der Pfad zur Datenbank
name - der Loginname zur Datenbank
pw - das Passwort zur Datenbank

Throws:

java.lang.ClassNotFoundException - Fehler beim Laden des jdbc Treibers
java.sql.SQLException - Datenbankfehler

Method Detail

createBook

```
public void createBook(java.lang.String bookName)  
    throws java.sql.SQLException
```

Diese Methode legt ein neues Buch in der Tabelle "Books" an

Specified by:

`createBook` in interface `IDataBase`

Parameters:

bookName - der Name des anzulegenden Buches

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Throws:

java.sql.SQLException - Datenbankfehler

createChapter

```
public void createChapter(java.lang.String chapterName,  
                          int bookID)  
    throws java.sql.SQLException
```

Diese Methode legt ein neues Kapitel in der Tabelle "Chapter" an

Specified by:

createChapter in interface IDataBase

Parameters:

chapterName - der Name des anzulegenden Kapitels

bookID - die id des Buches zu dem das anzulegende Kapitel gehören soll

Throws:

java.sql.SQLException - Datenbankfehler

createPage

```
public void createPage(PageInformation pageInfo,  
                      int chapterID)  
    throws java.sql.SQLException
```

Diese Methode legt eine neue Seite in der Tabelle "Pages" an

Specified by:

createPage in interface IDataBase

Parameters:

pageInfo - informationen zur Seite

chapterID - die id des Kapitels zu dem die anzulegende Seite gehören soll

Throws:

java.sql.SQLException - Datenbankfehler

createContent

```
public void createContent(byte[] contentBytes,  
                          int contentNumber,  
                          int pageID)  
    throws java.sql.SQLException
```

Diese Methode legt einen neuen Seiteninhalt in der Tabelle "Content" an

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Specified by:

`createContent` in interface `IDataBase`

Parameters:

`contentBytes` - das serialisierte Objekt

`contentNumber` - die nummer des Seiteninhalts innerhalb seiner Seite

`pageID` - die id der Seite zu welcher der Inhalt gehören soll

Throws:

`java.sql.SQLException` - Datenbankfehler

deleteBook

```
public void deleteBook(int bookID)
    throws java.sql.SQLException
```

Diese Methode löscht ein Buch aus der Tabelle "Books"

Specified by:

`deleteBook` in interface `IDataBase`

Parameters:

`bookID` - Die id des Buches, welches gelöscht werden soll

Throws:

`java.sql.SQLException` - Datenbankfehler

deleteChapter

```
public void deleteChapter(int chapterID)
    throws java.sql.SQLException
```

Diese Methode löscht ein Kapitel aus der Tabelle "Chapter"

Specified by:

`deleteChapter` in interface `IDataBase`

Parameters:

`chapterID` - Die id des Kapitels, welches gelöscht werden soll

Throws:

`java.sql.SQLException` - Datenbankfehler

deletePage

```
public void deletePage(int pageID)
```


3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

throws java.sql.SQLException

Diese Methode löscht eine Seite aus der Tabelle "Pages"

Specified by:

deletePage in interface IDataBase

Parameters:

pageID - Die id der Seite, welche gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

deleteContent

```
public void deleteContent(int contentID)
    throws java.sql.SQLException
```

Diese Methode löscht einen Seiteninhalt aus der Tabelle "Content"

Specified by:

deleteContent in interface IDataBase

Parameters:

contentID - Die id des Seiteninhaltes, welcher gelöscht werden soll

Throws:

java.sql.SQLException - Datenbankfehler

saveContent

```
public void saveContent(byte[] contentBytes,
    int contentID)
    throws java.sql.SQLException
```

Diese Methode speichert einen Seiteninhalt in der Tabelle "Content"

Specified by:

saveContent in interface IDataBase

Parameters:

contentBytes - das serialisierte Objekt

contentID - die innerhalb der Tabelle "Content"

Throws:

java.sql.SQLException - Datenbankfehler

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

savePage

```
public void savePage(PageInformation pageInfo,
    int chapterID,
    int pageID)
    throws java.sql.SQLException
```

Diese Methode speichert eine Seite in der Tabelle "Pages"

Specified by:

`savePage` in interface `IDataBase`

Parameters:

`pageInfo` - die Information zur Seite
`chapterID` - das Kapitel zu welchem die Seite gehört
`pageID` - die id der Seite innerhalb der Tabelle "Pages"

Throws:

`java.sql.SQLException` - Datenbankfehler

loadContent

```
public byte[] loadContent(int contentID)
    throws java.sql.SQLException
```

Diese Methode lädt einen Seiteninhalt aus der Tabelle "Content"

Specified by:

`loadContent` in interface `IDataBase`

Parameters:

`contentID` - die id des angefragten Seiteninhalts

Returns:

das serialisierte Objekt zur angefragten id

Throws:

`java.sql.SQLException` - Datenbankfehler

loadPageInformation

```
public PageInformation loadPageInformation(int pageID)
    throws java.sql.SQLException
```

Diese Methode lädt die Seiteninformation aus der Tabelle "Pages"

Specified by:

`loadPageInformation` in interface `IDataBase`

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Parameters:

pageID - die id der angefragten Seite

Returns:

die Seiteninformation zur angefragten id

Throws:

java.sql.SQLException - Datenbankfehler

existsBook

```
public boolean existsBook(java.lang.String bookName)
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Books" ein Buch mit dem Namen "bookName" gibt

Specified by:

`existsBook` in interface `IDataBase`

Parameters:

bookName - der Name des Buchs

Returns:

zum angefragten Buch existiert ein Eintrag in der Datenbank

Throws:

java.sql.SQLException - Datenbankfehler

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

existsChapter

```
public boolean existsChapter(java.lang.String chapterName,  
                             int bookID)  
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Chapter" ein Kapitel mit dem Namen "chapterName" gibt

Specified by:

`existsChapter` in interface `IDataBase`

Parameters:

`chapterName` - der Name des Kapitels

`bookID` - die id des Buches zu welchem das Kapitel gehört

Returns:

zum angefragten Kapitel existiert ein Eintrag in der Datenbank

Throws:

`java.sql.SQLException` - Datenbankfehler

existsPage

```
public boolean existsPage(java.lang.String pageName,  
                           int chapterID)  
    throws java.sql.SQLException
```

Diese Methode gibt zurück, ob es in der Tabelle "Pages" eine Seite mit dem Namen "pageName" gibt

Specified by:

`existsPage` in interface `IDataBase`

Parameters:

`pageName` - der Name der Seite

`chapterID` - die id des Kapitels zu welchem die Seite gehört

Returns:

zur angefragten Seite existiert ein Eintrag in der Datenbank

Throws:

`java.sql.SQLException` - Datenbankfehler

existsContent

```
public boolean existsContent(int contentNumber,  
                             int pageID)  
    throws java.sql.SQLException
```

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Diese Methode gibt zurück, ob es in der Tabelle "Content" ein Eintrag mit der Nummer "contentNumber" gibt

Specified by:

`existsContent` in interface `IDataBase`

Parameters:

`contentNumber` - die Nummer des Seiteninhalts

`pageID` - die id der Seite zu welcher der Inhalt gehört

Returns:

zum angefragten Seiteninhalt existiert ein Eintrag in der Datenbank

Throws:

`java.sql.SQLException` - Datenbankfehler

getBookID

```
public int getBookID(java.lang.String bookName)
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen eines Buches die entsprechende id in der Datenbank

Specified by:

`getBookID` in interface `IDataBase`

Parameters:

`bookName` - Der Name des angefragten Buches

Returns:

die id des angefragten Buches

Throws:

`java.sql.SQLException` - Datenbankfehler

getChapterID

```
public int getChapterID(java.lang.String chapterName,
    int bookID)
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen eines Kapitels die entsprechende id in der Datenbank

Specified by:

`getChapterID` in interface `IDataBase`

Parameters:

`chapterName` - Der Name des angefragten Kapitels

`bookID` - die id des Buches zu welchem das Kapitel gehört

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Returns:

die id des angefragten Kapitels

Throws:

java.sql.SQLException - Datenbankfehler

getPageID

```
public int getPageID(java.lang.String pageName,  
                    int chapterID)  
    throws java.sql.SQLException
```

Diese Methode liefert zu dem Namen einer Seite die entsprechende id in der Datenbank

Specified by:

getPageID in interface IDataBase

Parameters:

pageName - Der Name der angefragten Seite

chapterID - die id des Kapitels zu welchem die Seite gehört

Returns:

die id der angefragten Seite

Throws:

java.sql.SQLException - Datenbankfehler

getContentID

```
public int getContentID(int ContentNumber,  
                       int pageID)  
    throws java.sql.SQLException
```

Diese Methode liefert zu der Nummer eines Seiteinhalts die entsprechende id in der Datenbank

Specified by:

getContentID in interface IDataBase

Parameters:

ContentNumber - Die Nummer des angefragten Seiteninhalts

pageID - die id der Seite zu welchem der Inhalt gehört

Returns:

die id des angefragten Seiteinhalts

Throws:

java.sql.SQLException - Datenbankfehler

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

getBookChildren

```
public java.lang.String[] getBookChildren(int bookID)
                           throws java.sql.SQLException
```

Diese Methode liefert zur id eines Buches alle Kapitel welche zu diesem Buch gehören

Specified by:

`getBookChildren` in interface `IDataBase`

Parameters:

`bookID` - Die id des angefragten Buches

Returns:

der Name aller Kapitel, welche zum angefragten Buch gehören

Throws:

`java.sql.SQLException` - Datenbankfehler

getChapterChildren

```
public java.lang.String[] getChapterChildren(int chapterID)
                           throws java.sql.SQLException
```

Diese Methode liefert zur id eines Kapitels alle Seiten welche zu diesem Kapitel gehören

Specified by:

`getChapterChildren` in interface `IDataBase`

Parameters:

`chapterID` - Die id des angefragten Kapitels

Returns:

der Name aller Seiten, welche zum angefragten Kapitel gehören

Throws:

`java.sql.SQLException` - Datenbankfehler

getPageChildren

```
public int[] getPageChildren(int pageID)
                           throws java.sql.SQLException
```

Diese Methode liefert zur id einer Seite alle Inhalte welche zu dieser Seite gehören

Specified by:

`getPageChildren` in interface `IDataBase`

Parameters:

3.3. JAVADOC

MySqlDatabase

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

pageID - Die id der angefragten Seite

Returns:

der Name aller Inhalte, welche zur angefragten Seite gehören

Throws:

java.sql.SQLException - Datenbankfehler

getAllBooks

```
public java.lang.String[] getAllBooks()  
    throws java.sql.SQLException
```

Diese Methode liefert den Namen aller Bücher welche in der Datenbank angelegt sind

Specified by:

getAllBooks in interface IDataBase

Returns:

ein String[] mit den Namen aller angelegten Büchern

Throws:

java.sql.SQLException - Datenbankfehler

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

Page

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

pageData

Class Page

java.lang.Object
pageData.Page

Direct Known Subclasses:

[Startpage](#)

```
public class Page
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
static int	DEFAULT_HEIGHT
static int	DEFAULT_WIDTH

Constructor Summary

Constructors

Constructor and Description
Page (PageInformation pageInfo) Erstellt eine neue Seite mit parametrierbaren Seiteninformationen.

Method Summary

Methods

Modifier and Type	Method and Description
void	addContent (Content cont) Dies ist nur die Schnittstelle der Seite zum Model.

3.3. JAVADOC

Page

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

void	contentInstructionCompleted() Der Seite wird mitgeteilt, dass eine contentInstruction korrekt bearbeitet wurde.
Content	createContent(Content cont) Es wird ein neuer Seiteninhalt innerhalb der Seite gespeichert Der neue Inhalt wird mit einer Nummer versehen und zurückgegeben.
void	deleteContent(int number) Der übergebene Seiteninhalt wird von der Seite gelöscht.
Content[]	getContent()
Content	getContent(int number)
int	getContentInstructionSize() Diese Methode gibt die Anzahl der noch nicht in der Datenbank gespeicherten Änderungen zurück
ContentInstruction	getNextContentInstruction() Liefert die nächste ContentInstruction.
PageInformation	getPageInfo()
void	saveContent(Content cont) Der übergebene Seiteninhalt wird innerhalb der Seite gespeichert.
void	setPageInfo(PageInformation pageInfo)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

DEFAULT_WIDTH

public static final int DEFAULT_WIDTH

See Also:

Constant Field Values

DEFAULT_HEIGHT

public static final int DEFAULT_HEIGHT

See Also:

Constant Field Values

Constructor Detail

3.3. JAVADOC

Page

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Page

```
public Page(PageInformation pageInfo)
```

Erstellt eine neue Seite mit parametrierbaren Seiteninformationen. Die Nummer des ersten Seiteninhalts wird auf 0 gesetzt.

Method Detail

getPageInfo

```
public PageInformation getPageInfo()
```

setPageInfo

```
public void setPageInfo(PageInformation pageInfo)
```

getContent

```
public Content[] getContent()
```

getContent

```
public Content getContent(int number)
    throws DirectoryDoesNotExistException
```

Throws:

DirectoryDoesNotExistException

getNextContentInstruction

```
public ContentInstruction getNextContentInstruction()
```

Liefert die nächste ContentInstruction. Es sollte vorher überprüft werden, ob noch ungespeicherte Änderungen vorhanden sind. Achtung: wird die ContentInstruction korrekt bearbeitet muss zuerst "contentInstructionCompleted" aufgerufen werden, damit die nächste ContentInstruction bearbeitet werden kann. Somit wird sichergestellt, dass die Daten jederzeit konsistent sind, auch wenn es beim Speichern einer Änderung zu einem Fehler kommt.

Returns:

die nächste contentInstruction

3.3. JAVADOC

Page

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Throws:

`java.lang.RuntimeException` - es wurden bereits alle contentInstructions bearbeitet

getContentInstructionSize

```
public int getContentInstructionSize()
```

Diese Methode gibt die Anzahl der noch nicht in der Datenbank gespeicherten Änderungen zurück

Returns:

Anzahl der ungespeicherten Änderungen

createContent

```
public Content createContent(Content cont)
```

Es wird ein neuer Seiteninhalt innerhalb der Seite gespeichert. Der neue Inhalt wird mit einer Nummer versehen und zurückgegeben. Der Zähler für die Nummer des nächsten Seiteninhalts wird inkrementiert. Es wird eine contentInstruction angelegt.

Returns:

der Seiteninhalt mit Nummer

deleteContent

```
public void deleteContent(int number)
```

Der übergebene Seiteninhalt wird von der Seite gelöscht. Falls er nicht Teil der Seite ist, wird eine Exception geworfen. Falls der Seiteninhalt noch gar nicht in der Datenbank gespeichert wurde, wird die Anweisung den Inhalt zu erstellen aus den contentInstructions gelöscht. Falls nicht, wird eine neue contentInstruction angelegt, welche dokumentiert, dass der Inhalt aus der Datenbank gelöscht werden soll.

Parameters:

number - die Nummer des zu löschenden Seiteninhalts

Throws:

`DirectoryDoesNotExistsException` - es existiert kein Seiteninhalt mit der angegebenen Nummer

saveContent

```
public void saveContent(Content cont)
```

Der übergebene Seiteninhalt wird innerhalb der Seite gespeichert. Es wird den contentInstructions ein Eintrag hinzugefügt, sodass die Änderung in der Datenbank übernommen wird, falls diese Seite das nächste mal gespeichert wird.

Parameters:

3.3. JAVADOC

Page

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

cont - der zu speichernde Inhalt

contentInstructionCompleted

```
public void contentInstructionCompleted()
```

Der Seite wird mitgeteilt, dass eine contentInstruction korrekt bearbeitet wurde. Die contentInstruction wird gelöscht und es kann die nächste abgefragt werden.

Throws:

`java.lang.RuntimeException` - Es wurden bereits alle ContentInstructions abgeschlossen

addContent

```
public void addContent(Content cont)
```

Dies ist nur die Schnittstelle der Seite zum Model. Wird die Seite aus der Datenbank geladen, ruft das Model jeden dazugehörigen Seiteninhalt ab und fügt ihn mit dieser Methode zur Seite hinzu. Dem Inhalt wird einer Nummer hinzugefügt und der Nummerzähler wird inkrementiert

Parameters:

cont - Der hinzuzufügende Inhalt

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

PageInformation

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

pageData

Class PageInformation

java.lang.Object
pageData.PageInformation

```
public class PageInformation
extends java.lang.Object
```

Die PageInformation speichern Name des Buches, des Kapitels und der Seite. Zusätzlich wird zu jeder Seite die Höhe und die Breite gespeichert. Diese Informationen werden in der Datenbank gespeichert und beim Laden einer Seite dieser zugefügt. Die PageInformation wird beim Speichern einer Seite verwendet um den Pfad zur Seite zu erhalten.

Constructor Summary

Constructors

Constructor and Description

<p>PageInformation(java.lang.String pageName, java.lang.String chapterName, java.lang.String bookName, int width, int height)</p> <p>Es wird eine Instanz der Klasse PageInformation mit den übergebenen Parametern angelegt.</p>
--

Method Summary

Methods

Modifier and Type	Method and Description
java.lang.String	getBookName()
java.lang.String	getChapterName()
int	getHeight()
java.lang.String	getPageName()
int	getWidth()
void	setBookName(java.lang.String bookName)
void	setChapterName(java.lang.String chapterName)
void	setHeight(int height)
void	setPageName(java.lang.String pageName)
void	setWidth(int width)

3.3. JAVADOC

PageInformation

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PageInformation

```
public PageInformation(java.lang.String pageName,  
                       java.lang.String chapterName,  
                       java.lang.String bookName,  
                       int width,  
                       int height)
```

Es wird eine Instanz der Klasse PageInformation mit den übergebenen Parametern angelegt.

Parameters:

pageName - Name der Seite
chapterName - Name des Kapitels
bookName - Name des Buches
width - Breite der Seite
height - Höhe der Seite

Method Detail

getPageName

```
public java.lang.String getPageName()
```

setPageName

```
public void setPageName(java.lang.String pageName)
```

getChapterName

```
public java.lang.String getChapterName()
```

3.3. JAVADOC

PageInformation

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

setChapterName

```
public void setChapterName(java.lang.String chapterName)
```

getBookName

```
public java.lang.String getBookName()
```

setBookName

```
public void setBookName(java.lang.String bookName)
```

getHeight

```
public int getHeight()
```

setHeight

```
public void setHeight(int height)
```

getWidth

```
public int getWidth()
```

setWidth

```
public void setWidth(int width)
```

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

Serializer

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

dataManagement

Class Serializer

java.lang.Object
dataManagement.Serializer

```
public class Serializer  
extends java.lang.Object
```

Diese Klasse serialisiert ein Objekt der Klasse Content zu einem byte[] oder deserialisiert ein byte[] zu einem Objekt der Klasse Content. Diese Funktionalität wurde in diese Klasse ausgelagert um die Komplexität der anderen Klassen zu verringern, indem man die Funktionen um das serialisieren als Objekt kapselt.

Constructor Summary

Constructors

Constructor and Description

Serializer()

Method Summary

Methods

Modifier and Type	Method and Description
Content	deserialize (byte[] bytes) Diese Methode deserialisiert ein byte[]
byte[]	serialize (Content cont) Diese Methode serialisiert ein Objekt vom Typ Content

Methods inherited from class java.lang.Object

[equals](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Constructor Detail

3.3. JAVADOC

Serializer

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Serializer

```
public Serializer()
```

Method Detail

serialize

```
public byte[] serialize(Content cont)
                throws java.io.IOException
```

Diese Methode serialisiert ein Objekt vom Typ Content

Parameters:

cont - der zu serialisierende Content.

Returns:

das Object als bytecode

Throws:

java.io.IOException - Fehler beim serialisieren

deserialize

```
public Content deserialize(byte[] bytes)
                throws java.lang.ClassNotFoundException,
                       java.io.IOException
```

Diese Methode deserialisiert ein byte[]

Parameters:

bytes - der zu deserialisierende bytecode

Returns:

ein Objekt der Klasse Content

Throws:

java.io.IOException - Fehler beim deserialisieren

java.lang.ClassNotFoundException - fehler beim Lesen des Objekts

3.3. JAVADOC

Startpage

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

pageData

Class Startpage

java.lang.Object
 pageData.Page
 pageData.Startpage

public class **Startpage**
extends Page

Die Startpage gibt ist eine Seite, welche Informationen für den Benutzer über die Verwendung dieses Programms beinhaltet. Die Startpage wird angezeigt, wenn das Programm gestartet wird und wenn die Seite gelöscht wird, welche aktuell angezeigt wird.

Field Summary

Fields inherited from class pageData.Page

DEFAULT_HEIGHT, DEFAULT_WIDTH

Constructor Summary

Constructors

Constructor and Description

Startpage()

Method Summary

Methods inherited from class pageData.Page

addContent, contentInstructionCompleted, createContent, deleteContent, getContent, getContent, getContentInstructionSize, getNextContentInstruction, getPageInfo, saveContent, setPageInfo

Methods inherited from class java.lang.Object

3.3. JAVADOC

Startpage

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Startpage

`public Startpage()`

[Overview](#) [Package](#) **[Class](#)** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

TextBox

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

pageData

Class TextBox

java.lang.Object
 pageData.Content
 pageData.TextBox

All Implemented Interfaces:

java.io.Serializable

```
public class TextBox
    extends Content
    implements java.io.Serializable
```

Diese Klasse dient der internen Speicherung einer TextBox und ist von Content abgeleitet. Der Content wird durch diese Klasse um einen Text, eine Schriftgröße, eine Schriftfarbe und eine Hintergrundfarbe ergänzt. Diese Klasse kann auf einem vorgegebenem JPanel gezeichnet werden.

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description

TextBox(int x, int y) Es wird eine TextBox mit Defaultwerten an einer vorgegebenen Position erstellt.
TextBox(TextBox t) Der Copy-Constructor dient dazu eine tiefe Kopie einer TextBox anzulegen.

Method Summary

Methods

Modifier and Type	Method and Description
java.awt.Component	draw (javax.swing.JPanel panel) Der Seiteninhalt wird angewiesen sich auf das übergebene JPanel zu zeichnen.
java.awt.Color	getBackgroundColor ()
java.awt.Color	getFontColor ()

3.3. JAVADOC

TextBox

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

int	getFontSize()
java.lang.String	getText()
void	setBackgroundColor (java.awt.Color backgroundColor)
void	setFontColor (java.awt.Color fontColor)
void	setFontSize (int fontSize)
void	setText (java.lang.String text)
java.lang.String	toString()

Der Text der TextBox wird zurückgegeben.

Methods inherited from class **pageData.Content**

getContentType, getHeight, getNumber, getWidth, getX, getY, setContentType, setHeight, setNumber, setWidth, setX, setY

Methods inherited from class **java.lang.Object**

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

TextBox

```
public TextBox(int x,
               int y)
```

Es wird eine TextBox mit Defaultwerten an einer vorgegebenen Position erstellt.

Parameters:

x - die x-Koordinate

y - die y-Koordinate

TextBox

```
public TextBox(TextBox t)
```

Der Copy-Constructor dient dazu eine tiefe Kopie einer TextBox anzulegen.

Parameters:

t - Die TextBox welche kopiert werden soll.

3.3. JAVADOC

TextBox

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Method Detail

getText

```
public java.lang.String getText()
```

setText

```
public void setText(java.lang.String text)
```

getFontSize

```
public int getFontSize()
```

setFontSize

```
public void setFontSize(int fontSize)
```

getFontColor

```
public java.awt.Color getFontColor()
```

setFontColor

```
public void setFontColor(java.awt.Color fontColor)
```

getBackgroundColor

```
public java.awt.Color getBackgroundColor()
```

setBackgroundColor

```
public void setBackgroundColor(java.awt.Color backgroundColor)
```

toString

```
public java.lang.String toString()
```

3.3. JAVADOC

TextBox

file:///home/felilein/git/TwoNote/TwoNote/doc/pa...

Der Text der TextBox wird zurückgegeben.

Overrides:

`toString` in class `java.lang.Object`

Returns:

`TextBox.Text`

draw

```
public java.awt.Component draw(javax.swing.JPanel panel)
```

Der Seiteninhalt wird angewiesen sich auf das übergebene JPanel zu zeichnen.

Specified by:

`draw` in class `Content`

Parameters:

`panel` - das JPanel auf welches gezeichnet werden soll

Returns:

die gezeichnete Komponente, kann verwendet werden um listener zu initialisieren

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

TextEditTool

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Tools

Class TextEditTool

java.lang.Object
Tools.TextEditTool

All Implemented Interfaces:

Tool

```
public class TextEditTool
extends java.lang.Object
implements Tool
```

Das TextEditTool dient zum Bearbeiten einer TextBox. Dazu zeichnet sich das Tool auf ein vorgegebenes JPanel. Alle UI-Elemente, die benötigt werden um eine TextBox zu modifizieren, werden vom TextEditTool erstellt und später wieder entfernt. Es kann ein Listener hinzugefügt werden um informiert zu werden, wenn das Tool beendet wird. Das Tool wird beendet, falls es den Focus verliert, mit Enter beendet wird oder das zu bearbeitende Element entfernt wird.

Constructor Summary

Constructors

Constructor and Description

<code>TextEditTool(javax.swing.JPanel panel, TextBox content)</code> Es wird ein neues TextEditTool gestartet.

Method Summary

Methods

Modifier and Type	Method and Description
void	<code>addToolReadyHandler(ToolReadyHandler handler)</code> Möchte man wissen, wann das Tool beendet wurde, kann ein ToolReadyHandler hinzugefügt werden.
void	<code>drawTool()</code> Das Tool zeichnet sich auf dem übergebenen JPanel.
void	<code>endEdit(ToolEndsType type)</code> Diese Methode wird normalerweise automatisch aufgerufen sobald das Tool beendet wird.

3.3. JAVADOC

TextEditTool

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

Content	getContent() Hiermit kann der Seiteninhalt abgerufen werden, welcher durch das Tool bearbeitet wurde.
----------------	---

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

TextEditTool

```
public TextEditTool(javax.swing.JPanel panel,  
                    TextBox content)
```

Es wird ein neues TextEditTool gestartet.

Parameters:

panel - das JPanel auf dem das Tool gezeichnet wird
content - der Seiteninhalt, welcher vom Tool bearbeitet werden soll.

Method Detail

drawTool

```
public void drawTool()
```

Das Tool zeichnet sich auf dem übergebenen JPanel. Wird das Tool beendet entfernt es sich wieder. Hier werden auch alle Handler initialisiert, welche eine Interaktion mit dem Benutzer erlauben. Ein Tool sollte es ermöglichen ein Element zu verschieben es zu bearbeiten und zu löschen. Eine Verschiebung sollte mittels der Pfeiltasten und der Maus und das Löschen mittels "entfernen" möglich sein.

Specified by:

drawTool in interface Tool

endEdit

```
public void endEdit(ToolEndsType type)
```

Diese Methode wird normalerweise automatisch aufgerufen sobald das Tool beendet wird. Sollte man wünschen das Tool vorher manuell zu beenden, kann diese Methode aufgerufen werden.

Specified by:

3.3. JAVADOC

TextEditTool

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

endEdit in interface Tool

getContent

```
public Content getContent()
```

Hiermit kann der Seiteninhalt abgerufen werden, welcher durch das Tool bearbeitet wurde.

Specified by:

`getContent` in interface Tool

Returns:

der bearbeitete Seiteninhalt

addToolReadyHandler

```
public void addToolReadyHandler(ToolReadyHandler handler)
```

Möchte man wissen, wann das Tool beendet wurde, kann ein ToolReadyHandler hinzugefügt werden.

Specified by:

`addToolReadyHandler` in interface Tool

Parameters:

handler - ein ToolReadyHandler

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

Tool

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

Overview Package **Class** Use Tree Deprecated Index Help

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Tools

Interface Tool

All Known Implementing Classes:

[TextEditTool](#)

public interface **Tool**

Ein Tool dient zur Bearbeitung eines Seitenelements. Dazu muss das Tool mit dem Seitenelement und einem JPanel initialisiert werden. Ein Tool kann sich dann auf dem JPanel zeichnen und den Content modifizieren oder löschen. Wird das Tool beendet, so kann dies durch das Hinzufügen eines ToolReadyHandlers abgefangen und verarbeitet werden. Tool ist bewusst als Interface implementiert um das mächtigere Werkzeug der Vererbung für größere Tools verwendbar zu halten.

Method Summary

Methods

Modifier and Type	Method and Description
void	<code>addToolReadyHandler(ToolReadyHandler handler)</code> Möchte man wissen, wann das Tool beendet wurde, kann ein ToolReadyHandler hinzugefügt werden.
void	<code>drawTool()</code> Das Tool zeichnet sich auf dem übergebenen JPanel.
void	<code>endEdit(ToolEndsType type)</code> Diese Methode wird normalerweise automatisch aufgerufen sobald das Tool beendet wird.
Content	<code>getContent()</code> Hiermit kann der Seiteninhalt abgerufen werden, welcher durch das Tool bearbeitet wurde.

Method Detail

`drawTool`

void drawTool()

Das Tool zeichnet sich auf dem übergebenen JPanel. Wird das Tool beendet entfernt es sich wieder. Hier werden auch alle Handler initialisiert, welche eine Interaktion mit dem Benutzer erlauben. Ein Tool sollte es ermöglichen ein Element zu verschieben es zu bearbeiten und zu löschen. Eine Verschiebung sollte mittels

3.3. JAVADOC

Tool

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

der Pfeiltasten und der Maus und das Löschen mittels "entfernen" möglich sein.

endEdit

`void endEdit(ToolEndsType type)`

Diese Methode wird normalerweise automatisch aufgerufen sobald das Tool beendet wird. Sollte man wünschen das Tool vorher manuell zu beenden, kann diese Methode aufgerufen werden.

getContent

`Content getContent()`

Hiermit kann der Seiteninhalt abgerufen werden, welcher durch das Tool bearbeitet wurde.

Returns:

der bearbeitete Seiteninhalt

addToolReadyHandler

`void addToolReadyHandler(ToolReadyHandler handler)`

Möchte man wissen, wann das Tool beendet wurde, kann ein ToolReadyHandler hinzugefügt werden.

Parameters:

handler - ein ToolReadyHandler

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.3. JAVADOC

ToolReadyHandler

file:///home/felilein/git/TwoNote/TwoNote/doc/Too...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Tools

Interface ToolReadyHandler

public interface **ToolReadyHandler**

Dieser Handler kann als anonyme innere Klasse implementiert werden, falls es nötig ist darauf zu reagieren, wenn das Tool die Bearbeitung eines Seiteninhalts abgeschlossen hat. Für üblich wird dieser Handler in der View implementiert um das Panel mit dem bearbeiteten Seitenelement zu aktualisieren.

Method Summary

Methods

Modifier and Type	Method and Description
void	toolEnds (Content c, ToolEndsType type) Diese Methode wird vom Tool ausgeführt, falls dies mit der Bearbeitung fertig ist.

Method Detail

toolEnds

```
void toolEnds(Content c,  
              ToolEndsType type)
```

Diese Methode wird vom Tool ausgeführt, falls dies mit der Bearbeitung fertig ist.

Parameters:

c - der bearbeitete Seiteninhalt

type - gibt an, ob das Tool regulär beendet wurde oder den Fokus verloren hat

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

3.3. JAVADOC

Tree

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dataManagement

Class Tree

java.lang.Object
dataManagement.Tree

```
public class Tree
extends java.lang.Object
```

Diese Klasse dient dazu, die Information über den Verzeichnisbaum zu speichern. Die verwendete Datenstruktur ist ein Baum. Ein Tree speichert in einer ArrayList alle seiner Kindknoten. Jeder Kindknoten ist wiederum ein Objekt der Klasse Tree. Die höchste im Programm sinnvoll verwendbare Tiefe eines solchen Baumes ist 3. Im ersten Level liegen die Bücher, im zweiten liegen die Kapitel und im dritten die Seiten. Jeder Baum speichert seinen Titel, um später die Namen der einzelnen Elemente der Verzeichnisstruktur rekonstruieren zu können.

Constructor Summary

Constructors

Constructor and Description

<code>Tree()</code>

Der Baum bekommt den Defaulttitel "root" und die Liste wird initialisiert

<code>Tree(java.lang.String title)</code>

Der Baum wird mit einem vorgegebenen Titel erstellt.
--

Method Summary

Methods

Modifier and Type	Method and Description
void	addChildren (java.lang.String title) Erzeugt einen Kindbaum zu vorgegebenen Titel und nimmt diesen in die Liste aller Kindbäume auf
void	addChildren (Tree t) Nimmt den übergebenen Kindbaum in die Liste aller Kindbäume auf
java.util.ArrayList<Tree>	getChildren () Diese Methode gibt die Liste aller Kindbäume zurück, sodass darüber iteriert werden kann.
boolean	hasNoChildren () Diese Methode gibt zurück, ob dieser Baum Kindbäume gespeichert hat.

3.3. JAVADOC

Tree

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

java.lang.String	toString()
------------------	------------

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Tree

public Tree()

Der Baum bekommt den Defaulttitel "root" und die Liste wird initialisiert

Tree

public Tree(java.lang.String title)

Der Baum wird mit einem vorgegebenen Titel erstellt. Die Liste wird initialisiert

Parameters:

title - der Titel des Baums

Method Detail

addChildren

public void addChildren(java.lang.String title)

Erzeugt einen Kindbaum zu vorgegebenen Titel und nimmt diesen in die Liste aller Kindbäume auf

Parameters:

title - der Titel des Kindbaumes

3.3. JAVADOC

Tree

file:///home/felilein/git/TwoNote/TwoNote/doc/da...

addChildren

```
public void addChildren(Tree t)
```

Nimmt den übergebenen Kindbaum in die Liste aller Kindbäume auf

Parameters:

t - Der einzufügende Kindbaum

toString

```
public java.lang.String toString()
```

Overrides:

toString in class java.lang.Object

getChildren

```
public java.util.ArrayList<Tree> getChildren()
```

Diese Methode gibt die Liste aller Kindbäume zurück, sodass darüber iteriert werden kann.

Returns:

die Kindbäume

hasNoChildren

```
public boolean hasNoChildren()
```

Diese Methode gibt zurück, ob dieser Baum Kindbäume gespeichert hat.

Returns:

dieser Baum besitzt Kindbäume

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

[Summary: Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

3.4. Datenbankdokumentation

Als Datenbankmanagementsystem wurde MySQL verwendet. Die Schnittstelle zu Java lieferte jdbc. Es wurde eine relationale Datenbank verwendet. Die Datenbank besteht aus 4 Tabellen. Die Tabelle Books speichert die Bücher mittels id und Namen. Die Tabelle Chapter speichert die Kapitel mittels id, Namen und book. Der Fremdschlüssel book ist dabei die id des zugehörigen Buches. Die Tabelle Pages speichert Seiten mittels id, Namen, Chapter, Höhe und Breite. Der Fremdschlüssel chapter ist dabei die id des zugehörigen Buches. Die Tabelle Content speichert den Seiteninhalt. Dieser besteht aus einer id, einem serialisierten Objekt und der id der zugehörigen Seite(Fremdschlüssel).

3.4.1. ON DELETE CASCADE

Beim Anlegen der Tabellen wurde die Option ON DELETE CASCADE verwendet. Diese besagt das, dass ein Löschvorgang für ein übergeordnetes Verzeichnis, beispielsweise ein Buch, sofort das Löschen aller untergeordneten Verzeichnisse zur Folge hat. Am Beispiel eines Buches werden somit alle Kapitel, welche die id des Buches als Fremdschlüssel gespeichert haben gelöscht. Damit werden wiederum alle Seiten welche die id des Kapitels als Fremdschlüssel gespeichert haben gelöscht usw. Diese Option wird vollständig vom Datenbankmanagementsystem ausgeführt und muss im Quellcode nicht beachtet werden. Somit wird ein Großteil dieser sehr aufwendigen Aufgabe auf das Datenbankmanagementsystem übertragen, womit ein großer Laufzeitboost erzielt werden konnte.

4. Benutzerdokumentation

4.1. Installationsdokumentation

Damit TwoNote fehlerfrei funktioniert muss die Datenbank korrekt wie oben beschrieben eingerichtet werden. Es existiert dazu noch kein Installer. Des Weiteren muss der jdbc Treiber korrekt installiert sein, damit dieser von TwoNote geladen werden kann. Aufgrund von Java als Programmiersprache ist TwoNote sonst weitestgehend portabel und benötigt keiner Installation.

4.2. Benutzung

Die Benutzerschnittstelle ist weitestgehend intuitiv gehalten und relativ selbsterklärend. Es existiert eine Toolbar mittels welcher zwischen Textmodus und Cursermodus gewechselt werden kann. Im Cursermodus kann kein neuer Text erzeugt werden, jedoch kann per Klick auf eine Textnotiz in den Textmodus gewechselt werden. Befindet man sich im Textmodus, kann per Klick auf eine freie Fläche eine neue Textnotiz erstellt werden. Wird eine Textnotiz angelegt oder bearbeitet, so öffnet sich das passende Werkzeug. Der Text kann nun geändert, mittels Maus oder Pfeiltasten verschoben werden. Die Schriftgröße und Schriftfarbe sind dann ebenfalls auswählbar. Mittels der Taste Entfernen kann die Textnotiz gelöscht werden. Wenn der gesamte Text gelöscht wird, erfolgt ebenfalls ein Löschvorgang für die gesamte Textnotiz. Durch den Verzeichnisbaum kann bequem über ein Dropdown, welches das aktuelle Buch auswählbar macht, navigiert werden. Innerhalb dieses Buches kann dann im Verzeichnisbaum selber navigiert werden. Möchte man ein Verzeichnis löschen, so muss dies nur im Verzeichnisbaum ausgewählt werden und der Button löschen geklickt werden. Es existiert auch eine Startseite, welche dem Benutzer eine grobe Einführung in die Benutzung von TwoNote bietet.

Teil II.

Quellcode

5. dataManagement

5.1. IDataBase

```
package dataManagement;

import java.sql.SQLException;
import pageData.PageInformation;

public interface IDataBase {

    //create
    public void createBook(String bookName)
        throws SQLException;

    public void createChapter(String chapterName, int bookID)
        throws SQLException;

    public void createPage(PageInformation pageInfo, int chapterID)
        throws SQLException;

    public void createContent(byte[] contentBytes, int contentNumber,
        int pageID) throws SQLException;

    //delete
    public void deleteBook(int bookID)
        throws SQLException;

    public void deleteChapter(int chapterID)
        throws SQLException;

    public void deletePage(int pageID)
        throws SQLException;

    public void deleteContent(int contentID)
        throws SQLException;

    //save
```

```
public void saveContent(byte[] contentBytes, int contentID)
    throws SQLException;

public void savePage(PageInformation pageInfo, int chapterID,
    int pageID) throws SQLException;

//load
public byte[] loadContent(int contentID)
    throws SQLException;

public PageInformation loadPageInformation (int pageID)
    throws SQLException;

//exists
public boolean existsBook(String bookName)
    throws SQLException;

public boolean existsChapter(String chapterName, int bookID)
    throws SQLException;

public boolean existsPage(String pageName, int chapterID)
    throws SQLException;

public boolean existsContent(int contentNumber, int pageID)
    throws SQLException;

//getID
public int getBookID(String bookName)
    throws SQLException;

public int getChapterID(String chapterName, int bookID)
    throws SQLException;

public int getPageID(String pageName, int chapterID)
    throws SQLException;

public int getContentID(int contentNumber, int pageID)
    throws SQLException;

//getChildren
public String[] getBookChildren(int bookID)
    throws SQLException;

public String[] getChapterChildren(int chapterID)
    throws SQLException;
```

```
        public int [] getPageChildren(int pageID)
            throws SQLException;

        //getAllBooks
        public String [] getAllBooks() throws SQLException;
    }
```

5.2. Tree

```
package dataManagement;

import java.util.ArrayList;

public class Tree {

    private String title;
    private ArrayList<Tree> children;

    public Tree()
    {
        this.title = "root ";
        children = new ArrayList<>();
    }

    public Tree(String title)
    {
        this.title = title;
        children = new ArrayList<>();
    }

    public void addChildren(String title)
    {
        children.add(new Tree(title));
    }

    public void addChildren(Tree t)
    {
        children.add(t);
    }

    @Override
    public String toString()
```

```
        {
            return title;
        }

        public ArrayList<Tree> getChildren()
        {
            return children;
        }

        public boolean hasNoChildren()
        {
            return children.isEmpty();
        }
    }
}
```

5.3. Serializer

```
package dataManagement;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

import pageData.Content;

public class Serializer {

    //methods
    public byte[] serialize(Content cont) throws IOException
    {
        try(ByteArrayOutputStream bos = new ByteArrayOutputStream();
            ObjectOutput out = new ObjectOutputStream(bos)) {

            out.writeObject(cont);
            byte[] bytes = bos.toByteArray();
            return bytes;
        }
    }

    public Content deserialize(byte[] bytes)
```



```
        throws ClassNotFoundException, IOException
    {
        try (ByteArrayInputStream bis = new ByteArrayInputStream(bytes);
             ObjectInput in = new ObjectInputStream(bis)){

            Object o = in.readObject();
            return (Content)o;

        }
    }
}
```

5.4. MySQLDatabase

```
package dataManagement;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.xml.bind.DatatypeConverter;

import pageData.PageInformation;

public class MySQLDatabase implements IDatabase {

    private Connection con;

    //constructor
    public MySQLDatabase (String url, String name, String pw)
        throws ClassNotFoundException, SQLException
    {
        //load CDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //initialize connection
        con = DriverManager.getConnection(url, name, pw);
    }

    //create
    @Override
```

```
public void createBook(String bookName) throws SQLException {
    Statement stmt = null;
    try {
        stmt = con.createStatement();
        String query = "INSERT INTO 'Books' ('Name' "
            + "VALUES ('" + bookName + " '));";
        stmt.executeUpdate(query);
    }
    finally
    {
        if (stmt != null) stmt.close();
    }
}

@Override
public void createChapter(String chapterName, int bookID)
    throws SQLException {
    Statement stmt = null;
    try {
        stmt = con.createStatement();
        String query = "INSERT INTO 'Chapter' ('Name', 'Book' "
            + "VALUES ('" + chapterName + "', '" + bookID + " '));";
        stmt.executeUpdate(query);
    }
    finally
    {
        if (stmt != null) stmt.close();
    }
}

@Override
public void createPage(PageInformation pageInfo, int chapterID)
    throws SQLException {
    Statement stmt = null;
    String pageName = pageInfo.getPageName();
    int width = pageInfo.getWidth();
    int height = pageInfo.getHeight();
    try {
        stmt = con.createStatement();
        String query = "INSERT INTO 'Pages' "
            + " ('Name', 'Chapter', 'Width', 'Height') " +
            + "VALUES ('" + pageName + "', '" + chapterID + "', "
            + " '" + width + "', '" + height + " '));";
        stmt.executeUpdate(query);
    }
}
```

```
        finally
        {
            if (stmt!=null) stmt.close();
        }
    }

    @Override
    public void createContent(byte[] contentBytes, int contentNumber,
        int pageID) throws SQLException {
        Statement stmt =null;
        String byteString = DatatypeConverter.
            printBase64Binary(contentBytes);
        try {
            stmt = con.createStatement();
            String query="INSERT INTO 'Content '
                +('Objekt ', 'Page ', 'Number ')"
                +" VALUES ('"+byteString+"
                +","+pageID+","+contentNumber+"));";
            stmt.executeUpdate(query);
        }
        finally
        {
            if (stmt!=null) stmt.close();
        }
    }

    //delete
    @Override
    public void deleteBook(int bookID) throws SQLException {
        Statement stmt =null;
        try {
            stmt = con.createStatement();
            String query="DELETE FROM 'Books ' WHERE ID="+bookID+"";
            stmt.executeUpdate(query);
        }
        finally
        {
            if (stmt!=null) stmt.close();
        }
    }

    @Override
    public void deleteChapter(int chapterID) throws SQLException {
        Statement stmt =null;
        try {
```

```
        stmt = con.createStatement();
        String query="DELETE FROM 'Chapter '
                    +WHERE ID="+chapterID +"";
        stmt.executeUpdate(query);
    }
    finally
    {
        if(stmt!=null) stmt.close();
    }
}

@Override
public void deletePage(int pageID) throws SQLException {
    Statement stmt =null;
    try {
        stmt = con.createStatement();
        String query="DELETE FROM '
                    +Pages ' WHERE ID="+pageID +"";
        stmt.executeUpdate(query);
    }
    finally
    {
        if(stmt!=null) stmt.close();
    }
}

@Override
public void deleteContent(int contentID) throws SQLException {
    Statement stmt =null;
    try {
        stmt = con.createStatement();
        String query="DELETE FROM 'Content '
                    + WHERE ID="+contentID +"";
        stmt.executeUpdate(query);
    }
    finally
    {
        if(stmt!=null) stmt.close();
    }
}

//save
@Override
public void saveContent(byte[] contentBytes, int contentID)
    throws SQLException {
```

```
Statement stmt = null;
String byteString = DatatypeConverter.
    printBase64Binary(contentBytes);
try {
    stmt = con.createStatement();
    String query="UPDATE 'Content'
        +SET Objekt = '"+byteString+"
        + WHERE ID = '"+contentID+"";
    stmt.executeUpdate(query);
}
finally
{
    if(stmt!=null) stmt.close();
}
}

@Override
public void savePage(PageInformation pageInfo, int chapterID,
    int pageID) throws SQLException {
    Statement stmt = null;
    String pageName = pageInfo.getPageName();
    int width = pageInfo.getWidth();
    int height = pageInfo.getHeight();
    try {
        stmt = con.createStatement();
        String query="UPDATE 'Pages' SET Name =
            + '"+pageName+"', Chapter = '"+chapterID
            + " ', Height = '"+height+"', Width = "
            + width+" WHERE ID = '"+pageID+"";
        stmt.executeUpdate(query);
    }
    finally
    {
        if(stmt!=null) stmt.close();
    }
}

//load
@Override
public byte[] loadContent(int contentID) throws SQLException {
    Statement stmt = null;
    ResultSet rst = null;
    try {
        stmt = con.createStatement();
        String query = "SELECT * FROM
```

```
        + 'Content ' WHERE ID= "+contentID + "';";
        rst = stmt.executeQuery(query);
        rst.next();
        String byteString = rst.getString("Objekt");
        return DatatypeConverter.parseBase64Binary(byteString);
    }
    finally
    {
        if (rst != null) rst.close();
        if (stmt != null) stmt.close();
    }
}

@Override
public PageInformation loadPageInformation(int pageID)
    throws SQLException {
    Statement stmt = null;
    ResultSet rst = null;
    try {
        stmt = con.createStatement();
        String query = "SELECT * FROM 'Pages '
            + WHERE ID= "+pageID + "';";
        rst = stmt.executeQuery(query);
        //get attributes

        rst.next();
        int width = rst.getInt("Width");
        int height = rst.getInt("Height");
        String pageTitle = rst.getString("Name");

        //returns the PageInformation to pageTitle
        return new PageInformation
            (pageTitle, null, null, width, height);
    }
    finally
    {
        if (rst != null) rst.close();
        if (stmt != null) stmt.close();
    }
}

//exists
@Override
public boolean existsBook(String bookName) throws SQLException {
    Statement stmt = null;
```

```
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Books'
+ WHERE Name= '"+bookName+"';";
            rst = stmt.executeQuery(query);
            if(rst.next())
                return true;
            else
                return false;
        }
        finally
        {
            if(rst!=null) rst.close();
            if(stmt!=null) stmt.close();
        }
    }

    @Override
    public boolean existsChapter(String chapterName, int bookID)
        throws SQLException {
        Statement stmt =null;
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Chapter' WHERE
+Name= '"+chapterName+"',
+ AND Book= '"+bookID+"';";
            rst = stmt.executeQuery(query);
            if(rst.next())
                return true;
            else
                return false;
        }
        finally
        {
            if(rst!=null) rst.close();
            if(stmt!=null) stmt.close();
        }
    }

    @Override
    public boolean existsPage(String pageName, int chapterID)
        throws SQLException {
        Statement stmt =null;
```

```
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Pages' WHERE
                           +Name= '"+pageName+"',
                           + AND Chapter= '"+chapterID+"";
            rst = stmt.executeQuery(query);
            if(rst.next())
                return true;
            else
                return false;
        }
        finally
        {
            if(rst!=null) rst.close();
            if(stmt!=null) stmt.close();
        }
    }

    @Override
    public boolean existsContent(int contentNumber, int pageID)
        throws SQLException {
        Statement stmt = null;
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Content'
                           +WHERE Number= '"+contentNumber+"',
                           +AND Page= '"+pageID+"";
            rst = stmt.executeQuery(query);
            if(rst.next())
                return true;
            else
                return false;
        }
        finally
        {
            if(rst!=null) rst.close();
            if(stmt!=null) stmt.close();
        }
    }

    //get ID
    @Override
    public int getBookID(String bookName) throws SQLException {
```



```
Statement stmt =null;
ResultSet rst = null;
try {
    stmt = con.createStatement();
    String query = "SELECT * FROM 'Books '
+ WHERE Name= '"+bookName+"'";
    rst = stmt.executeQuery(query);
    rst.next();
    int bookID=rst.getInt(1);
    return bookID;
}
finally
{
    if(rst!=null) rst.close();
    if(stmt!=null) stmt.close();
}
}

@Override
public int getChapterID(String chapterName, int bookID)
throws SQLException {
    Statement stmt =null;
    ResultSet rst = null;
    try {
        stmt = con.createStatement();
        String query = "SELECT * FROM 'Chapter '
+WHERE Book= '"+bookID+"'
+AND Name= '"+chapterName+"'";
        rst = stmt.executeQuery(query);
        rst.next();
        int chapterID=rst.getInt(1);
        return chapterID;
    }
    finally
    {
        if(rst!=null) rst.close();
        if(stmt!=null) stmt.close();
    }
}

@Override
public int getPageID(String pageName, int chapterID)
throws SQLException {
    Statement stmt =null;
    ResultSet rst = null;
```

```
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Pages '
+ WHERE Chapter= '" + chapterID + "'
+AND Name= '" + pageName + "'";
            rst = stmt.executeQuery(query);
            rst.next();
            int pageID = rst.getInt(1);
            return pageID;
        }
        finally
        {
            if (rst != null) rst.close();
            if (stmt != null) stmt.close();
        }
    }

    @Override
    public int getContentID(int ContentNumber, int pageID)
        throws SQLException {
        Statement stmt = null;
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM 'Content '
+WHERE Page= '" + pageID + "'
+AND Number= '" + ContentNumber + "'";
            rst = stmt.executeQuery(query);
            rst.next();
            int contentID = rst.getInt("ID");
            return contentID;
        }
        finally
        {
            if (rst != null) rst.close();
            if (stmt != null) stmt.close();
        }
    }

    //get Children
    @Override
    public String[] getBookChildren(int bookID)
        throws SQLException {
        Statement stmt = null;
        ResultSet rst = null;
```

```
        try {
            stmt = con.createStatement();
            String query = "SELECT * FROM
                            + 'Chapter ' WHERE Book= "+bookID + " ";
            rst = stmt.executeQuery(query);
            rst.last();
            int childrenNumber = rst.getRow();
            String [] children = new String [childrenNumber];
            rst.beforeFirst();
            for (int i = 0; i < children.length; i++) {
                rst.next();
                children[i] = rst.getString("Name");
            }
            return children;
        }
        finally
        {
            if (rst != null) rst.close();
            if (stmt != null) stmt.close();
        }
    }

    @Override
    public String[] getChapterChildren(int chapterID)
        throws SQLException {
        Statement stmt = null;
        ResultSet rst = null;
        try {
            stmt = con.createStatement();
            String query = "SELECT *
                            + FROM 'Pages ' WHERE
                            + Chapter= "+chapterID + " ";
            rst = stmt.executeQuery(query);
            rst.last();
            int childrenNumber = rst.getRow();
            String [] children = new String [childrenNumber];
            rst.beforeFirst();
            for (int i = 0; i < children.length; i++) {
                rst.next();
                children[i] = rst.getString("Name");
            }
            return children;
        }
        finally
        {

```

```
        if (rst!=null) rst.close();
        if (stmt!=null) stmt.close();
    }
}

@Override
public int[] getPageChildren(int pageID) throws SQLException {
    Statement stmt = null;
    ResultSet rst = null;
    try {
        stmt = con.createStatement();
        String query = "SELECT * FROM
            + 'Content ' WHERE Page= "+pageID +"";
        rst = stmt.executeQuery(query);
        rst.last();
        int childrenNumber = rst.getRow();
        int [] children = new int [childrenNumber];
        rst.beforeFirst();
        for (int i = 0; i < children.length; i++) {
            rst.next();
            children[i] = rst.getInt("ID");
        }
        return children;
    }
    finally
    {
        if (rst!=null) rst.close();
        if (stmt!=null) stmt.close();
    }
}

/**
 * {@inheritDoc}
 */
@Override
public String[] getAllBooks() throws SQLException {
    Statement stmt = null;
    ResultSet rst = null;
    try {
        stmt = con.createStatement();
        String query = "SELECT * FROM 'Books '";
        rst = stmt.executeQuery(query);
        rst.last();
        int bookNumber = rst.getRow();
        String [] books = new String [bookNumber];
    }
}
```

```
        rst.beforeFirst();
        for (int i = 0; i < books.length; i++) {
            rst.next();
            books[i] = rst.getString("Name");
        }
        return books;
    }
    finally
    {
        if (rst != null) rst.close();
        if (stmt != null) stmt.close();
    }
}
}
```

5.5. Model

```
package dataManagement;

import java.io.IOException;
import java.sql.SQLException;

import pageData.ActionType;
import pageData.Content;
import pageData.ContentInstruction;
import pageData.DirectoryAlreadyExistsException;
import pageData.DirectoryDoesNotExistsException;
import pageData.DirectoryType;
import pageData.Page;
import pageData.PageInformation;

public class Model {
    private Serializer serializer;
    private IDatabase db;

    //constructor
    public Model() throws ClassNotFoundException, SQLException
    {
        serializer = new Serializer();
        db = new MySQLDatabase
            ("jdbc:mysql://localhost/TwoNote", "TwoNote", "1dNfdTF");
    }
}
```

```
}

//methods

//create methods
public void createBook(String bookName)
    throws DirectoryAlreadyExistsException, SQLException
{
    //check existence
    if (db.existsBook(bookName))
        throw new DirectoryAlreadyExistsException
            (DirectoryType.Book, ActionType.create, bookName);

    db.createBook(bookName);
}

public void createChapter(String chapterName, String bookName)
    throws DirectoryAlreadyExistsException,
    SQLException, DirectoryDoesNotExistsException
{
    int bookID = this.getBookID(ActionType.create, bookName);
    if (db.existsChapter(chapterName, bookID))
        throw new DirectoryAlreadyExistsException
            (DirectoryType.Chapter, ActionType.create, chapterName);

    db.createChapter(chapterName, bookID);
}

public Page createPage
    (String pageName, String chapterName, String bookName)
    throws DirectoryAlreadyExistsException,
    SQLException, DirectoryDoesNotExistsException
{
    int chapterID = this.getChapterID
        (ActionType.create, chapterName, bookName);
    if (db.existsPage(pageName, chapterID))
        throw new DirectoryAlreadyExistsException
            (DirectoryType.Page, ActionType.create, pageName);

    //create empty default Page
    int width = Page.DEFAULT_WIDTH;
    int height = Page.DEFAULT_HEIGHT;
    PageInformation pageInfo = new PageInformation
        (pageName, chapterName, bookName, width, height);
}
```

```
        //save page in DB
        db.createPage(pageInfo , chapterID);

        return new Page(pageInfo);
    }

    private void createContent(Content cont, int pageID)
        throws SQLException, IOException,
        DirectoryAlreadyExistsException
    {
        byte[] contentBytes = serializer.serialize(cont);
        int contentNumber = cont.getNumber();
        if(db.existsContent(contentNumber, pageID))
            throw new DirectoryAlreadyExistsException
                (DirectoryType.Content, ActionType.create, contentNumber);
        db.createContent(contentBytes, cont.getNumber(), pageID);
    }

    //delete methods
    public void deleteBook(String bookName)
        throws SQLException, DirectoryDoesNotExistsException
    {
        int bookID = getBookID(ActionType.delete, bookName);

        //delete the book
        db.deleteBook(bookID);
    }

    public void deleteChapter(String chapterName, String bookName)
        throws SQLException, DirectoryDoesNotExistsException
    {
        int chapterID = this.getChapterID
            (ActionType.delete, chapterName, bookName);

        //delete the chapter
        db.deleteChapter(chapterID);
    }

    public void deletePage
        (String pageName, String chapterName, String bookName)
        throws SQLException, DirectoryDoesNotExistsException
    {
        int pageID = this.getPageID
            (ActionType.delete, pageName, chapterName, bookName);
    }
}
```

```
        //delete the page
        db.deletePage(pageID);
    }

    private void deleteContent(int contentNumber, int pageID)
        throws SQLException
    {
        if (!db.existsContent(contentNumber, pageID))
            throw new DirectoryDoesNotExistsException
                (DirectoryType.Content, ActionType.delete, contentNumber);
        int contentID = db.getContentID(contentNumber, pageID);
        db.deleteContent(contentID);
    }

    // save methods
    public void savePage(Page page)
        throws DirectoryDoesNotExistsException,
            SQLException, IOException
    {
        PageInformation pageInfo = page.getPageInfo();
        String bookName = pageInfo.getBookName();
        String chapterName = pageInfo.getChapterName();
        String pageName = pageInfo.getPageName();

        int pageID = this.getPageID
            (ActionType.save, pageName, chapterName, bookName);
        int chapterID = this.getChapterID
            (ActionType.save, chapterName, bookName);
        int instruSize = page.getContentInstructionSize();

        for (int i = 0; i < instruSize; i++) {
            ContentInstruction instru =
                page.getNextContentInstruction();
            int contentNumber = instru.getContentNumber();
            switch (instru.getActionType()) {
                case create:
                    createContent(page.getContent(contentNumber),
                                pageID);
                    break;
                case save:
                    saveContent(page.getContent(contentNumber),
                               pageID);
                    break;
                case delete:
                    deleteContent(contentNumber, pageID);
            }
        }
    }
}
```



```
                break;
            default:
                throw new RuntimeException
                    ("Ungueltiger ActionType");
            }
            page.contentInstructionCompleted();
        }

        db.savePage(pageInfo, chapterID, pageID);
    }

    private void saveContent(Content cont, int pageID)
        throws SQLException, IOException {

        byte[] contentBytes = serializer.serialize(cont);
        int contentNumber = cont.getNumber();
        int contentID = db.getContentID(contentNumber, pageID);
        if (db.existsContent(contentNumber, pageID))
            db.saveContent(contentBytes, contentID);
    }

    // load methods
    public Page loadPage(String pageName, String chapterName,
        String bookName)
        throws SQLException,
        DirectoryDoesNotExistsException,
        ClassNotFoundException, IOException
    {
        int pageID = getPageID
            (ActionType.load, pageName, chapterName, bookName);

        PageInformation pageInfo = db.loadPageInformation(pageID);
        pageInfo.setBookName(bookName);
        pageInfo.setChapterName(chapterName);

        Page page = new Page(pageInfo);

        int[] pageChildren = db.getPageChildren(pageID);
        for (int i = 0; i < pageChildren.length; i++) {
            page.addContent(loadContent(pageChildren[i]));
        }
        return page;
    }
}
```

```
private Content loadContent(int contentID)
    throws ClassNotFoundException,
        IOException, SQLException {
    byte[] contentBytes = db.loadContent(contentID);
    Content cont = serializer.deserialize(contentBytes);
    return cont;
}

//get ID methods
private int getBookID(ActionType type, String bookName)
    throws SQLException, DirectoryDoesNotExistsException
{
    if (!db.existsBook(bookName))
        throw new DirectoryDoesNotExistsException
            (DirectoryType.Book, type, bookName);
    int bookID = db.getBookID(bookName);
    return bookID;
}

private int getChapterID(ActionType type, String chapterName ,
    String bookName)
    throws SQLException, DirectoryDoesNotExistsException
{
    int bookID = this.getBookID(type, bookName);
    if (!db.existsChapter(chapterName, bookID))
        throw new DirectoryDoesNotExistsException
            (DirectoryType.Chapter, type, chapterName);
    int chapterID = db.getChapterID(chapterName, bookID);
    return chapterID;
}

private int getPageID
    (ActionType type, String pageName, String chapterName ,
    String bookName)
    throws SQLException, DirectoryDoesNotExistsException
{
    int chapterID = this.getChapterID(type, chapterName, bookName);
    if (!db.existsPage(pageName, chapterID))
        throw new DirectoryDoesNotExistsException
            (DirectoryType.Page, type, pageName);
    int pageID = db.getPageID(pageName, chapterID);
    return pageID;
}
```

```
//get Tree
public Tree getTree() throws SQLException
{
    String[] books = db.getAllBooks();
    Tree t = new Tree();
    for (int i = 0; i < books.length; i++) {
        Tree chapterTree = new Tree(books[i]);
        String[] chapters = db.getBookChildren
            (getBookID(ActionType.load, books[i]));
        for (int j = 0; j < chapters.length; j++) {
            Tree pageTree = new Tree(chapters[j]);
            String[] pages = db.getChapterChildren
                (getChapterID(ActionType.load,
                    chapters[j], books[i]));
            for (int k = 0; k < pages.length; k++) {
                pageTree.addChildren(pages[k]);
            }
            chapterTree.addChildren(pageTree);
        }
        t.addChildren(chapterTree);
    }
    return t;
}
}
```

6. dataProcessing

6.1. Controller

```
package dataProcessing;

import java.io.IOException;
import java.sql.SQLException;

import dataManagement.Model;
import dataManagement.Tree;
import pageData.Content;
import pageData.ContentType;
import pageData.DirectoryDoesNotExistsException;
import pageData.Page;
import pageData.PageInformation;
import pageData.Startpage;
import pageData.TextBox;

public class Controller {

    private Page currentPage;
    private Model model;

    private Tree dataTree;

    //constructor
    public Controller(Model model) throws SQLException
    {
        this.model = model;
        refreshTree();
    }

    //load Page
    public void loadPage
        (String pageName, String chapterName, String bookName)
        throws DirectoryDoesNotExistsException,
            SQLException, IOException, ClassNotFoundException
```

```
{
    if (pageName == "Startseite")
        currentPage = new Startpage();
    else
        currentPage = model.loadPage
            (pageName, chapterName, bookName);
}

//getContent : content []
public Content[] getContent() throws NullPointerException
{
    if (currentPage == null)
        throw new NullPointerException
            ("Es wurde keine Seite geoeffnet");
    return currentPage.getContent();
}

//createContent
public Content createContent(ContentType type, int x, int y)
{
    Content content = null;
    //create
    switch (type) {
    case textBox:
        content = new TextBox(x, y);
        break;

    default:
        new RuntimeException
            ("Gewahlter content ist nicht verfuegbar");
    }
    //create content on page
    currentPage.createContent(content);
    return content;
}

public void saveContent(Content c)
    throws DirectoryDoesNotExistsException,
        SQLException, IOException
{
    if (currentPage.getPageInfo().getPageName().equals("Startseite"))
        return;
    currentPage.saveContent(c);
    model.savePage(currentPage);
}
```

```
        public void deleteContent(Content c)
            throws DirectoryDoesNotExistsException,
                SQLException, IOException
        {
            if (currentPage.getPageInfo().getPageName().equals("Startseite"))
                return;
            currentPage.deleteContent(c.getNumber());
            model.savePage(currentPage);
        }

        public void refreshTree() throws SQLException
        {
            dataTree = model.getTree();
        }

        public Tree getDataTree()
        {
            return dataTree;
        }

        public PageInformation getCurrentPageInformation()
        {
            return currentPage.getPageInfo();
        }
    }
}
```

7. GUI

7.1. GUI

Die GUI wurde von NetBeans generiert.

8. pageData

8.1. ActionType

```
package pageData;

public enum ActionType {
    delete,
    create,
    load,
    save,
}
```

8.2. Content

```
package pageData;

import java.awt.Component;
import java.io.Serializable;
import javax.swing.JPanel;

public abstract class Content implements Serializable{

    private static final long serialVersionUID = 1L;
    private int number;
    protected int x;
    protected int y;
    protected int width;
    protected int height;

    private ContentType contentType;

    //constructor
    public Content
        (int x, int y, int width, int height, ContentType contentType)
    {
        this.x = x;
        this.y = y;
    }
}
```



```
        this.width = width;
        this.height = height;
        this.contentType = contentType;
    }

    //getter and setter
    public int getNumber() {
        return number;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public int getWidth() {
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public void setNumber(int number) {
        this.number = number;
    }
```

```
    }

    public ContentType getContentType() {
        return contentType;
    }

    public void setContentType(ContentType contentType) {
        this.contentType = contentType;
    }

    //Methods
    public abstract Component draw(JPanel panel);
}
```

8.3. ContentInstruction

```
package pageData;

public class ContentInstruction {

    private int contentNumber;
    private ActionType actionType;

    //Constructor
    public ContentInstruction(int contentNumber, ActionType actionType)
    {
        this.contentNumber = contentNumber;
        this.actionType = actionType;
    }

    //getter and setter
    public int getContentNumber() {
        return contentNumber;
    }

    public void setContentNumber(int contentNumber) {
        this.contentNumber = contentNumber;
    }

    public ActionType getActionType() {
        return actionType;
    }

    public void setActionType(ActionType actionType) {
        this.actionType = actionType;
    }
}
```

```
    }  
}
```

8.4. ContentType

```
package pageData;  
  
public enum ContentType {  
    textBox  
}
```

8.5. DirectoryAlreadyExistsException

```
package pageData;  
  
public class DirectoryAlreadyExistsException extends RuntimeException {  
    private static final long serialVersionUID = 1L;  
  
    private DirectoryType dirType;  
    private ActionType actionType;  
    private String directoryTitle;  
  
    // constructors  
    public DirectoryAlreadyExistsException (DirectoryType dirType,  
                                           ActionType actionType, String directoryTitle) {  
        super("Es existiert bereits ein Verzeichnis mit diesem Namen");  
        this.dirType = dirType;  
        this.actionType = actionType;  
        this.directoryTitle = directoryTitle;  
    }  
  
    public DirectoryAlreadyExistsException (DirectoryType dirType,  
                                           , ActionType actionType, String directoryTitle,  
                                           String msg) {  
        super(msg);  
        this.dirType = dirType;  
        this.actionType = actionType;  
        this.directoryTitle = directoryTitle;  
    }  
  
    // getter and setter  
    public DirectoryType getDirType() {  
        return dirType;  
    }  
}
```

```
    }

    public void setDirType(DirectoryType dirType) {
        this.dirType = dirType;
    }

    public String getDirectoryTitle() {
        return directoryTitle;
    }

    public void setDirectoryTitle(String directoryTitle) {
        this.directoryTitle = directoryTitle;
    }

    public ActionType getActionType() {
        return actionType;
    }

    public void setActionType(ActionType actionType) {
        this.actionType = actionType;
    }
}
```

8.6. DirectoryDoesNotExistsException

```
package pageData;

public class DirectoryDoesNotExistsException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    private DirectoryType dirType;
    private ActionType actionType;
    private String directoryTitle;

    //constructors
    public DirectoryDoesNotExistsException
        (DirectoryType dirType, ActionType actionType,
         String directoryTitle) {
        super("Das angeforderte Verzeichnis existiert nicht");
        this.dirType = dirType;
        this.actionType = actionType;
        this.directoryTitle = directoryTitle;
    }
}
```

```
    public DirectoryDoesNotExistsException
        (DirectoryType dirType, ActionType actionType,
         String directoryTitle, String msg) {
        super(msg);
        this.dirType = dirType;
        this.actionType = actionType;
        this.directoryTitle = directoryTitle;
    }

    //getter and setter
    public DirectoryType getDirType() {
        return dirType;
    }

    public void setDirType(DirectoryType dirType) {
        this.dirType = dirType;
    }

    public String getDirectoryTitle() {
        return directoryTitle;
    }

    public void setDirectoryTitle(String directoryTitle) {
        this.directoryTitle = directoryTitle;
    }

    public ActionType getActionType() {
        return actionType;
    }

    public void setActionType(ActionType actionType) {
        this.actionType = actionType;
    }
}
```

8.7. DirectoryType

```
package pageData;

public enum DirectoryType {
    Book,
    Chapter,
    Page,
```

```
        Content
    }
}
```

8.8. Page

```
package pageData;

import java.util.ArrayList;
import java.util.HashMap;

public class Page {

    private PageInformation pageInfo;
    private ArrayList<ContentInstruction> contentInstructions;
    private HashMap<Integer, Content> pageContent;
    private int nextContentNumber;

    public static final int DEFAULT_WIDTH= 800;
    public static final int DEFAULT_HEIGHT= 600;

    //constructor
    public Page(PageInformation pageInfo)
    {
        this.pageInfo = pageInfo;
        this.contentInstructions = new ArrayList<ContentInstruction>();
        this.pageContent = new HashMap<Integer, Content>();
        nextContentNumber = 0;
    }

    //getter and setter
    public PageInformation getPageInfo() {
        return pageInfo;
    }

    public void setPageInfo(PageInformation pageInfo) {
        this.pageInfo = pageInfo;
    }

    public Content[] getContent()
    {
        Content [] back = new Content[pageContent.size()];
        pageContent.values().toArray(back);
        return back;
    }
}
```

```
public Content getContent(int number)
throws DirectoryDoesNotExistsException
{
    if (!pageContent.containsKey(number))
        throw new DirectoryDoesNotExistsException
            (DirectoryType.Content, ActionType.load, number+"");
    return pageContent.get(number);
}

public ContentInstruction getNextContentInstruction()
{
    if (contentInstructions.size()==0)
        throw new RuntimeException
            ("Es wurden bereits alle ContentInstructions bearbeitet");
    return contentInstructions.get(0);
}

public int getContentInstructionSize()
{
    return contentInstructions.size();
}

//methods
public Content createContent(Content cont)
{
    //interface for the controller
    cont.setNumber(nextContentNumber);
    pageContent.put(nextContentNumber, cont);
    contentInstructions.add
        (new ContentInstruction(nextContentNumber, ActionType.create));
    nextContentNumber++;
    return cont;
}

public void deleteContent(int number)
{
    if (!pageContent.containsKey(number))
        throw new DirectoryDoesNotExistsException
            (DirectoryType.Content, ActionType.delete, number+"");
    for (ContentInstruction instru : contentInstructions)
    {
        //no need to create and delete
        if (instru.getContentNumber() == number
            && instru.getActionType() == ActionType.create)
```

```
        {
            contentInstructions.remove(instru);
            pageContent.remove(number);
            return;
        }
        pageContent.remove(number);
        contentInstructions.add
            (new ContentInstruction(number, ActionType.delete));
    }

    public void saveContent(Content cont)
    {
        int number = cont.getNumber();
        contentInstructions.add
            (new ContentInstruction(number, ActionType.save));
    }

    public void contentInstructionCompleted()
    {
        if (contentInstructions.size()==0)
            throw new RuntimeException
                ("Es wurden bereits alle ContentInstructions abgeschlossen");
        contentInstructions.remove(0);
    }

    public void addContent(Content cont)
    {
        //interface for the model
        int number = cont.getNumber();
        pageContent.put(number, cont);
        if (nextContentNumber<=number)
            nextContentNumber=number+1;
    }
}
```

8.9. PageInformation

```
package pageData;

public class PageInformation {

    private String pageName;
    private String chapterName;
```



```
private String bookName;
private int width;
private int height;

// constructor
public PageInformation
    (String pageName, String chapterName,
     String bookName, int width, int height)
{
    this.pageName = pageName;
    this.chapterName = chapterName;
    this.bookName = bookName;
    this.height = height;
    this.width = width;
}

// getter and setter
public String getPageName() {
    return pageName;
}

public void setPageName(String pageName) {
    this.pageName = pageName;
}

public String getChapterName() {
    return chapterName;
}

public void setChapterName(String chapterName) {
    this.chapterName = chapterName;
}

public String getBookName() {
    return bookName;
}

public void setBookName(String bookName) {
    this.bookName = bookName;
}

public int getHeight() {
    return height;
}
```

```
        public void setHeight(int height) {
            this.height = height;
        }

        public int getWidth() {
            return width;
        }

        public void setWidth(int width) {
            this.width = width;
        }
    }
```

8.10. TextBox

```
package pageData;

import java.awt.Color;
import java.awt.Component;
import java.awt.Font;
import java.io.Serializable;

import javax.swing.JLabel;
import javax.swing.JPanel;

public class TextBox extends Content implements Serializable {

    private static final long serialVersionUID = 1L;

    //default values
    private static final int DEFAULT_WIDTH = 200;
    private static final int DEFAULT_HEIGHT = 30;
    private static final int DEFAULT_FONT_SIZE = 15;
    private static final Color DEFAULT_FONT_COLOR = Color.black;
    private static final Color DEFAULT_BACKGROUND_COLOR = Color.black;
    private static final String DEFAULT_TEXT = "";

    private String text;
    private int fontSize;
    private Color fontColor;
    private Color backgroundColor;
```

```
//constructor
public TextBox(int x, int y) {
    super(x, y, DEFAULT_WIDTH, DEFAULT_HEIGHT, ContentType.textBox);
    initDefault();
}

//copy-constructor
public TextBox(TextBox t)
{
    super(t.x, t.y, t.width, t.height, ContentType.textBox);
    this.backgroundColor = t.backgroundColor;
    this.text = t.text;
    this.fontColor = t.fontColor;
    this.backgroundColor = t.backgroundColor;
}

//getter and setter
public String getText() {
    return text;
}

public void setText(String text) {
    this.text = text;
}

public int getFontSize() {
    return fontSize;
}

public void setFontSize(int fontSize) {
    this.fontSize = fontSize;
}

public Color getFontColor() {
    return fontColor;
}

public void setFontColor(Color fontColor) {
    this.fontColor = fontColor;
}

public Color getBackgroundColor() {
    return backgroundColor;
}
```

```
    public void setBackgroundColor(Color backgroundColor) {
        this.backgroundColor = backgroundColor;
    }

    //methods
    private void initDefault ()
    {
        this.text = DEFAULT_TEXT;
        this.backgroundColor = DEFAULT_BACKGROUNDCOLOR;
        this.fontColor = DEFAULT_FONTCOLOR;
        this.fontSize = DEFAULT_FONTSIZE;
        this.width = DEFAULT_WIDTH;
        this.height = DEFAULT_HEIGHT;
    }

    @Override
    public String toString()
    {
        return text;
    }

    @Override
    public Component draw(JPanel panel) {
        if (text.length() == 0)
            return null;

        //create Label
        JLabel label = new JLabel(text);
        label.setForeground(fontColor);
        label.setBackground(backgroundColor);
        label.setFont(new Font("Ubuntu", 0, fontSize));

        //add label to panel
        panel.add(label);
        label.setBounds(x,y,width, height);
        return label;
    }
}
```

8.11. Startpage

```
package pageData;

import java.awt.Color;
```

```
public class Startpage extends Page{

    public Startpage() {
        super(new PageInformation("Startseite", "", "", 800, 600));
        TextBox welcomeText = new TextBox(20, 20);
        welcomeText.setFontSize(50);
        welcomeText.height = 60;
        welcomeText.width = 600;
        welcomeText.setText("Willkommen bei TwoNote");
        super.createContent(welcomeText);

        TextBox toolbox = new TextBox(20, 100);
        toolbox.setText("Um eine Textnotiz anzulegen ,  
+ waehlen Sie in der Toolbox den Textmodus aus  
+ und klicken Sie auf die Seite.");
        toolbox.width = 1000;
        toolbox.setFontSize(20);
        super.createContent(toolbox);

        TextBox editText = new TextBox(20, 150);
        editText.setText("Um eine Textnotiz zu bearbeiten ,  
+ klicken Sie auf die betreffende Notiz.");
        editText.width = 1000;
        editText.setFontSize(20);
        super.createContent(editText);

        TextBox hierarchie = new TextBox(20, 200);
        hierarchie.setText("Legen Sie ein Buch, innerhalb  
+ eines Buches ein Kapitel und innerhalb eines Kapitels  
+ eine neue Seite an.");
        hierarchie.width = 1000;
        hierarchie.setFontSize(20);
        super.createContent(hierarchie);

        TextBox delete = new TextBox(20, 250);
        delete.setText("Um eine Seite zu loeschen , markieren  
+ Sie diese im Baum und waehlen loeschen.");
        delete.width = 1000;
        delete.setFontSize(20);
        super.createContent(delete);

        TextBox color = new TextBox(20, 300);
        color.setText("Um die Farbe eines Textes zu aendern ,
```

```
+ waehlen Sie im Textwerkzeug \"Schriftfarbe festlegen \".");
color.setFontColor(Color.RED);
color.width = 1000;
color.setFontSize(20);
super.createContent(color);

TextBox tree = new TextBox(550, 400);
tree.setText("Klicken Sie hier auf eine Seite um
+ diese zu oeffnen. -->");
tree.width = 1000;
tree.setFontSize(20);
super.createContent(tree);

TextBox haveFun = new TextBox(20, 550);
haveFun.setText("Viel Spass mit TwoNote!");
haveFun.width = 1000;
haveFun.setFontSize(20);
super.createContent(haveFun);

    }

}
```

9. Tools

9.1. TextEditTool

```
package Tools;

import java.awt.Component;
import java.awt.Font;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;

import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import pageData.Content;
import pageData.TextBox;

public class TextEditTool implements Tool{

    //swing
    private JPanel panel;
    private JTextField textField;
    private JComboBox<Integer> fontSize;
    private JLabel fontSizeLabel;
    private JButton buttonChooseFontColor;
    private JColorChooser colorChooser;

    //editor
```

```
private TextBox content;
private ToolReadyHandler handler;
private boolean isClosed;

// create
public TextEditTool(JPanel panel, TextBox content)
{
    this.panel = panel;
    this.content = content;
    this.textField = new JTextField();
    this.isClosed = false;
}

@Override
public void drawTool() {
    // color chooser
    colorChooser = new JColorChooser();
    panel.add(colorChooser);
    colorChooser.setBounds(20, 30, 700, 400);
    colorChooser.setVisible(false);

    // font size
    fontSizeLabel = new JLabel("Schriftgroesse: ");
    panel.add(fontSizeLabel);
    fontSizeLabel.setBounds(20, 0, 100, 30);

    fontSize = new JComboBox<>();
    for (int i = 0; i < 100; i++) {
        fontSize.addItem(i);
    }
    fontSize.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
            textField.setFont(new Font("Ubuntu", 0,
                (int)fontSize.getSelectedItem()));
            content.setFontSize
                ((int)fontSize.getSelectedItem());
        }
    });

    fontSize.setSelectedItem(content.getFontSize());
    panel.add(fontSize);
    fontSize.setBounds(120, 0, 100, 30);
```



```
//textField

//init listeners
textField.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        endEdit(ToolEndsType.ExitTool);
    }
});

textField.addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent evt)
    {
        if (evt.getKeyCode() == KeyEvent.VK_RIGHT )
        {
            content.setX(content.getX() + 2);
            Rectangle rect =
                textField.getBounds();
            rect.setBounds
                (rect.x + 2, rect.y,
                 rect.width, rect.height);
            textField.setBounds(rect);
        }
        if (evt.getKeyCode() == KeyEvent.VK_LEFT)
        {
            content.setX(content.getX() - 2);
            Rectangle rect =
                textField.getBounds();
            rect.setBounds
                (rect.x - 2, rect.y,
                 rect.width, rect.height);
            textField.setBounds(rect);
        }
        if (evt.getKeyCode() == KeyEvent.VK_UP)
        {
            content.setY(content.getY() - 2);
            Rectangle rect =
                textField.getBounds();
            rect.setBounds
                (rect.x, rect.y - 2,
                 rect.width, rect.height);
            textField.setBounds(rect);
        }
    }
});
```

```
        if (evt.getKeyCode() == KeyEvent.VK_DOWN)
        {
            content.setY(content.getY() + 2);
            Rectangle rect =
                textField.getBounds();
            rect.setBounds
                (rect.x, rect.y + 2,
                 rect.width, rect.height);
            textField.setBounds(rect);
        }

        if (evt.getKeyCode() == KeyEvent.VK_DELETE)
        {
            textField.setText("");
            endEdit (ToolEndsType.ExitTool);
        }
    }

});

textField.addFocusListener(new FocusAdapter() {
    @Override
    public void focusLost (FocusEvent evt)
    {
        Component c = evt.getOppositeComponent();
        if (c!=fontSize && c!=buttonChooseFontColor
            && c!=colorChooser && c!=textField)
            endEdit (ToolEndsType.FocusLost);
    }
});

textField.addMouseMotionListener(new MouseMotionListener() {
    @Override
    public void mouseMoved (MouseEvent arg0) {
    }

    @Override
    public void mouseDragged (MouseEvent arg0) {
        content.setX(arg0.getXOnScreen() - 60);
        content.setY(arg0.getYOnScreen() - 180);
        Rectangle rect = textField.getBounds();
        rect.setBounds(arg0.getXOnScreen() -
                       60, arg0.getYOnScreen() - 180,
                       rect.width, rect.height);
        textField.setBounds(rect);
    }
});
```

```
});
//end init listeners

textField.setText(content.getText());
textField.setBounds
    (content.getX(), content.getY(),
     content.getWidth(), content.getHeight());
textField.setFont
    (new Font("Ubuntu", 0, (int)fontSize.getSelectedItem()));
textField.setForeground(content.getFontColor());
panel.add(textField);
textField.requestFocusInWindow();

//color chooser
buttonChooseFontColor = new JButton("Schriftfarbe festlegen");
buttonChooseFontColor.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent arg0) {
        if (colorChooser.isVisible())
        {
            colorChooser.setVisible(false);
            content.setFontColor
                (colorChooser.getColor());
            textField.setForeground
                (colorChooser.getColor());
        }
        else
        {
            colorChooser.setVisible(true);
        }
    }
});
panel.add(buttonChooseFontColor);
buttonChooseFontColor.setBounds(220, 0, 200, 30);
}

@Override
public void endEdit(ToolEndsType type) {
    if (isClosed)
        return;
    else
        isClosed = true;

    //remove the tool
    textField.setVisible(false);
}
```

```
        fontSizeLabel.setVisible(false);
        fontSize.setVisible(false);
        colorChooser.setVisible(false);
        buttonChooseFontColor.setVisible(false);
        panel.remove(textField);
        panel.remove(fontSizeLabel);
        panel.remove(fontSize);
        panel.remove(buttonChooseFontColor);
        panel.remove(colorChooser);

        //change the content
        content.setText(textField.getText());

        //call the handler
        if(handler != null)
            handler.toolEnds(content, type);
    }

    @Override
    public Content getContent() {
        return content;
    }

    @Override
    public void addToolReadyHandler(ToolReadyHandler handler) {
        this.handler = handler;
    }
}
```

9.2. Tool

```
package Tools;

import pageData.Content;

public interface Tool {

    void drawTool();

    void endEdit(ToolEndsType type);

    Content getContent();

    void addToolReadyHandler(ToolReadyHandler handler);
}
```

```
}
```

9.3. ToolEndsType

```
package Tools;

public enum ToolEndsType {
    FocusLost, ExitTool, Draw
}
```

9.4. ToolReadyHandler

```
package Tools;

import pageData.Content;

public interface ToolReadyHandler {

    void toolEnds(Content c, ToolEndsType type);
}
```