DACOM
data analyzed - comunity

Version 1.0

Developer Dokumentation

Francesco Carello 854396
David Detzler 854575
Timo Pauli 854450

Projekt der Fachhochschule Kaiserslautern, Sommersemester 2006, Lector: Prof. Hendrik Speck
www.da-com.sourceforge.net

Inhalt:

1. Einleitung

1.1. Da-Com

Da-Com ist eine Software zur Analyse und Visualisierung des sozialen Netzwerkes von www.gesichterparty.de. Sie basiert auf einer Web Crawler-Technologie, d.h. die benötigten Informationen werden zunächst über die Webseiten von gesichterparty gesammelt und die geparsten Daten in einer Datenbank gespeichert. Diese werden von der Visualisierungskomponente aufbereitet und in einem dynamischen Graphen dargestellt. Es werden Verbindungen zwischen einzelnen Usern und Usergruppen deutlich. Der Graf lässt sich dynamisch erweitern, abbauen und transformieren. Dabei können permanent Userinformationen abgerufen werden. Desweiteren kann der Graf durch verschiedene Filter genauer eingegrenzt werden und somit genauer analysiert werden. Die Software entstand im Sommersemester 2006 im Rahmen der Veranstaltung Medienkonzeption und Produktion, an der Fachhochschule Kaiserslautern, Standort Zweibrücken. Die Aufgabe bestand in der Realisierung einer Software zur Visualisierung eines sozialen Netzwerkes. Die zu entwickelnde Software sollte folgende 3 Bausteine enthalten:

- Crawler
- Datenbank
- Visualisierung

Die Entwicklung musste unter einer Open Source Lizenz realisiert werden. Somit sollte eine Weiterentwicklung auch nach Beendigung der Lehrveranstaltung ermöglicht werden.

1.2. Lizenz und Disclaimer

Da-Com wird unter der GNU General Public License (GPL) entwickelt. Mehr Informationen hierzu unter http://www.opensource.org. Da-Com übernimmt keine Haftung für eventuelle Schäden, die bei der Nutzung bzw. Weiterentwicklung des Crawlers entstehen.

1.3. Weiterentwicklung

Durch die Zusammenarbeit mit gesichterparty und einer Anbindung an deren Datenbank wäre eine genauere Zuordnung der Userpaare über die interne Buddy-Liste möglich. Weiterhin wären Features wie beispielsweise Userbild, Einbindung eines Live-Chats, sowie weitere Eigenschaftsfilter denkbar.

1.4. Entwicklerdokumentation

Ziel dieser Dokumentation ist es, zukünftigen Entwicklern bei der Programmierung, d.h. der Verwendung von Teilen des Quellcodes zu unterstützen. Durch die detaillierte Kommentare des Quellcodes der einzelnen Klassen, Funktionen und Variablen, dient diese Dokumentation lediglich zur weiteren Erläuterung und Verständnis der Gesamtstruktur der Anwendung. Zusätzlich befindet sich eine JavaDoc des Crawlers in dem da-com Sourcecode Package. Zum Verständnis dieser Dokumentation werden grundlegende Kenntnisse in der Programmiersprache Java 5.0 sowie ActionScript 2.0 vorrausgesetzt.

2. Entwicklungstechniken und Umgebung

2.1. Systemvorraussetzungen für die Entwicklung

Zur Weiterentwicklung von Da-Com werden folgende Komponenten benötigt:

- Java Development Kit JDK 5.0 der Java Standard Edition J2SE
- MySQL Datenbank
- MySQL JNDI Treiber (MySQL-Connector-Java-3.1.12)
- PHPMyAdmin oder ähnliches
- Testserver (Apache mit PHP Modul)
- Macromedia Flash 8

2.2. Programmiersprachen

Der Crawler wurde in Java 5.0 geschrieben. Desweiteren wurde ein Verbindungsscript in PHP benötigt, um die Kommunikation zwischen der MySQL Datenbank und Flash zu ermöglichen. Diese Script generiert eine XML- Datei mit den erforderlichen Informationen zur Erstellung des Grafen. Die Visualisierungskomponente, d.h. GUI und Graf wurden mit Flash 8 und ActionScript 2.0 realisiert.

2.3. Entwicklungsumgebung

Der Crawler wurde mit der OpenSource Entwicklungsumgebung Eclipse entwickelt. Unter der Verwendung des CVS Plugins konnte eine fortlaufender Versionsabgleich unter den Entwicklern ermöglicht. Für die Scriptsprachen SQL, PHP und XML wurde ein einfacher Editor(notepad++) verwendet. Die Visualisierungskomponente wurde ausschließlich Flash 8 entwickelt.

2.4. Software Dritter Parteien

Aufgrund mangelnder OpenSource Software im Bereich Netzwerkvisualisierung mit Flash, bestand keine Möglichkeit der Einbindung von anderen OpenSource Komponenten. Somit bietet Da-Com die erste OpenSource Lösung zur Visualisierung von Netzwerken unter Verwendung von Flash und ActionScript 2.0.

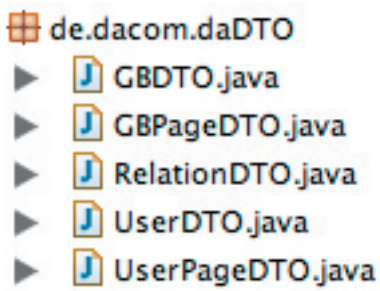
3. Softwarekomponenten

3.1. Crawler

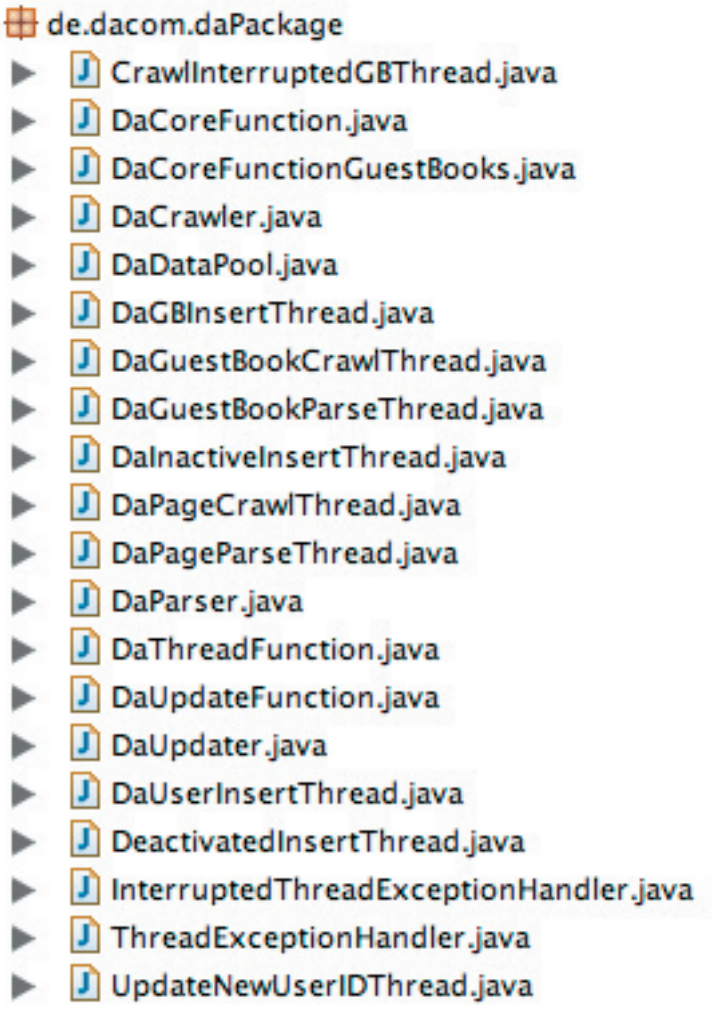
3.1.1. Packages und Klassen

Package: de.dacom.daDB
Dieses Package beinhaltet alle Funktionen die zur Datenbankanbindung und der Speicherung der Daten in der MySQL-Datenbank nötig sind. Eine detaillierte Beschreibung der einzelnen Klassen und Funktionen befindet sich im Kapitel 3.1.2 und 3.1.3.

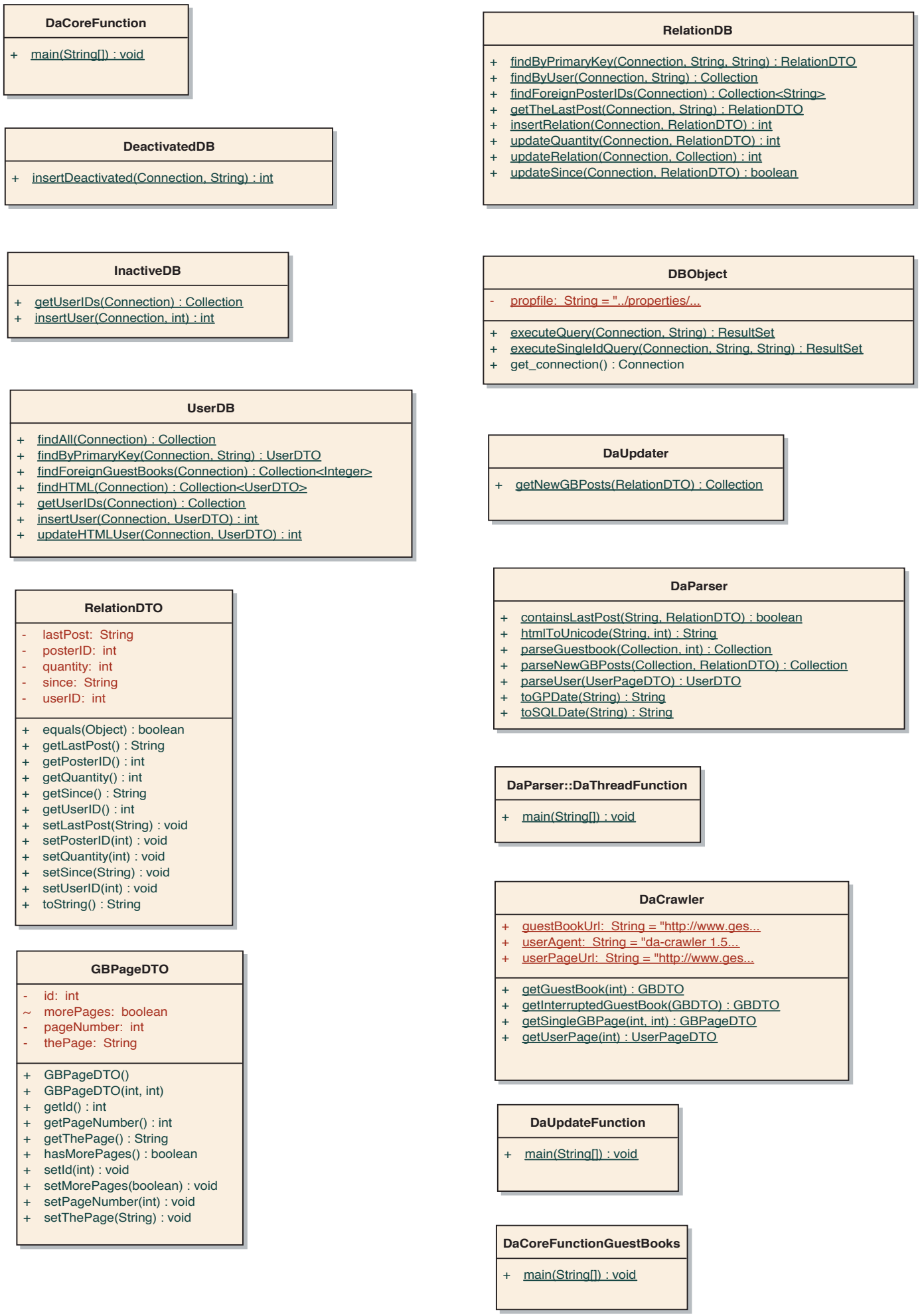
Package: de.dacom.daDTO
Dieses Package beinhaltet alle Klassen zum Transport spezifischer Informationen. Eine detaillierte Beschreibung der einzelnen Klassen und Funktionen befindet sich im Kapitel 3.1.2 und 3.1.3.

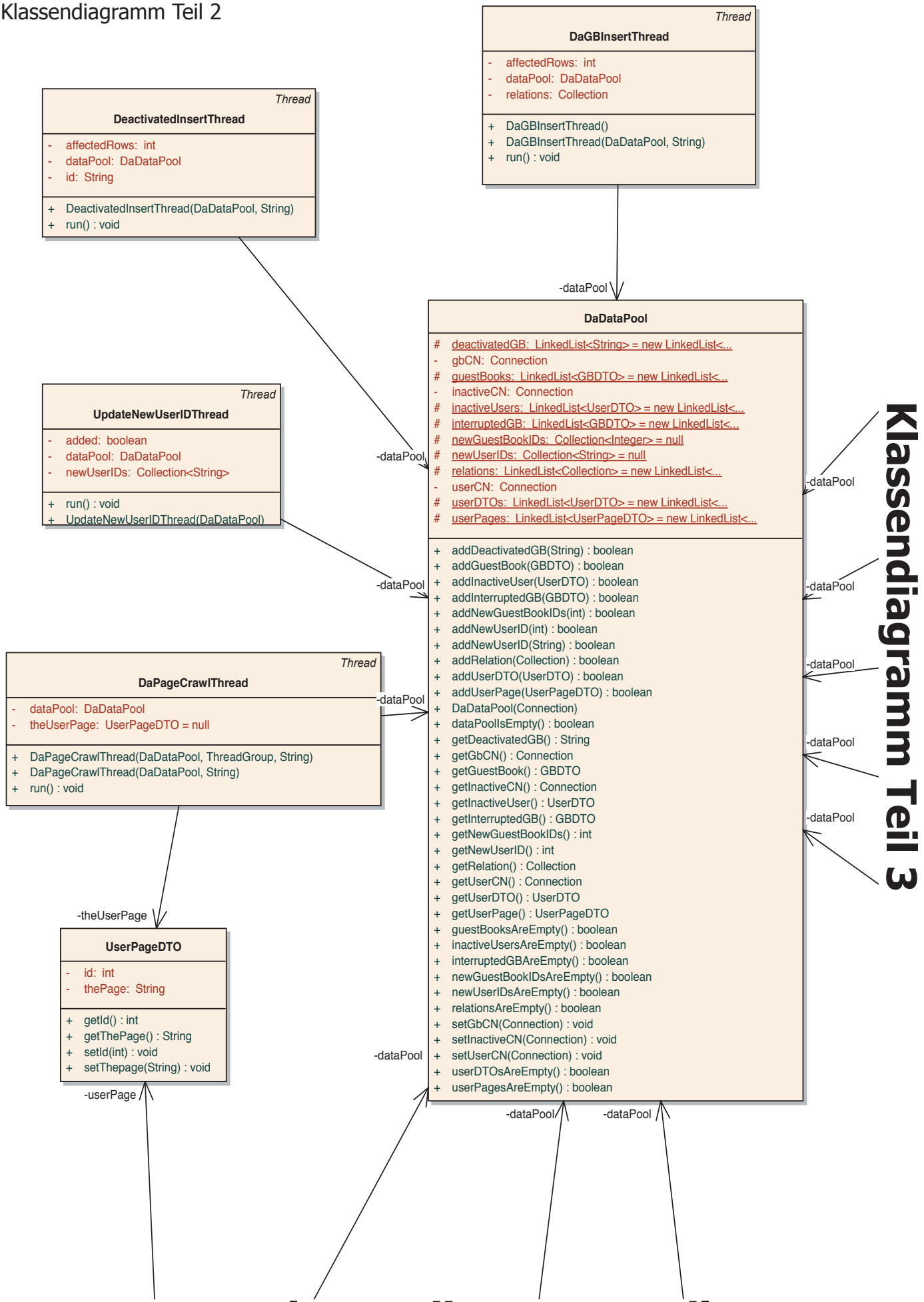


Package: de.dacom.daPackage
Dieses Package beinhaltet alle Klassen und Funktionen die zum Multithreading, crawlen und parsen benötigt werden. Eine detaillierte Beschreibung der einzelnen Klassen und Funktionen befindet sich im Kapitel 3.1.2 und 3.1.3.

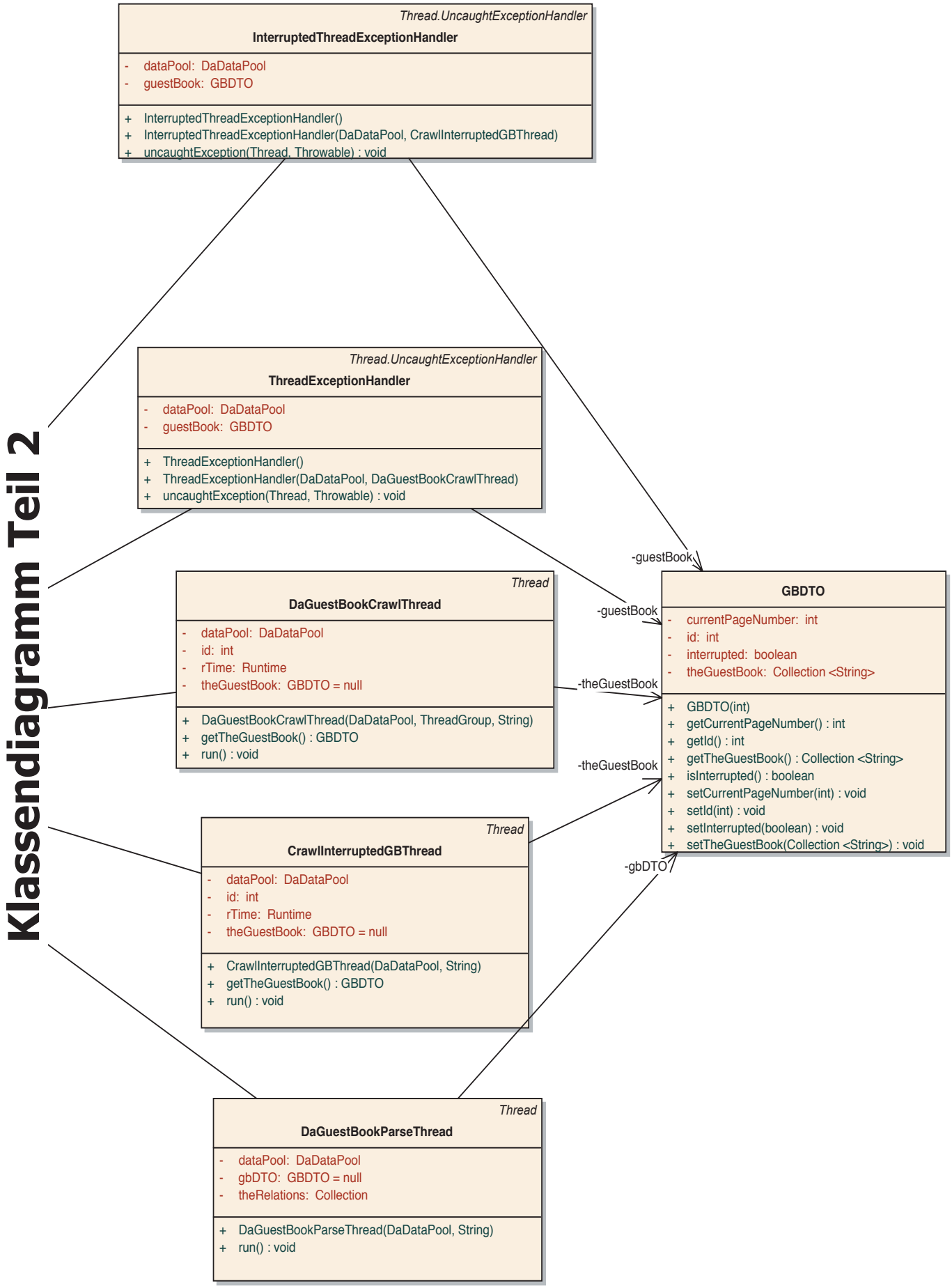


Klassendiagramm Teil 1

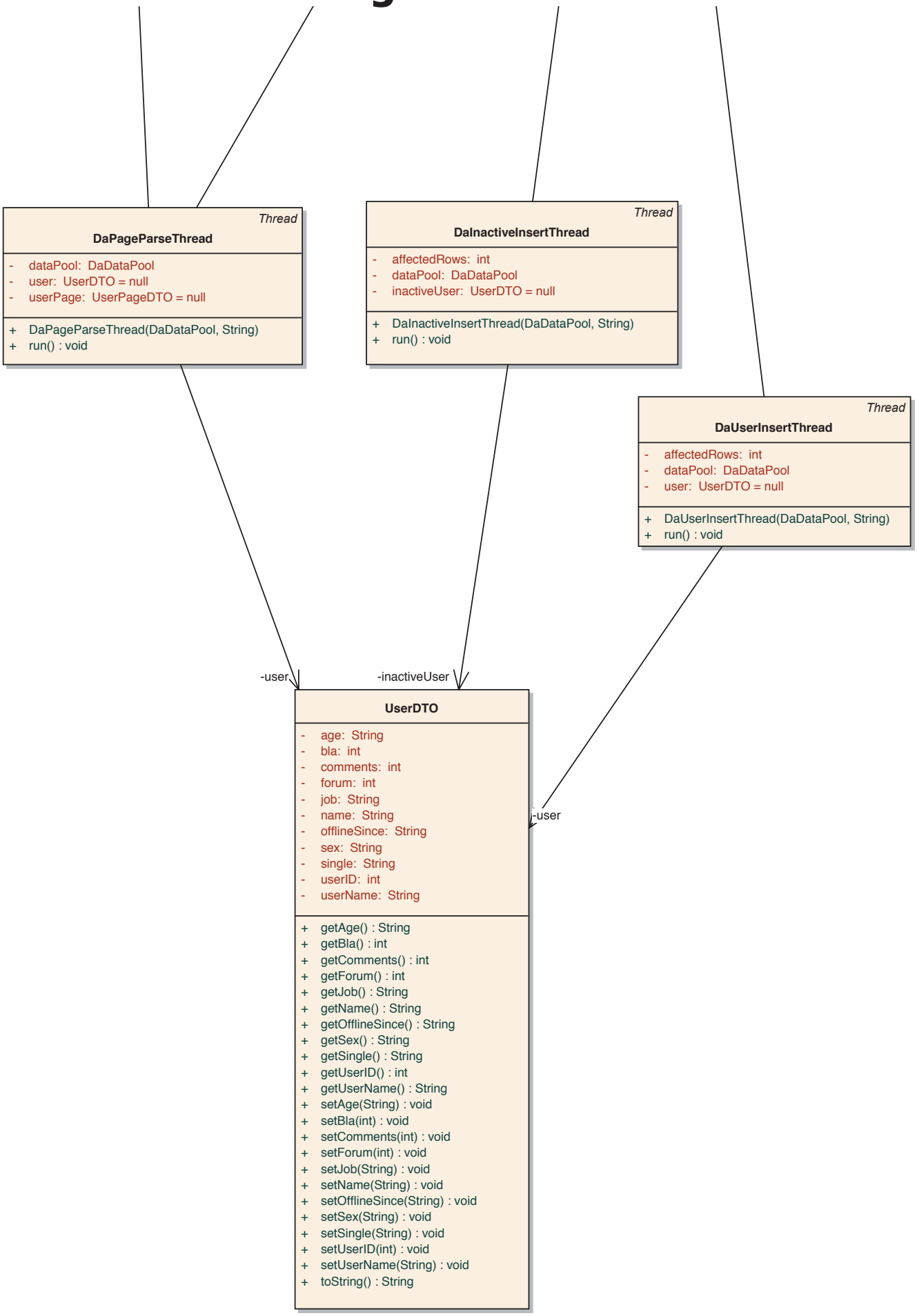



Thread
DaGBInsertThread
- affectedRows: int
- dataPool: DaDataPool
- relations: Collection
+ DaGBInsertThread()
+ DaGBInsertThread(DaDataPool, String)
+ run() : void
Thread
DeactivatedInsertThread
- affectedRows: int
- dataPool: DaDataPool
- id: String
+ DeactivatedInsertThread(DaDataPool, String)
+ run() : void
-dataPool
DaDataPool
deactivatedGB: LinkedList<String> = new LinkedList<...
- gbCN: Connection
guestBooks: LinkedList<GBDTO> = new LinkedList<...
- inactiveCN: Connection
inactiveUsers: LinkedList<UserDTO> = new LinkedList<...
interruptedGB: LinkedList<GBDTO> = new LinkedList<...
newGuestBookIDs: Collection<Integer> = null
newUserIDs: Collection<String> = null
relations: LinkedList<Collection> = new LinkedList<...
- userCN: Connection
userDTOs: LinkedList<UserDTO> = new LinkedList<...
userPages: LinkedList<UserPageDTO> = new LinkedList<...
+ addDeactivatedGB(String) : boolean
+ addGuestBook(GBDTO) : boolean
+ addInactiveUser(UserDTO) : boolean
+ addInterruptedGB(GBDTO) : boolean
+ addNewGuestBookIDs(int) : boolean
+ addNewUserID(int) : boolean
+ addNewUserID(String) : boolean
+ addRelation(Collection) : boolean
+ addUserDTO(UserDTO) : boolean
+ addUserPage(UserPageDTO) : boolean
+ DaDataPool(Connection)
+ dataPoolIsEmpty() : boolean
+ getDeactivatedGB() : String
+ getGbCN() : Connection
+ getGuestBook() : GBDTO
+ getInactiveCN() : Connection
+ getInactiveUser() : UserDTO
+ getInterruptedGB() : GBDTO
+ getNewGuestBookIDs() : int
+ getNewUserID() : int
+ getRelation() : Collection
+ getUserCN() : Connection
+ getUserDTO() : UserDTO
+ getUserPage() : UserPageDTO
+ guestBooksAreEmpty() : boolean
+ inactiveUsersAreEmpty() : boolean
+ interruptedGBAreEmpty() : boolean
+ newGuestBookIDsAreEmpty() : boolean
+ newUserIDsAreEmpty() : boolean
+ relationsAreEmpty() : boolean
+ setGbCN(Connection) : void
+ setInactiveCN(Connection) : void
+ setUserCN(Connection) : void
+ userDTOsAreEmpty() : boolean
+ userPagesAreEmpty() : boolean
Thread
UpdateNewUserIDThread
- added: boolean
- dataPool: DaDataPool
- newUserIDs: Collection<String>
+ run() : void
+ UpdateNewUserIDThread(DaDataPool)
-dataPool
Thread
DaPageCrawlThread
- dataPool: DaDataPool
- theUserPage: UserPageDTO = null
+ DaPageCrawlThread(DaDataPool, ThreadGroup, String)
+ DaPageCrawlThread(DaDataPool, String)
+ run() : void
-theUserPage
UserPageDTO
- id: int
- thePage: String
+ getId() : int
+ getThePage() : String
+ setId(int) : void
+ setThepage(String) : void
-userPage
-dataPool
-dataPool
-dataPool
-dataPool
-dataPool
-dataPool
-dataPool
-dataPool
Klassendiagramm Teil 3
Klassendiagramm Teil 4

Klassendiagramm Teil 3



Klassendiagramm Teil 2



3.1.2 Aufbau und Funktionsweise

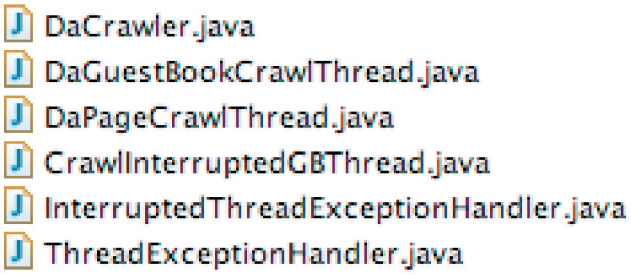
Der Crawler ist auf Grund seiner verschiedenen Aufgaben, das Beschaffen der HTML Seiten, das Parsen dieser und die anschließende Speicherung in der Datenbank, modular aufgebaut. Die Module wurden nach den spezifischen Aufgaben entwickelt, um so die einzelnen Arbeitsschritte klar von einander zu trennen und sowohl die Performance des Crawlers, als auch dessen Fehlerbehandlung und Fehleranfälligkeit zu optimieren bzw. reduzieren. Im folgenden werden die Module, deren Funktionsweise und deren wichtigsten Klassen genauer beschrieben.

3.1.2.1 DataPool

Die Klasse `DaDataPool.java` dient als „Controller" für die einzelnen Module. Sie synchronisiert den Datenaustausch der einzelnen Threads, damit es zu keinen Deadlocks kommt. Alle Daten werden im DataPool gespeichert und somit den anderen Modulen zur Verfügung gestellt. Sobald Daten von den CrawlThreads in den DataPool geschrieben werden, startet dieser neue ParsingThreads, um die benötigten Informationen aus den Rohdaten zu lesen und diese wiederrum in Form von DTO's (DataTransferObject) im DataPool zu speichern. Durch das Speichern der DTO's im DataPool werden neue InsertThreads gestartet, welche die DTO Objekte aus dem DataPool entnehmen und die gesammelten Informationen in den entsprechenden Datenbanktabellen speichern.Eine detaillierte Auflistung der Methoden befindet sich sowohl im JavaDoc, als auch im Klassendiagramm im Abschnitt 3.1.1 Packages und Klassen.

3.1.2.2 Crawlermodul

Das Crawlermodul besteht aus folgenden Klassen:



Die Klasse `DaCrawler.java` enthält alle Funktionen, die für das Crawlen der HTML Seiten nötig sind.

```
public static UserPageDTO getUserPage(int userID)
public static GBDTO getGuestBook(int id)
public static GBPageDTO getSingleGBPage(int id, int pageNumber)
public static GBDTO getInterruptedGuestBook(GBDTO guestBook)
```

Diese Funktionen bauen eine TCP/IP Verbindung auf und lesen die Daten der angefragten Site in einen BufferedReader. Dieser speichert zeilenweise die Daten in einem StringBuffer, welcher die gelesene HTML Seite in Form eines String speichert. Dieses Prinzip ist bei allen Crawl-Methoden gleich.

Die Aufgabe des `DaPageCrawlThread.java` ist es, die Methode getUserPage() der Klasse

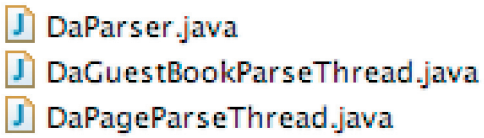
`DaCrawler.java` aufzurufen. Die Methode getUserPage() bekommt die ID eines Users, welche aus dem DataPool entnommen wird, als Argument. Diese ID wird an die statische URL angehängt um somit die Verbindung zu dieser Seite aufzubauen. Der Rückgabewert von getUserPage() ist ein UserDTO Objekt, welches den HTML String der User Page und die User ID enthält. Der DaPage-CrawlThread speichert dieses Objekt zur weiteren Verarbeitung im DatenPool.

Die Klasse `DaGuestBookCrawlThread.java` hat die Aufgabe das Gästebuch eines Users zu Crawlen, es in einem GBDTO Objekt zu speichern und es in den DataPool zu schreiben. Ein Thread dieser Klasse entnimmt eine User ID aus dem DataPool und ruft die Methode getGuestBook() der Klasse `DaCrawler.java` mit der User ID als Argument auf. Diese Methode ruft so lange die get-SinglePage() bis das Ende des Gästebuchs erreicht ist. Ist das Ende des Gästebuchs erreicht so wird ein GBDTO, welches alle HTML Seiten in einer String Collection und die User Id enthält zurück gegeben. Tritt während des Crawlens ein Fehler auf, z.B. durch Timeout der TCP Verbindung, wird geprüft ob schon Seiten gespeichert worden sind. Ist dies der Fall und das Gästebuch hätte noch weitere Seiten gehabt, so wird das GBDTO Objekt in der InterruptedGB Collection im DataPool gespeichert. Daraufhin wird ein Thread der Klasse `CrawlInterruptedGBThread.java` gestartet, welcher ein GBDTO aus dem DataPool entnimmt und das entsprechende Gästebuch zu ende crawlt. Sollte während dessen wieder ein Fehler auftreten, so wird das GBDTO wieder zurück in den Data-Pool geschrieben und von einem neuen Thread zu ende gecrawlt.

Der `ThreadExceptionHandler.java` dient dazu, die Anzahl der CrawlThreads konstant zu halten. Tritt eine Exception auf, welche nicht abgefangen wurde, wird wieder ein neuer CrawlThread gestartet.

3.1.2.3 Parsermodul

Das Parsermodul besteht aus folgenden Klassen:



Die wichtigste Klasse des Parsingmoduls ist die Klasse `DaParser.java`, welche den Thread Klassen folgende Methoden zum Parsen bietet:

```
public static UserDTO parseUser(UserPageDTO pageDTO)
public static Collection parseGuestbook(Collection allPages, int id)
public static String toSQLDate(String date)
public static String toGPDate(String date)
```

Sobald neue Rohdaten in Form von DTO Objekten in den DatenPool geschrieben werden, starten neue Parsing Threads. Je nach DTO Objekt startet ein `DaGuestBookParseThread.java` zum parsen eines Gästebuchs oder ein `DaPageParseThread.java` zum parsen einer User Page.

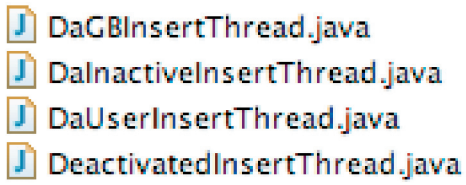
Ein DaGuestBookParseThread entnimmt dem DataPool ein GBDTO Objekt. Mit dessen Attributen, allPages (Collection)und id (int) wird die Methode parseGuestBook() der Klasse `DaParser.java` aufgerufen. Das Attribut allPages ist eine String Collection mit allen Seiten eines Gästebuchs. In der parseGuestBook() Methode werden nun mit Hilfe eines Iterator Objekts alle Gästebuchseiten aus der Collection entnommen und folgende Informationen extrahiert. Die User ID, die Anzahl an Einträgen, das Datum des ersten und des letzten Eintrages eines jeden Schreibers. Alle diese Informa-

tionen werden anschließend in einem RelationDTO gespeichert und in eine Collection geschrieben. Die Collection mit allen RelationDTO Objekten wird nun wieder an den DaGuestBookThread zurück gegeben. Dieser Thread schreibt die Collection in den DataPool.

Sobald ein neues UserPageDTO Objekt in den DataPool geschrieben wird, startet ein neuer DaPageParseThread. Dieser entnimmt das UserPageDTO Objekt und ruft die Methode parseUser(), welche ein UserPageDTO Objekt als Argument erwartet, auf. In dieser Methode wird das String Attribut thePage, welches die User Page enthält, entnommen und geparst und folgende Informationen in einem neuen UserDTO Objekt gespeichert: Nickname, Name, Alter, Beruf, Geschlecht, Single, Forum, BlaBox und Kommentare. Das UserDTO Objekt wird an den Thread zurück gegeben und von diesem in den DataPool geschrieben. Wird während dem parsen festgestellt, dass der User nicht mehr aktiv ist, so wird er in die Collection inactiveUser des DataPools geschrieben.

3.1.2.4 Databasemodul

Das Databasemodul dient der Speicherung der geparsten Informationen in der Datenbank. Folgende Klassen gehören zu diesem Modul:



Sobald das Parsermodul neue DTO Objekte in den DataPool schreibt wird je nach Object ein neuer Thread gestartet. Dieser Thread entnimmt das DTO Objekt und speichert die enthaltenen Informationen in der entsprechenden Datenbanktabelle. Zur Speicherung werden die insert Methoden der verschiedenen DBKlassen verwendet.

```
public static int insertUser(Connection cn, int id)
public static int insertDeactivated(Connection cn, String id)
public static int insertRelation(Connection cn, RelationDTO Relation)
public static int insertRelation(Connection cn, RelationDTO Relation)
public static int insertUser(Connection cn, UserDTO user)
```

Zwei wichtige DBMethoden seien an dieser Stelle noch erwähnt, die bei der initialisierung und aktualisierung des DataPools eine wichtige Rolle spielen.

```
public static Collection<Integer> findForeignGuestBooks(Connection cn)
public static Collection<String> findForeignPosterIDs(Connection cn)
```

Diese Methoden der Klassen `UserDB.java` und `RelationDB.java` finden alle User und Gästebücher welche noch nicht in der Datanbank gespeichert worden sind. Die entsprechenden ID's werden im DataPool gespeichert um dann von den anderen Modulen verarbeitet zu werden. Weitere Informationen befindet sich im sowohl im JavaDoc, als auch im Kapitel 3.1.1 Packages und Klassen.

3.2. Visualisierung und GUI

3.2.1 Packages und Klassen

Package: *de.dacom.flash.as.events*

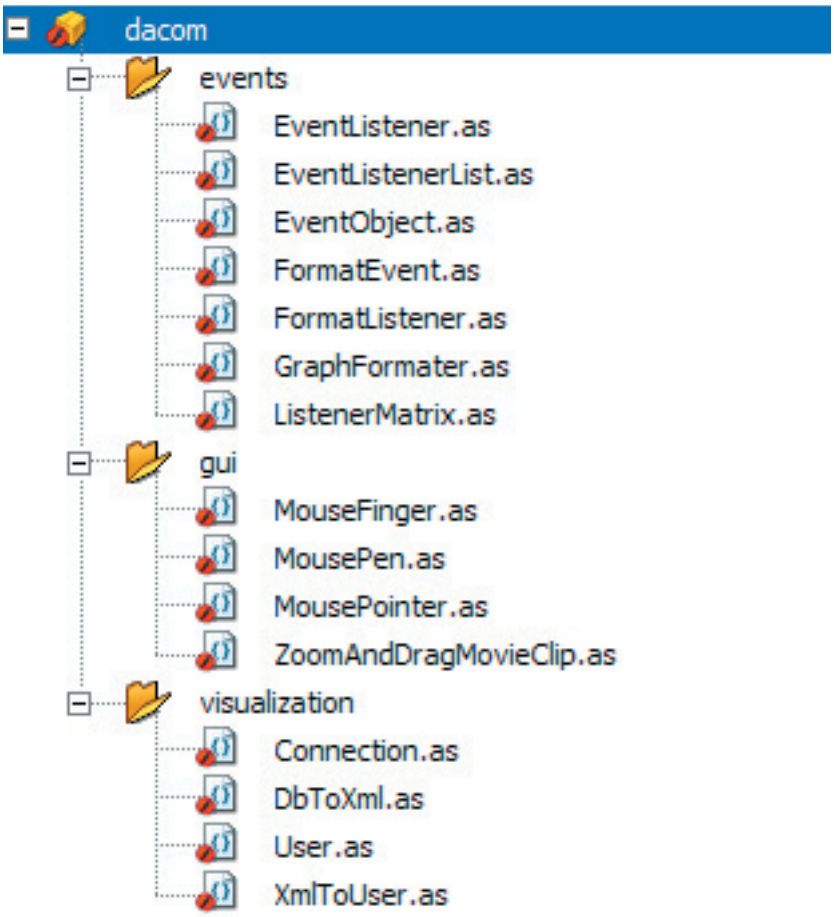
In diesem Package sind alle Klassen zur Kommunikation zwischen dem GUI und dem Grafen enthalten. Die Struktur entspricht der eines Delegation-Event-Models. Die Hauptklassen dieses Packages sind GraphFormater(regelt die korrekte Darstellung des Graphen und des GUIs) und ListenerMatrix(stellt den Grafen, d.h User und deren Connections in einer Matrix dar).

Package: de.dacom.flash.as.gui

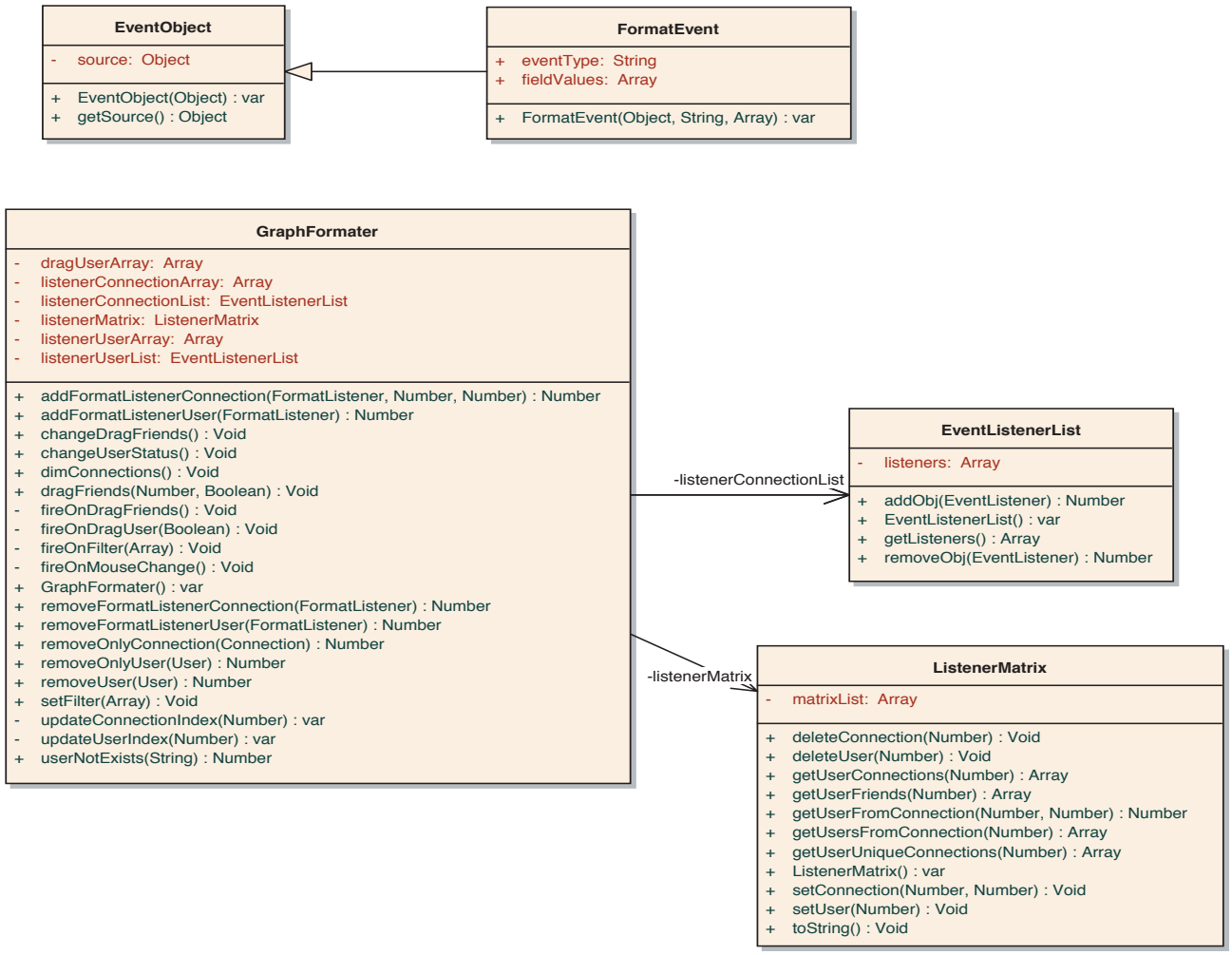
Diese Package enthält Klassen zur Darstellung des GUI. Die Klasse ZoomAndDragMovieClip beinhaltet den kompletten Grafen, d.h. alle User und deren Connections untereinander. Dadurch lassen sich Transformationen des gesamten Grafen(zoomen, verschieben, anpassen) einfacher bewältigen.

Package: da.dacom.flash.as.visualization

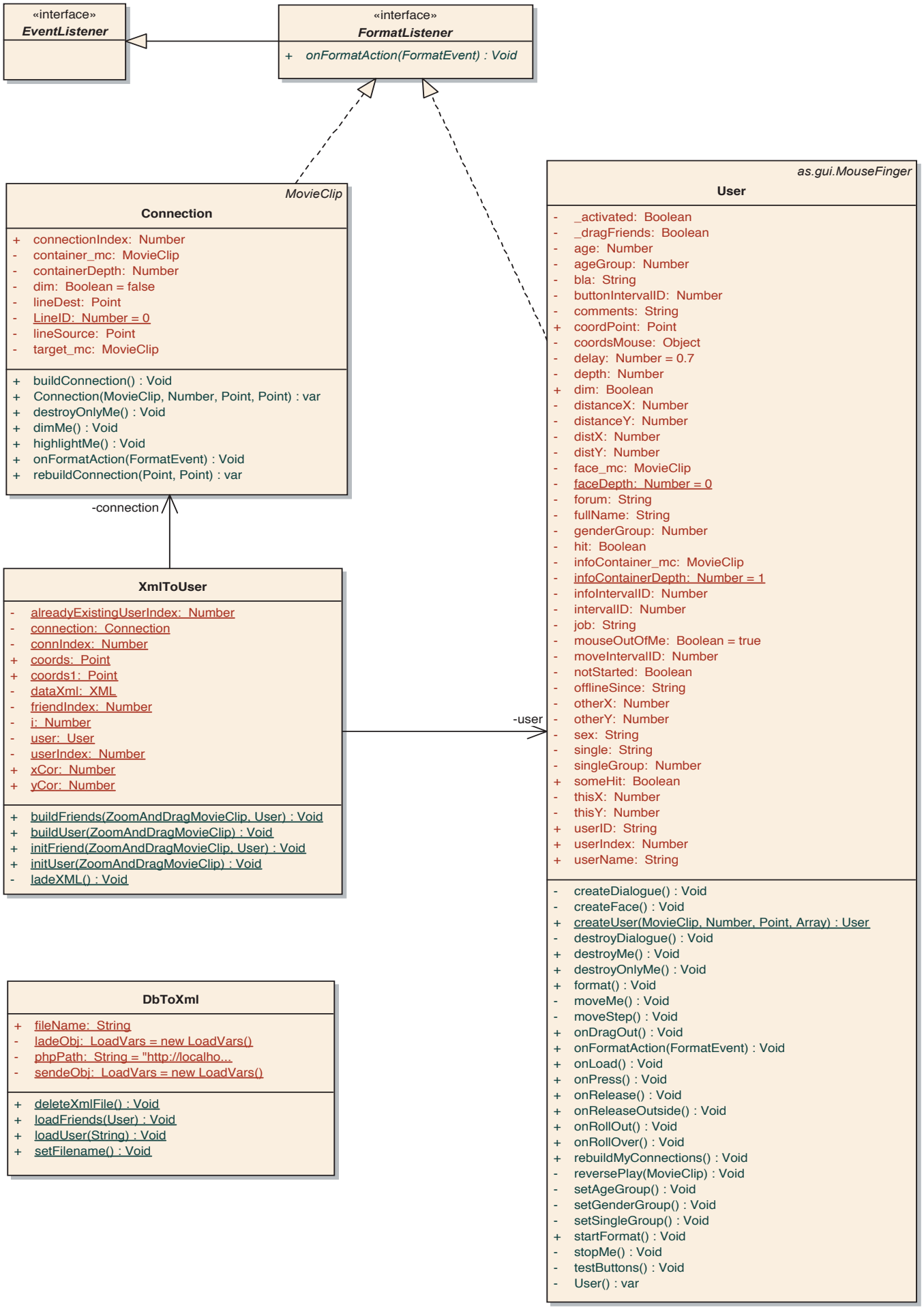
Die in diesem Package enthaltenen Klassen User und Connection dienen der Darstellung der einzelnen Grafenelementen. Die beiden anderen Klassen ermöglichen den Datenaustausch zwischen der Datenbank und Flash, sowie der Erzeugung der entsprechenden User bzw. Connections anhand der erzeugten XML-Datei.



Klassendiagramm Teil 1



Klassendiagramm Teil 2



Klassendiagramm Teil 3

3.1.2 Aufbau und Funktionsweise

3.1.1 Aufbau des Grafen

Der Graf und seine Funktionsweise bzw. Steuerung durch den User wird durch das Zusammenspiel zwischen den Klassen `User.as`, `Connection.as`, `XmlToUser.as`, `DbToXml.as`, `GraphFormater.as`, der `ListenerMatrix.as` und den dazugehörigen Event-Klassen realisiert. Die Kommunikation besteht unter dem Konzept eines Delegation-Event-Models. Hierbei werden Aktionen, die auf einen User einwirken, wie das Anzeigen des Informations-Dialogs eines Users, das Ausklappen seiner Freunde bzw. das Entfernen eines Users und seinen Freunden, über die onPress() bzw. onRelease() Funktionen zunächst an die Klasse GraphFormater geleitet. Jeder User bzw. Connection, die zuvor in der Klasse XmlToUser erzeugt werden, registrieren sich sofort durch die Methode addFormatListenerUser() bzw. addFormatListenerConnection() bei der Klasse GraphFormater. Um eine Verknüpfung zwischen den erzeugten Usern und Connection zu erhalten, wird in der Klasse GraphFormater eine ListenerMatrix erzeugt, die eine Inzidenz-Matrix darstellt, wobei zu einem bestimmten User Index die dazugehörenden Connection Indexes zugeordnet werden.

User Index \ Connection Index	0	1	2	3
0	1	1	1	1
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Hierbei ist es notwendig, dass die User bzw. Connections den gleichen Index aufweisen, den sie zuvor auch in den Listener Listen beim registrieren erhalten haben, damit eine eindeutige Verbindung der Matrix und der Listen besteht. Die ListenerMatrix ermöglicht es mit Hilfe ihrer Funktionen bestimmte Connections oder Freunde eines Users zu ermitteln. Der GraphFormater bedient sich dieser Funktionen, um aus den registrierten Usern (listenerUserList) und Connections (listenerConnectionList) die gesuchten Objekte zu ermitteln und danach die Funktion onFormatAction() der jeweiligen FormatListener Klassen (User und Connection) aufzurufen. Dieser Funktion wird ein FormatEvent Objekt übergeben, welches ein Attribut Type besitzt, das über die im User oder Connection auszuführende Funktion bestimmt. Das jeweilige FormatListerner Objekt (User/Connection) entscheidet nun in der onFormatAction() Funktion anhand des FormatEvent.type, welche Aktion ausgeführt werden soll. Die Listener Listen in der Klasse GraphFormater werden auch oftmals dazu verwendet, um eine bestimmte Methode eines User bzw. Connection Objekts direkt aufzurufen. Dies geschiet beispielsweise, um die Connections zu dimmen. Wenn ein User aus dem Graf mittels dem minus Buttons entfernt wird, wird er, seine Freunde und Connections ebenfalls aus den Listener Listen und der Matrix entfernt.

3.1.2 Aufbau des Grafen

Die Verteilung der User um einen bestimmten User herum geschieht durch die Methoden format() und rebuildMyConnections(). Die Methode format() untersucht, ob der User MovieClip in Kontakt mit einem anderen User kommt. Ist das der Fall wird auch bei jenem User die Funktion format() aktiviert. Dies führt zu einer Kettenreaktion, wobei jeder User den anderen überwacht. Das führt bei vielen Userkontakten zu einer höheren CPU Auslastung. Sobald kein Kontakt mehr eines Users mit einem anderen besteht, wird die Methode format() gestoppt und damit die CPU Auslastung sofort wieder reduziert. Letzdenendes hat die CPU Auslastung den Tiefpunkt erreicht, wenn kein User

mehr einen anderen berührt. Durch die Methode rebuildMyConnections() wird mit Hilfe des Graph-Formaters dafür gesorgt, dass ständig die Verbindungen zwischen den Usern aktualisiert werden.

3.1.3 Aufbau der GUI

ZoomAndDragMovieClip:

Dieser MovieClip beinhaltet letztendlich den gesamten Grafen. Damit konnten auf effektivem Weg Transformationen, die den gesamten Graf betreffen realisiert werden. Die Kernfunktionen sind: zoomMovie(), zoomToPoint(), zoomToStage(), startDragMovie() und stopDragMovie(). Die Funktion zoomMovie() ermöglicht es, den Grafen bis zu 100% zu vergrößern oder bis zu einem bestimmten Wert zu verkleinern. Die Werte können auch über den Konstruktor der Klasse übergeben werden. ZoomToPoint() zoomt den Grafen zum einen auf 100% und zum anderen werden die Koordinaten des Grafen, die mit der Maus anvisiert wurden zur Mitte des Bildschirms verschoben. Durch zoom-ToStage() wird der gesamte Graf auf dem Bildschirm sichtbar gemacht und angepasst. Die beiden letzten Drag Funktionen dienen dazu den Grafen zu bewegen.

MousePointer:
Um die verschiedenen erläuterten Funktionen zu aktivieren, müssen die dazu gehörenden Schalter _enableDrag, _enableZoom, _enableZoomToPoint auf true gesetzt werden. Diese Aufgabe übernimmt die Klasse MousePointer, die sich ebenfalls im package gui befindet. Hier werden durch die verschiedenen Funktionen, wie zum Beispiel getZoomMovie(), diese Schalter in der Klasse ZoomAndDragMovieClip entsrechend gesetzt und gleichzeitig der korrekte Mauszeiger aus dem Bedien-Panel als Mauszeiger aktiviert, d.h im Panel unsichtbar gemacht und dem Mauszeiger angehängt. Hier werden auch durch die Funktionen getFinger(), getHand() bzw. getPen() die entsprechenden RollOver für die GUI realisiert.

MouseFinger/MousePen:
Diese Klassen dienen lediglich der entsprechenden Darstellung des Mauszeigers bei den jeweiligen GUI Elementen.