|  |
| --- |
|  |

|  |
| --- |
| **Task 08** |
| RMI Auctionsystem |
|  |
|  |
|  |
| **Krepela, Lipovits, Reichmann, Tattyrek, Traxler** |
| **29.01.2014** |
|  |

Insert Specification Here

Designüberlegung

# Testing Component

**Reading Property File:**

# TODO: adjust these values

clients = 100

auctionsPerMin = 1

auctionDuration = 2\*60

updateIntervalSec: 20

bidsPerMin = 2

Lines, which start with a '#' are comments and do not affect any functionality.

The next line describes the number of clients which should be used within this test.

Afterwards, the auctions per minute, the auction duration, the update interval in seconds and the bids per minute are given.

The attribute and the value can be split by '=' or ':', additionally the number can consist of two multiplicators, which have to be multiplicated before it can be saved.

**Exceptions (Lipovits only)**

CommandNotFoundException()

→ Thrown if a Command does not exist

IllegalNumberOfArgumentsException()

→ Thrown, if the userinput conists of a wrong number of arguments for the command

WrongInputEception()

→ Thrown, if the command exists and has the right number of arguments, but one or more arguments are of a wrong type. e.g. '!removeSteps 1 miau'

# Management Client

**Commands**

The following management client - commands where implemented as a prototype:

* !login
* !logout
* !steps
* !addStep
* !removeStep
* !bill
* !subscribe
* !unsubscribe

Those commands are recognized and checked by the client and print a response.

# Billing Server

**Saving price steps:**

***private*** *ConcurrentSkipListMap<CompositeKey,PriceStep> priceSteps;*

CompositeKey has 2 attributes (pk’s): start and end price. It also provides methods to compare these keys. The function overlaps() tests if they collide with each other.

Price Step contains all 4 attributes. The shutdown method saves data persistent to a file.

**Saving bills:**

***private*** *ConcurrentHashMap<String,Bill> bills;*

If the list already contains the user (key), the bill will be added to the Bill Object. The Bill class contains a synchronized LinkedList<BillingLine>, which holds all BillingLines. The shutdown method saves data persistent to a file

**Exceptions**

IllegalValueException ()

→One or more arguments are invalid!

e.g. :

values below zero

endprice must be bigger than startprice

PriceStepIntervalOverlapException()

→The provided price interval overlaps with an existing price step

(delete the other price step first)!

# JUnitTests

**createPriceStepTest**

Creates four pricesteps (unordered) and look with getPriceSteps() if they have been created and if they are ordered correctly.

**createPriceStepTestPriceStepIntervalOverlapException**

Expects: PriceStepIntervalOverlapException

Creates two PriceSteps. The end prices overlap.

**createPriceStepTestPriceStepIntervalOverlapException2**

Expects: PriceStepIntervalOverlapException

Creates two PriceSteps. The end prices are infinity and the start prices overlap.

**createPriceStepTestPriceStepIntervalOverlapException3**

Expects: PriceStepIntervalOverlapException

Creates two PriceSteps. The first price step has as end price infnity and the second start price is higher than the start price from the first price step. The price steps overlap.

**createPriceStepTestIllegalValueException**

Expects: IllegalValueException

Create a price step with negative value. Throws IllegalValueException.

**deletePriceStepTest**

Creates a price step. Deletes a price step. Then creates the same step again and some additional. No exception is called because the step was successfully deleted and could be created again.

**toStringTest**

Creates a price step and tests if the toString method output format is correct.

**billAuctionAndGetBillTest**

Creates some price steps and bills. Tests if the bill functions calculation is correct. (calls getBill).

**billAuctionAndGetBillTestIntervalDoesNotExist**

Creates some price steps and bills. Tests if the bill functions calculation is correct if no price step exists for the given interval. (calls getBill)

**getBillTestFalse**

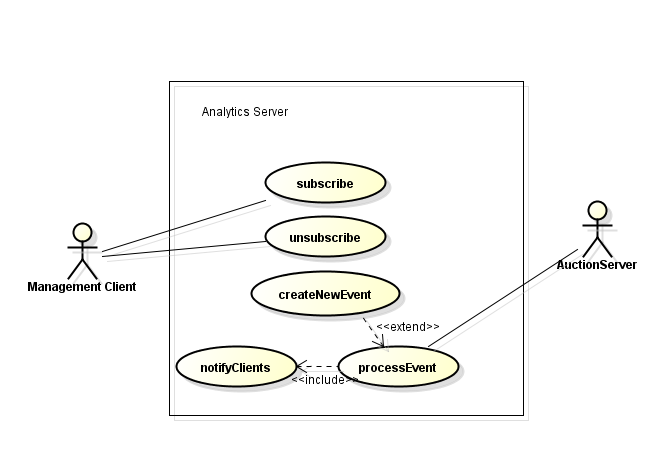
Calls getBill for a user that does not exist. Tests if the output is correct (user gets informed that there are no bills for him)

**shutdownAndLoad**

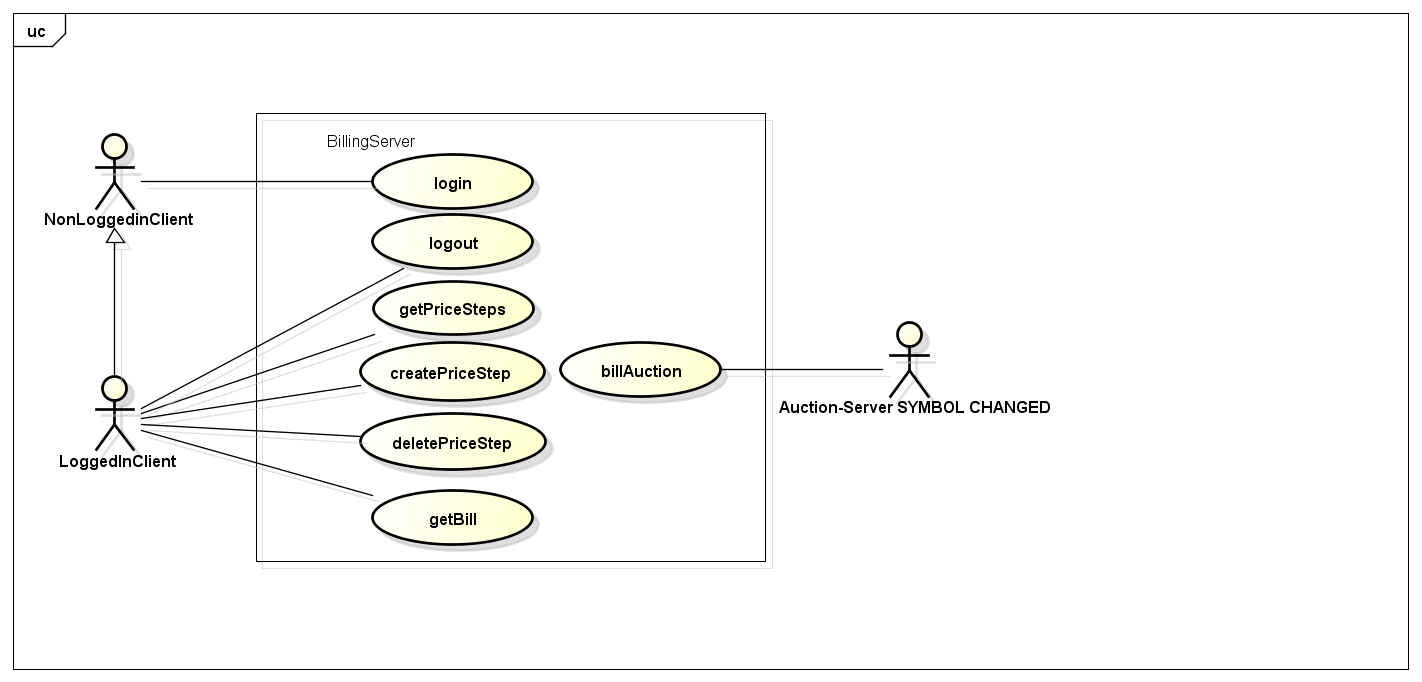
Creates some price steps and bills. Calls getBill to show that the data exists in the Map. Tests if shutdown saves the data to a file and if the data is loaded again after creating a new instance. These functions work correct if the same data is loaded again. That means that the getBill method is called again after saving and loading the data.

## Use Case

Analytic Server:



BillingServer:



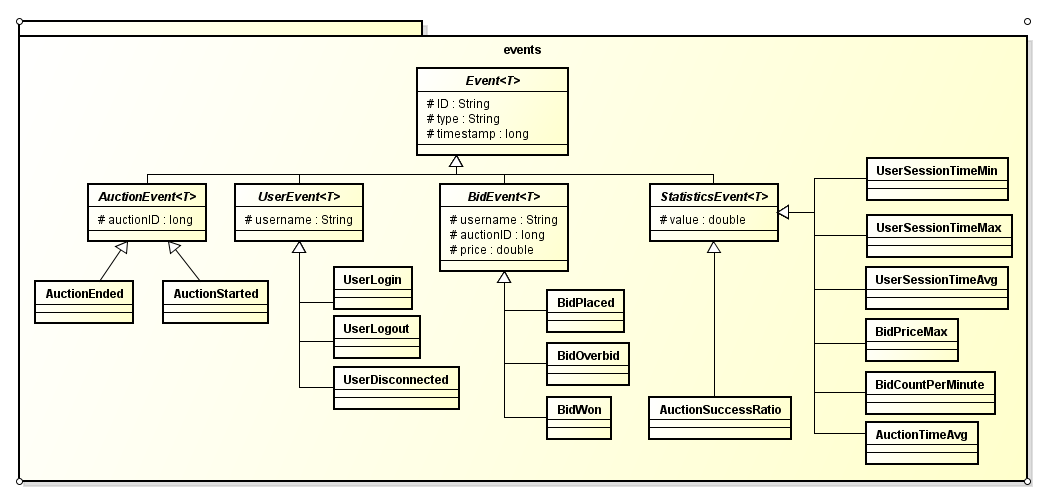
## AktivitätsdiagrammAuktionen anzeigen.jpgAuktionsende.jpgBieten.jpgClient Login.jpgClient Logout.jpgEinschreiben.jpgErstellen .jpgEvent verarbeiten.jpgManagement Login.jpgManagement Logout.jpgStep bearbeiten.jpgSteps anzeigen.jpg

## Klassendiagramme

### Analytics

### 

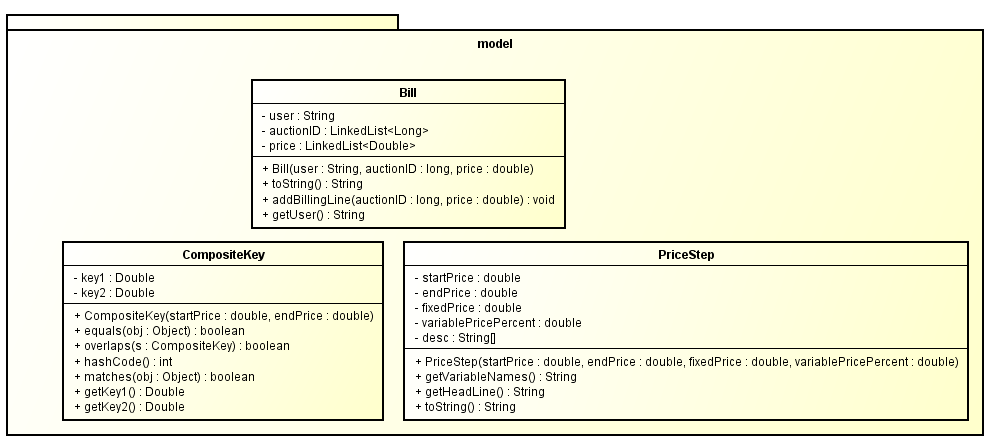
### Events



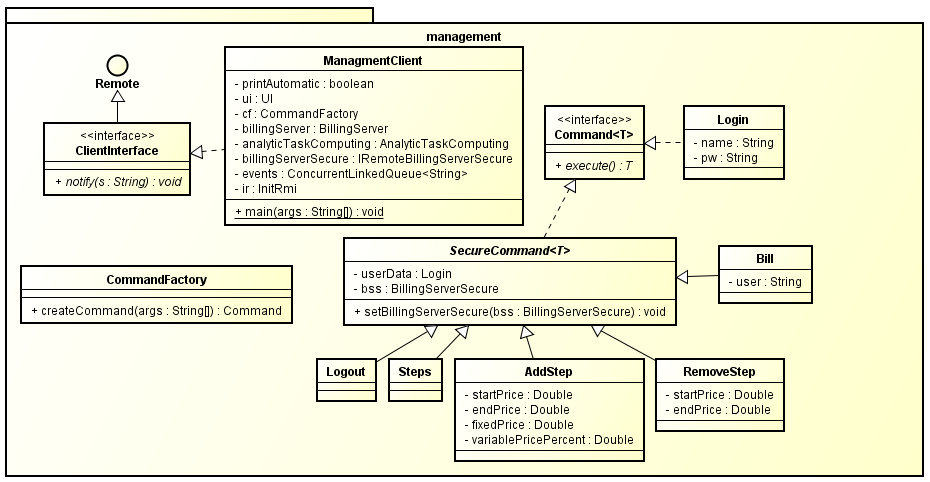
### Billing

### 

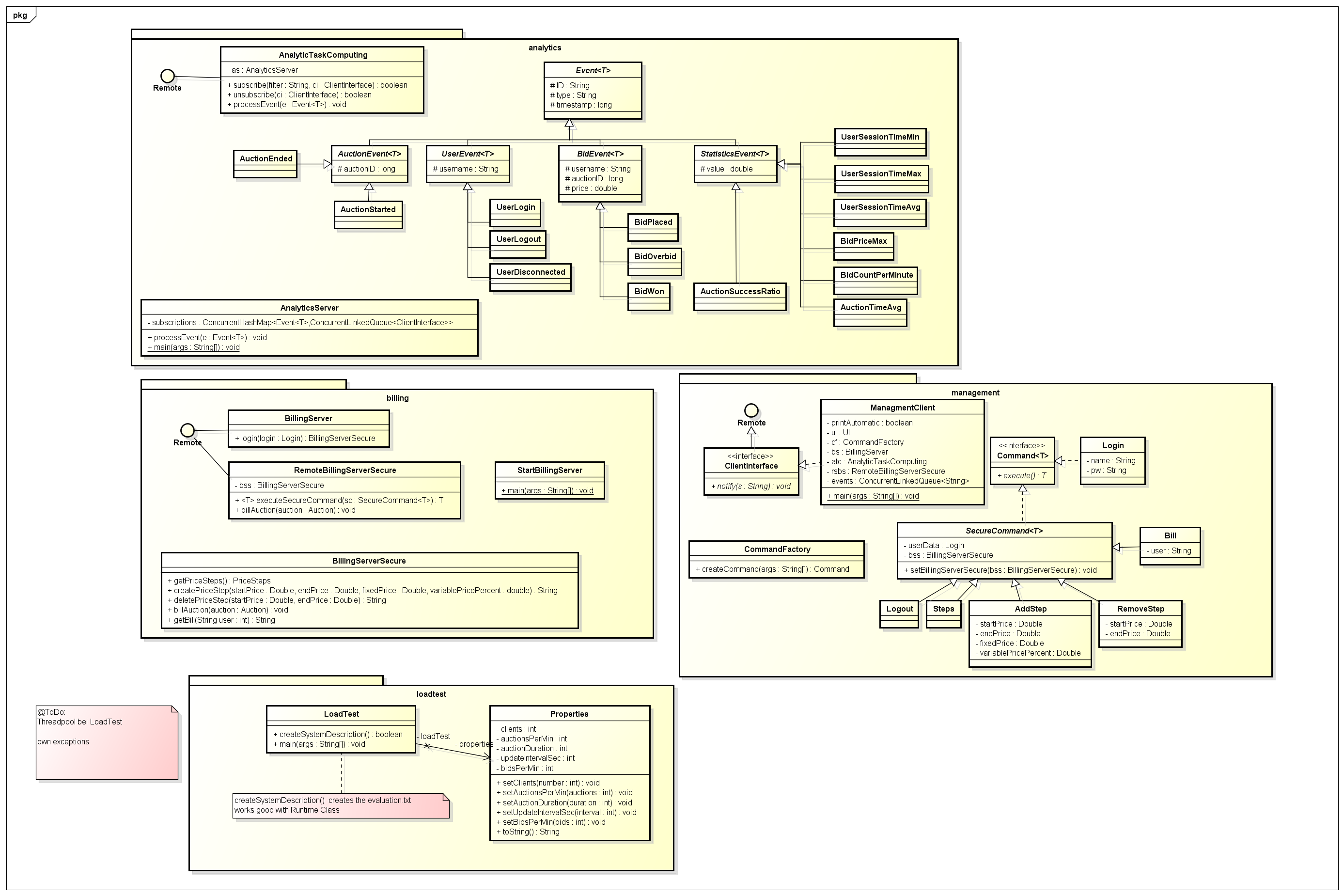
### Billing Model



### Managemenclient



### Testing Component



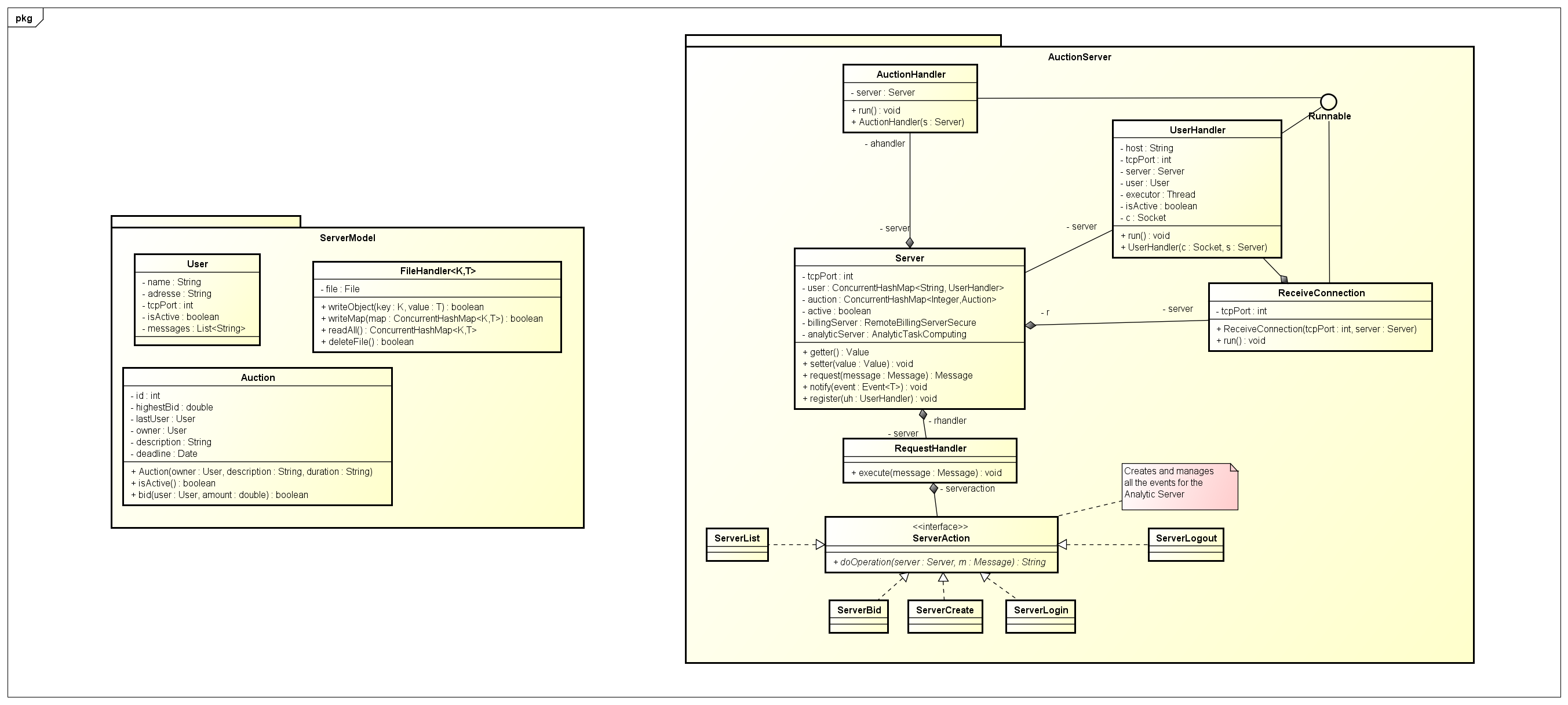
### Client



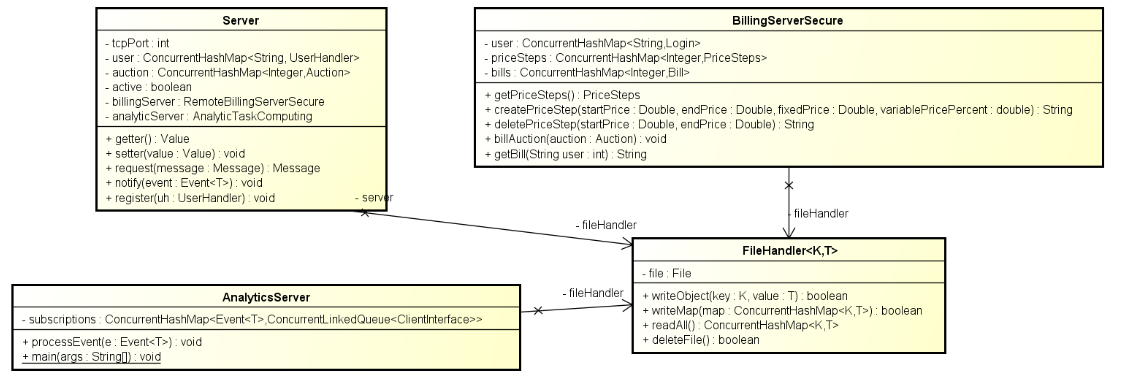
### Message-Model



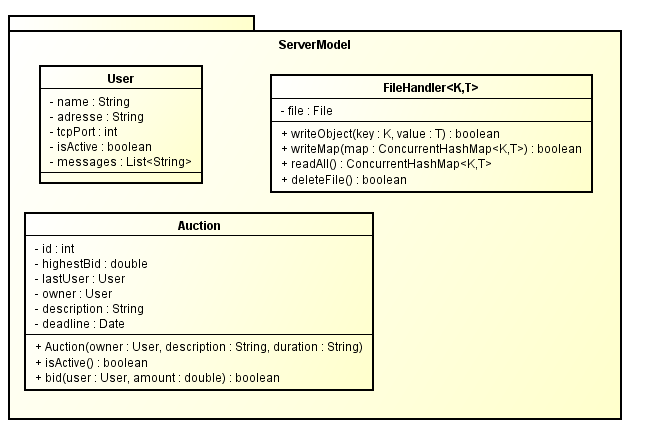
### Server



### FileHandler



### Model Server + FileHander



# Zeitschätzungen und Arbeitsaufteilung



# Absprachen

-> Analytic Server

-> Billing Server (Frage GRAFIK?) schritte setzen, abrechnung erstellen -> keine persistenz

-> Testing Load Client (Viele Clients machen bids etc.)

-> Management Client (Befehle für Billing Server, Benachrichtungungen etc)

Altes Programm → List-> ConcurrentHashMap, Eigene Exceptions → UDP Notification brauchen wir nicht mehr

UML-Klassendiagramm → Krepela, Traxler

Aktivitätdiagrann → Lipovits

Use Case Diagramme → Reichmann

Checker → Tattyrek

Tasks:

* RMI-Verbindungen → Traxler
* Analytics Server → Reichmann, Tattyrek
* Billing Server → Krepela
* Management Client → Lipovits
* Testing Component → Lipovits
* Model →Tattyrek (Model JUnitTests)
* Ausbesssern alten Code → Traxler, Reichmann
* Ant,Protokoll → Reichmann

JEDER TESTET SEINEN TEIL + TECHNOLOGIEBESCHREIBUNG FÜRS PROTOKOLL!!!

**Arbeitspakete bis Montag, 10. Februar**

Liebes Team,

damit auch was beim Projekt weitergeht, wird es notwendig die Aufgaben zu definieren:

Jeder sollte bis zum 10. Februar einen Prototypen (= 70% Funktionalität) umgesetzt haben. Dieser soll bis Montag fertig sein, da ich einen Tag benötige die Arbeitspakete zu kontrollieren und etwaige Sachen auszubessern. Ich befinde mich selbst auf Urlaub, werde jedoch per Mail erreichbar sein und kann auf auftretende Fragen antworten.

Die betreffenden Arbeitspakete pro Person könnt ihr aus der Liste auslesen. Bitte haltet euch bei der Umsetzung an das UML und schreibt mir wenn(am besten bevor) ihr Änderungen an diesem machen müsst.

Bitte sprecht euch mit eurem jeweiligen Mitarbeiter ab, wer welchen Teil übernimmt. Es gibt pro Server einen Hauptverantwortlichen der im Notfall das Sagen hat.

Dokumentiert bitte eure Vorgehensweisen, damit wir später ein gutes Protokoll haben.

**Wichtig:** Da wir vom Borko die Tests durchgeführt bekommen, muss die Funktionalität bis zum 17. Gegeben sein! Sonst haben wir nichts von unserem Vorteil. Unit-Tests bitte auch beachten, können aber bis zum 20. Fertiggestellt werden. Und: Ihr dürft ruhig mehr machen, dann sind wir halt schneller fertig^^

**Gewünschter Fortschritt bis zur Deadline:**

Management-Client: (Lipovits)

* Usereingaben werden erkannt (alle geforderten Befehle) und auf Syntax kontrolliert (Trockene Ausgabe ohne Senden)
* Bei falschen Eingaben wird eine **eigene** Exception geworden (außer nicht beötigt)

Load-Testing Component: (Lipovits)

* Properties aus dem File lesen und entsprechend speichern
* Liste an Clients (Package Client aus altem src) erstellen
* Implementieren Fake\_Cli (Stichwort InputStream)

BillingServer + Secure (Krepela)

* Erstellen von PriceSteps
* Löschen von PriceSteps
* Anzeigen der PriceSteps schön formatiert
* Anmelden mittels Username + pwd, auslesen aus Property-File (siehe Angabe); Vermittlung muss noch nicht erfolgen
* Rechnung erstellen lassen

AnalyticsServer (Reichmann)

* Alle Events als Models erstellt und entsprechende HashMap angelegt
* Dokumentieren der Abhängigkeiten (bei welchem Eingehenden Event werden welche erzeugt)
* Berechnung von neuen Events
* Passende Events zu einem Regex finden
* Überlegen von Notifications zu einem Event

RMI-Verbindungen (Traxler)

* Alle Interfaces definiert
* Stubs implementiert
* Sozusagen fertig um die Componenten nurnoch verbinden zu müssen

Alter Server: (Traxler/Reichmann)

* ArrayList durch HashMap ersetzen
* Eigene Exceptions definieren
* ThreadPool einsetzen
* Ev TimerService

**Diagramme:** (Reichmann)

* Kein Schatten
* Include bei Use-Case entfernen
* Sichtbarer machen

**Mfg Daniel**

P.S.: Wenn euch die Aufgaben zu viel sind, dann bitte ich um Rückmeldung, damit ich das eine oder andere Arbeitspaket überdenken kann.

Hallo liebes Team,

wie ihr wisst, müssen wir nächste Woche mit unserer Aufgabe fertig sein,

da der liebe Herr Professor Borko uns die Tests abnimmt. Ich muss ihm

nurnoch seine Arbeitspakete senden (überlegen wir am Freitag)

Was heißt das für uns?

Jeder muss seinen Teil am Sonntag(! 16.02.14) fertig haben. Wie ich

gesehen habe, fehlt euch allen nichtmehr viel und das sollte auf jeden

Fall machbar sein. (AnalyseServer is am wenigsten weit, aber den muss

sowieso ich auch machen).

Montag schau ich mir den Fortschritt an und teste euren Code.

Thomas: Bereite die Interfaces so vor, dass es funktioniert, wenn man

die Teile zusammenfügt.

Nanak: Du übernimmst erstmal den FileHandler, ich überleg mir was zum

AnalyseServer und sag dir wenn ich etwas brauch.

Es ist gut, wenn ihr vorher fertig seid, dann haben wir noch Zeit zum

Debuggen.

Jeder schreibt bitte seine Dokumentation für seinen Part. Testing kann

in der letzten Woche erfolgen, aber auch nicht bis irgendwann sondern

rechtzeitig.

Das build.xml werde ich übernehmen

mfg

Daniel

P.S.: Ich bitte um Rückmeldung und schreibt mich bitte an, wenn ihr

Probleme jeglicher Art habt oder Hilfe braucht. Fragen bez. UML auch an

Traxler/Krepela weiterleiten.

Liebes Team,

da bis gestern alle Eigenfunktionalitäten fertigzustellen waren, möchte

ich bitte von euch bis morgen, dass die Funktionalitäten zusammengefügt

werden.

Das bedeutet für Traxler, dass er die RMI-Objekte überall in die

Registry schreiben muss, und für alle Server + MgmtClient verfügbar

machen muss.

Bitte bis morgen 19:00.

Setzt euch morgen in NWSY/Deutsch zusammen und schaut, dass ihr dann

etwaige Fehler gemeinsam Lösen könnt. Am Abend werde ich mir das ganze

ansehen und testen.

mfg,

Daniel