Software Engineering Übung, LVNr: 050052
Gruppe 7, Hotelreservierung 2
Übungsleiter: Hans Moritsch

Designmodell v.1.0

Projekt Grasshopper

Projektteam

Matrikelnummer	Nachname	Vorname	E-Mail-Adresse
1201759	Anreiter	Simon	Anreiter.Simon@gmail.com
0302840	Kocman	Andreas	a0302840@unet.univie.ac.at
1200628	Moser	Victoria	vmoser@gmail.com

Projekthomepage
http://unet.univie.ac.at/~a1200628/

Git-Repository
https://github.com/swe2013hotel2/grashopper

Aufgabenstellung:

Erstellen sie ein Designmodell gemäß *Unified Process* das zumindest folgende Aspekte umfasst:

- Klassendesign
- Use-Case-Realization-Design
- Übersichtsklassendiagramm
- Architekturbeschreibungen

1 Klassendesign

Beschreiben sie die wichtigsten Klassen und spezifizieren sie die öffentlichen Methoden und Attribute. Begründen sie die grundlegenden Designentscheidungen.

LocationDAO:

- speichert und lest Locations
- zu den Locations gehören Städte und Hotels
- ist das Interface welches sich um die Speicherung von Location kümmert

```
ArrayList<Hotel> getHotelList( );
```

- gibt eine Liste von allen Hotels zurück.

```
ArrayList<City> getCityList( );
```

- gibt eine Liste von allen Städten zurück.

```
Location getLocationByName(String locationname, boolean city);
```

- man kann ein Hotel oder Stadt nach Namen suchen
- wenn man nach einer Stadt sucht, muss man den boolean Wert true für city übergeben
- wenn man nach einem Hotel sucht, muss man den boolean Wert false für city übergeben

```
void saveLocation(Location location);
void deleteLocation(Location location);
void updateLocation(Location location);
```

SqlLocationDAO:

- Implementiert Location DAO und speichert die Daten in eine SQL Datenbank

```
public ArrayList<Location> getLocationList() { }
public void deleteLocation(Location location) { }
public void updateLocation(Location location) { }
public Location getLocationByName(String locationname, boolean
city) { }
public void saveLocation(Location location) {}
```

UserDAO:

- Interface welches alle user daten speichert und lest

```
ArrayList<User> getUserList() throws
  IllegalArgumentException;
```

- liefert eine Liste von allen Usern

```
User getUserbyUsername(String username);
void saveUser(User user) throws IllegalArgumentException;
void deleteUser(User user) throws IllegalArgumentException;
void updateUser(User user) throws IllegalArgumentException;
```

SqlUserDAO:

- Implementiert UserDAO und speichert die Daten in eine SQL Datenbank

```
public ArrayList<User> getUserList() throws
  IllegalArgumentException { }
public User getUserbyUsername(String username) { }
public void saveUser(User user) throws
  IllegalArgumentException { }
public void deleteUser(User user) throws
  IllegalArgumentException { }
public void updateUser(User user) throws
  IllegalArgumentException { }
```

Bookings:

- Buchungen für einen bestimmten Raum
- Es wird angegeben wer die Buchung durchführt und das Datum von-bis
- Buchungen werden nach Datum sortiert und in der SQL Datenbank gespeichert
- Jeder Raum hat eine ArrayList wo die Buchungen für sich gespeichert werden

```
void bookForTimeFrame(Customer customer, Date beginDate, Date
  endDate){_}

boolean freeForTimeFrame(Date beginDate, Date endDate){ }
```

City:

- extends Location
- Jede Stadt hat ein Land zu welchem es gehört
- Hotels welche in einer Stadt sind werden in einem Array gespeichert

```
public void addHotel(Hotel hotel){ }
public void removeHotel(Hotel hotel){ }
public ArrayList<Hotel> getHotels(){ }
```

Hotel:

- extends Location
- Jedes Hotel hat Räume welche in einem ArrayList gespeichert werden

```
public ArrayList<Room> getRooms(){ }
public void bookRoomForTimeFrame(Customer customer,Room room,
  Date beginDate, Date endDate){ }
```

Location:

- Abstrakte Überklasse von Hotel und City

Review:

- Objektwertig
- Die Reviews werden direkt in die SQL Datenbank gespeichert
- Man kann für jede Location, Hotel oder Stadt kann man alle Reviews suchen
- Beim speichern kann jeder User zu einer Location immer nur ein Review schreiben
- Sollte ein User zu einer Location ein erneutes Review schreiben, wird das vorherige überschrieben
- Bei Kunden wird für das Hotelreview vorher überprüft, ob eine Buchung für diesen Kunden existiert

```
public static void saveNewReview(User creator, Location location,
  int stars, String reviewText){ }
public static ArrayList<Review> getReviewsForLocation(Location
  location){ }
```

Statistik:

- Statistik hat 3 Instanzvariablen: Beginn- EndDatum der Statistik und ein Array
- Im Array wird gespeichert, wieviele Buchungen an dem Tag stattgefunden haben
- Statistiken werden jeweils für bestimmte Tage erstellt
- Die Hotelstatistik wird aus den Statistiken der Räume dieses Hotels erstellt

- Die Stadtstatistiken werden aus den Statistiken der Hotels welche sich in dieser Stadt befinden erstellt

```
private void mergeWithStatistic(Statistic statistic){ }
```

- mit dieser Methode kann man zwei Statistiken zusammenfügen
- Das EndDatum und EndDatum werden angepasst und die Buchungen werden aufsummiert

```
public Statistic statisticForHotel(Hotel hotel){ }
public Statistic statisticForCity(City city){ }
public Statistic statisticForRoom(Room room){ }
```

HotellierManagement:

- extends Usermanagement

```
public ArrayList<Bookings> bookingsForAssignedHotel(){ }
```

- für sein eigenes Hotel alle Buchungen einsehen

```
public void createHotel(String name, int oneBedRooms, int
  twoBedRooms){_}
```

- beim erstellen vom Hotel müssen Anzahl von Einbett- und Zweibett Zimmern angegeben werden
- Hotel Ids werden automatisch vergeben
- Sollte der Hotellier bereits ein Hotel besitzen wird das alte überschrieben

```
public void editOwnHotel(String name, int oneBedRooms, int
  twoBedRooms){ }
```

- wenn man als Hotellier eingeloggt ist hat man die Möglichkeit das Hotel welches man besitzt zu bearbeiten

```
public Statistic getStatisticForOwnHotel(){
      return null;}
```

CustomerManagement

- extends Usermanagement

```
public void reviewHotel(Hotel hotel, int stars, String
  reviewText){ }
```

- überprüft ob Kunde mind. Ein mal gebucht hat

```
public void bookRoomForTimeFrame(Hotel hotel, Room room,
  Date beginDate, Date endDate){ }
```

Session

- Gibt die Pages an den Browser weiter

```
public void loginAction(String email, String password){ }
```

- User wird gesucht und entsprechendes Usermanagement (Customer, TA, Hotellier) wird gesetzt

```
public void logoutAction(){ }
public void searchAction(/*params*/){ }
public void editAction(){ }
```

TAManagement

- extends Usermanagement

```
public void reviewAssignedCity(int stars, String
  reviewText){ }
```

- erlaubt dem Tourismusverband Reviews für die zugehörige Stadt zu erstellen

```
public Review getReviewForAssignedCity(){ }
```

- erlaubt dem Tourismusverband das erstellte Review für die dazugehörige Stadt anzusehen

```
public void editAssignedCityReview(int stars, String
  reviewText){ }
```

- erlaubt dem Tourismusverband ein existierendes Review zu ändern

```
public Statistic statisticForAssignedCity(){ }
```

- erlaubt dem Tourismusverand die Statistik der zugehörigen Stadt einzusehen

Usermanagement

- Sind die Funktionen die alle User ausführen können

```
ArrayList<Hotel>getHotelsWithFreeRoomsWithParameters{ }
```

- Nach Hotels suchen die den suchparametern entsprechen und

freie Räume haben

```
Room getFreeRoomWithParameters{ }
```
- Nach Räumen in einem Hotel suchen die den suchparametern entsprechen

```
int getNumberOfFreeRoomsWithParameters{ }
```
- Gibt die Anzahl der freien Räme zurück die den Suchparametern entsprechen

```
ArrayList<Review> getReviewsForHotel(Hotel hotel){ }
```
- Gibt alle Reviews für ein Hotel zurück

```
public void editOwnAccount(/*params*/){ }
```
- Eigenen Account bearbeiten(falls vorhanden)

```
public void login(/*login*/){ }
```
- Einloggen (UserAccount erforderlich)

```
public void logout(){ }
```
- Ausloggen

```
public void deleteOwnAccount(){ }
```
- Eigenen Account löschen (UserAccount erforderlich)

```
public void registerUser(User user){ }
```
- Neuen Account erstellen und in Datenbank speichern

Customer
- extends user

Hotellier
- extends user

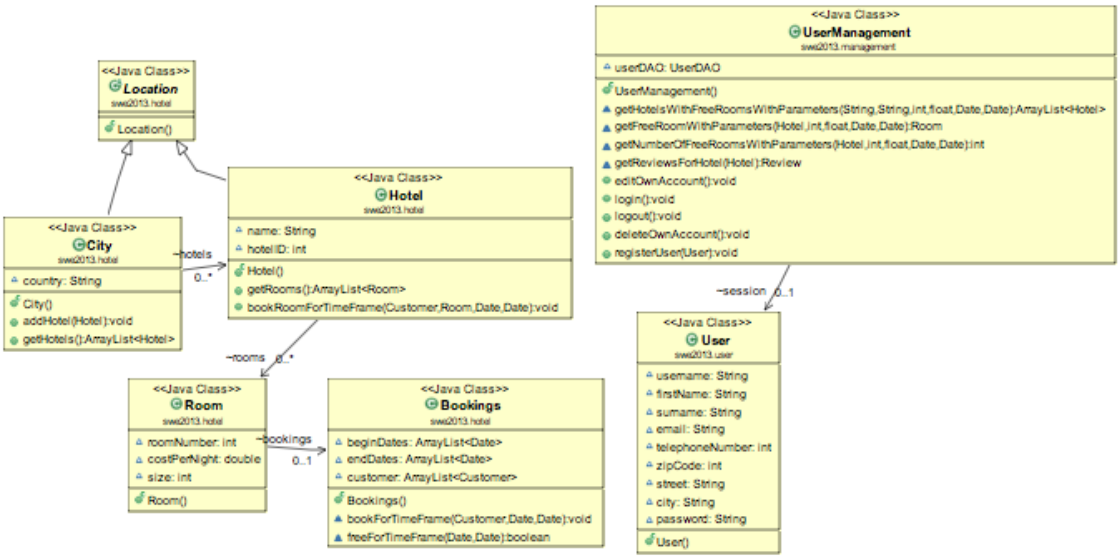
TourismAssosication
- extends user

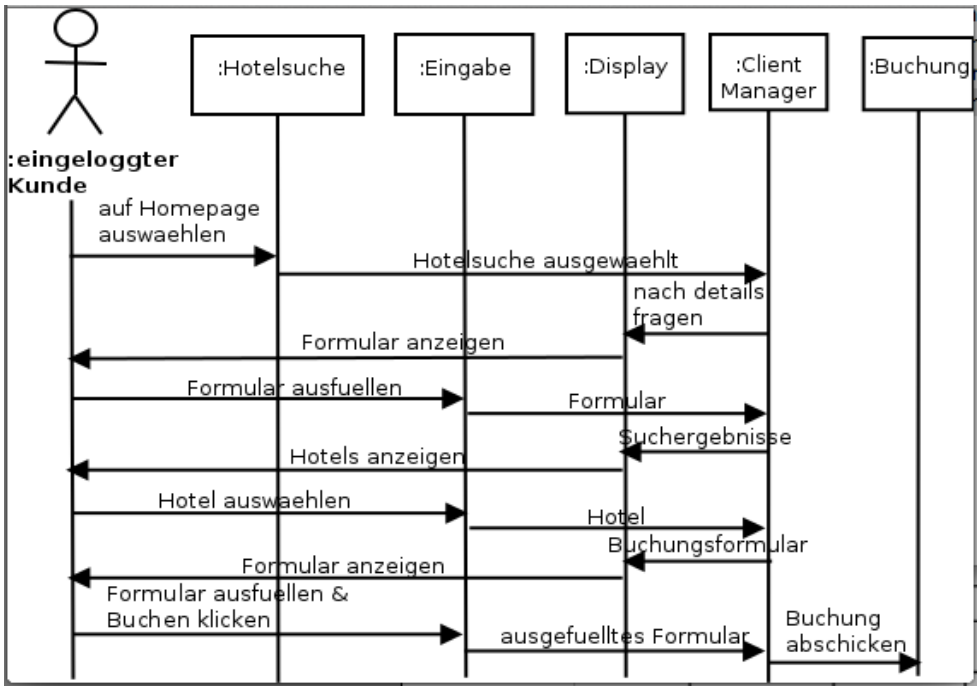
User
- public abstract class;

2 Use Case Realization Design

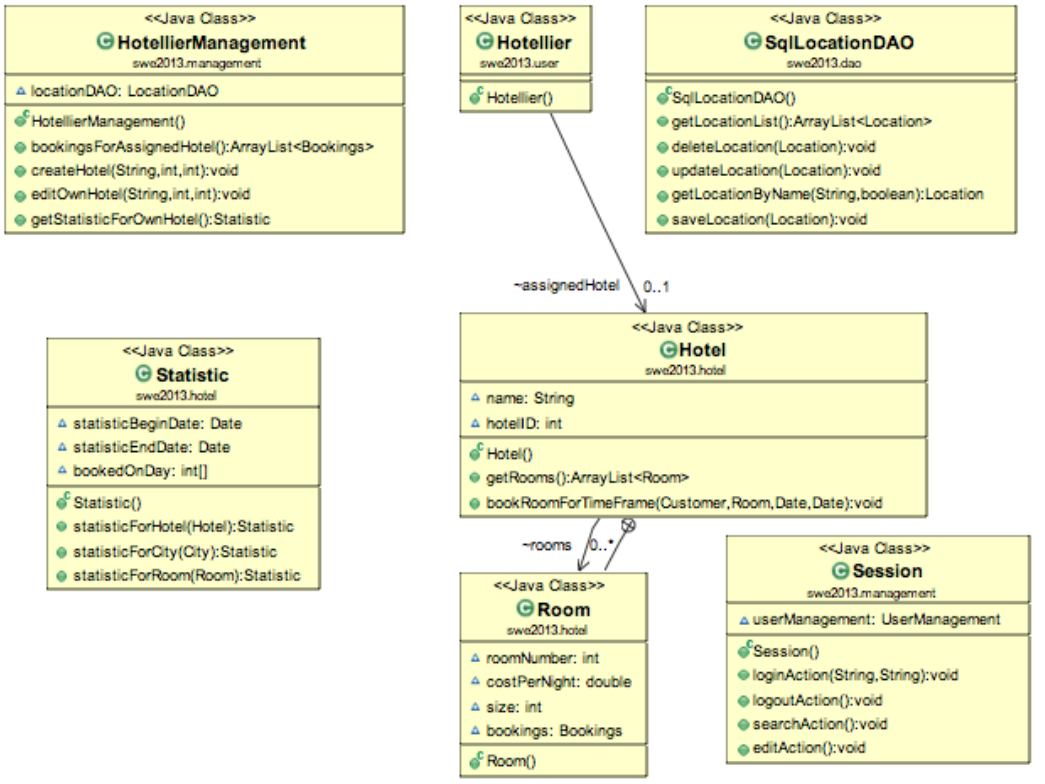
Modellieren sie, wie die zentralen Use Cases ihres Systems mittels Designklassen realisiert werden. Modellieren sie dabei die wesentlichen statischen und dynamischen Aspekte jedes Use Cases mit geeigneten Klassendiagrammen und Sequenzdiagrammen.

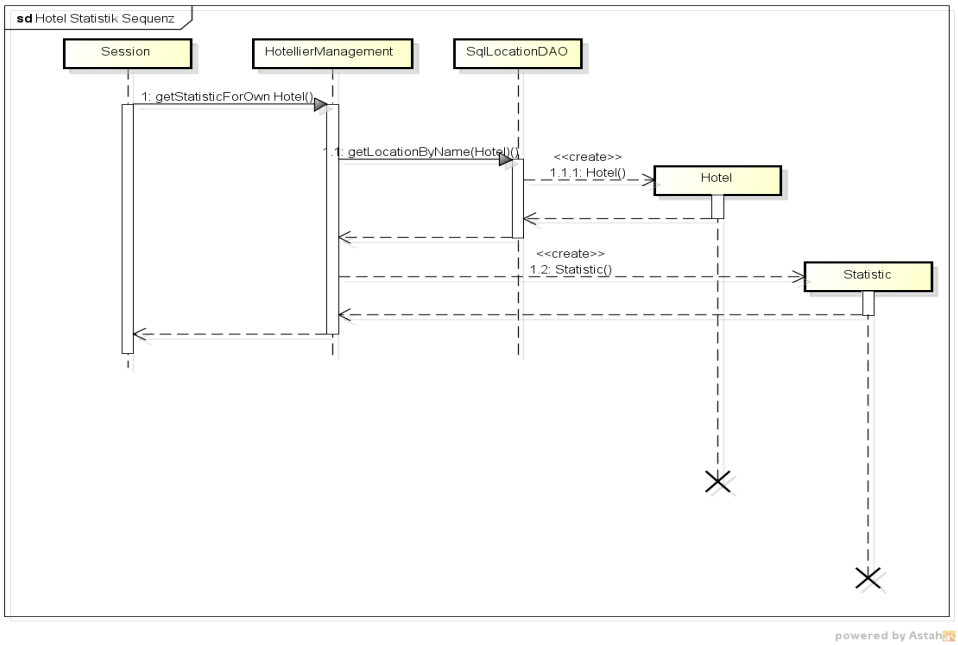
2.1 Buchung



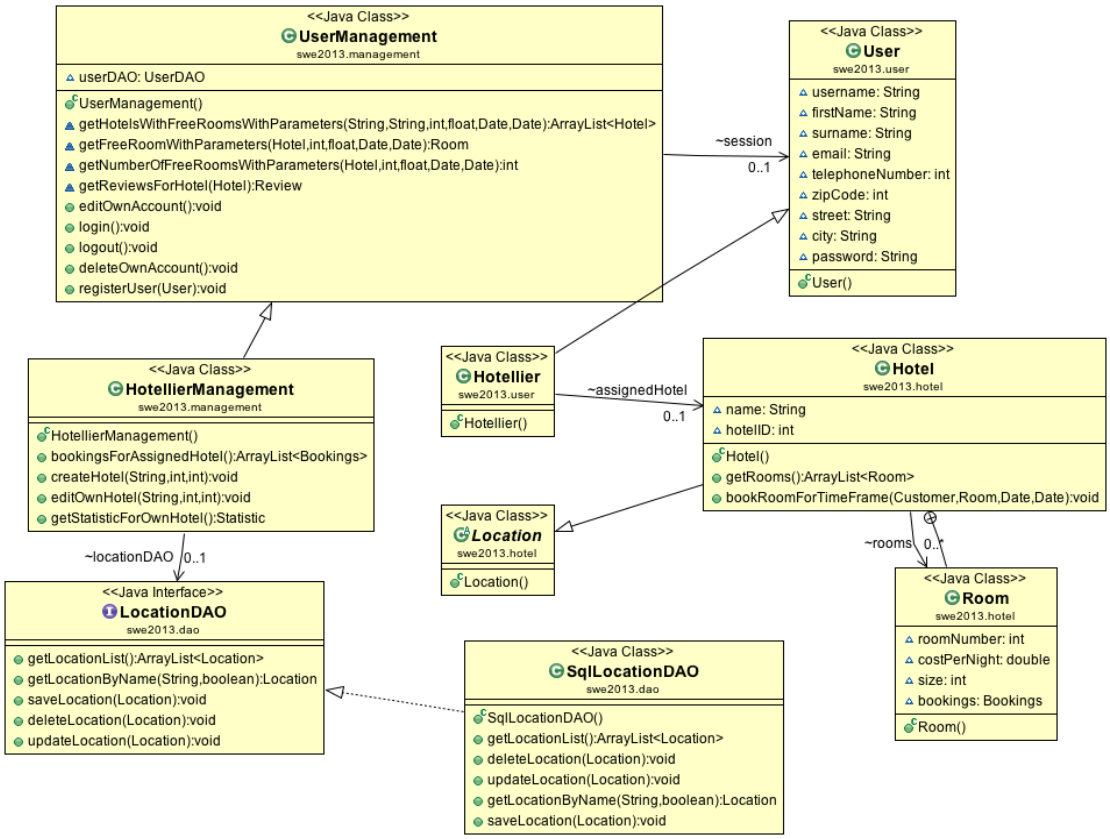


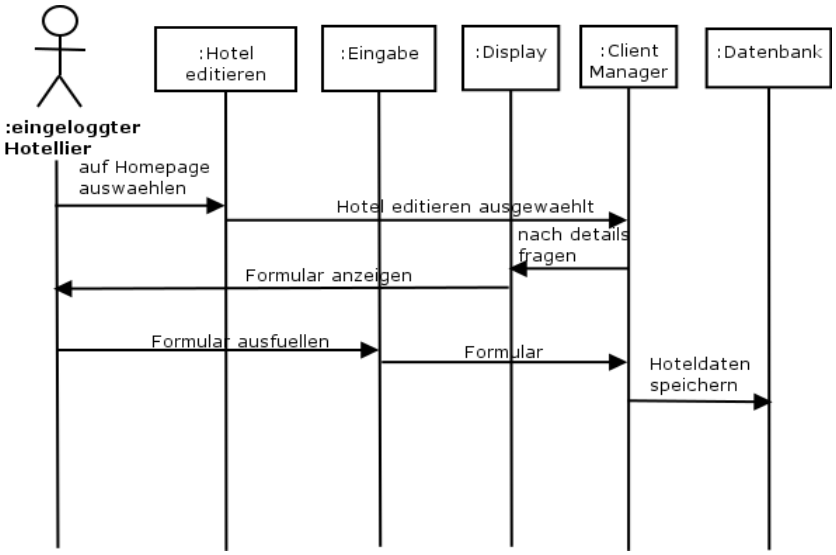
2.2 Statistik einsehen Hotellier



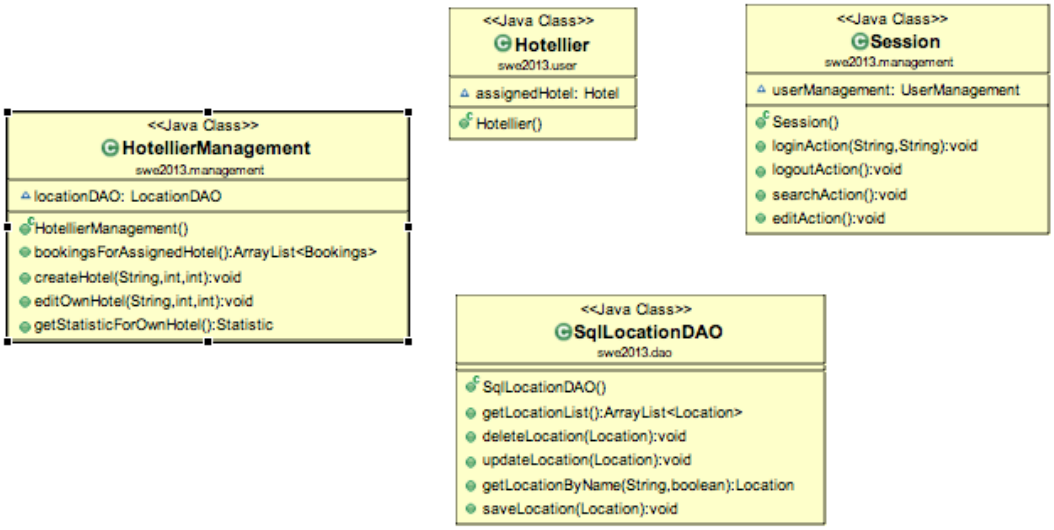


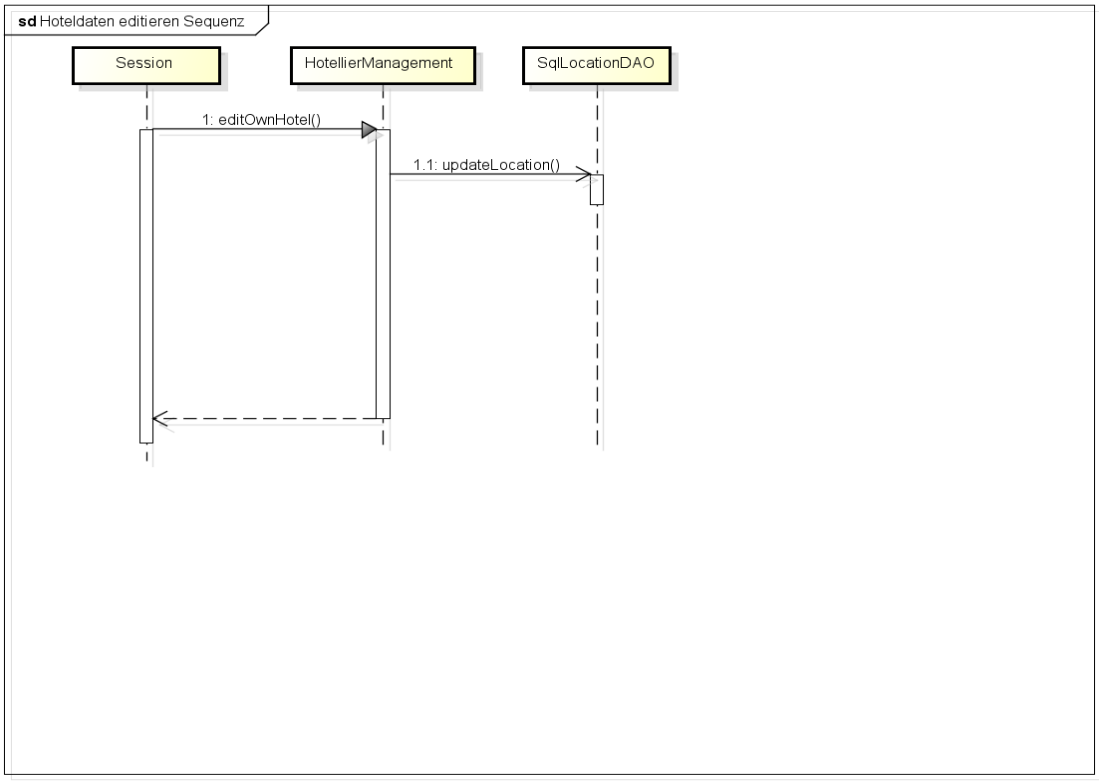
2.3 Hotelübersicht editieren



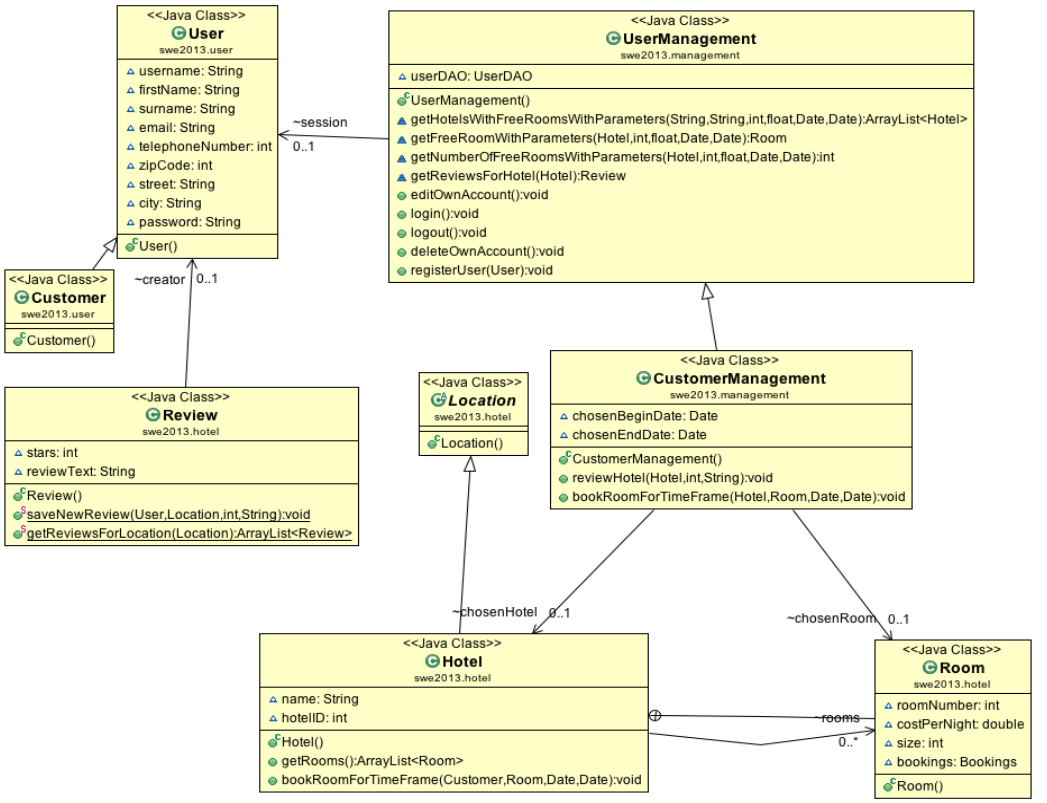


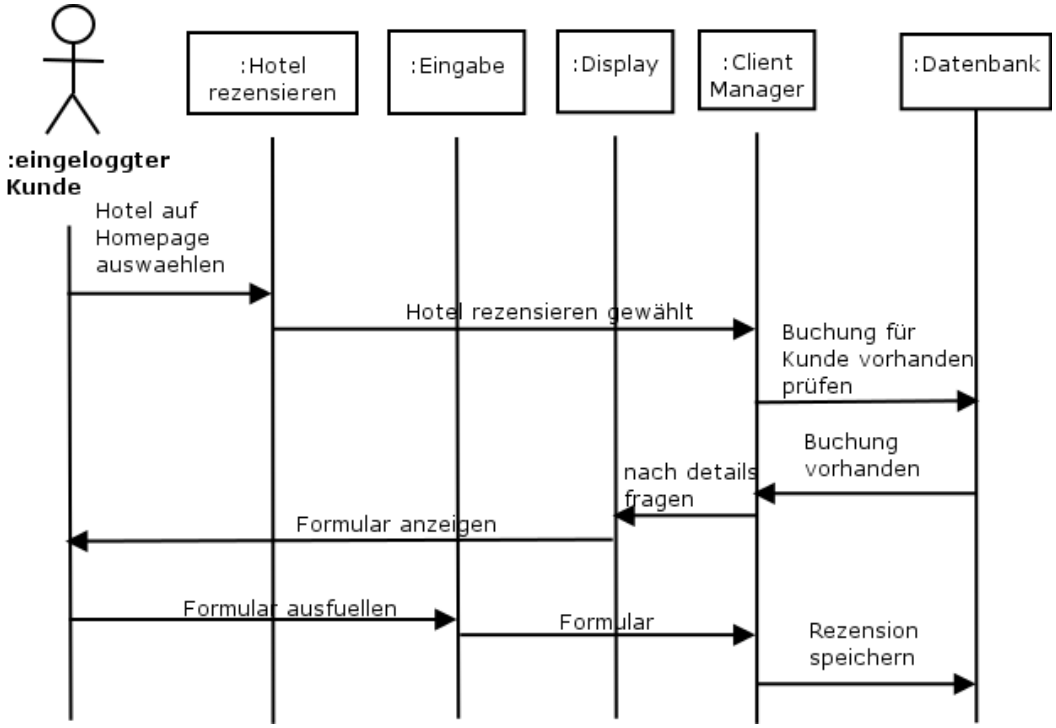
2.4 Hoteldaten editieren



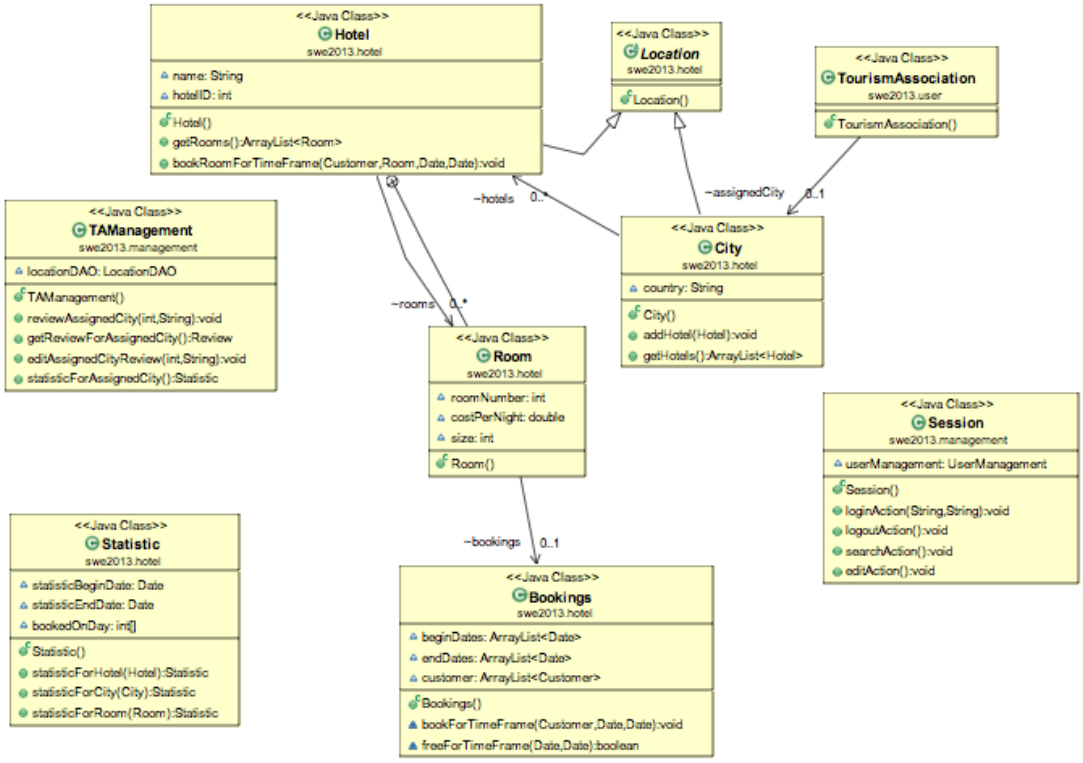


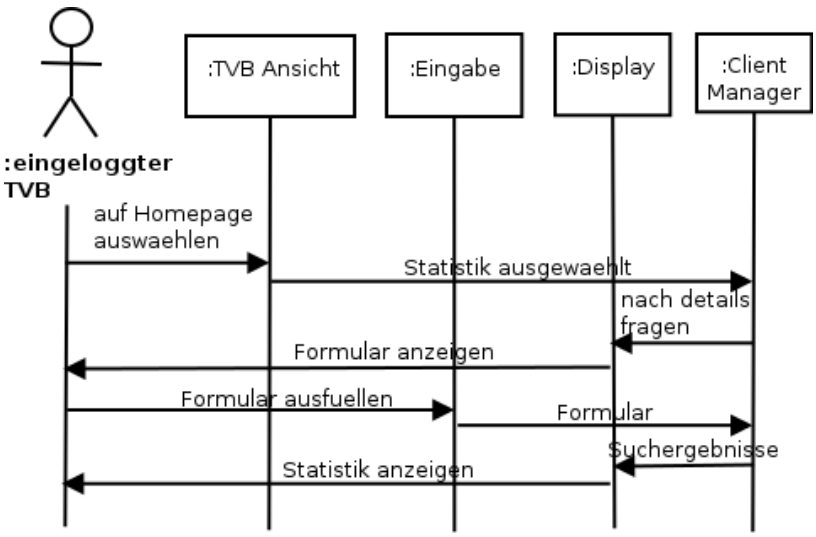
2.5 Rezension schreiben





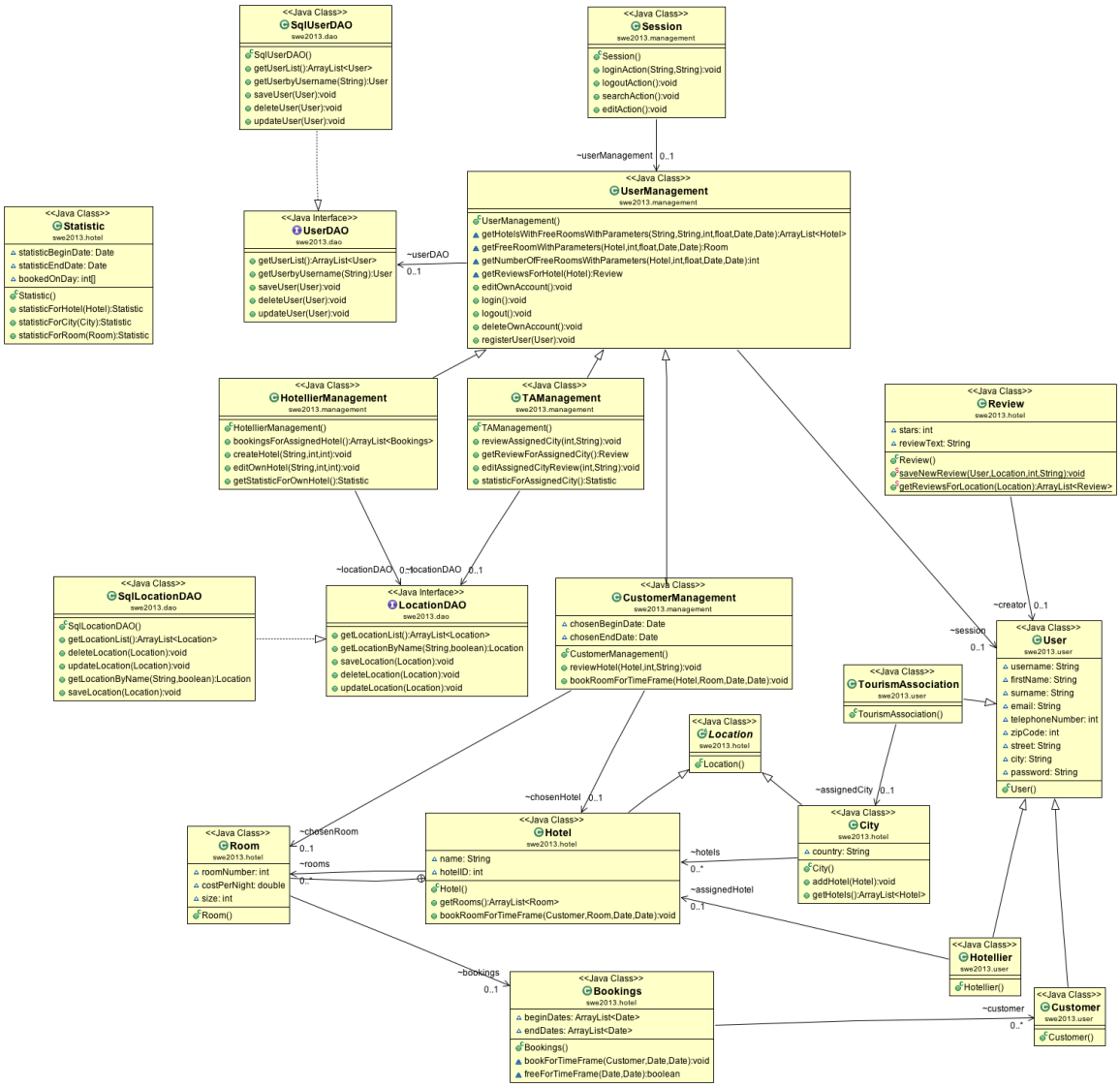
2.6 Statistik einsehen TVB





3 Übersichtsklassendiagramm

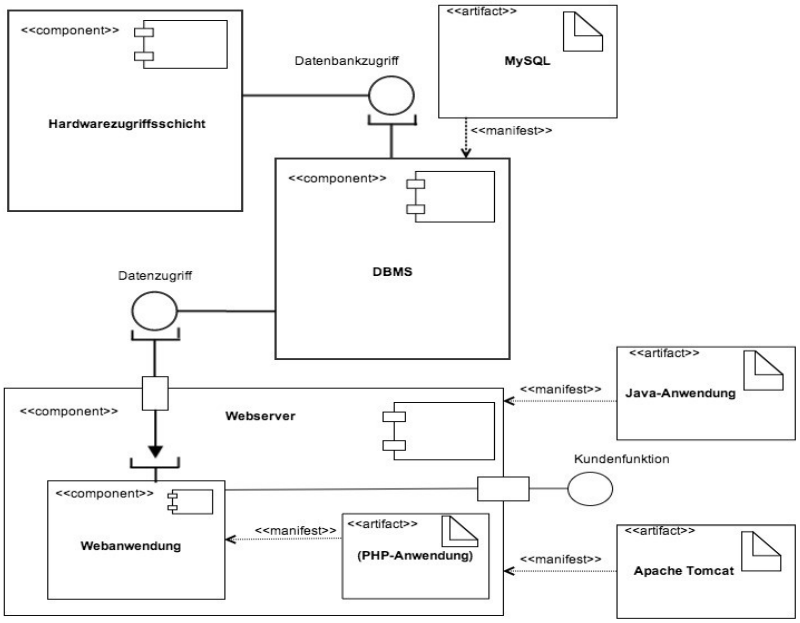
Erstellen sie ein Klassendiagram das *alle* wesentlichen Klassen ihres Systems und deren Beziehungen möglichst übersichtlich darstellt.



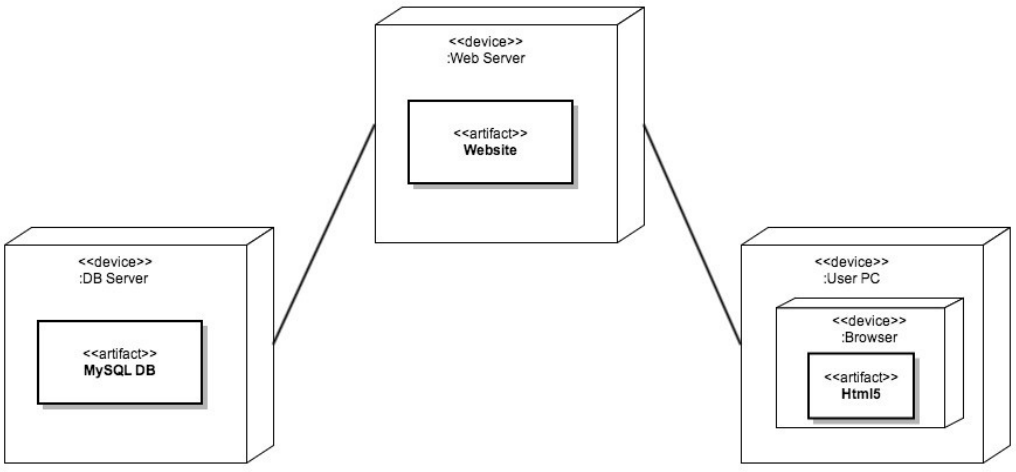
4 Architekturbeschreibung

Beschreiben sie die Architektur ihres Systems mittels Komponentendiagrammen und Deploymentdiagrammen.

Komponentendiagramm:



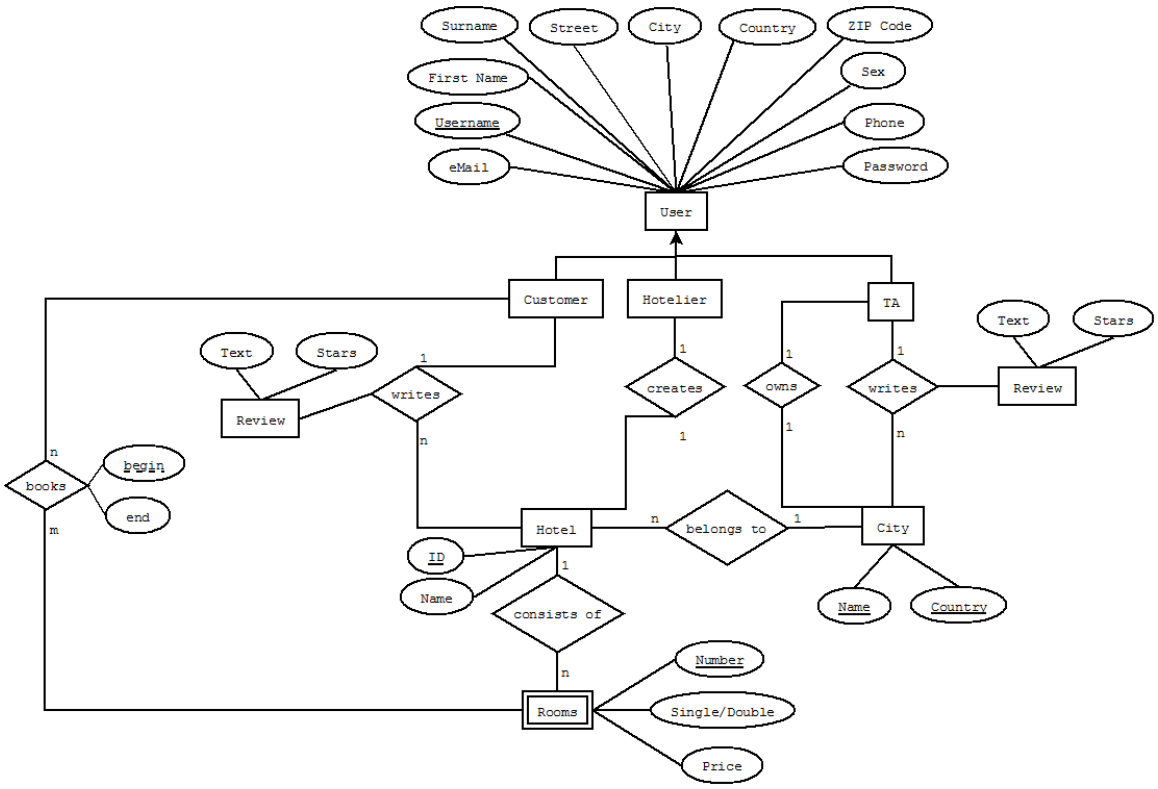
Deploymentdiagramm:



5 Datenspeicherung

User, Hotel und Buchungsdaten werden im Rahmen unserer Aufgabenstellung persistent mittels MySQL in einer Datenbank gespeichert. Die nötige Infrastruktur wird durch den ZID zur Verfügung gestellt (http://zid.univie.ac.at/mysql/).

Die exakte Datenbankstruktur soll durch das folgende Entity-Relationship Diagramm veranschaulicht werden:



Programmintern werden die Zugriffe auf die SQL Datenbank im Rahmen der Klassen SqlUserDAO und SqlLocationDAO mittels der *Java Database Connectivity (JDBC) API* Version 3.0 realisiert.