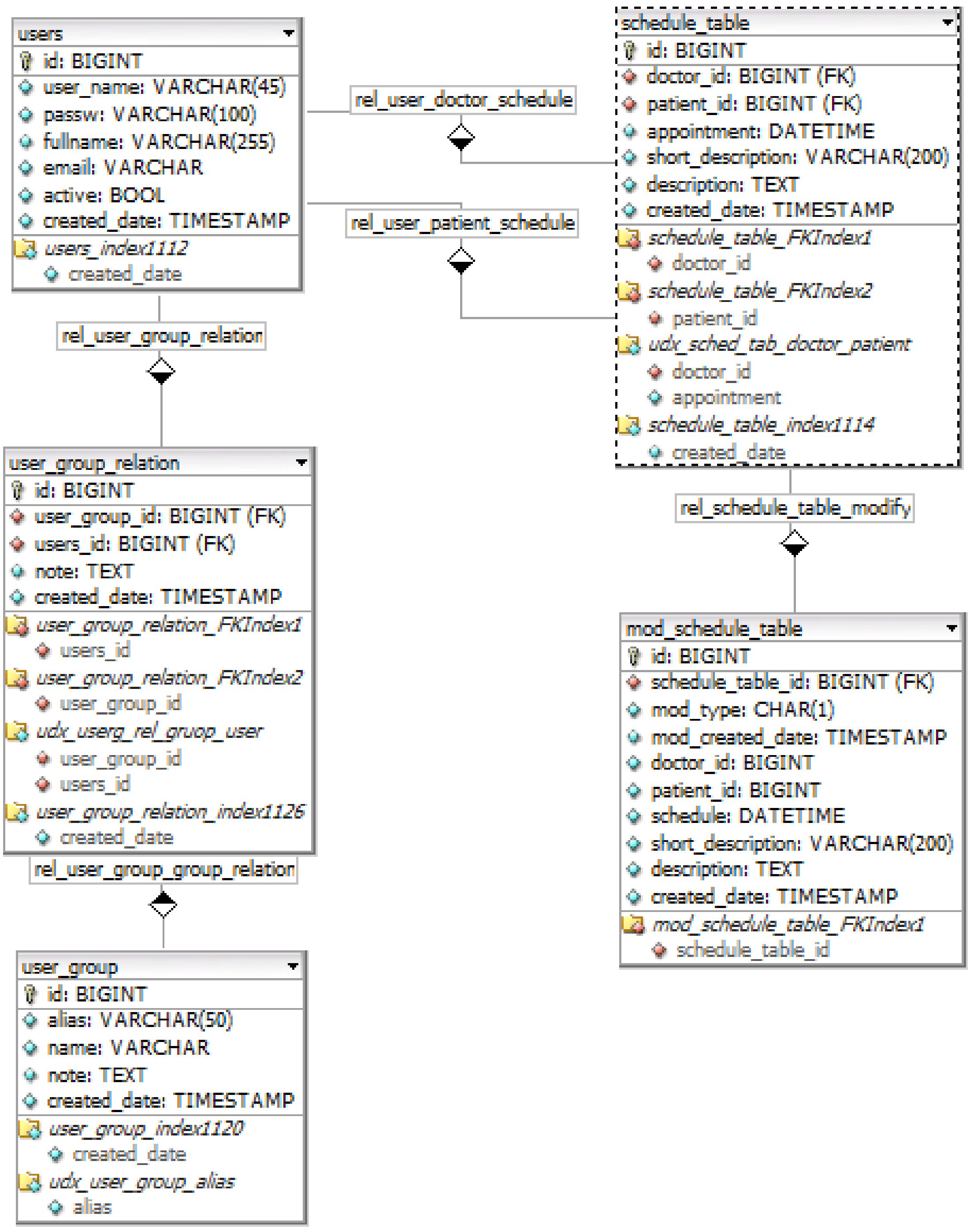
Patient Management System – Schedule feature

This documentation will represent the new feature. Each doctor has a calendar. The doctor has only one patient at a time. The system must be able to that the doctor can change the schedule or appointment if he wants’.

The Database structure

This chapter contains the details of database table. The full database structure represents the 1. figure. There are two columns where all tables are included.
The first columns name is id. This is a primary key of table and auto incrementing. This is the unique key.
The another column’s name is created_date. This contains creating time of rows. This column has got default value, so we don’t have to manipulate manually.



1. figure

Users table

This table contains all patients and all doctors. Currently, the table included just a few columns. These columns are the most important. Of course, this table can be expandable.

Columns:

- **user_name:** this is a user's unique name
- **passw:** this is the user's password. The content has to be encoded. e.g.: with SHA-1 or MD5 hash function
- **fullname:** first and last name
- **email:** user's email address
- **active(false):** It is possible to disable the user. false: disable, true: enable

User_group table

This table contains all of the users groups. These groups represent the permissions and that what kind of functions are available. e.g.: User is a doctor or patient. The doctor is able to create new schedule.

Columns:

- **alias:** This is a unique string that clearly identifies a row. This is a simpler use in program code like id.
- **name:** the group name
- **note:** description of group

User_group_relation table

In this table it can be defined that the user is a doctor or a patient. Of course, another relation of group can be creating, too. The group can contain more than one user.

Columns:

- **user_group_id:** unique id of row from user_group table
- **user_id:** unique id of row from user table
- **note:** description of relation

Schedule table

This is a schedule table. Here can be recorded when the doctor will meet with patient.

Columns:

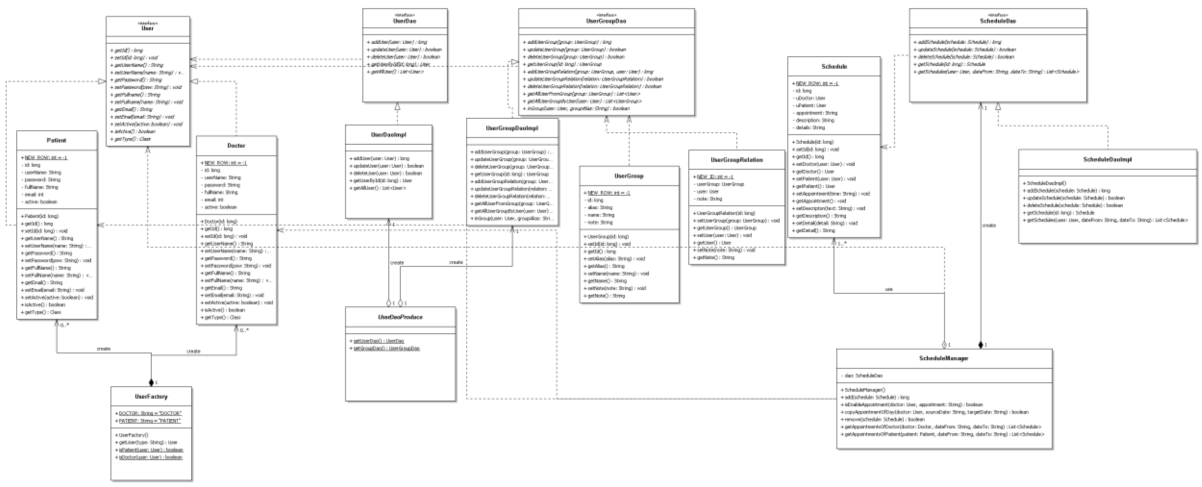
- **doctor_id:** unique id of row from user table
- **patient_id:** unique id of row from user table
- **appointment:** date of appointment
- **short_description:** short description, keywords
- **description:** thorough details about meeting

Mod_schedule table

This table contains all modifications of main table(schedule_table). So we can restore or recheck previous data, that changed by doctors. We load contents with a trigger, so don't have to manipulate with manual.

Java Class Design

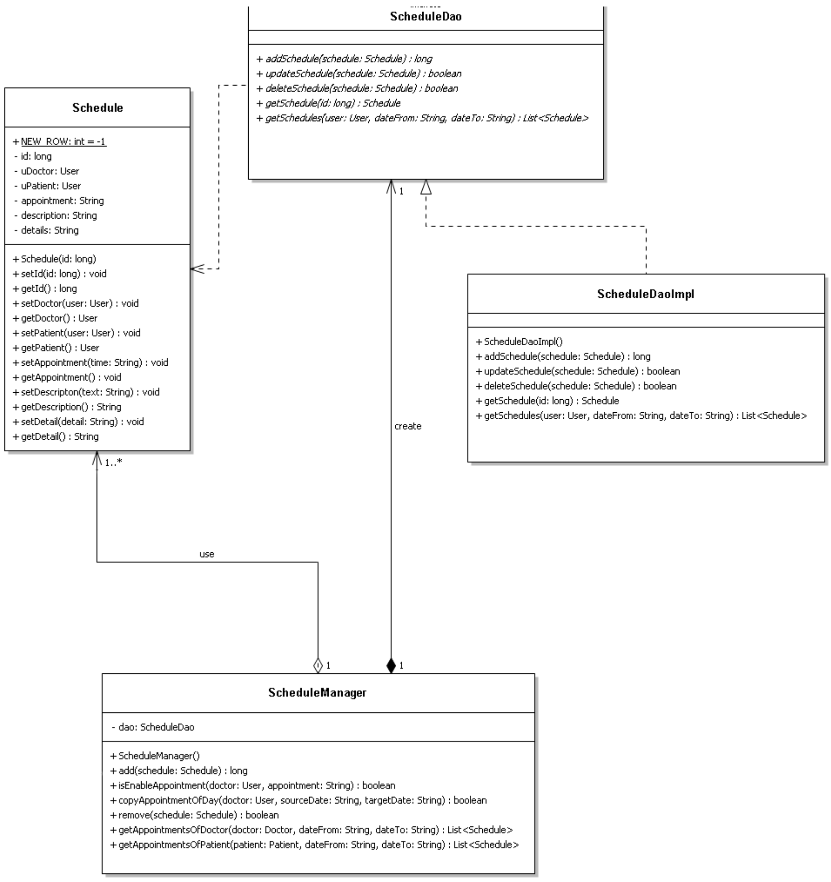
This picture represent the full java design which is important for the new feature.



2. figure

Schedule design

This chapter deals with the schedule manager. We have an interface which name is *ScheduleDao*. The *ScheduleDaoImpl* class implements this interface. This class uses the *Schedule* model class which will pass information to *ScheduleDaoImpl* object to get data it needs. For managing of data we use the *ScheduleManager* class. Through the manager class we can reach the creator, the updater, controller and copy functions.


ScheduleDao
+ addSchedule(schedule: Schedule) : long
+ updateSchedule(schedule: Schedule) : boolean
+ deleteSchedule(schedule: Schedule) : boolean
+ getSchedule(id: long) : Schedule
+ getSchedules(user: User, dateFrom: String, dateTo: String) : List<Schedule>
Schedule
+ NEW_ROW: int = -1
- id: long
- uDoctor: User
- uPatient: User
- appointment: String
- description: String
- details: String
+ Schedule(id: long)
+ setId(id: long) : void
+ getId() : long
+ setDoctor(user: User) : void
+ getDoctor() : User
+ setPatient(user: User) : void
+ getPatient() : User
+ setAppointment(time: String) : void
+ getAppointment() : void
+ setDescripton(text: String) : void
+ getDescription() : String
+ setDetail(detail: String) : void
+ getDetail() : String
ScheduleDaoImpl
+ ScheduleDaoImpl()
+ addSchedule(schedule: Schedule) : long
+ updateSchedule(schedule: Schedule) : boolean
+ deleteSchedule(schedule: Schedule) : boolean
+ getSchedule(id: long) : Schedule
+ getSchedules(user: User, dateFrom: String, dateTo: String) : List<Schedule>
1
create
1..*
use
1
1
ScheduleManager
- dao: ScheduleDao
+ ScheduleManager()
+ add(schedule: Schedule) : long
+ isEnableAppointment(doctor: User, appointment: String) : boolean
+ copyAppointmentOfDay(doctor: User, sourceDate: String, targetDate: String) : boolean
+ remove(schedule: Schedule) : boolean
+ getAppointmentsOfDoctor(doctor: Doctor, dateFrom: String, dateTo: String) : List<Schedule>
+ getAppointmentsOfPatient(patient: Patient, dateFrom: String, dateTo: String) : List<Schedule>

3. figure

Schedule class

This object is simple POJO containing get/set methods. This object is going to work as a Model.

<u>Schedule(long id)</u>

Create new instance.

<u>Parameters:</u>

- id(long): identification of one schedule. If the value is -1 then it means that the set data will be new data. This isn’t in database, yet.

ScheduleDao interface

This interface defines the standard operations to be performed on a model object(s).

<u>addSchedule(Schedule schedule): long</u>

Insert new schedule into database.

<u>Parameters:</u>

- schedule(Schedule): Object.Schedule is simple POJO object.

<u>Return:</u>

This is new row id.

<u>updateSchedule(Schedule schedule): Boolean</u>

Update one row in database.

<u>Parameters:</u>

- schedule(Schedule): Object.Schedule is simple POJO object. This object contains the data which we have to modify.

<u>Return:</u>

Returns true for success. Otherwise return false.

<u>deleteSchedule(Schedule schedule): Boolean</u>

Delete one row in database.

<u>Parameters:</u>

- schedule(Schedule): Object.Schedule is simple POJO object. This object contains the data which we have to delete.

<u>Return:</u>

Returns true for success. Otherwise return false.

getSchedule(long id): Schedule

Get one row from database.

Parameters:

- id(long): the row unique identification

Return:

Object.Schedule is simple POJO object.

getSchedules(User user, String dateFrom, String dateTo): List<Schedule>

Get more rows from database. These rows contain the schedule of the doctor between two dates.

Parameters:

- user(User): Object.User is simple POJO object. This object contains the data of user whose schedule is necessary.
- dateFrom(String): begin of filter date. If this is null then the begin date is open.
- dateTo(String): end of filter date. If this is null then the end date is open.

Return:

Object.List object that it is contain Schedule objects.

ScheduleDaoImpl class

This class implements above interface. This class is responsible to get data from a data source.

ScheduleManager class

This class manages scheduling processes. Through the object we can add new schedule, change and checking the schedule time that can be inserted under different conditions.

add(Schedule schedule): long

Add new schedule into database.

Parameters:

- schedule(Schedule): Object.Schedule is simple POJO object. If in this object the value of id is -1 then it will be inserted otherwise it will be updated.

Return:

New or updated row id.

isEnableAppointment (Schedule schedule): boolean

This method checks that the schedule can be inserted or updated with different condition.

Parameters:

- schedule(Schedule): Object.Schedule is simple POJO object. If in this object value of id is -1 then it will be inserted otherwise it will be updated.

Return:

true if the schedule can be inserted or updated otherwise false.

copyAppointmentOfDay (User doctor, String sourceDate, String targetDate): boolean

A schedule is copyed to another day.

Parameters:

- doctor(User): Object.User is simple POJO object. The user whose data we would like to copy.
- sourceDate(String): the day of copy
- targetDate(String): target day

Return:

true for success otherwise false.

remove (Schedule schedule): boolean

The schedule is deleted from data source.

Parameters:

- schedule(Schedule): Object.Schedule is simple POJO object.

Return:

true for success otherwise false.

getAppointmentsOfDoctor (Doctor doctor, String dateFrom, String dateTo): List<Schedule>

Get more rows from data source. These rows contain the schedule of doctor between two dates.

Parameters:

- doctor(Doctor): Object.Doctor is simple POJO object. The user whose data we want to queried.
- dateFrom(String): begin of filter date. If this is null then the begin date is open.
- dateTo(String): end of filter date. If this is null then the end date is open.

Return:

Object.List object that it is contain Schedule objects.

getAppointmentsOfPatient (Patient patient, String dateFrom, String dateTo): List<Schedule>

Get more rows from data source. These rows contain the schedule of patient between two dates.

Parameters:

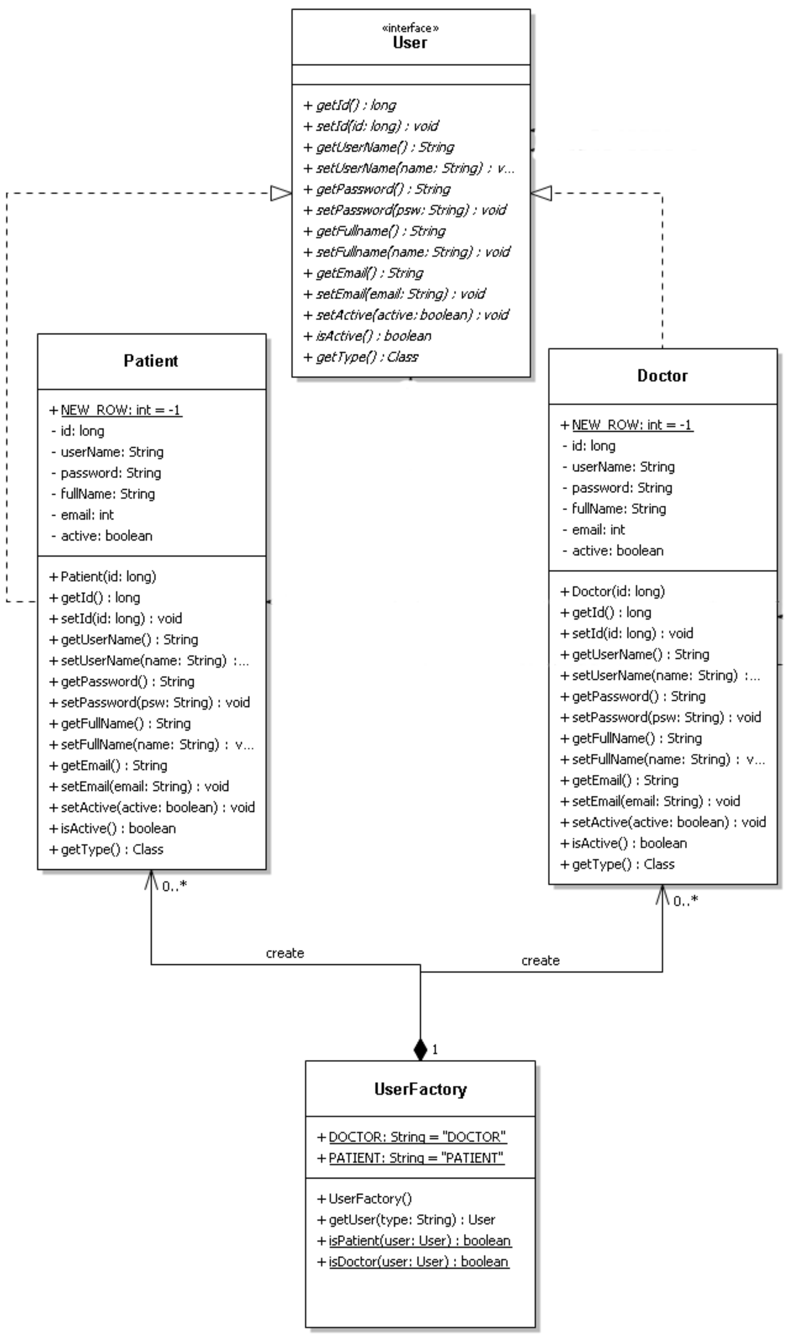
- patient(Patient): Object.Patient is simple POJO object. The user whose data we want to queried.
- dateFrom(String): begin of filter date. If this is null then the begin date is open.
- dateTo(String): end of filter date. If this is null then the end date is open.

Return:

Object.List object that it is contain Schedule objects.

User Factory design

We are going to create a *User* interface and the *Doctor* and *Patient* model classes implementing the *User* interface. Next step we create a factory class *UserFactory* to get a *User* object.



4. figure

User interface

This interface defines the standard operations to be performed on a Doctor or a Patient model object(s).

getType(): Class

This method returns the class type of Model object. So we can define that this user is a doctor or a patient.

Doctor class

This object is simple POJO containing get/set methods. This object is going to work as a Model.

<u>Doctor (long id)</u>

Create new model instance.

<u>Parameters:</u>

- id(long): Unique row id of user in database. If this id is -1 then this object contains data of new user. If the id is greater than -1 then this object contains data of existing user.

Patient class

This object is simple POJO containing get/set methods. This object is going to work as a Model.

<u>Patient (long id)</u>

Create new model instance.

<u>Parameters:</u>

- id(long): Unique row id of user in database. If this id is -1 then this object contains data of new user. If the id is greater than -1 then this object contains data of existing user.

UserFactory class

This class create new instance from *User* or *Doctor* Model class without exposing the creation logic to the client.

<u>getUser (String type): User</u>

Create new model object from Doctor or Patient class.

<u>Parameters:</u>

- type(String): type of object that is creating. For the value we can use the static variables of class, too: UserFactory.DOCTOR or UserFactory.PATIENT

<u>Return:</u>

an instance of Doctor or Patient model class

<u>isPatient (User user): boolean</u>

Check the user is a patient.

<u>Parameters:</u>

- user(User): Object.User model class is simple POJO object.

Return:

true if the user is a patient otherwise it is false

isDoctor (User user): boolean

Check the user is a doctor.

Parameters:

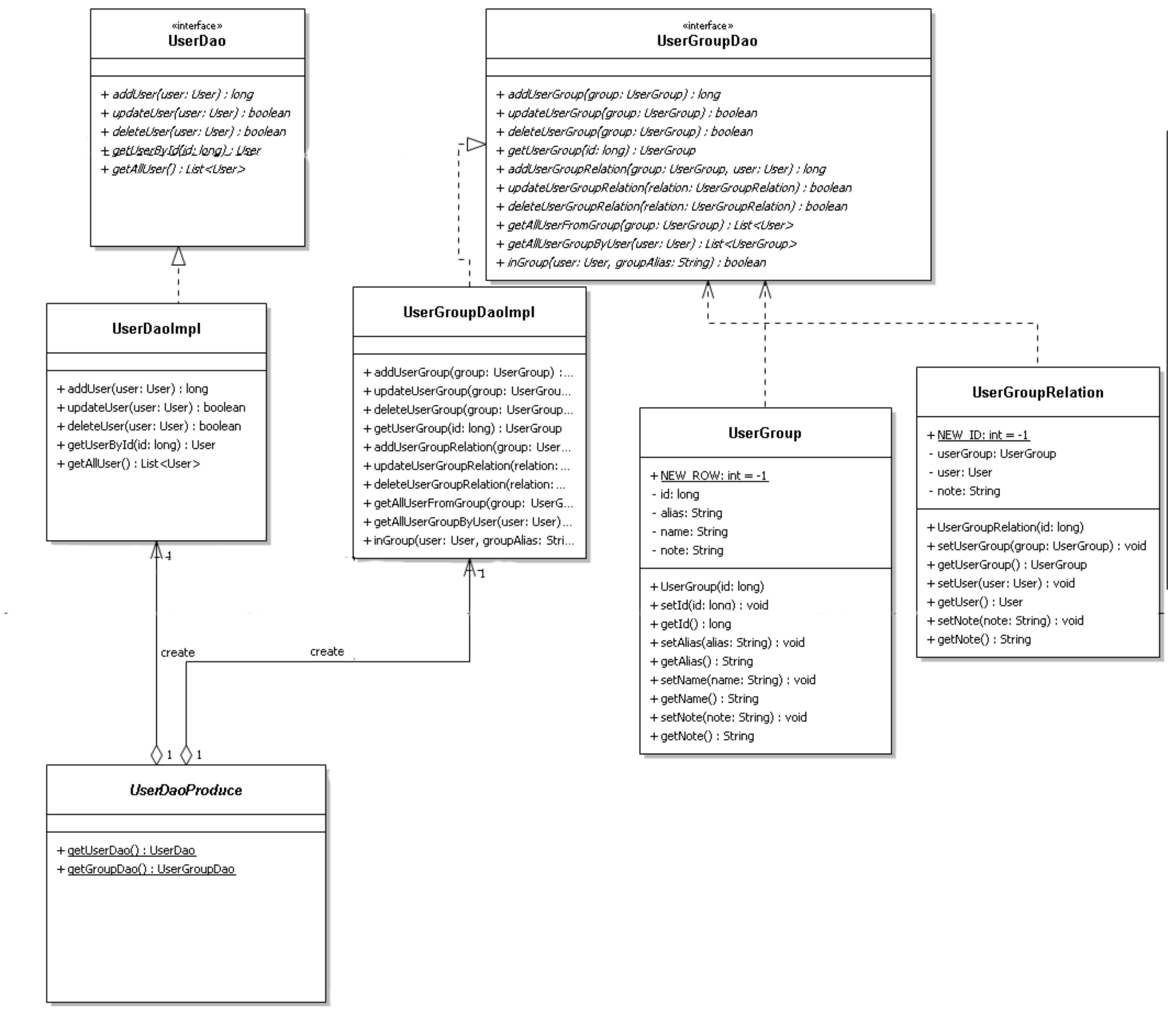
- user(User): Object.User model class is simple POJO object.

Return:

true if the user is a doctor otherwise it is false

User and User Group data access objects

This chapter explains that how to handle the users and user groups through the objects. We are going to create a *UserDao* and *UserGroupDao* interfaces. These interfaces implement concrete classes(*UserDaoImpl*, *UserGroupImpl*). Next step we create a static class. This class is *UserDaoProduce*. Through the *UserDaoProduce* class we can use the methods of *UserDaoImpl* and methods of *UserGroupDaoImpl*. We are going to create two model class (*UserGroup*, *UserGroupRelation*). Both classes will pass information to *UserGroupDao* object to get the data it needs.



«interface»
UserDao
+ addUser(user: User) : long
+ updateUser(user: User) : boolean
+ deleteUser(user: User) : boolean
+ getUserById(id: long) : User
+ getAllUser() : List<User>
«interface»
UserGroupDao
+ addUserGroup(group: UserGroup) : long
+ updateUserGroup(group: UserGroup) : boolean
+ deleteUserGroup(group: UserGroup) : boolean
+ getUserGroup(id: long) : UserGroup
+ addUserGroupRelation(group: UserGroup, user: User) : long
+ updateUserGroupRelation(relation: UserGroupRelation) : boolean
+ deleteUserGroupRelation(relation: UserGroupRelation) : boolean
+ getAllUserFromGroup(group: UserGroup) : List<User>
+ getAllUserGroupByUser(user: User) : List<UserGroup>
+ inGroup(user: User, groupAlias: String) : boolean
UserDaoImpl
+ addUser(user: User) : long
+ updateUser(user: User) : boolean
+ deleteUser(user: User) : boolean
+ getUserById(id: long) : User
+ getAllUser() : List<User>
UserGroupDaoImpl
+ addUserGroup(group: UserGroup) : ...
+ updateUserGroup(group: UserGrou...
+ deleteUserGroup(group: UserGroup...
+ getUserGroup(id: long) : UserGroup
+ addUserGroupRelation(group: User...
+ updateUserGroupRelation(relation: ...
+ deleteUserGroupRelation(relation: ...
+ getAllUserFromGroup(group: UserG...
+ getAllUserGroupByUser(user: User) ...
+ inGroup(user: User, groupAlias: Stri...
UserGroup
+ NEW_ROW: int = -1
- id: long
- alias: String
- name: String
- note: String
+ UserGroup(id: long)
+ setId(id: long) : void
+ getId() : long
+ setAlias(alias: String) : void
+ getAlias() : String
+ setName(name: String) : void
+ getName() : String
+ setNote(note: String) : void
+ getNote() : String
UserGroupRelation
+ NEW_ID: int = -1
- userGroup: UserGroup
- user: User
- note: String
+ UserGroupRelation(id: long)
+ setUserGroup(group: UserGroup) : void
+ getUserGroup() : UserGroup
+ setUser(user: User) : void
+ getUser() : User
+ setNote(note: String) : void
+ getNote() : String
1
1
create
create
1
1
UserDaoProduce
+ getUserDao() : UserDao
+ getGroupDao() : UserGroupDao

5. figure

UserDao interface

This interface defines the standard operations to be performed on a UserDaoImpl model object.

addUser(User user): long

Insert new user into database.

Parameters:

- user(User): Object.User is simple POJO object(Patient or Doctor model object).

Return:

This is new row id.

updateUser(User user): Boolean

Update one row in database.

Parameters:

- user(User): Object.User is simple POJO object.

Return:

Returns true for success. Otherwise return false.

deleteUser(User user): Boolean

Delete one row in database.

Parameters:

- user(User): Object.User is simple POJO object.

Return:

Returns true for success. Otherwise return false.

getUserById(long id): User

Get one row from database.

Parameters:

- id(long): unique identification of the row

Return:

Object.User is simple POJO object.

getAllUser(): List<User>

Get more rows from database. These rows contain the users whose there are in database.

Return:

Object.List object which contains User objects.

UserGroupDao interface

This interface defines the standard operations to be performed on a UserGroupDaoImpl model object.

addUserGroup(UserGroup group): long

Insert new user group into database.

Parameters:

- group(UserGroup): Object.User is simple POJO object.

Return:

This is new row id.

updateUserGroup(UserGroup group): Boolean

Update one row in database.

Parameters:

- group(UserGroup): Object.UserGroup is simple POJO object.

Return:

Returns true for success. Otherwise return false.

deleteUserGroup(UserGroup group): Boolean

Delete one row in database.

Parameters:

- group(UserGroup): Object.UserGroup is simple POJO object.

Return:

Returns true for success. Otherwise return false.

getUserGroup(long id): UserGroup

Get a row from database by id.

Parameters:

- id(Long): the unique value of row in database.

Return:

Object.UserGroup is simple POJO object.

addUserGroupRelation(UserGroup group, User user): long

This method inserts a user into a new group.

Parameters:

- group(UserGroup): Object.UserGroup is simple POJO object. This is new group of user.
- user(User): Object.User is simple POJO object. This user who will be inserted into the group.

Return:

Returns true for success. Otherwise return false.

updateUserGroupRelation(UserGroupRelation relation): boolean

Update a row in database.

Parameters:

- relation(UserGroupRelation): Object. UserGroupRelation is simple POJO object. This object contains the new data which will be updated.

Return:

Returns true for success. Otherwise return false.

deleteUserGroupRelation(UserGroupRelation relation): boolean

Delete a row from database.

Parameters:

- relation(UserGroupRelation): Object. UserGroupRelation is simple POJO object. This object contains the row of data that have to be delete.

Return:

Returns true for success. Otherwise return false.

getAllUserFromGroup(UserGroup group): List<User>

Return all user who are in this group.

Parameters:

- group(UserGroup): Object. UserGroup is simple POJO object. This object contains the data of group. Get a user list from this group.

Return:

Object.List object which it is contains User objects.

getAllUserGroupByUser(User user): List<UserGroup>

This method returns all groups of a user.

Parameters:

- user(User): Object. User is simple POJO object. This object contains data of a user.

Return:

Object.List object which it is contains UserGroup objects.

inGroup(User user, String groupAlias): boolean

Check if the user is in the group.

Parameters:

- user(User): Object. User is simple POJO object. This object contains data of a user.
- groupAlias(String): unique identification of group

Return:

true if the user is in the group otherwise false.

UserDaoImpl class

This is a concrete class which implements the UserDao interface.

UserGroupDaoImpl class

This is a concrete class which implements the UserGroupDao interface. This class uses the UserGroup and UserGroupRelation model classes. The class handle the connect of users and groups, too.

UserGroup class

This object is simple POJO containing get/set methods. This object is going to work as a Model and will pass information another to classes.

UserGroup (long id)

Create new model instance.

Parameters:

- id(long): Unique row id of user group in database. If this id is -1 then this object contains data of new user. If the id is greater than -1 then this object contains data of exists user.

UserGroupRelation class

This object is simple POJO containing get/set methods. This object is going to work as a Model and will pass information another to classes.

UserGroupRelation (long id)

Create new model instance.

Parameters:

- id(long): Unique row id of user group relation in database. If this id is -1 then this object contains data of new user. If the id is greatest than -1 then this object contains data of exists user.

UserDaoProduce class
Through the class we can use the methods of UserDaoImpl and methods of UserGroupDaoImpl. This class gets dao by passing information such as UserDao or UserGroupDao.

getUserDao(): UserDao

This method creates a new UserDaoImpl instance.

Return:

Object.UserDao object.

getUserGroupDao(): UserGroupDao

This method creates a new UserGroupDaoImpl instance.

Return:

Object.UserGroupDao object.