Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

***ПРОЕКТНА РОБОТА***

з дисципліни «Організація баз даних»

Тема «Підтримка клієнтів будівельної компанії»

*Виконав:*

Штанько А.О.

*Факультет:* ІОТ

*Група:* ІО-31,

*Перевірив:*  
 Болдак А.О.

**Київ**

**2014р.**

# *Зміст*

[Зміст - 2 -](#_Toc404855851)

[Запити зацікавлених осіб - 3 -](#_Toc404855852)

[Вступ - 3 -](#_Toc404855853)

[**1. Мета** - 3 -](#_Toc404855854)

[**2. Короткий огляд продукту** - 3 -](#_Toc404855855)

[**3. Контекст** - 3 -](#_Toc404855856)

[**4. Ділові правила** - 3 -](#_Toc404855857)

[4.1 Призначення системи - 3 -](#_Toc404855858)

[4.2 Політика взаємовідносин - 3 -](#_Toc404855859)

[4.3 Характеристика ділового процесу - 4 -](#_Toc404855860)

[4.4 Сценарії - 4 -](#_Toc404855861)

[4.4.1 Сценарій першої зустрічі з контакт-менеджером - 4 -](#_Toc404855862)

[4.4.2 Сценарій призначення зустрічі з контакт-менеджером - 4 -](#_Toc404855863)

[1) Інвестор авторизується в системі. - 4 -](#_Toc404855864)

[2) Інвестор заходить у розділ системи «Робота з клієнтами». - 4 -](#_Toc404855865)

[3) Інвестор натискає на посилання «Залишити повідомлення для адміністрації». - 4 -](#_Toc404855866)

[4) Інвестор заповнює поле з назвою « Залишити повідомлення», де вказує причину зустрічі та та натискає кнопку «Надіслати повідомлення». - 4 -](#_Toc404855867)

[5) Контакт-менеджер отримує повідомлення. - 4 -](#_Toc404855868)

[6) Контакт-менеджер зв’язується з інвестором та призначає зустріч. - 4 -](#_Toc404855869)

[4.4.3. Сценарій залишення повідомлення на форумі - 4 -](#_Toc404855870)

[1) Інвестор авторизується в системі. - 4 -](#_Toc404855871)

[2) Інвестор заходить у форум системи. - 4 -](#_Toc404855872)

[3) Інвестор знаходить потрібний розділ форуму. - 4 -](#_Toc404855873)

[4) Інвестор заповнює поле з назвою « Залишити коментар» та натискає кнопку «Надіслати повідомлення». - 4 -](#_Toc404855874)

[**5. Функціональність** - 4 -](#_Toc404855875)

[5.1 Структура облікового запису інвестора - 4 -](#_Toc404855876)

[5.2 Можливості гостей - 4 -](#_Toc404855877)

[**6. Практичність** - 4 -](#_Toc404855878)

[6.1 Мобільність - 4 -](#_Toc404855879)

[6.2 Інтерфейс користувача - 4 -](#_Toc404855880)

[**7. Надійність** - 4 -](#_Toc404855881)

[Прецеденти - 5 -](#_Toc404855882)

[**Список прецедентів** - 5 -](#_Toc404855883)

[**Діаграми Use Case** - 6 -](#_Toc404855884)

[**Use Case** - 8 -](#_Toc404855885)

[Діаграма бізнес сутностей - 12 -](#_Toc404855886)

[**Глосарій** - 13 -](#_Toc404855887)

[Реляційна схема - 14 -](#_Toc404855888)

[SQL запити - 14 -](#_Toc404855889)

[Java скрипт - 18 -](#_Toc404855890)

***Запити зацікавлених осіб***

**Вступ**

У цьому документі описуються запити зацікавлених осіб по відношенню до розроблювальної системи «Будівельна компанія» в якості яких виступають будь-яка фізична чи юридична особа, контакт-менеджер та клієнти цієї компанії.

* Гість – майбутній клієнт;
* Клієнт – фізична особа, що користується послугами будівельної компанії;
* Контакт-менеджер – відповідає за зв’язок компанії з клієнтом;

**1. Мета**

Метою документу є визначення основних вимог до функціональності, продуктивності, експлуатаційної придатності, а також визначення бізнес-правил і технологічних обмежень, пред’явлених предмету розробки.

**2. Короткий огляд продукту**

Система «Будівельна компанія» надає інтерфейс взаємодії між замовником, персоналом та клієнтами цієї компанії. Замовник має найвищий рівень доступу до організації системи та може контролювати усі внутрішні процеси. Персонал (контакт-менеджер) має доступ до бази даних клієнтів (редагування та оновлення даних клієнта, реєстрація нового клієнта), що дозволяє працівникам напряму спілкуватися з клієнтами. Клієнт має доступ лише до своєї анкети у базі даних та загальної інформації. Гість має доступ лише до загальної інформації.

**3. Контекст**

Перелік вимог, зазначених у даному документі є основою технічного завдання для розробки системи «Будівельна компанія».

**4. Ділові правила**

4.1 Призначення системи

Система призначена для зберігання інформації про клієнтів компанії із подальшим використанням на різних рівнях доступу для полегшення організації комунікаційного процесу та економії часу при автоматичному заповненні необхідних документів.

4.2 Політика взаємовідносин

Система «Будівельна компанія» надає можливість усім працівникам використовувати програмне забезпечення для фіксування результатів переговорів контакт-менеджер – інвестори.

Гість може переглядати список наявних пропозицій та читати форум.

Інвестор має постійний доступ до процесу будівництва за допомогою засобів відеоспостереження, має можливість призначати зустріч з контакт-менеджером, переглядати історію переговорів, а також отримувати повну інформацію про інвестоване майно. За політикою конфіденційності системи інвестор має доступ лише до свого профілю, маючи пару логін-пароль, яку він отримує після успішного підписання договору. Інвестор є повноцінним учасником форуму, де може залишати свої повідомлення, оцінювати якість роботи компанії.

Контакт-менеджер має власний робочий кабінет в системі у якій оброблює заявки та повідомлення від інвесторів.

4.3 Характеристика ділового процесу

Обліковий запис інвестора у системі створюється після укладання угоди з компанією. Після створення нового облікового запису клієнт повинен перенести дані з договору в профіль. По завершенню процедури реєстрації дані відправляються у базу даних компанії.

4.4 Сценарії

4.4.1 Сценарій першої зустрічі з контакт-менеджером

1) Гість контактує з компанією, досягають консенсусу про зустріч у назначений час;

2) Контакт-менеджер веде перемовини щодо вибору нерухомості;

3) Контакт-менеджер укладає договір купівлі-продажу між компанією та гостем;

4) Гість отримує статус інвестора та пару логін-пароль від власного облікового запису;

5) Інвестор авторизується у системі і заповнює усі необхідні дані про договір;

6) Контакт-менеджер перевіряє правильність уведених даних інвестором.

4.4.2 Сценарій призначення зустрічі з контакт-менеджером

1) Інвестор авторизується в системі.

2) Інвестор заходить у розділ системи «Робота з клієнтами».

3) Інвестор натискає на посилання «Залишити повідомлення для адміністрації».

4) Інвестор заповнює поле з назвою « Залишити повідомлення», де вказує причину зустрічі та та натискає кнопку «Надіслати повідомлення».

5) Контакт-менеджер отримує повідомлення.

6) Контакт-менеджер зв’язується з інвестором та призначає зустріч.

4.4.3. Сценарій залишення повідомлення на форумі

1) Інвестор авторизується в системі.

2) Інвестор заходить у форум системи.

3) Інвестор знаходить потрібний розділ форуму.

4) Інвестор заповнює поле з назвою « Залишити коментар» та натискає кнопку «Надіслати повідомлення».

**5. Функціональність**

Основні вимоги до функціональності, пред’явлені зацікавленими особами, відносяться до трьох категорій:

1. Гість;
2. Інвестор;
3. Контакт-менеджер.

5.1 Структура облікового запису інвестора

1) Паспортні дані – прізвище, ім’я та по-батькові інвестора, дата народження, адреса проживання, сімейний стан, прізвище, ім’я та по-батькові жінки (чоловіка) тощо.

2) Контактні дані – skype, e-mail, vk, facebook, мобільні номери чоловіка та/або жінки.

3) Дані про угоду – дата підписання, дата прийняття в експлуатацію, умови оплати, фотокопії попереднього договору та інших документів;

4) Зв’язок з контакт-менеджером – призначення зустрічі, надсилання повідомлень;

5) Доступ до залишених повідомлень на форумі, можливість редагування або видалення своїх повідомлень;

6) Переглядання онлайн відео з будівельного майданчику (різні ракурси);

7) Фінансовий баланс – чіткий звіт про вкладені інвестиції та відображення поточного залишку до оплати.

Система забезпечує інвестору використання зазначених вище інформації та можливостей.

* 1. Можливості гостей

1. Перегляд форуму;
2. Перегляд наявних пропозицій;
3. Можливість призначати зустріч із контакт-менеджером.

5.3 Можливості контакт-менеджера

1) Оброблювати запити – назначати час та місце зустрічі;

2) Оброблювати повідомлення;

3) Підписання договору купівлі-продажу;

4) Перевірка валідності введених даних інвестором та їх конфірмація

**6. Практичність**

6.1 Мобільність

Веб-сайт повинен бути оптимізованим для роботи не тільки із комп’ютера, а також із мобільних пристроїв.

6.2 Інтерфейс користувача

Інтерфейс облікового запису повинен відповідати наступним вимогам:

1. Бути зрозумілим і не допускати двозначного тлумачення;
2. Бути виконаним з урахуванням ергономічних вимог, бути інтуїтивно зрозумілим;
3. Усі кодовані параметри або елементи, та наведені скорочення повинні мати тлумачення або вікно-підказку, що буде з’являтися після наведення курсору на елемент або після натискання спеціальної клавіші;
4. При виконанні електронного цифрового підпису в інтерфейс повинен бути включений результат перевірки цілісності цифрового підпису відповідно з сертифікатом. Відомості про порушення цілісного підпису повинні бути виділені окремо.

**7. Надійність**

Протягом усього терміну зберігання для облікових записів повинна бути забезпечена їх цілісність, незмінність і достовірність.

Для забезпечення збереження та цілісності використовуватиметься метод резервного копіювання.

Система повинна бути добре захищена від різного роду зловмисних атак із метою заволодіння інформації чи атак типу DDoS. Має використовуватися комплекс технологічних і адміністративних процедур, що перешкоджають випадковій або навмисній зміні збережених записів. Найкращим рішенням є використання електронного цифрового підпису, що дозволяє у будь-який момент перевірити незмінність збереженого запису в порівнянні з моментом його підписання.

Також система повинна витримувати великі навантаження, обслуговуючи значну кількість користувачів.

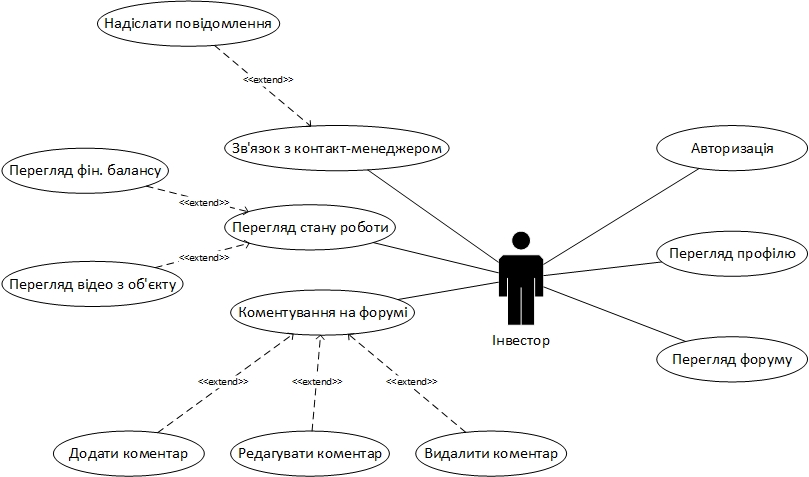
Крім цього повинна забезпечуватись конфіденційність персональної інформації. Надання доступу до збережених підписаних записів та персональних даних інвесторів здійснюють у відповідності з правами доступу.

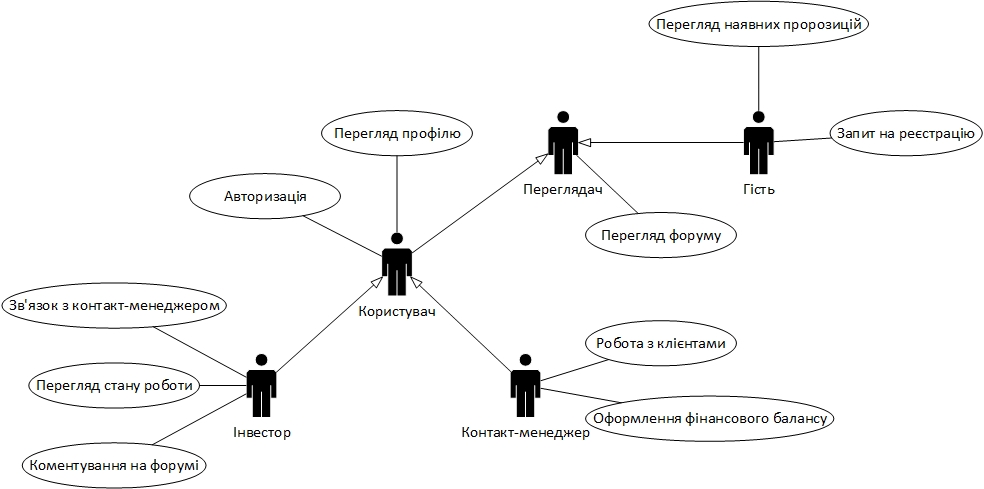
# *Прецеденти*

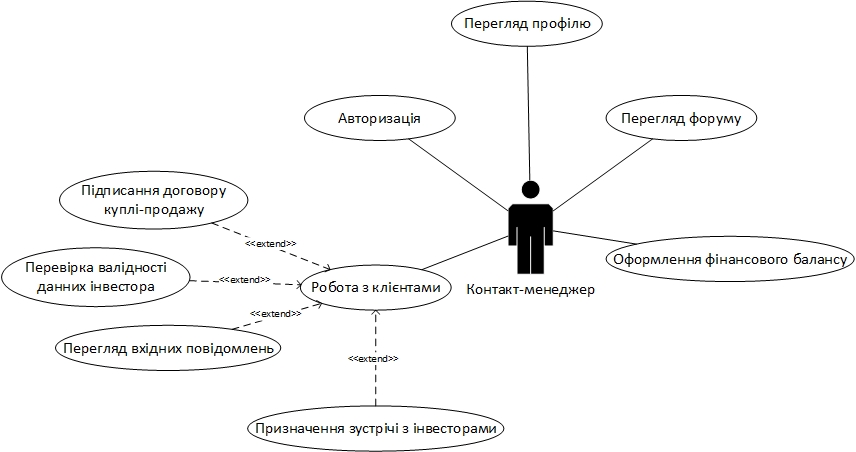
**Список прецедентів**:

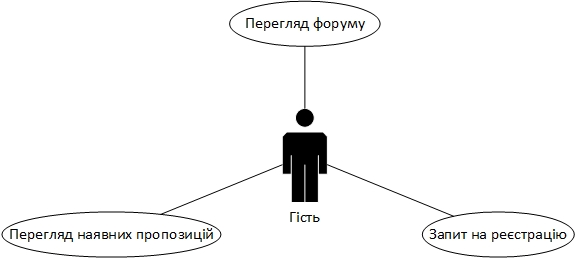
* Діаграма Use Case
* Use Case 001 Запит на реєстрацію гостя
* Use Case 002 Переглядання форуму
* Use Case 003 Переглядання пропозиції
* Use Case 004 Перша авторизація інвестора
* Use Case 005 Перевірка валідності введених даних інвестора
* Use Case 006 Авторизація користувача
* Use Case 007 Перегляд профілю
* Use Case 008 Надсилання інвестором повідомлення контакт-менеджеру
* Use Case 009 Отримання контакт-менеджером зворотнього зв’язку
* Use Case 010 Перегляд відео з об’єкту
* Use Case 011 Перегляд фінансового балансу
* Use Case 012 Оформлення фінансового балансу
* Use Case 013 Додавання коментаря
* Use Case 014 Видалення коментаря
* Use Case 015 Редагування коментаря

**Діаграми Use Case**

****

****

****

****

**Use Case**

**ID: UC 001**

**Назва:** Запит на реєстрацію гостя.

**Учасники:** Гість, система.

**Передумови:** Гість бажає укласти договір із будівельною компанією.

**Результат:** Система надсилає запит на укладання договору контакт-менеджеру.

**Основний сценарій:**

1. Гість натискає на кнопку “Запит на укладання договору”.
2. Система надає форму із параметрами «Дата і час зустрічі», «Бажаний об’єкт».
3. Гість обирає дату та час зустрічі і пункті «Обрати час зустрічі».
4. Гість вводить інформацію у поле «Бажаний об’єкт».
5. Гість натискає кнопку «Зареєструвати».
6. Система надсилає запит на укладання договору контакт-менеджеру.

**Виключні ситуації:**

1. Гість не заповнив якесь із полів.
2. Гість не має змоги зустрітись із контакт-менеджером через зайнятість у години незайнятості контакт-менеджера.

**ID: UC 002**

**Назва:** Переглядання форуму.

**Учасники:** Переглядач, система.

**Передумови:** Переглядач бажає переглянути повідомлення на форумі.

**Результат:** Переглядач отримує доступ до повідомлень форуму.

**Основний сценарій:**

1. Переглядач натискає на кнопку “Перейти на форум”.
2. Система відправляє переглядача на форум.
3. Переглядач обирає необхідну гілку форуму.
4. Система відправляє переглядача на обрану гілку форуму.
5. Переглядач переглядає повідомлення обраної гілки форуму.

**ID: UC 003**

**Назва:** Переглядання пропозиції.

**Учасники:** Гість, система.

**Передумови:** Гість бажає переглянути наявні пропозиції будівельної компанії.

**Результат:** Гість переглядає наявні пропозиції.

**Основний сценарій:**

1. Гість натискає на кнопку “Перейти до пропозицій”.
2. Система відправляє гостя на сторінку «Поточні проекти».
3. Гість обирає пропозицію.
4. Система відправляє гостя на сторінку обраної пропозиції.
5. Гість переглядає інформацію про обрану пропозицію.

**Виключні ситуації:**

1. Будівельна компанія не має поточних проектів.
2. Система не має поточних проектів, що влаштовують гостя.

**ID: UC 004**

**Назва:** Перша авторизація інвестора.

**Учасники:** Інвестор, система.

**Передумови:** Інвестор уклав договір із будівельною компанією.

**Результат:** Система надсилає запит на підтвердження даних про інвестора контакт-менеджеру.

**Основний сценарій:**

1. Інвестор відправляє запит на авторизацію.
2. Система надає форму із параметрами «Логін» та «Пароль» для авторизації.
3. Інвестор вводить свої дані (логін/пароль).
4. Інвестор натискає кнопку «Ввійти в систему».
5. Система перевіряє дані авторизації.
6. Система відправляє інвестора на сторінку «Паспортні дані».
7. Інвестор заповнює параметри сторінки «Паспортні дані» («Прізвище», «Ім’я» «По-батькові», заповнює параметри сторінки «Контактні дані»«Дата народження», «Адреса проживання», «Сімейний стан», «Прізвище, ім’я та по-батькові жінки (чоловіка)») та натискає «Далі».
8. Система відправляє інвестора на сторінку «Контактні дані».
9. Інвестор («Мобільний номер» , «Skype», «e-mail», «Vk», «Facebook», «Мобільний номер жінки(чоловіка)»)та натискає «Далі».
10. Система відправляє інвестора на сторінку «Дані про угоду».
11. Інвестор заповнює параметри сторінки «Дані про угоду» («Дата підписання», «Дата прийняття в експлуатацію», «Умови оплати», «Фотокопії попереднього договору та інших документів») та натискає «Завершити ввід даних інвестора».
12. Система відправляє запит на перевірку валідності введених даних інвестора контакт-менеджеру.

**Виключні ситуації:**

1. Введені персональні дані не вірні (логін/пароль).
2. Інвестор не заповнив якесь з полів.

**ID: UC 005**

**Назва:** Перевірка валідності введених даних інвестора.

**Учасники:** Контакт-менеджер, система.

**Передумови:** Система відправила запит на перевірку валідності введених даних інвестора контакт-менеджеру.

**Результат:** Інвестор отримує права інвестора у системі.

**Основний сценарій:**

1. Контакт-менеджер натискає на кнопку «Робота з клієнтами».
2. Система відправляє контакт-менеджера на сторінку «Робота з клієнтами».
3. Контакт-менеджер вводить логін інвестора та натискає кнопку «Перевірити дані інвестора».
4. Система відправляє контакт-менеджера на сторінку із даними обраного інвестора.
5. Контакт-менеджер перевіряє введені дані інвестора.
6. Контакт-менеджер натискає кнопку «Перевірено».
7. Система надає інвестору повні права інвестора.

**Виключні ситуації:**

1. Введені дані інвестора не валідні.

**ID: UC 006**

**Назва:** Авторизація користувача.

**Учасники:** Користувач, система.

**Передумови:** Користувач не має доступа до системи.

**Результат:** Користувач отримує доступ до системи.

**Основний сценарій:**

1. Користувач відправляє запит на авторизацію.
2. Система надає форму із параметрами «Логін» та «Пароль» для авторизації.
3. Користувач вводить свої дані (логін/пароль).
4. Користувач натискає кнопку «Ввійти в систему».
5. Система перевіряє дані авторизації.
6. Система дає доступ користувачу до системи.

**Виключні ситуації:**

1. Введені персональні дані не вірні (логін/пароль).

**ID: UC 007**

**Назва:** Перегляд профілю.

**Учасники:** Користувач, система.

**Передумови:** Користувач авторизований у системі.

**Результат:** Користувач переглядає свій профіль.

**Основний сценарій:**

1. Користувач натискає на кнопку “Переглянути профіль”.
2. Система відправляє користувача на сторінку «Мій профіль».
3. Користувач обирає категорію персональних даних користувача та натискає кнопку «Переглянути».
4. Система відправляє користувача на потрібну сторінку.
5. Користувач переглядає обрану інформацію.

**ID: UC 008**

**Назва:** Надсилання інвестором повідомлення контакт-менеджеру.

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор має потребу задати питання/ призначити зустріч контакт-менеджеру.

**Результат:** Система надсилає повідомлення контакт-менеджеру.

**Основний сценарій:**

1. Інвестор натискає на кнопку «Робота з клієнтами».
2. Система відправляє інвестора на сторінку «Ваші повідомлення».
3. Інвестор натискає на посилання «Залишити повідомлення для адміністрації».
4. Система відправляє інвестора на сторінку «Написати нове повідомлення адміністрації».
5. Інвестор заповнює поле з назвою « Залишити повідомлення», де вказує причину зустрічі та натискає кнопку «Надіслати повідомлення».
6. Система надсилає повідомлення контакт-менеджеру.

**Виключні ситуації:**

1. Інвестор не заповнив поле повідомлення.

**ID: UC 009**

**Назва:** Отримання контакт-менеджером зворотнього зв’язку.

**Учасники:** Контакт-менеджер, система.

**Передумови:** Контакт-менеджер авторизований у системі, контакт-менеджер має необхідність переглянути повідомлення від інвесторів.

**Результат:** Контакт-менеджер переглядає повідомлення.

**Основний сценарій:**

1. Контакт-менеджер натискає на кнопку «Переглянути повідомлення».
2. Система переправляє контакт-менеджера на сторінку повідомлень.
3. Контакт-менеджер обирає непрочитане повідомлення і переглядає його.

**Виключні ситуації:**

1. Контакт-менеджер не має непрочитаних повідомлень.

**ID: UC 010**

**Назва:** Перегляд відео з об’єкту .

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор бажає проконтролювати стан роботи на об’єкті.

**Результат:** Інвестор переглядає відео з об’єкта.

**Основний сценарій:**

1. Інвестор натискає на кнопку «Переглянути відео з об’єкта».
2. Система відправляє інвестора на сторінку «Ваші об’єкти».
3. Інвестор обирає потрібний об’єкт та натискає на кнопку «Перегляд ракурсів об’єкта».
4. Система відправляє інвестора на сторінку «Відео-ракурси об’єкта».
5. Інвестор обирає потрібний ракурс та натискає на кнопку «Перегляд відео».
6. Система відправляє інвестора на сторінку із потрібною відео-трансляцією з об’єкта.
7. Інвестор дивиться шукане відео.

**Виключні ситуації:**

1. На об’єкті ще не встановлені відеокамери.
2. Проблеми із потрібною веб-камерою.
3. На об’єкті приміщенні ведуться роботи, що унеможливлюють відео трансляцію.

**ID: UC 011**

**Назва:** Перегляд фінансового балансу.

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор бажає переглянути фінансовий баланс.

**Результат:** Інвестор переглядає свій фінансовий баланс.

**Основний сценарій:**

1. Інвестор натискає на кнопку «Переглянути фінансовий баланс».
2. Система відправляє інвестора на сторінку «Переглянути фінансовий баланс».
3. Інвестор обирає фінансовий баланс за потрібний термін та натискає на посилання із потрібним балансом.
4. Система відправляє інвестора на сторінку із потрібним фінансовим балансом.
5. Інвестор переглядає свій фінансовий баланс

**ID: UC 012**

**Назва:** Оформлення фінансового балансу.

**Учасники:** Контакт-менеджер, система.

**Передумови:** Контакт-менеджер авторизований у системі, інвестор має відредагувати фінансовий баланс певного користувача.

**Результат:** Система обновлює фінансовий баланс інвестора.

**Основний сценарій:**

1. Контакт-менеджер натискає на кнопку «Робота з клієнтами».
2. Система відправляє контакт-менеджера на сторінку «Робота з клієнтами».
3. Контакт-менеджер вводить логін інвестора та натискає кнопку «Оновити фінансовий баланс».
4. Система відправляє контакт-менеджера на сторінку із фінансовим балансом обраного інвестора.
5. Контакт-менеджер натискає кнопку «Оновити».
6. Система надає форму контакт-менеджеру для зміни фінансового балансу інвестора.
7. Контакт-менеджер заносить оновлені фінансові дані та дату зміни у форму та натискає на кнопку «Оновити фінансовий баланс».
8. Система оновлює фінансовий баланс інвестора. .

**Виключні ситуації:**

1. Контакт-менеджер неправильно ввів логін інвестора.
2. Контакт-менеджер не заповнив якесь з полів.

**ID: UC 013**

**Назва:** Додавання коментаря.

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор бажає додати коментар на форумі.

**Результат:** Система додає коментар інвестора.

**Основний сценарій:**

1. Інвестор натискає кнопку “Перейти на форум”.
2. Система відправляє інвестора на форум.
3. Інвестор обирає необхідну гілку форуму.
4. Система відправляє інвестора на обрану гілку форуму.
5. Інвестор переглядає повідомлення обраної гілки форуму та натискає на кнопку « Залишити коментар»
6. Система надає інвесторові форму з параметром «Текст коментаря».
7. Інвестор заповнює поле з назвою «Текст коментаря» та натискає кнопку «Надіслати коментар».
8. Система публікує коментар інвестора у обраній гілці форума.

**Виключні ситуації:**

1. Інвестор не заповнив поле «Відредагований текст коментаря».

**ID: UC 014**

**Назва:** Видалення коментаря.

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор бажає видалити коментар на форумі.

**Результат:** Система видаляє коментар інвестора.

**Основний сценарій:**

1. Інвестор натискає кнопку “Перейти на форум”.
2. Система відправляє інвестора на форум.
3. Інвестор обирає необхідну гілку форуму.
4. Система відправляє інвестора на обрану гілку форуму.
5. Інвестор переглядає коментарі обраної гілки форуму, шукає необхідний коментар та натискає на кнопку біля нього «Видалити коментар»
6. Система видаляє коментар інвестора у обраній гілці форума.

**Виключні ситуації:**

1. Інвестор не має коментарів на форумі.

**ID: UC 015**

**Назва:** Редагування коментаря

**Учасники:** Інвестор, система.

**Передумови:** Інвестор авторизований у системі, інвестор бажає редагувати коментар на форумі.

**Результат:** Система редагує коментар інвестора.

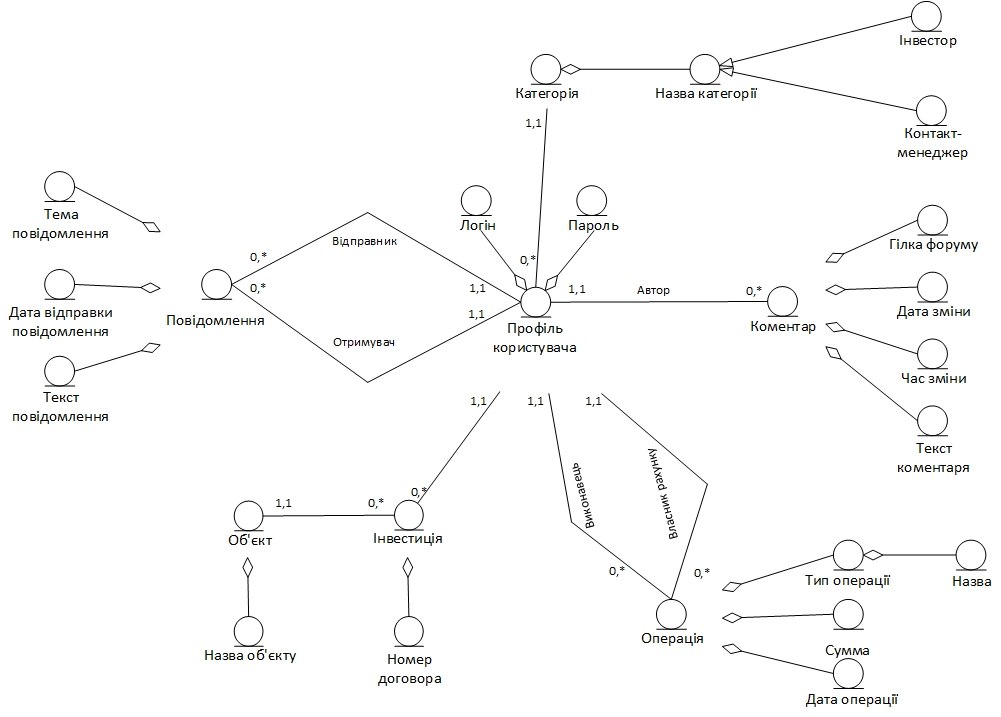
**Основний сценарій:**

1. Інвестор натискає кнопку “Перейти на форум”.
2. Система відправляє інвестора на форум.
3. Інвестор обирає необхідну гілку форуму.
4. Система відправляє інвестора на обрану гілку форуму.
5. Інвестор переглядає коментарі обраної гілки форуму, шукає необхідний коментар та натискає на кнопку біля нього «Редагувати коментар»
6. Система надає інвесторові форму з параметром «Відредагований текст коментаря».
7. Інвестор заповнює поле з назвою «Відредагований текст коментаря» та натискає кнопку «Відредагувати коментар».
8. Система редагує коментар інвестора у обраній гілці форума.

**Виключні ситуації:**

1. Інвестор не має коментарів на форумі.
2. Коментар було вже відредаговано.

# *Діаграма бізнес сутностей*



**Глосарій**

***Профіль користувача*** – це сутність, яка містить інформацію про користувача та має атрибути «Логін» та «Пароль».

***Повідомлення*** – це сутність, що є текстом, відправленим іншому користувачу, та має атрибути «Тема повідомлення», «Дата відправки повідомлення» та «Текст повідомлення».

***Інвестиція*** – це сутність, яка містить інформацію про інвестиції користувача та містить атрибут «Номер договора».

***Об’єкт*** - це сутність, що є певним об’єктом нерухомості, у який інвестував свої гроші інвестор, та має атрибут «Назва об’єкту».

***Операція*** – це сутність, яка є одиничною зміною фінансового балансу користувача та містить атрибути «Тип операції», «Сумма» та «Дата операції».

***Коментар*** – це сутність, що є повідомленням на форумі та містить атрибути «Гілка форуму», «Дата зміни», «Час зміни» та «Текст коментаря».

***Категорія*** – це сутність, що є повідомленням на форумі та містить атрибут «Назва категорії».

# *Реляційна схема*

# *SQL запити*

Navicat MySQL Data Transfer

Source Server : ODB

Source Server Version : 50619

Source Host : localhost:3306

Source Database : ODB\_Lab4

Target Server Type : MYSQL

Target Server Version : 50619

File Encoding : 65001

Date: 2014-11-25 23:14:17

\*/

SET FOREIGN\_KEY\_CHECKS=0;

-- ----------------------------

-- Table structure for category

-- ----------------------------

DROP TABLE IF EXISTS `category`;

CREATE TABLE `category` (

`category\_id` int(255) NOT NULL AUTO\_INCREMENT,

`category\_name` varchar(250) NOT NULL,

PRIMARY KEY (`category\_id`),

KEY `profile\_id` (`category\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of category

-- ----------------------------

INSERT INTO `category` VALUES ('1', 'Investor');

INSERT INTO `category` VALUES ('2', 'Contact-manager');

-- ----------------------------

-- Table structure for comment

-- ----------------------------

DROP TABLE IF EXISTS `comment`;

CREATE TABLE `comment` (

`comment\_id` int(255) NOT NULL AUTO\_INCREMENT,

`comment\_branch\_of\_forum` varchar(250) NOT NULL,

`comment\_date\_of\_change` date NOT NULL,

`comment\_time\_of\_change` datetime(6) NOT NULL,

`comment\_text` varchar(250) NOT NULL,

`profile\_id` int(11) NOT NULL,

PRIMARY KEY (`comment\_id`),

KEY `profile\_id` (`profile\_id`) USING BTREE,

CONSTRAINT `comment\_pp` FOREIGN KEY (`profile\_id`) REFERENCES `profile` (`profile\_id`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of comment

-- ----------------------------

INSERT INTO `comment` VALUES ('1', 'Fresh news', '2014-11-25', '2014-11-25 21:18:23.000000', 'so interesting', '1');

INSERT INTO `comment` VALUES ('2', 'FAQ', '2014-11-20', '2014-11-20 21:42:51.000000', 'useful', '2');

INSERT INTO `comment` VALUES ('3', 'Prices', '2014-11-21', '2014-11-21 21:43:05.000000', 'wtf?!', '3');

-- ----------------------------

-- Table structure for investment

-- ----------------------------

DROP TABLE IF EXISTS `investment`;

CREATE TABLE `investment` (

`investment\_id` int(255) NOT NULL AUTO\_INCREMENT,

`investment\_contract\_number` int(255) NOT NULL,

`object\_id` int(11) NOT NULL,

`profile\_id` int(11) NOT NULL,

PRIMARY KEY (`investment\_id`),

KEY `object\_id` (`object\_id`) USING BTREE,

KEY `investment\_pp` (`profile\_id`),

CONSTRAINT `investment\_pp` FOREIGN KEY (`profile\_id`) REFERENCES `profile` (`profile\_id`),

CONSTRAINT `investment\_oo` FOREIGN KEY (`object\_id`) REFERENCES `object` (`object\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of investment

-- ----------------------------

INSERT INTO `investment` VALUES ('1', '5123151', '1', '1');

INSERT INTO `investment` VALUES ('2', '5516516', '1', '1');

INSERT INTO `investment` VALUES ('3', '5621533', '2', '2');

INSERT INTO `investment` VALUES ('4', '4566138', '2', '2');

INSERT INTO `investment` VALUES ('5', '8815531', '3', '3');

-- ----------------------------

-- Table structure for message

-- ----------------------------

DROP TABLE IF EXISTS `message`;

CREATE TABLE `message` (

`message\_id` int(11) NOT NULL AUTO\_INCREMENT,

`message\_subject` varchar(250) NOT NULL,

`message\_date` date NOT NULL,

`message\_text` varchar(250) NOT NULL,

`message\_sender\_profile\_id` int(11) NOT NULL,

`message\_recipient\_profile\_id` int(11) NOT NULL,

PRIMARY KEY (`message\_id`),

KEY `profile\_id` (`message\_sender\_profile\_id`,`message\_recipient\_profile\_id`),

KEY `message\_mrp` (`message\_recipient\_profile\_id`),

CONSTRAINT `message\_mrp` FOREIGN KEY (`message\_recipient\_profile\_id`) REFERENCES `profile` (`profile\_id`),

CONSTRAINT `message\_msp` FOREIGN KEY (`message\_sender\_profile\_id`) REFERENCES `profile` (`profile\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of message

-- ----------------------------

INSERT INTO `message` VALUES ('1', 'buying', '2014-11-19', 'want to buy', '1', '3');

INSERT INTO `message` VALUES ('2', 'meeting', '2014-11-21', 'want to meet', '2', '3');

INSERT INTO `message` VALUES ('3', 'askingfor question', '2014-11-20', 'how to arrange a meeting with conatct-manager', '2', '1');

-- ----------------------------

-- Table structure for object

-- ----------------------------

DROP TABLE IF EXISTS `object`;

CREATE TABLE `object` (

`object\_id` int(11) NOT NULL AUTO\_INCREMENT,

`object\_name` varchar(250) NOT NULL,

PRIMARY KEY (`object\_id`),

KEY `investment\_id` (`object\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of object

-- ----------------------------

INSERT INTO `object` VALUES ('1', 'new building 1');

INSERT INTO `object` VALUES ('2', 'new building 2 ');

INSERT INTO `object` VALUES ('3', 'new building 3');

-- ----------------------------

-- Table structure for operation

-- ----------------------------

DROP TABLE IF EXISTS `operation`;

CREATE TABLE `operation` (

`operation\_id` int(11) NOT NULL AUTO\_INCREMENT,

`operation\_type\_name` varchar(250) NOT NULL,

`operation\_summ` int(11) NOT NULL,

`operation\_date` date NOT NULL,

`profile\_id` int(11) NOT NULL,

PRIMARY KEY (`operation\_id`),

KEY `operation\_pp` (`profile\_id`),

CONSTRAINT `operation\_pp` FOREIGN KEY (`profile\_id`) REFERENCES `profile` (`profile\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of operation

-- ----------------------------

INSERT INTO `operation` VALUES ('1', 'payment', '1000', '2014-11-19', '1');

INSERT INTO `operation` VALUES ('2', 'payment', '1250', '2014-11-20', '2');

INSERT INTO `operation` VALUES ('3', 'transaction', '2250', '2014-11-20', '3');

-- ----------------------------

-- Table structure for profile

-- ----------------------------

DROP TABLE IF EXISTS `profile`;

CREATE TABLE `profile` (

`profile\_id` int(11) NOT NULL AUTO\_INCREMENT,

`profile\_login` varchar(250) NOT NULL,

`profile\_password` varchar(250) NOT NULL,

`category\_id` int(11) NOT NULL,

PRIMARY KEY (`profile\_id`),

KEY `category\_id` (`category\_id`),

CONSTRAINT `profile\_cc` FOREIGN KEY (`category\_id`) REFERENCES `category` (`category\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

-- ----------------------------

-- Records of profile

-- ----------------------------

INSERT INTO `profile` VALUES ('1', 'ololosh', '123456', '1');

INSERT INTO `profile` VALUES ('2', 'ololoev', '234567', '1');

INSERT INTO `profile` VALUES ('3', 'Nick23', 'qwerty', '2');

# *Java скрипт*

**package** abstractions;

/\*\*

\* Created by oleh on 26.11.14.

\*/

**public** **interface** Condition<T> {

**public** **boolean** satisfyTo(T object);

}

**package** abstractions;

**import** java.sql.SQLException;

**import** java.util.HashSet;

/\*\*

\* Created by oleh on 26.11.14.

\*/

**public** **abstract** **class** DAO<T> {

**public** **abstract** HashSet<T> get(Condition<T> condition) **throws** SQLException;

**public** **abstract** T create(T object) **throws** SQLException;

**public** **abstract** T update(T object) **throws** SQLException;

**public** **abstract** **boolean** delete(T object) **throws** SQLException;

**public** **abstract** T getById(Object id) **throws** SQLException;

**protected** **abstract** HashSet<T> getFromDatabase() **throws** SQLException;

}

**package** access;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** Connector {

**private** String config;

**private** String driver;

**private** Connection connection;

**public** Connector() {

String url = "jdbc:mysql://host-176-36-112-223.la.net.ua/";

String db = "ODB\_Lab4";

String login = "oter";

String password = "12345";

**this**.driver = "com.mysql.jdbc.Driver";

**this**.config = url + db + "?user=" + login + "&password=" + password;

**try** {

Class.*forName*(driver).newInstance();

connection = DriverManager.*getConnection*(config);

} **catch** (InstantiationException | IllegalAccessException

| ClassNotFoundException e) {

e.printStackTrace();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**public** Connection getConnection() {

**return** connection;

}

}

**package** access;

**import** managers.\*;

/\*\*

\* Factory for data base

\* Created by oter on 26.11.14.

\*/

**public** **class** DAOFactory {

**private** **static** Connector *connection* = **new** Connector();

**private** **static** CategoryDAO *categoryDAO*;

**private** **static** CommentDAO *commentDAO*;

**private** **static** InvestmentDAO *investmentDAO*;

**private** **static** MessageDAO *messageDAO*;

**private** **static** ObjectDAO *objectDAO*;

**private** **static** OperationDAO *operationDAO*;

**private** **static** ProfileDAO *profileDAO*;

/\*\*

\* Getter for CategoryDAO

\* **@return** CategoryDAO

\*/

**public** **static** CategoryDAO getCategoryDAO() {

**if** (*categoryDAO* == **null**) {

*categoryDAO* = **new** CategoryDAO(*connection*.getConnection());

}

**return** *categoryDAO*;

}

/\*\*

\* Getter for CommentDAO

\* **@return** CommentDAO

\*/

**public** **static** CommentDAO getCommentDAO() {

**if** (*commentDAO* == **null**) {

*commentDAO* = **new** CommentDAO(*connection*.getConnection());

}

**return** *commentDAO*;

}

/\*\*

\* Getter for InvestmentDAO

\* **@return** InvestmentDAO

\*/

**public** **static** InvestmentDAO getInvestmentDAO() {

**if** (*investmentDAO* == **null**) {

*investmentDAO* = **new** InvestmentDAO(*connection*.getConnection(), *objectDAO*, *profileDAO*);

}

**return** *investmentDAO*;

}

/\*\*

\* Getter for MessageDAO

\* **@return** MessageDAO

\*/

**public** **static** MessageDAO getMessageDAO() {

**if** (*messageDAO* == **null**) {

*messageDAO* = **new** MessageDAO(*connection*.getConnection(), *profileDAO*, *profileDAO*);

}

**return** *messageDAO*;

}

/\*\*

\* Getter for ObjectDAO

\* **@return** ObjectDAO

\*/

**public** **static** ObjectDAO getObjectDAO() {

**if** (*objectDAO* == **null**) {

*objectDAO* = **new** ObjectDAO(*connection*.getConnection());

}

**return** *objectDAO*;

}

/\*\*

\* Getter for OperationDAO

\* **@return** OperationDAO

\*/

**public** **static** OperationDAO getOperationDAO() {

**if** (*operationDAO* == **null**) {

*operationDAO* = **new** OperationDAO(*connection*.getConnection(), *profileDAO*);

}

**return** *operationDAO*;

}

/\*\*

\* Getter for ProfileDAO

\* **@return** ProfileDAO

\*/

**public** **static** ProfileDAO getProfileDAO() {

**if** (*profileDAO* == **null**) {

*profileDAO* = **new** ProfileDAO(*connection*.getConnection(), *categoryDAO*);

}

**return** *profileDAO*;

}

}

**package** cash;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** managers.AbstractEntity;

**import** java.sql.SQLException;

**import** java.util.HashMap;

**import** java.util.HashSet;

/\*\*

\* Cash class which allows increase performance of DAO pattern by

\* getting SELECT information from this cash without connection to the database

\* **@author** oleh

\*

\* **@param** <K> special types of keys

\* **@param** <V> special types of cash values

\*/

**public** **class** СacheDAO<K, V **extends** AbstractEntity<K>> **extends** DAO<V> {

**private** HashMap<K, V> cash;

**public** СacheDAO(HashSet<V> cash) {

**this**.cash = **new** HashMap<K, V>();

cash.forEach(v -> **this**.cash.put(v.getId(), v));

}

@Override

**public** HashSet<V> get(Condition<V> condition) {

HashSet<V> hs = **new** HashSet<>();

**for** ( V entity : cash.values())

**if**(condition.satisfyTo(entity))

hs.add(entity);

**return** hs;

}

@Override

**public** V create(V object) **throws** SQLException {

**this**.cash.put(object.getId(), object);

**return** object;

}

@Override

**public** V update(V object) **throws** SQLException {

**this**.cash.replace(object.getId(), **this**.cash.get(object.getId()), object);

**return** object;

}

@Override

**public** **boolean** delete(V object) **throws** SQLException {

**if**(**this**.cash.remove(object.getId(), object) == **false**)

**throw** **new** SQLException();

**return** **true**;

}

@Override

**public** V getById(Object id) **throws** SQLException {

**return** **this**.cash.get(id);

}

@Override

**protected** HashSet<V> getFromDatabase() **throws** SQLException {

**throw** **new** UnsupportedOperationException();

}

}

**package** managers;

/\*\*

\* This class should be inherited by everyone entity

\* **@author** oleh

\* **@param** <T> type of keys which uses with concrete entity

\*/

**public** **abstract** **class** AbstractEntity<T> {

/\*\*

\* Key of concrete entity object

\*/

**private** T id;

/\*\*

\* **@return** key

\*/

**public** T getId() {

**return** id;

}

/\*\*

\* **@param** id key

\*/

**protected** **void** setId(T id) {

**this**.id = id;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Category;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Category Class

\* Created by oleh on 26.11.14.

\*/

**public** **class** CategoryDAO **extends** DAO<Category> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** CacheDAO<Integer, Category> cash;

/\*\*

\* Constructor for category

\* **@param** connection is a pointer for our connection

\*/

**public** CategoryDAO(Connection connection ) {

**super**();

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO category"

+ "(nameOfCategory)"

+ "VALUES(?)");

**this**.update = connection.prepareStatement("UPDATE category SET "

+ "nameOfCategory = ? "

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM category WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Category> get(Condition<Category> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Category create(Category object) **throws** SQLException {

create.setString(2, object.getNameOfCategory());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM discount");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Category update(Category object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getNameOfCategory());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Category object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Category getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Category> getFromDatabase() **throws** SQLException {

HashSet<Category> categories = **new** HashSet<Category>();

ResultSet rs = statement.executeQuery("SELECT \* FROM category");

**while** (rs.next()) {

Category d = **new** Category(rs.getString(2));

d.setId(rs.getInt(1));

categories.add(d);

}

**return** categories;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Comment;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Comment Class

\* Created by oleh on 26.11.14.

\*/

**public** **class** CommentDAO **extends** DAO<Comment> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** CacheDAO<Integer, Comment> cash;

/\*\*

\* Constructor for comment

\* **@param** connection is a pointer for our connection

\*/

**public** CommentDAO(Connection connection ) {

**super**();

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO comment"

+ "(branchOfForum, dateOfChange, timeOfChange, textOfComment)"

+ "VALUES(?,?,?,?)");

**this**.update = connection.prepareStatement("UPDATE comment SET "

+ "branchOfForum = ?, dateOfChange = ?, timeOfChange = ?, textOfComment = ? "

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM comment WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Comment> get(Condition<Comment> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Comment create(Comment object) **throws** SQLException {

create.setString(2, object.getBranchOfForum());

create.setString(3, object.getDateOfChange());

create.setString(4, object.getTextOfComment());

create.setString(5, object.getTextOfComment());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM comment");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Comment update(Comment object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getBranchOfForum());

update.setString(3, object.getDateOfChange() );

update.setString(4, object.getTextOfComment() );

update.setString(5, object.getTextOfComment());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Comment object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Comment getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Comment> getFromDatabase() **throws** SQLException {

HashSet<Comment> comments = **new** HashSet<Comment>();

ResultSet rs = statement.executeQuery("SELECT \* FROM comment");

**while** (rs.next()) {

Comment d = **new** Comment(rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5));

d.setId(rs.getInt(1));

comments.add(d);

}

**return** comments;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Investment;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Investment Class

\* Created by oleh on 26.11.14.

\*/

**public** **class** InvestmentDAO **extends** DAO<Investment> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** ObjectDAO oObjectDAO;

**private** ProfileDAO profileDAO;

**private** CacheDAO<Integer, Investment> cash;

/\*\*

\* Constructor for investment

\* **@param** connection is a pointer for our connection

\*/

**public** InvestmentDAO(Connection connection, ObjectDAO objectDAO, ProfileDAO profileDAO ) {

**super**();

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO investment"

+ "(contractNumber, oobject, profile)"

+ "VALUES(?,?,?)");

**this**.update = connection.prepareStatement("UPDATE investment SET "

+ "contractNumber = ?, oobject = ?, profile = ?"

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM investment WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Investment> get(Condition<Investment> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Investment create(Investment object) **throws** SQLException {

create.setString(2, object.getContractNumber());

create.setInt(3, object.getOObject().getId());

create.setInt(4, object.getProfile().getId());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM investment");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Investment update(Investment object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getContractNumber() );

update.setInt(3, object.getOObject().getId());

update.setInt(4, object.getProfile().getId());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Investment object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Investment getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Investment> getFromDatabase() **throws** SQLException {

HashSet<Investment> investments = **new** HashSet<>();

ResultSet rs = statement.executeQuery("SELECT \* FROM investment");

**while** (rs.next()) {

Investment d = **new** Investment(profileDAO.getById(rs.getInt(4)), oObjectDAO.getById(rs.getInt(3)), rs.getString(2));

d.setId(rs.getInt(1));

investments.add(d);

}

**return** investments;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Message;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Message Class

\* Created by oleh on 26.11.14.

\*/

**public** **class** MessageDAO **extends** DAO<Message> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** ProfileDAO profileDAOsend;

**private** ProfileDAO profileDAOreceip;

**private** CacheDAO<Integer, Message> cash;

/\*\*

\* Constructor for message

\* **@param** connection is a pointer to our connection

\* **@param** profileDAOsend points to ProfileDAO of sender

\* **@param** profileDAOreceip points to ProfileDAO of recipient

\*/

**public** MessageDAO(Connection connection, ProfileDAO profileDAOsend, ProfileDAO profileDAOreceip ) {

**super**();

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO message"

+ "(themeOfMessage, dateOfSending, textOfSending, sender, recipient)"

+ "VALUES(?,?,?,?,?)");

**this**.update = connection.prepareStatement("UPDATE message SET "

+ "themeOfMessage = ?, dateOfSending = ?, textOfSending = ?, sender = ?, recipient = ?"

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM message WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Message> get(Condition<Message> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Message create(Message object) **throws** SQLException {

create.setString(2, object.getThemeOfMessage());

create.setDate(3, object.getDateOfSending());

create.setString(4, object.getTextOfMesssage());

create.setInt(5, object.getSender().getId());

create.setInt(6, object.getRecipient().getId());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM message");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Message update(Message object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getThemeOfMessage());

update.setDate(3, object.getDateOfSending());

update.setString(4, object.getTextOfMesssage());

update.setInt(5, object.getSender().getId());

update.setInt(6, object.getRecipient().getId());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Message object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Message getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Message> getFromDatabase() **throws** SQLException {

HashSet<Message> messages = **new** HashSet<>();

ResultSet rs = statement.executeQuery("SELECT \* FROM message");

**while** (rs.next()) {

Message d = **new** Message(rs.getString(2), rs.getDate(3), rs.getString(4), profileDAOsend.getById(rs.getInt(5)),

profileDAOreceip.getById(rs.getInt(6)));

d.setId(rs.getInt(1));

messages.add(d);

}

**return** messages;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.OObject;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to OObject Class

\* Created by oleh on 26.11.14.

\*/

**public** **class** ObjectDAO **extends** DAO<OObject> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** CacheDAO<Integer, OObject> cash;

/\*\*

\* Constructor for object

\* **@param** connection is a pointer to connection to DB

\*/

**public** ObjectDAO(Connection connection) {

**super**();

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO object"

+ "(name)"

+ "VALUES(?)");

**this**.update = connection.prepareStatement("UPDATE object SET "

+ "name = ?"

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM object WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<OObject> get(Condition<OObject> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** OObject create(OObject object) **throws** SQLException {

create.setInt(1, object.getId());

create.setString(2, object.getName());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM discount");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** OObject update(OObject object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getName());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(OObject object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** OObject getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<OObject> getFromDatabase() **throws** SQLException {

HashSet<OObject> objects = **new** HashSet<>();

ResultSet rs = statement.executeQuery("SELECT \* FROM object");

**while** (rs.next()) {

OObject d = **new** OObject(rs.getString(2));

d.setId(rs.getInt(1));

objects.add(d);

}

**return** objects;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Operation;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Operation Class

\* Created by OTER on 11/26/2014.

\*/

**public** **class** OperationDAO **extends** DAO<Operation> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** CacheDAO<Integer, Operation> cash;

**private** ProfileDAO profileDAO;

/\*\*

\* Constructor for operation

\* **@param** connection is a pointer to connection to DB

\* **@param** profileDAO points to ProfileDAO of the user

\*/

**public** OperationDAO(Connection connection, ProfileDAO profileDAO) {

**super**();

**this**.profileDAO = profileDAO;

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO operation"

+ "(typeOfOperation,sum,dateOfOperation,profile)"

+ "VALUES(?,?,?,?)");

**this**.update = connection.prepareStatement("UPDATE operation SET "

+ "typeOfOperation = ?,sum = ?,dateOfOperation = ?,profile = ?"

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM operation WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Operation> get(Condition<Operation> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Operation create(Operation object) **throws** SQLException {

create.setString(4, object.getTypeOfOperation());

create.setDouble(1, object.getSum());

create.setDate(2, object.getDateOfOperation());

create.setInt(3, object.getProfile().getId());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM operation");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Operation update(Operation object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setString(2, object.getTypeOfOperation());

update.setDouble(3, object.getSum());

update.setDate(4, object.getDateOfOperation());

update.setInt(1, object.getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Operation object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Operation getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Operation> getFromDatabase() **throws** SQLException {

HashSet<Operation> objects = **new** HashSet<>();

ResultSet rs = statement.executeQuery("SELECT \* FROM object");

**while** (rs.next()) {

Operation d = **new** Operation(rs.getString(2), rs.getInt(3), rs.getDate(4), profileDAO.getById( rs.getInt(5)) );

d.setId(rs.getInt(1));

objects.add(d);

}

**return** objects;

}

}

**package** managers;

**import** abstractions.Condition;

**import** abstractions.DAO;

**import** cash.CacheDAO;

**import** tableClasses.Profile;

**import** java.sql.\*;

**import** java.util.HashSet;

/\*\*

\* implements pattern DAO to Profile Class

\* Created by OTER on 11/26/2014.

\*/

**public** **class** ProfileDAO **extends** DAO<Profile> {

**private** Connection connection;

**private** Statement statement;

**private** PreparedStatement update;

**private** PreparedStatement create;

**private** PreparedStatement delete;

**private** CacheDAO<Integer, Profile> cash;

**private** CategoryDAO categoryDAO;

/\*\*

\* Constructor for object

\* **@param** connection is a pointer to connection to DB

\*

\* **@param** connection to which we connect

\* **@param** categoryDAO refers to the line in the table

\*/

**public** ProfileDAO(Connection connection, CategoryDAO categoryDAO) {

**super**();

**this**.categoryDAO = categoryDAO;

//this.profileDAO = profileDAO;

**this**.connection = connection;

**try** {

**this**.statement = connection.createStatement();

**this**.create = connection.prepareStatement("INSERT INTO profile"

+ "(login,password,category)"

+ "VALUES(?,?,?)");

**this**.update = connection.prepareStatement("UPDATE profile SET "

+ "login = ?,password = ?,category = ?"

+ "WHERE id = ?");

**this**.delete = connection.prepareStatement("DELETE FROM profile WHERE id = ?");

//load into cash

**this**.cash = **new** CacheDAO<>(getFromDatabase());

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* Getting method for already read categories

\* **@param** condition

\* **@return** HashSet of certain class

\* **@throws** SQLException if happened something wrong

\*/

@Override

**public** HashSet<Profile> get(Condition<Profile> condition)

**throws** SQLException {

**return** cash.get(condition);

}

/\*\*

\* create a line with certain name

\* **@param** object which to add to the table

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Profile create(Profile object) **throws** SQLException {

create.setInt(1, object.getId());

create.setString(2, object.getLogin());

create.setString(3, object.getPassword());

create.setInt(4, object.getCategory().getId());

**if** (create.executeUpdate() != 1) {

**throw** **new** IllegalStateException();

}

ResultSet rs = statement.executeQuery("SELECT Max(id) FROM profile");

rs.next();

object.setId(rs.getInt(1));

//create in cash

cash.create(object);

**return** object;

}

/\*\*

\* update the certain line in the table

\*

\* **@param** object what we add to the table instead of what we have

\* **@return** improved object

\* **@throws** SQLException

\*/

@Override

**public** Profile update(Profile object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

update.setInt(1, object.getId());

update.setString(2, object.getLogin());

update.setString(3, object.getPassword());

update.setInt(4, object.getCategory().getId());

**if** (update.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//update in cash

cash.update(object);

**return** object;

}

/\*\*

\* delete the object from the table

\*

\* **@param** object what to delete

\* **@return** true if deleted

\* **@throws** SQLException

\*/

@Override

**public** **boolean** delete(Profile object) **throws** SQLException {

**if** (object.getId() == -1)

**throw** **new** IllegalStateException();

delete.setInt(1, object.getId());

**if** (delete.executeUpdate() != 1)

**throw** **new** IllegalStateException();

//delete from cash

**return** cash.delete(object);

}

/\*\*

\* get element by id

\*

\* **@param** id of the object

\* **@return** the element due to it id

\* **@throws** SQLException

\*/

@Override

**public** Profile getById(Object id) **throws** SQLException {

**return** cash.getById(id);

}

/\*\*

\* get the elements from the data base

\*

\* **@return** HashSet of the certain class

\* **@throws** SQLException

\*/

@Override

**protected** HashSet<Profile> getFromDatabase() **throws** SQLException {

HashSet<Profile> objects = **new** HashSet<>();

ResultSet rs = statement.executeQuery("SELECT \* FROM object");

**while** (rs.next()) {

Profile d = **new** Profile( rs.getString(2),rs.getString(3), categoryDAO.getById( rs.getInt(4) ) );

d.setId(rs.getInt(1));

objects.add(d);

}

**return** objects;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

/\*\*

\* represent the tableClasses.Category table

\* Created by oleh on 25.11.14.

\*/

**public** **class** Category **extends** AbstractEntity<Integer> {

**private** String nameOfCategory;

**public** Category(String nameOfCategory) {

**this**.nameOfCategory = nameOfCategory;

}

/\*\*

\* get name of category

\* **@return** field nameOfCategory

\*/

**public** String getNameOfCategory() {

**return** nameOfCategory;

}

/\*\*

\* set name of category

\* **@param** nameOfCategory what to set for the field

\*/

**public** **void** setNameOfCategory(String nameOfCategory) {

**this**.nameOfCategory = nameOfCategory;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

/\*\*

\* represent the comment table

\* Created by oleh on 25.11.14.

\*/

**public** **class** Comment **extends** AbstractEntity<Integer> {

**private** String branchOfForum;

**private** String dateOfChange;

**private** String timeOfChange;

**private** String textOfComment;

**public** Comment(String branchOfForum, String dateOfChange, String timeOfChange, String textOfComment) {

**this**.branchOfForum = branchOfForum;

**this**.dateOfChange = dateOfChange;

**this**.timeOfChange = timeOfChange;

**this**.textOfComment = textOfComment;

}

/\*\*

\* get the String value of branchOfForum

\* **@return** branchOfForum

\*/

**public** String getBranchOfForum() {

**return** branchOfForum;

}

/\*\*

\* set the String value for field branchOfForum

\* **@param** branchOfForum to add

\*/

**public** **void** setBranchOfForum(String branchOfForum) {

**this**.branchOfForum = branchOfForum;

}

/\*\*

\* get the String value of dateOfChange

\* **@return** dateOfChange

\*/

**public** String getDateOfChange() {

**return** dateOfChange;

}

/\*\*

\* set the String value of dateOfChange

\* **@param** dateOfChange

\*/

**public** **void** setDateOfChange(String dateOfChange) {

**this**.dateOfChange = dateOfChange;

}

/\*\*

\* get the time of change of the comment

\* **@return** timeOfChange

\*/

**public** String getTimeOfChange() {

**return** timeOfChange;

}

/\*\*

\* set the time of change of the comment

\* **@param** timeOfChange which to set

\*/

**public** **void** setTimeOfChange(String timeOfChange) {

**this**.timeOfChange = timeOfChange;

}

/\*\*

\* get the text in the comment

\* **@return** textOfComment

\*/

**public** String getTextOfComment() {

**return** textOfComment;

}

/\*\*

\* set the text of the comment

\* **@param** textOfComment which to set to the field

\*/

**public** **void** setTextOfComment(String textOfComment) {

**this**.textOfComment = textOfComment;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

/\*\*

\* represent investment table in the database

\* Created by oleh on 25.11.14.

\*/

**public** **class** Investment **extends** AbstractEntity<Integer>{

**private** Profile profile;

**private** OObject oobject;

**private** String contractNumber;

/\*\*

\* constructor

\*

\* **@param** profile for which investment belongs to

\* **@param** oobject which object

\* **@param** contractNumber number of contract

\*/

**public** Investment(Profile profile, OObject oobject, String contractNumber) {

**this**.profile = profile;

**this**.oobject = oobject;

**this**.contractNumber = contractNumber;

}

/\*\*

\* get the Profile object

\* **@return** profile field

\*/

**public** Profile getProfile() {

**return** profile;

}

/\*\*

\* set profile to the field

\* **@param** profile which field to set

\*/

**public** **void** setProfile(Profile profile) {

**this**.profile = profile;

}

/\*\*

\* getter for oobject

\* **@return** oobject

\*/

**public** OObject getOObject() {

**return** oobject;

}

/\*\*

\* setter for oobject

\* **@param** oobject which oobject to set

\*/

**public** **void** setOObject(OObject oobject) {

**this**.oobject = oobject;

}

/\*\*

\* getter for contract number

\* **@return** contractNumber

\*/

**public** String getContractNumber() {

**return** contractNumber;

}

/\*\*

\* setter for contract number

\* **@param** contractNumber which to set

\*/

**public** **void** setContractNumber(String contractNumber) {

**this**.contractNumber = contractNumber;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

**import** java.sql.Date;

/\*\*

\* represent message table in the database

\* Created by oleh on 25.11.14.

\*/

**public** **class** Message **extends** AbstractEntity<Integer> {

**private** String themeOfMessage;

**private** Date dateOfSending;

**private** String textOfMessage;

**private** Profile sender;

**private** Profile recipient;

/\*\*

\* constructor for Message

\*

\* **@param** themeOfMessage the theme of the message

\* **@param** dateOfSending date of sending

\* **@param** textOfMessage the text of the message

\* **@param** sender profile of a sender

\* **@param** recipient profile of a recipient

\*/

**public** Message(String themeOfMessage, Date dateOfSending, String textOfMessage, Profile sender, Profile recipient) {

**this**.themeOfMessage = themeOfMessage;

**this**.dateOfSending = dateOfSending;

**this**.textOfMessage = textOfMessage;

**this**.sender = sender;

**this**.recipient = recipient;

}

/\*\*

\* getter for theme of the message

\* **@return** themeOfMessage

\*/

**public** String getThemeOfMessage() {

**return** themeOfMessage;

}

/\*\*

\* setter for theme of the message

\* **@param** themeOfMessage which to set for the field

\*/

**public** **void** setThemeOfMessage(String themeOfMessage) {

**this**.themeOfMessage = themeOfMessage;

}

/\*\*

\* getter for date of Sending of the message

\* **@return** dateOfSending

\*/

**public** Date getDateOfSending() {

**return** dateOfSending;

}

/\*\*

\* setter for date of sending

\* **@param** dateOfSending which to set for the field

\*/

**public** **void** setDateOfSending(Date dateOfSending) {

**this**.dateOfSending = dateOfSending;

}

/\*\*

\* getter for text of message

\* **@return** textOfMessage

\*/

**public** String getTextOfMesssage() {

**return** textOfMessage;

}

/\*\*

\* setter for text of the message

\* **@param** textOfMesssage which to set for the field

\*/

**public** **void** setTextOfMesssage(String textOfMesssage) {

**this**.textOfMessage = textOfMesssage;

}

/\*\*

\* getter for Profile sender

\* **@return** sender

\*/

**public** Profile getSender() {

**return** sender;

}

/\*\*

\* setter for Profile field sender

\* **@param** sender who send a message

\*/

**public** **void** setSender(Profile sender) {

**this**.sender = sender;

}

/\*\*

\*

\* **@return**

\*/

**public** Profile getRecipient() {

**return** recipient;

}

/\*\*

\* setter for Profile field recipient

\* **@param** recipient who receive a message

\*/

**public** **void** setRecipient(Profile recipient) {

**this**.recipient = recipient;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

/\*\*

\* represent object table in the database

\* Created by oleh on 25.11.14.

\*/

**public** **class** OObject **extends** AbstractEntity<Integer> {

**private** String name;

/\*\*

\* constructor

\*

\* **@param** name name of the object

\*/

**public** OObject(String name) {

**super**();

setId(-1);

**this**.name = name;

}

/\*\*

\* getter for name

\* **@return** name of the object

\*/

**public** String getName() {

**return** name;

}

/\*\*

\* setter for name

\* **@param** name which name to set

\*/

**public** **void** setName(String name) {

**this**.name = name;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

**import** java.sql.Date;

/\*\*

\* represent operation table in the database

\* Created by oleh on 25.11.14.

\*/

**public** **class** Operation **extends** AbstractEntity<Integer> {

**private** Profile profile;

**private** String typeOfOperation;

**private** **double** sum;

**private** Date dateOfOperation;

/\*\*

\* constructor

\*

\* **@param** typeOfOperation name of the type of the operation

\* **@param** sum sum of money for operation

\* **@param** dateOfOperation date the operation was completed

\* **@param** profile to which the operation belongs to

\*/

**public** Operation(String typeOfOperation, **double** sum, Date dateOfOperation, Profile profile ) {

**this**.profile = profile;

**this**.typeOfOperation = typeOfOperation;

**this**.sum = sum;

**this**.dateOfOperation = dateOfOperation;

}

/\*\*

\* getter for dateOfOperation

\* **@return** value of field dateOfOperation

\*/

**public** Date getDateOfOperation() {

**return** dateOfOperation;

}

/\*\*

\* setter for dateOfOperation

\* **@param** dateOfOperation which to set

\*/

**public** **void** setDateOfOperation(Date dateOfOperation) {

**this**.dateOfOperation = dateOfOperation;

}

/\*\*

\* getter for profile

\* **@return** profile

\*/

**public** Profile getProfile() {

**return** profile;

}

/\*\*

\* setter for profile

\* **@param** profile which to set

\*/

**public** **void** setProfile(Profile profile) {

**this**.profile = profile;

}

/\*\*

\* getter for type of the operation

\* **@return** typeOfOperation

\*/

**public** String getTypeOfOperation() {

**return** typeOfOperation;

}

/\*\*

\* setter for type of the operation

\* **@param** typeOfOperation which to set

\*/

**public** **void** setTypeOfOperation(String typeOfOperation) {

**this**.typeOfOperation = typeOfOperation;

}

/\*\*

\* getter for sum

\* **@return** sum

\*/

**public** **double** getSum() {

**return** sum;

}

/\*\*

\* setter for field sum

\* **@param** sum which to set

\*/

**public** **void** setSum(**double** sum) {

**this**.sum = sum;

}

}

**package** tableClasses;

**import** managers.AbstractEntity;

/\*\*

\* represent profile table in the database

\* Created by oleh on 25.11.14.

\*/

**public** **class** Profile **extends** AbstractEntity<Integer>{

**private** String login;

**private** String password;

**private** Category category;

/\*\*

\* constructor

\*

\* **@param** login of the user

\* **@param** password of the user

\* **@param** category investor or contact-manager

\*/

**public** Profile(String login, String password, Category category) {

**this**.login = login;

**this**.password = password;

**this**.category = category;

}

/\*\*

\* getter for login

\* **@return** login

\*/

**public** String getLogin() {

**return** login;

}

/\*\*

\* setter for login

\* **@param** login which to set

\*/

**public** **void** setLogin(String login) {

**this**.login = login;

}

/\*\*

\* getter for password

\* **@return** password

\*/

**public** String getPassword() {

**return** password;

}

/\*\*

\* setter for password

\* **@param** password which to set

\*/

**public** **void** setPassword(String password) {

**this**.password = password;

}

/\*\*

\* getter for category

\* **@return** category

\*/

**public** Category getCategory() {

**return** category;

}

/\*\*

\* setter for category

\* **@param** category which to set

\*/

**public** **void** setCategory(Category category) {

**this**.category = category;

}

}

**package** main;

**import** access.Connector;

**import** access.DAOFactory;

**import** managers.CategoryDAO;

/\*\*

\* Created by oleh on 19.11.14.

\*/

**public** **class** Main {

**public** **static** **void** main(String[] args) **throws** Exception {

DAOFactory daoFactory = **new** DAOFactory();

}

}