Змі

[Вступ 4](#_Toc388954826)

[1 Огляд MVC 5](#_Toc388954827)

[1.1 Загальний огляд 5](#_Toc388954828)

[1.2 Історія 6](#_Toc388954829)

[1.3 Призначення 7](#_Toc388954830)

[1.4 Концепція 7](#_Toc388954831)

[1.5 Найчастіші помилки 9](#_Toc388954832)

[1.6 Реалізація 10](#_Toc388954833)

[1.6.1 Java 10](#_Toc388954834)

[2 Проектування програмного забезпечення 11](#_Toc388954835)

[2.1 Прецеденти 11](#_Toc388954836)

[2.1.1 Авторизація у системі 11](#_Toc388954837)

[2.1.2 Створення нового акаунту 11](#_Toc388954838)

[2.1.3 Видалення акаунту 12](#_Toc388954839)

[2.1.4 Редагування акаунту 12](#_Toc388954840)

[2.1.5 Перегляд розкладу групи 13](#_Toc388954841)

[2.1.6 Перегляд історії активності користувачів 13](#_Toc388954842)

[2.1.7 Редагування розкладу групи 14](#_Toc388954843)

[2.1.8 Вихід із системи 14](#_Toc388954844)

[2.1.9 Перегляд посібника користувача 15](#_Toc388954845)

[2.2 Сценарії використання програми 15](#_Toc388954846)

[2.2.1 Отримання інформації про профілі 15](#_Toc388954847)

[2.2.2 Створення нової категорії у БД 16](#_Toc388954848)

[2.2.3 Налаштування програми 16](#_Toc388954849)

[2.2.4 Отримання загальної інформації про програму. 16](#_Toc388954850)

[2.3 Діаграма граничних класів 16](#_Toc388954851)

[2.4 Проектування графічного інтерфейсу користувача 17](#_Toc388954852)

[2.5 Таблиця відповідності елементів інтерфейсу 20](#_Toc388954853)

[3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ 21](#_Toc388954854)

[3.1 Структура програмного додатку 21](#_Toc388954855)

[3.2 Специфікація класів програмного додатка 21](#_Toc388954856)

[3.2.1 Package com.Oyster.app.model 21](#_Toc388954860)

[3.2.2 Package com.Oyster.config 37](#_Toc388954863)

[3.2.3 Package com.Oyster.core.controller 43](#_Toc388954864)

[3.2.4 Package com.Oyster.core.controller.command 47](#_Toc388954865)

[3.2.5 Package com.Oyster.dao 51](#_Toc388954868)

[3.2.6 com.Oyster.ui 56](#_Toc388954869)

[3.3 Інструкція для користувача 63](#_Toc388954870)

[4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ 65](#_Toc388954871)

[ВИСНОВКИ 69](#_Toc388954872)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70](#_Toc388954873)

[ДОДАТОК А. ДІАГРАМИ КЛАСІВ ДОДАТКУ 72](#_Toc388954874)

[ДОДАТОК Б. ВИХІДНИЙ КОД ДОДАТКУ 76](#_Toc388954875)

# Вступ

Курсова робота спрямована на закріпення знань та отримання навичок із розробки програмних додатків на Java із використанням бібліотеки Swing. У курсовій роботі я розроблю програмний додаток згідно вимог MVC для управління навчальним процесом в університеті (реалізувати функціонал для адміністратора).

У роботі розгляну основні принципи MVC, принципи побудови графічного інтерфейсу користувача, основні елементи інтерфейсу, організацію обробки подій, систему відлову помилок і організацію багатопоточності.

Робота міститиме повну документацію, а також програмний код проекту та UML діаграму класів.

# Огляд MVC

## Загальний огляд

**Модель-вид-контролер** (або **Модель-представлення-контролер** [англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Model-view-controller*, **MVC**) — [архітектурний шаблон](http://uk.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%96_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), який використовується під час проектування та розробки [програмного забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F).



Рис 1.1 Концепція Model-View-Controller

Цей шаблон поділяє систему на три частини: [модель даних](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), вигляд даних та [керування](http://uk.wikipedia.org/wiki/%D0%9A%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). Застосовується для відокремлення даних (модель) від [інтерфейсу користувача](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0)(вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах призводить до певної впорядкованості їх структури і робить їх зрозумілішими завдяки зменшенню складності.

Дана схема проектування часто використовується для побудови архітектурного каркасу, коли переходять від теорії до реалізації в конкретній предметній області.

## Історія

Концепція MVC була описана у 1979 році Трюгве Реенскауг (англ. Trygve Reenskaug), який тоді працював над мовою програмування Smalltalk в Xerox PARC. Оригінальна реалізація описана в статті «Applications Programming in Smalltalk-80: How to use Model-View-Controller». Потім Джим Алтоф з командою розробників реалізували версію MVC для бібліотеки класів Smalltalk-80.

В оригінальній концепції була описана сама ідея і роль кожного із елементів: моделі, представлення і контролера. Та зв’язки між ними були описані без конкретизації. Крім того, розрізняли дві основні модифікації:

* + 1. Пасивна модель – модель не має ніяких способів впливу на представлення чи контролер і використовується в якості джерела даних для відображення. Всі зміни моделі відстежуються контролером і він же відповідає за перемальовування представлення, якщо це необхідно. Така модель частіше всього використовується в структурному програмуванні, оскільки у цьому випадку модель являє собою просто структуру даних, без методів їх обробки.
    2. Активна модель – модель сповіщає представлення про те, що в ній виникли зміни, а представлення, які зацікавленні в сповіщенні, підписуються на ці сповіщення. Це дозволяє зберегти незалежність моделі як від контролера, так і від представлення.

Класичною реалізацією концепції MVC вважають версію з активною моделлю.

У даній роботі використаємо класичну модель (Активну), оскільки вона дозволяє значно краще зберегти незалежність моделі від контролера та представлення, а також спростить реалізацію та забезпечить гнучкість системи.

Із розвитком Об’єктно-орієнтованого програмування і поняття про шаблони проектування був створений ряд модифікацій концепції MVC, які при реалізації у різних авторів можуть відрізнятися від оригінальної. Так, наприклад, Еріан Вермі в 2004 році писав приклад узагальненого MVC.

## Призначення

Основна мета використання цієї концепції у розділенні бізнес-логіки (моделі) від її візуалізації(представлення, вигляду). За рахунок такого розмежування підвищується можливість повторного використання. Найбільш корисне застосування даної теорії у тих випадках, коли користувач повинен бачити ті ж самі дані одночасно у різних контекстах та/або з різних точок зору. Виконуються наступні задачі:

1. До однієї моделі можна приєднати декілька видів, при цьому на чіпаючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді електронної таблиці, гістограми і кругової діаграми.
2. Не чіпаючи реалізацію вигляду, можна змінити реакції на дії користувача(натиснення мишкою на копку, введення даних), для цього достатньо використовувати інший контролер.
3. Ряд розробників спеціалізуються лише в одній із областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо досягнути того, що програмісти, які займаються розробкою бізнес-логіки(моделі), взагалі не будуть обізнані в тому, яке представлення використовуватиметься.

## Концепція

Концепція MVC дозволяє розділити дані, представлення і обробку дій користувача на три окремих компоненти:

1. Модель(англ. Model) Модель представляє інформацію: дані і методи роботи з цими даними, реагує на запити, змінюючи свій стан. Не містить інформації, про те, як ці знання можна візуалізувати.
2. Представлення, вигляд (англ. View). Відповідає за відображення інформації(візуалізацію). Часто в якості представлення виступає форма (вікно) з графічними елементами.
3. Контролер (англ. Controller). Забезпечує зв’язок між користувачем і системою: контролює введення даних користувачем і використовує модель і представлення для реалізації необхідної реакції, керує компонентами, отримує сигнали у вигляді реакції на дії користувача, і повідомляє про зміни компоненту Модель.

Така внутрішня структура в цілому поділяє систему на самостійні частини і розподіляє відповідальність між різними компонентами.

Важливо зазначити, що як представлення, так і контролер залежать від моделі. Однак модель не залежить ні від представлення, ні від контролера. Тим самим досягається призначення такого розмежування: воно дозволяє будувати модель незалежно від візуального представлення, а також створювати декілька різних представлень для однієї моделі

Для реалізації схеми Model-View-Controller використовується достатньо велике число шаблонів проектування(в залежності від складності архітектурного рішення), основні з яких стратегія, композит, спостерігач.

Найбільш типова реалізація відділяє вигляд від моделі шляхом встановлення між ними протоколу взаємодії, використовуючи апарат подій (підписка/сповіщення). При кожній зміні внутрішніх даних в моделі вона сповіщає всі залежні від неї представлення і представлення обновлюється. Для цього використовують шаблон спостерігач. При обробці реакції користувача вигляд обирає, в залежності від потрібної реакції, потрібний контролер, який забезпечує той чи інший зв’язок з моделлю. Для цього використовується шаблон стратегія, або замість цього може бути модифікація із використанням шаблону команда. А для можливості однотипної роботи з підоб’єктами складно-скомпонованого ієрархічного виду може використовуватись шаблон композит. Крім того, можуть використовуватись і інші шаблони проектування, наприклад, фабричний метод, який дозволить задати за замовчуванням тип контролера для відповідного виду.

## Найчастіші помилки

Програмісти-початківці (особливо в веб-програмуванні, де абревіатура MVC стала популярною) дуже часто трактують архітектурну модель MVC як пасивну модель MVC. У цьому випадку модель виступає виключно сукупністю функцій для доступу до даних, а контролер містить бізнес логіку. В результаті код моделі по факту являється засобом для отримання даних із СУБД, а контролер являє собою типовий модуль, наповнений бізнес-логікою, або скрипт у термінології веб-програміста. В результаті такого розуміння MVC розробники стали писати код, який Pádraic Brady, відомий у колах спільноти Zend Framework, охарактеризував як ТТПК – «Товсті тупі потворні контролери» (Fat Stupid Ugly Controllers)

Середньостатистичний ТТПК отримував дані із БД (використоруючи рівень абстракції бази даних, враховуючи, що ще модель) або маніпулював, перевіряв, записував, а також передавав дані у вигляд. Такий підхід став дуже популярним тому, що використання таких контролерів схоже на класичну практику використання окремого php-файлу для кожної сторінки додатку.

Але в об’єктно-орієнтованому програмуванні використовується активна модель MVC, де модель – це не тільки сукупність коду доступу до даних і СУБД, але і вся бізнес логіка. Варто зазначити про можливості моделі інкапсулювати в собі інші моделі. В свою чергу контролери являють собою лише елементи системи, в безпосередні обов’язки яких входить прийом даних із запиту і передача їх іншим елементам системи. Тільки у цьому випадку контролер стає «тонким» і виконує виключно функції зв’язкової ланки(glue layer) між певними компонентами системи.

## Реалізація

Концепція MVC вперше застосувалася при проектуванні мови програмування Smalltalk як модель для інтерфейсу користувача . Також в область застосування концепції входить реалізація каркаса Документ-Вид (Document-View) в рамках бібліотеки MFC для мови Visual C++ . У сучасних технологіях концепція MVC представлена схемою JSP Model 1/2 для динамічної обробки Web-змісту на основі Java Server Pages (JSP).

### Java

У мові програмування Java концепція MVC підтримується на рівні стандартних класів-бібліотек. В результаті використання парадигми MVC програміст отримує в своє розпорядження могутню структуру об'єктів-компонентів, функції яких чітко розмежовані, що гарантує надійність і розширюваність системи, що розробляється.

# Проектування програмного забезпечення

## Прецеденти

### Авторизація у системі

Адміністратор може увійти до системи, заповнивши поля “Логін” та “Пароль” та натиснувши кнопку “Вхід” (Рис. 2.1).

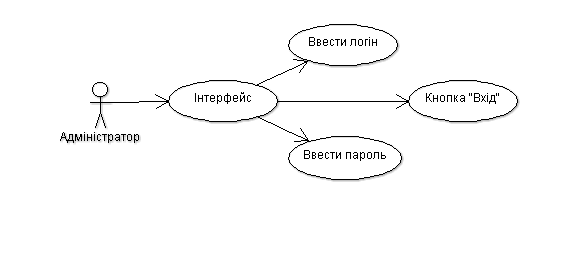
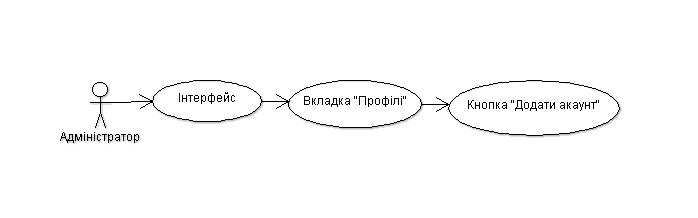


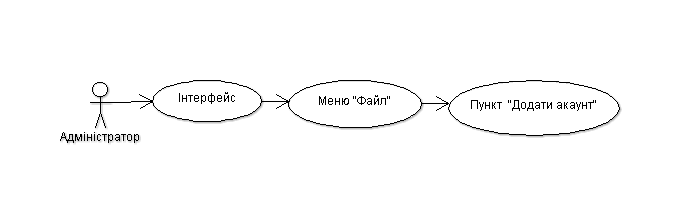
Рис. 2.1 Діаграма прецеденту операції входу до системи

### Створення нового акаунту

Адміністратор може додати нови акаунт, натиснувши кнопку “Додати акаунт у вкладці “Профілі” (Рис. 2.2.а) або у меню “Файл”, натиснувши підпункт “Додати акаунт” (Рис. 2.2.б), або із допомогою комбінації клавіш Alt + N (Рис. 2.2.в)



а) створення акаунту через вкладку “Профілі”



б) створення акаунту чарез головне меню



в) створення акаунту із допомогою комбінації клавіш Alt + N

Рис. 2.2 Діаграма прецедентів операції створення акаунту

### Видалення акаунту

Адміністратор може видалити акаунт, вибравши його та натиснувши кнопку “Видалити” (Рис. 2.3.а)

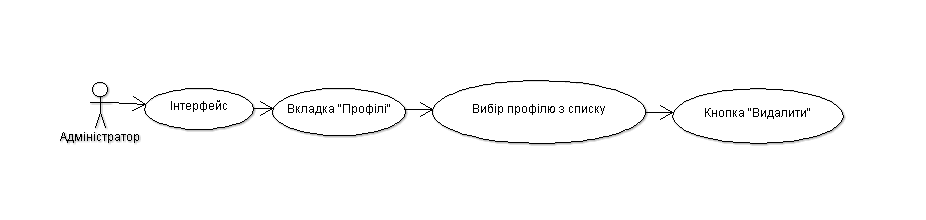


Рис. 2.3 Діаграма прецеденту операції видалення акаунту

### Редагування акаунту

Адміністратор може редагувати акаунт, вибравши його, змінивши поля та натиснувши кнопку “Зберегти” (Рис. 2.4)

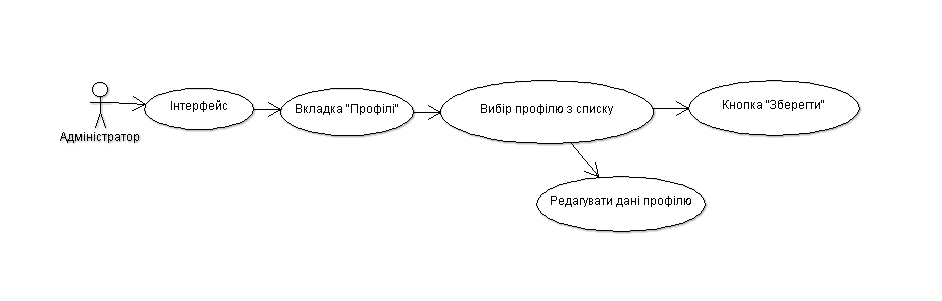


Рис. 2.4 Діаграма прецеденту операції редагування акаунту

### Перегляд розкладу групи

Адміністратор може переглянути розклад групи, перейшовши у вкладку “Розклад”, вибравши групу та предмет із списку (Рис. 2.5).



Рис. 2.5 Діаграма прецеденту перегляду розкладу групи

### Перегляд історії активності користувачів

Адміністратор може переглянути історію активності всіх користувачі, перейшовши у вкладку “Історія” вибравши пункт “Повна” (Рис. 2.6)



Рис. 2.6 Діаграма прецеденту операції перегляду історії

### Редагування розкладу групи

Адміністратор може редагувати розклад групи, здійснивши прецедент перегляду розкладу групи (Рис 2.5), редагувавши поля розкладу та натиснувиши кнопку “Зберегти”(Рис. 2.7).

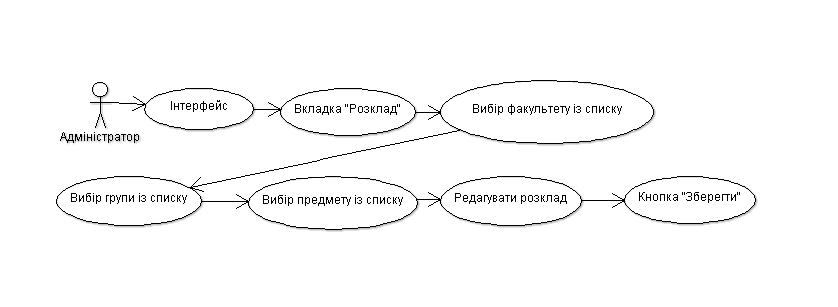


Рис. 2.7 Діаграма прецеденту операції редагування розкладу

### Вихід із системи

Адміністратор може вийти із системи, натиснувши пункт “Вихід” у меню “Файл” (Рис. 2.8.а,в) або із допомогою комбінації клавіш Alt + W (Рис. 2.8.б).



а) вихід із системи через головне меню



б) вихід із системи або із допомогою комбінації клавіш Alt + W

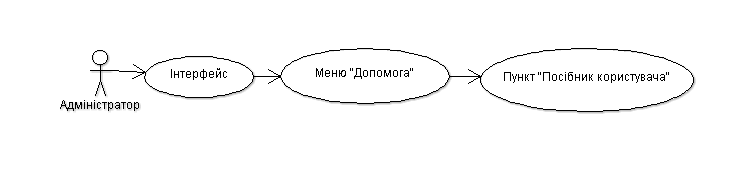


в) вихід із системи із допомогою комбінації клавіш Alt + F та пункту меню

Рис. 2.8 Діаграма прецедентів операції виходу із системи

### Перегляд посібника користувача

Адміністратор може переглянути посібник користувача, натиснувши пункт “Посібник користувача” у меню “Допомога” (Рис. 2.9.а) або із допомогою комбінації клавіш Alt + Н (Рис. 2.9.б) .



а) перегляд посібника користувача із допомогою головного меню



б) перегляд посібника користувача із допомогою комбінації клавіш Alt + H

Рис. 2.9 Діаграма прецедентів операції перегляду посібника користувача

## Сценарії використання програми

### Отримання інформації про профілі

Передумова: програма запущена, адміністратор пройшов аутентифікацію.

1. Користувач переходить на вкладку “Профілі”.
2. Програма надає список профілів відсортованих по категоріям.
3. Користувач виділяє певний профіль.
4. Програма відображає інформацію про нього.

### Створення нової категорії у БД

Передумова: програма запущена, адміністратор пройшов аутентифікацію.

1. Користувач натискає на кнопку “Додати акаунт”.
2. Програма відображає типи акаунтів.
3. Користувач вибирає один із типів.
4. Програма відображає діалогове вікно.
5. Користувач заповнює всі необхідні поля, натискає на кнопку “Створити”.

### Налаштування програми

Передумова: програма запущена, користувач пройшов аутентифікацію.

1. Користувач натискає на меню “Налаштування”.
2. Програма зображує пункти меню “Налаштування”.
3. З'являється вікно з налаштуваннями.
4. Користувач редагує налаштування, натискає на кнопку “Зберегти”.

### Отримання загальної інформації про програму.

Передумова: програма запущена, користувач пройшов аутентифікацію.

1. Користувач натискає на меню “Допомога”.
2. Програма відображає пункти меню “Допомога”.
3. Користувач натискає на пункт “Про програму”.
4. Програма відображає загальну інформацію про додаток.

## Діаграма граничних класів

На рис. 2.10 зображена діаграма граничних класів програмного додатку “KPI City”.

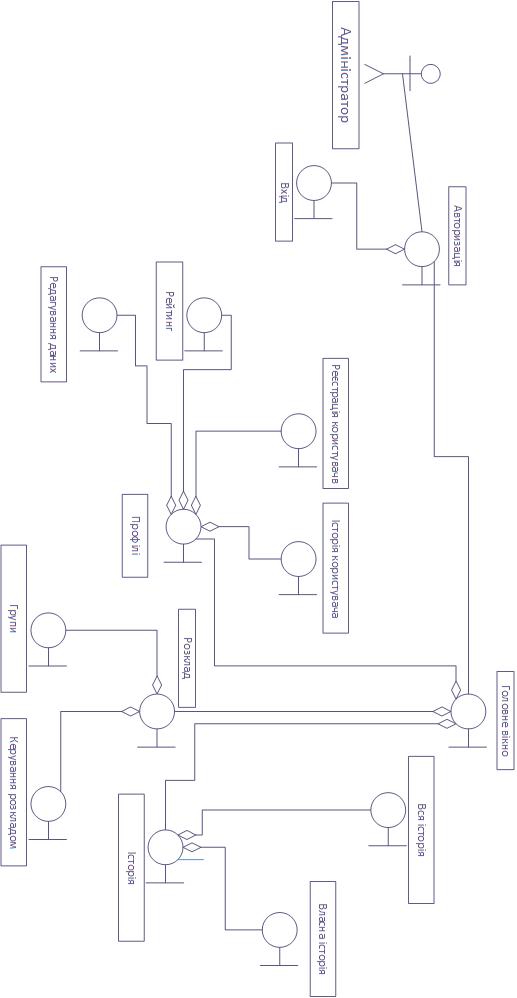


Рис. 2.10 Діаграма граничних класів

## Проектування графічного інтерфейсу користувача

На рисунках 2.3.1 зображено ескізи моделей графічного інтерфейсу програмного додатку “KPI City”. На ньому виділено області для розташування всіх елементів. Графічний інтерфейс розроблявся таким чином, щоб адміністратор мав можливість швидко і легко знайти потрібну йому інформацію чи вибрати та здійснити операцію. Також при розробці інтерфейсу брались до уваги праці [8, 9], де автори описують тонкі аспекти побудови моделі вікна програмного додатку, які суттєво впливають на його привабливість і легкість сприйняття.

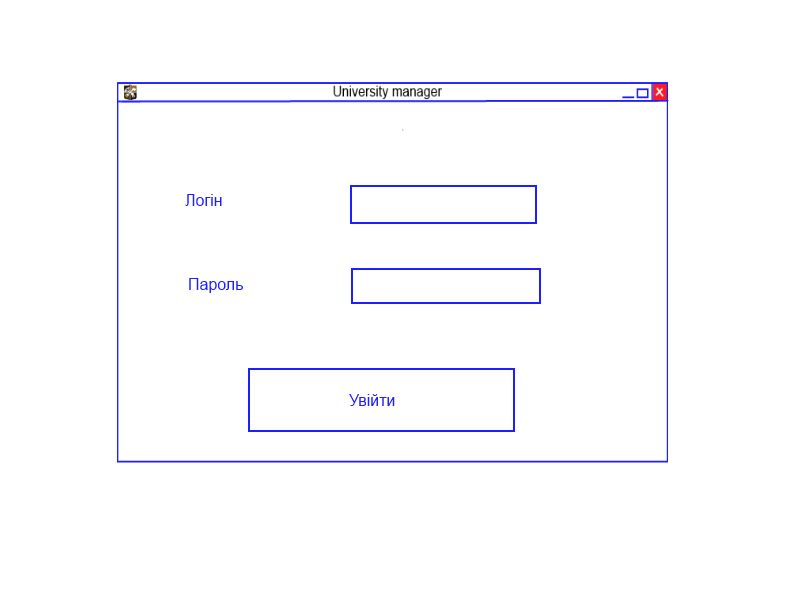


Рис 2.11 Ескіз вікна авторизації програми

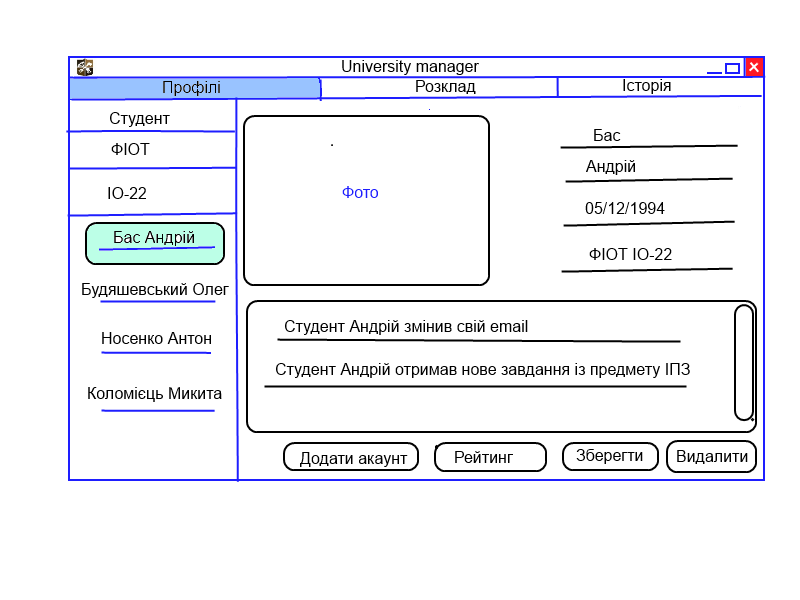


Рис 2.12 Ескіз вкладки “Профілі” головного вікна програми

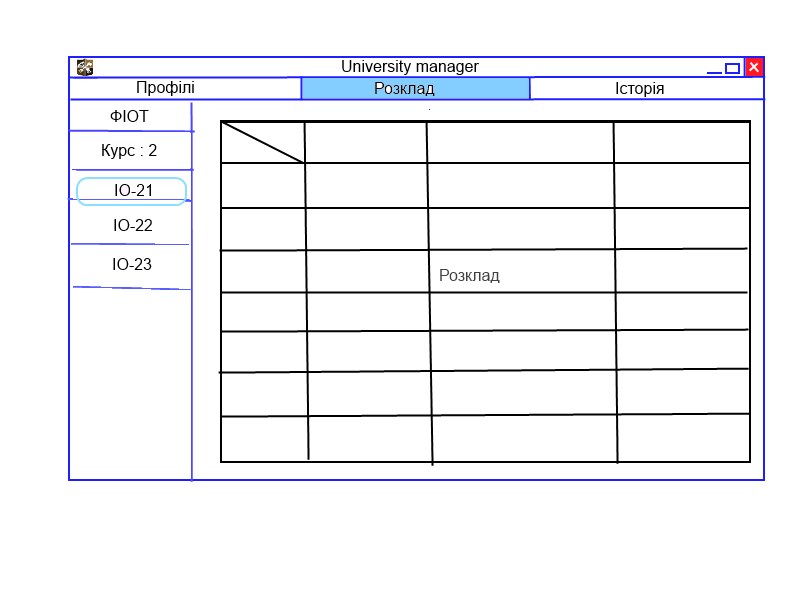


Рис 2.13 Ескіз вкладки “Розклад” головного вікна програми

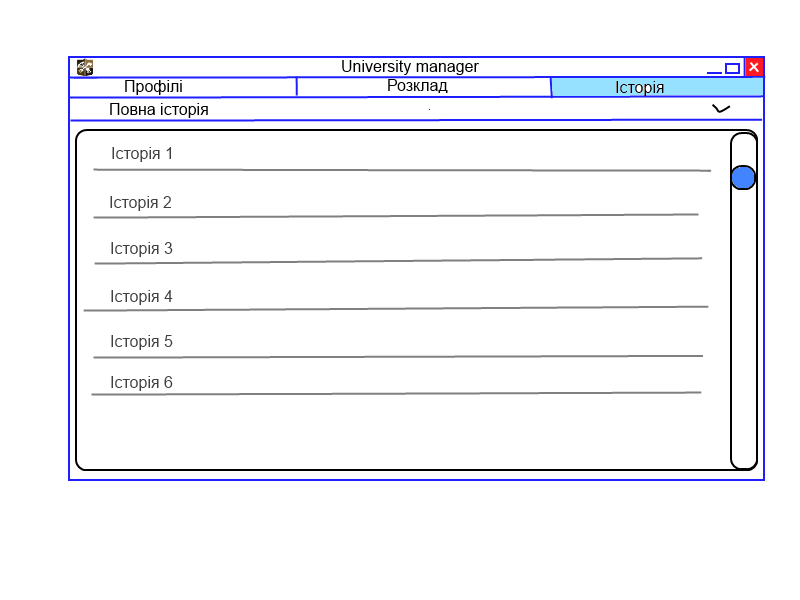


Рис 2.14 Ескіз вкладки “Історія” головного вікна програми

## Таблиця відповідності елементів інтерфейсу

У таблиці 2.1 показані відповідності елементів ескізу графічного інтерфейсу рисунків 2.11 - 2.14 та класів пакету Swing, що їх реалізують.

Таблиця 2.1. Відповідність елементів інтерфейсу до класів

|  |  |
| --- | --- |
| ***Елемент інтерфейсу*** | ***Класи пакету java.swing*** |
| Головне вікно | java.swing .JFrame |
| Панель «Меню» | java.swing .JMenuBar |
| Панель «Вкладки» | java.swing .JTabbedPane |
| Список вибору факультету | java.swing .JComboBox |
| Список вибору групи | java.swing .JComboBox |
| Список вибору профілю | java.swing .JList |
| Область історії | java.swing .JList |
| Кнопка «Файл» | java.swing .JMenu |
| Кнопка «Допомога» | java.swing .JMenu |
| Кнопка «Про програму» | java.swing .JMenuItem |
| Кнопка «Посібник користувача» | java.swing .JMenuItem |
| Кнопка «Додати акаунт | java.swing .JMenuItem |
| Кнопка «Вихід» | java.swing .JMenuItem |
| Бічна панель кнопок | java.swing .JPanel |
| Кнопка «Додати акаунт» | java.swing .JButton |
| Кнопка «Рейтинг» | java.swing .JButton |
| Кнопка «Зберегти» | java.swing .JButton |
| Кнопка «Видалити» | java.swing .JButton |
| Поля профілів | java.swing .JTextEdit |
| Назви полів профілів | java.swing .JLable |
| Область розкладу | java.swing .JTableLayout |
| Область рейтингу | java.swing .JTableLayout |

# 3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

## 3.1 Структура програмного додатку

На діаграмі класів додатку, що розробляється, зображується структура додатку (див додаток А). Програма складається із класів пакетів ui, model, core а також config, dao. При розробці класів я використовував знання шаблонів проектування набуті в праці [1]. Також я використовував інформацію про мову програмування Java, принципи ООП та принципи побудови графічних інтерфейсів набуті з праць [2-10].

## 3.2 Специфікація класів програмного додатка



### Package com.Oyster.app.model

|  |  |
| --- | --- |
| **Перелік Інтерфейсів** | |
| **Інтерфейс** | **Пояснення** |
| [**IProfile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\IProfile.html) | Інтерфейс для узагальнення сутностей, що містять граничний клас Profile серед полів Created by oleg on 5/11/14. |

|  |  |
| --- | --- |
| **Перелік класів** | |
| **Class** | **Пояснення** |
| [**Absence**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html) | Граничний клас, що представляє таблицю ABSENCE\_TBL у базі даних Проміжна сутність для отримання розкладу |
| [**Classes**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html) | Граничний клас, що представляє таблицю CLASSES\_TBL у базі даних |
| [**Faculty**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html) | Граничний клас, що представляє таблицю FACULTY\_TBL у базі даних |
| [**Group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html) | Граничний клас, що представляє таблицю GROUP\_TBL у базі даних |
| [**Mark**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Mark.html) | Граничний клас, що представляє таблицю MARK\_TBL у базі даних Created by oleg on 5/22/14. |
| [**Message**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html) | Граничний клас, що представляє таблицю MESSAGE\_TBL у базі даних Created by oleg on 5/11/14. |
| [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | Граничний клас, що представляє таблицю PROFILE\_TBL у базі даних |
| [**Sending**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html) | Граничний клас, що представляє таблицю SENDING\_TBL у базі даних Created by oleg on 5/11/14. |
| [**Student**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html) | Граничний клас, що представляє таблицю SUBJECT\_TBL у базі даних |
| [**Subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html) | Граничний клас, що представляє таблицю TEACHER\_TBL у базі даних |
| [**Task**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html) | Граничний клас, що представляє таблицю SUBJECT\_TBL у базі даних Created by oleg on 5/22/14. |
| [**Teacher**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html) | Граничний клас, що представляє таблицю TEACHER\_TBL у базі даних |

#### Interface IProfile

Method Summary

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод і Пояснення** |
| [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | [**getProfile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\IProfile.html#getProfile())()  повертає профіль класу |

#### getProfile

[Profile](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) getProfile()

повертає профіль класу

**Returns:**

граничний клас Profile

#### Class Absence

public class **Absence**

extends java.lang.Object

Граничний клас, що представляє таблицю ABSENCE\_TBL у базі даних Проміжна сутність для отримання розкладу

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**classId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#classId) |
| private java.util.UUID | [**groupId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#groupId) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#id) |

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**Absence**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#Absence())()  конструктор зазамовчуванням |
| [**Absence**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#Absence(java.util.UUID, java.util.UUID, java.util.UUID))(java.util.UUID id, java.util.UUID groupId, java.util.UUID classId)  Конструктор класу Absence |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| java.util.UUID | [**getClassId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#getClassId())()  метод, що повертає id заняття |
| java.util.UUID | [**getGroupId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#getGroupId())()  метод, що повертає id групи |
| java.util.UUID | [**getId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#getId())()  метод, що повертає id присутності |
| void | [**setClassId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#setClassId(java.util.UUID))(java.util.UUID classId)  метод, що встановлює id заняття |
| void | [**setGroupId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#setGroupId(java.util.UUID))(java.util.UUID groupId)  метод, що встановлює id групи |
| void | [**setId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Absence.html#setId(java.util.UUID))(java.util.UUID id)  метод, що встановлює id присутності |

#### Absence

public Absence()

конструктор зазамовчуванням

#### Absence

public Absence(java.util.UUID id,

java.util.UUID groupId,

java.util.UUID classId)

Конструктор класу Absence

**Parameters:**

id - ідентифікатор об’єкту

groupId - Ідетифікатор групи

classId - ідентифікатор заняття

#### Class Classes

Граничний клас, що представляє таблицю CLASSES\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.lang.String | [**audience**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#audience) |
| private java.lang.String | [**building**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#building) |
| private int | [**date**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#date) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#id) |
| private java.util.UUID | [**subjectId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#subjectId) |
| private java.util.UUID | [**teacherId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#teacherId) |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| java.lang.String | [**getAudience**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getAudience())()  метод, що повертає номер аудиторії |
| java.lang.String | [**getBuilding**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getBuilding())()  метод, що повертає номер корпусу |
| int | [**getDate**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getDate())()  метод, що повертає час заняття |
| java.util.UUID | [**getId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getId())()  поветрає ідентифікатор заняття |
| java.util.UUID | [**getSubjectId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getSubjectId())()  повертає id предмету |
| java.util.UUID | [**getTeacherId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#getTeacherId())()  повертає id викладача |
| void | [**setAudience**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setAudience(java.lang.String))(java.lang.String audience)  метод, що встановлює номер аудиторії |
| void | [**setBuilding**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setBuilding(java.lang.String))(java.lang.String building)  метод, що встановлює номер корпусу |
| void | [**setDate**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setDate(int))(int date)  метод, що встановлює час заняття |
| void | [**setId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setId(java.util.UUID))(java.util.UUID id)  встановлює ідентифікатор заняття |
| void | [**setSubjectId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setSubjectId(java.util.UUID))(java.util.UUID subjectId)  встановює id предмету |
| void | [**setTeacherId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Classes.html#setTeacherId(java.util.UUID))(java.util.UUID teacherId)  встановює id викладача |

#### Classes

public Classes()

#### Classes

public Classes(java.util.UUID id,

java.util.UUID subjectId,

java.util.UUID teacherId,

java.lang.String building,

java.lang.String audience,

int date)

**Parameters:**

id - ідентифікатор поля заняття в таблиці бази даних

subjectId - ідентифікатор поля предмету в таблиці бази даних

teacherId - ідентифікатор поля викладач в таблиці бази даних

building - ідентифікатор поля заняття в таблиці бази даних

audience - номер аудиторії

date - час проведення заняття

#### Class Faculty

Граничний клас, що представляє таблицю FACULTY\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html#id) |
| private java.lang.String | [**name**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html#name) |
| **Constructors** | | |
| **Constructor and Description** | | |
| [**Faculty**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html#Faculty())() | | |
| [**Faculty**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html#Faculty(java.util.UUID, java.lang.String))(java.util.UUID id, java.lang.String name)  Конструктор класу Faculty | | |

#### Faculty

public Faculty(java.util.UUID id,

java.lang.String name)

Конструктор класу Faculty

**Parameters:**

id - ідентифікатор факультету

name - назва факультету

#### Class Group

Граничний клас, що представляє таблицю GROUP\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private [**Faculty**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Faculty.html) | [**faculty**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html#faculty) |
| private java.util.UUID | [**facultyId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html#facultyId) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html#id) |
| private java.lang.String | [**name**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html#name) |

#### Group

public Group(java.util.UUID id,

java.util.UUID facultyId,

java.lang.String name)

**Parameters:**

id - ідентифікатор поля групи в таблиці бази даних

facultyId - ідентифікатор поля факультету в таблиці бази даних

name - назва групи

#### Class Mark

Граничний клас, що представляє таблицю MARK\_TBL у базі даних Created by oleg on 5/22/14.

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Mark.html#id) |
| private java.lang.String | [**mark**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Mark.html#mark) |
| private java.util.UUID | [**studentId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Mark.html#studentId) |
| private java.util.UUID | [**taskId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Mark.html#taskId) |

#### Mark

public Mark()

#### Mark

public Mark(java.util.UUID id,

java.util.UUID taskId,

java.util.UUID studentId,

java.lang.String mark)

**Parameters:**

id - ідентифікатор поля оцінки в таблиці бази даних

taskId - ідентифікатор поля завдання в таблиці бази даних

studentId - ідентифікатор поля студента в таблиці бази даних

mark - значення оцінки

#### Class Message

Граничний клас, що представляє таблицю MESSAGE\_TBL у базі даних Created by oleg on 5/11/14.

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#id) |
| private java.lang.String | [**message**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#message) |
| private [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | [**recipient**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#recipient) |
| private [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | [**sender**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#sender) |
| private java.lang.String | [**senderStr**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#senderStr) |
| private java.lang.String | [**topic**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html#topic) |

### Constructor Detail

#### Message

public Message()

#### Message

public Message(java.util.UUID id,

java.lang.String topic,

java.lang.String message)

**Parameters:**

id - ідентифікатор поля повідомлення в таблиці бази даних

topic - тема повідомлення

message - повідомлення

#### Class Profile

Граничний клас, що представляє таблицю PROFILE\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.lang.String | [**address**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#address) |
| private java.lang.String | [**email**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#email) |
| private java.lang.String | [**firstName**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#firstName) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#id) |
| private java.lang.String | [**password**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#password) |
| private java.lang.String | [**phoneNom**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#phoneNom) |
| private java.lang.String | [**secondName**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html#secondName) |

#### Profile

public Profile()

#### Profile

public Profile(java.util.UUID id,

java.lang.String firstName,

java.lang.String secondName,

java.lang.String email,

java.lang.String password,

java.lang.String phoneNom,

java.lang.String address)

конструктор класу Profile

**Parameters:**

id - ідентифікатор поля профіль в таблиці бази даних

firstName - ім’я куристувача

secondName - прізвище користувача

email - електронна адреса користувача

password - пароль користувача

phoneNom - номер телефону користувача

address - адреса користувача

#### Class Sending

Граничний клас, що представляє таблицю SENDING\_TBL у базі даних Created by oleg on 5/11/14.

Field Summary

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#id) |
| private [**Message**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Message.html) | [**message**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#message) |
| private java.util.UUID | [**messageId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#messageId) |
| private [**IProfile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\IProfile.html) | [**recipient**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#recipient) |
| private java.util.UUID | [**recipientId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#recipientId) |
| private [**IProfile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\IProfile.html) | [**sender**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#sender) |
| private java.util.UUID | [**senderId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Sending.html#senderId) |

#### Sending

public Sending()

#### Sending

public Sending(java.util.UUID id,

java.util.UUID senderId,

java.util.UUID recipientId,

java.util.UUID messageId)

**Parameters:**

id - ідентифікатор поля розсилки в таблиці бази даних

senderId - ідентифікатор відправника

recipientId - ідентифікатор поля отримувача

messageId - ідентифікатор поля повідомлення в таблиці бази даних

#### Class Student

Граничний клас, що представляє таблицю SUBJECT\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private int | [**bookNum**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#bookNum) |
| private int | [**course**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#course) |
| private [**Group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html) | [**group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#group) |
| private java.util.UUID | [**groupId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#groupId) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#id) |
| private [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | [**profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#profile) |
| private java.util.UUID | [**profileId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html#profileId) |

#### Student

public Student()

#### Student

public Student(java.util.UUID id,

java.util.UUID profileId,

java.util.UUID groupId,

int course,

int bookNum)

констуктор класу Student

**Parameters:**

id - ідентифікатор поля студента в таблиці бази даних

profileId - ідентифікатор профіля студента

groupId - ідентифікатор групи студента

course - курс студента

bookNum - номер залікової нижки студента

#### Class Subject

Граничний клас, що представляє таблицю TEACHER\_TBL у базі даних

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html#id) |
| private java.lang.String | [**name**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html#name) |

#### Subject

public Subject()

#### Subject

public Subject(java.util.UUID id,

java.lang.String name)

**Parameters:**

id - ідентифікатор поля предмет в таблиці бази даних

name - ім’я предмету

#### Class Task

Граничний клас, що представляє таблицю SUBJECT\_TBL у базі даних Created by oleg on 5/22/14.

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**groupId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html#groupId) |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html#id) |
| private java.lang.String | [**name**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html#name) |
| private [**Subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html) | [**subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html#subject) |
| private java.util.UUID | [**subjectId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Task.html#subjectId) |

#### Task

public Task()

#### Task

public Task(java.util.UUID id,

java.util.UUID groupId,

java.util.UUID subjectId,

java.lang.String name)

конструктор класу Task

**Parameters:**

id - ідентифікатор поля завдання в таблиці бази даних

groupId - ідетифікатор групи, до якої відноситься завдання

subjectId - ідетифікатор предмету, до якого відноситься завдання

name - назва завдання

#### Class Teacher

Граничний клас, що представляє таблицю TEACHER\_TBL у базі даних

**Since:**

4/15/14 11:03 PM

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| private java.util.UUID | [**id**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html#id) |
| private [**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) | [**profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html#profile) |
| private java.util.UUID | [**profileId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html#profileId) |
| private java.util.UUID | [**workerInfoId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html#workerInfoId) |

### Constructor Detail

#### Teacher

public Teacher()

#### Teacher

public Teacher(java.util.UUID id,

java.util.UUID profileId,

java.util.UUID workerInfoId)

Конструктор класу Teacher

**Parameters:**

id - ідентифікатор викладача

profileId - ідентифікатор профіля

workerInfoId - ідентифікатор інформації про робітника

### Package com.Oyster.config

|  |  |
| --- | --- |
| **Перелік Інтерфейсів** | |
| **Інтерфейс** | **Пояснення** |
| [**IConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) | Інтерфейс для всіх об’єктів конфігурації декларує методи для отримання та запису полів конфігурації |

|  |  |
| --- | --- |
| **Перелік класів** | |
| **Class** | **Пояснення** |
| [**AppConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html) | Клас конфігурації |
| [**ConfigReader**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigReader.html) | Абстрактний клас для уніфікованого доступу до об’єкту конфігурації |
| [**ConfigWriter**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigWriter.html) | Абстрактний клас для уніфікованого доступу до об’єкту конфігурації |

#### Interface IConfig

Інтерфейс для всіх об’єктів конфігурації декларує методи для отримання та запису полів конфігурації

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| java.util.Set<java.lang.String> | [**getAllKeys**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html#getAllKeys())()  повертає усі ключі із карти конфігурації прешого рівня |
| java.lang.Object | [**getValue**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html#getValue(java.lang.String))(java.lang.String key)  дістає значення із ключем key |
| void | [**setValue**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html#setValue(java.lang.String, java.lang.Object))(java.lang.String key, java.lang.Object value)  встановлює значення для поля із ключем key |

#### setValue

void setValue(java.lang.String key,

java.lang.Object value)

встановлює значення для поля із ключем key

**Parameters:**

key - ключ поля

value - значення поля

#### getValue

java.lang.Object getValue(java.lang.String key)

дістає значення із ключем key

**Parameters:**

key - ключ поля

**Returns:**

значення поля

#### getAllKeys

java.util.Set<java.lang.String> getAllKeys()

повертає усі ключі із карти конфігурації прешого рівня

**Returns:**

усі ключі першого рівня

#### Class AppConfig

Клас конфігурації

Method Summary

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| java.util.Set<java.lang.String> | [**getAllKeys**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html#getAllKeys())()  Повертає множину всіх ключів |
| static [**AppConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html) | [**getInstance**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html#getInstance())()  Метод для отримання об’єкту класу конфігурацій, якщо такого немає, створює його, використовує метод подвійної перевірки, тому може безпечно використовуватись у багатопоточних застосунках |
| java.lang.Object | [**getValue**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html#getValue(java.lang.String))(java.lang.String key)  дістає значення із ключем key |
| java.lang.String | [**out**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html#out())()  Виводить усі значення конфігурацій |
| void | [**setValue**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\AppConfig.html#setValue(java.lang.String, java.lang.Object))(java.lang.String key, java.lang.Object value)  встановлює значення для поля із ключем key |

#### Class ConfigReader

Абстрактний клас для уніфікованого доступу до об’єкту конфігурації

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| abstract void | [**loadFromFile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigReader.html#loadFromFile(java.lang.String))(java.lang.String path)  завантажує зовнішній файл конфігурації |
| void | [**setConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigReader.html#setConfig(com.kpi_sity.config.IConfig))([**IConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)  встановлює об’єкт конфігурації |

#### loadFromFile

public abstract void loadFromFile(java.lang.String path)

завантажує зовнішній файл конфігурації

**Parameters:**

path - шлях до файлу

#### setConfig

public void setConfig([IConfig](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)

встановлює об’єкт конфігурації

**Parameters:**

config - об’єкт конфігурації

#### Class ConfigWriter

Абстрактний клас для уніфікованого доступу до об’єкту конфігурації

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| abstract void | [**save**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigWriter.html#save(java.lang.String))(java.lang.String path)  записує конфігураіцю у JSON - файл |
| void | [**setConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigWriter.html#setConfig(com.kpi_sity.config.IConfig))([**IConfig**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)  встановлює об’єкт конфігурації |

#### save

public abstract void save(java.lang.String path)

записує конфігураіцю у JSON - файл

**Parameters:**

path - шлях до файлу

#### setConfig

public void setConfig([IConfig](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)

встановлює об’єкт конфігурації

**Parameters:**

config - об’єкт конфігурації

#### JSONConfigReader

Клас зчитування конфігурації із JSON – файлу

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| void | [**loadFromFile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\impl\JSONConfigReader.html#loadFromFile(java.lang.String))(java.lang.String path)  завантажує зовнішній файл конфігурації |

#### JSONConfigReader

public JSONConfigReader([IConfig](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)

Конструктор класу JSONConfigReader

**Parameters:**

config - об’єкт, у якому зберігатиметься конфігурація

#### loadFromFile

public void loadFromFile(java.lang.String path)

завантажує зовнішній файл конфігурації

**Specified by:**

[loadFromFile](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigReader.html#loadFromFile(java.lang.String)) in class [ConfigReader](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigReader.html)

**Parameters:**

path - шлях до файлу

#### Class JSONConfigWriter

Клас запису конфігурації в JSON – файлу

#### JSONConfigWriter

public JSONConfigWriter([IConfig](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\IConfig.html) config)

Конструктор класу JSONConfigWriter

**Parameters:**

config - об’єкт, у якому зберігатиметься конфігурація

#### save

public void save(java.lang.String path)

записує конфігураіцю у JSON - файл

**Specified by:**

[save](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigWriter.html#save(java.lang.String)) in class [ConfigWriter](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\config\ConfigWriter.html)

**Parameters:**

path - шлях до файлу

### Package com.Oyster.core.controller

Клас відповідає за реєстрацію, валідацію та виконання команд, реалізує паттерн Singleton

|  |  |
| --- | --- |
| **Constructors** | |
| **Modifier** | **Constructor and Description** |
| private | [**CommandExecutor**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#CommandExecutor())()  створює екземпляр класу CommandExecutor |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| void | [**addCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#addCommand(java.lang.Class))(java.lang.Class command)  реєструє нову команду в CommandExecutor |
| protected java.lang.Object | [**clone**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#clone())()  викидає помилку при спробі клонувати об’єкт, оскільки CommandExecutor повинен бути одиночкою |
| void | [**execute**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#execute(java.lang.String, com.kpi_sity.core.controller.command.Context, java.lang.Runnable))(java.lang.String action, [**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) params, java.lang.Runnable onPostExecute)  виконує команду у фоновому режимі |
| private java.lang.Class | [**findCommandByAction**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#findCommandByAction(java.lang.String))(java.lang.String action)  шукає команду за ключем |
| static [**CommandExecutor**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html) | [**getInstance**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html#getInstance())()  повертає змінну класу CommandExecutor |

#### CommandExecutor

private CommandExecutor()

створює екземпляр класу CommandExecutor

#### getInstance

public static [CommandExecutor](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\CommandExecutor.html) getInstance()

повертає змінну класу CommandExecutor

**Returns:**

екземпляр класу CommandExecutor

#### execute

public void execute(java.lang.String action,

[Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) params,

java.lang.Runnable onPostExecute)

throws [CommandNotFoundException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\CommandNotFoundException.html),

java.lang.InstantiationException,

java.lang.IllegalAccessException,

[InvalidCommandParameterException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\InvalidCommandParameterException.html)

виконує команду у фоновому режимі

**Parameters:**

action - ключ команди

params - контекст команди

**Throws:**

[CommandNotFoundException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\CommandNotFoundException.html)

java.lang.InstantiationException

java.lang.IllegalAccessException

[InvalidCommandParameterException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\InvalidCommandParameterException.html)

#### findCommandByAction

private java.lang.Class findCommandByAction(java.lang.String action)

шукає команду за ключем

**Parameters:**

action - ключ команди

**Returns:**

команду, якщо знаходить, інакше null

#### addCommand

public void addCommand(java.lang.Class command)

реєструє нову команду в CommandExecutor

**Parameters:**

command - class of the new command

#### clone

protected java.lang.Object clone()

throws java.lang.CloneNotSupportedException

викидає помилку при спробі клонувати об’єкт, оскільки CommandExecutor повинен бути одиночкою

**Overrides:**

clone in class java.lang.Object

**Returns:**

нічого

**Throws:**

java.lang.CloneNotSupportedException

#### Class Validator

Клас, що відповідає за валідацію команди

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| static boolean | [**validate**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\Validator.html#validate(java.lang.Class, com.kpi_sity.core.controller.command.Context))(java.lang.Class command, [**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context)  виконує валідацію команди |

#### validate

public static boolean validate(java.lang.Class command,

[Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context)

throws [InvalidCommandParameterException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\InvalidCommandParameterException.html)

виконує валідацію команди

**Parameters:**

command - команда для валідації

context - контекст команди

**Returns:**

true - якщо команда пройшла валідацію, якщо ні - викидає помилку

**Throws:**

[InvalidCommandParameterException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\exception\InvalidCommandParameterException.html)

### Package com.Oyster.core.controller.command

|  |  |
| --- | --- |
| **Перелік класів** | |
| **Class** | **Пояснення** |
| [**AbstractCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html) | Абстпрактний клас, що реалізує базову команду, містить контекст, та екземпляр інтерфейсу Runnable, та методи його отримання |
| [**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) | Клас відображає контекст команди |
| [**DeleteIProfileCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\DeleteIProfileCommand.html) | реалізує команду видалення профілю Created by oleg on 5/12/14. |
| [**FillRecipientsCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\FillRecipientsCommand.html) | команда, для заповнення списку отримувачів повідомлення |
| [**LoadClassesCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LoadClassesCommand.html) | команда, що реалізує завантаження занять з бази даних |
| [**LoadMaksCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LoadMaksCommand.html) | команда для завантаження оцінок з бази даних |
| [**LoadMessageCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LoadMessageCommand.html) |  |
| [**LogInCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LogInCommand.html) | Реалізує команду авторизації в системі |
| [**RegisterStudentCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\RegisterStudentCommand.html) | Реалізує команду для реєстрації користуваача |
| [**SendMessageCommand**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\SendMessageCommand.html) | команда для відправлення повідомлень |

#### Class AbstractCommand

Абстпрактний клас, що реалізує базову команду, містить контекст, та екземпляр інтерфейсу Runnable, та методи його отримання

|  |  |
| --- | --- |
| **Поля** | |
| **Модифікатор і тип** | **Поле та пояснення** |
| protected [**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) | [**context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#context)  контекст, який передається команді |
|  |  |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| java.lang.Object | [**addParameter**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#addParameter(java.lang.String, java.lang.Object))(java.lang.String name, java.lang.Object value)  додає параметер до контексту |
| [**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) | [**getContext**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#getContext())()  повертає контекст команди |
| java.lang.Runnable | [**getOnPostExecute**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#getOnPostExecute())()  повертає екземпляр інтерфейсу Runnable, метод якого run() буде виконаний у UI потоці |
| java.lang.Object | [**removeParametr**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#removeParametr(java.lang.String))(java.lang.String name)  видаляє параметер з контексту |
| void | [**setContext**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#setContext(com.kpi_sity.core.controller.command.Context))([**Context**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context)  встановлює контекст команди |
| void | [**setOnPostExecute**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\AbstractCommand.html#setOnPostExecute(java.lang.Runnable))(java.lang.Runnable runnable)  встановлює екземпляр інтерфейсу Runnable, метод якого run() буде виконаний у UI потоці |

#### Class DeleteIProfileCommand

реалізує команду видалення профілю Created by oleg on 5/12/14.

#### DeleteIProfileCommand

public DeleteIProfileCommand()

конструктор за замовчуванням

#### DeleteIProfileCommand

public DeleteIProfileCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

**Parameters:**

context1 - параметри для видалення профілю

#### Class FillRecipientsCommand

Команда, для заповнення списку отримувачів повідомлення

#### FillRecipientsCommand

public FillRecipientsCommand()

#### FillRecipientsCommand

public FillRecipientsCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

конструктор класу FillRecipientsCommand

**Parameters:**

context1 - параметри для заповнення списку отримувачів

#### Class LoadClassesCommand

Команда, що реалізує завантаження занять з бази даних

#### LoadClassesCommand

public LoadClassesCommand()

#### LoadClassesCommand

public LoadClassesCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

конструктор класу LoadClassesCommand

**Parameters:**

context1 - параметри, необхідні для завантаження занять

#### Class LoadMaksCommand

Команда для завантаження оцінок з бази даних

### Constructor Detail

#### LoadMaksCommand

public LoadMaksCommand()

#### LoadMaksCommand

public LoadMaksCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

конструктор класу LoadMaksCommand

**Parameters:**

context1 - параметри, необхідня для завантаження оцінок

#### Class LoadMessageCommand

### Constructor Detail

#### LoadMessageCommand

public LoadMessageCommand()

#### LoadMessageCommand

public LoadMessageCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

конструктор класу LoadMessageCommand

**Parameters:**

context1 - параметри, необхідня для завантаження повідомлень

#### Class LogInCommand

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| static void | [**centreWindow**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LogInCommand.html#centreWindow(javax.swing.JFrame))(javax.swing.JFrame frame)  встановлює вікно в центер екрану |
| void | [**run**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\LogInCommand.html#run())() |

#### LogInCommand

public LogInCommand()

#### LogInCommand

public LogInCommand([Context](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\core\controller\command\Context.html) context1)

Конструктор для класу LogInCommand

**Parameters:**

context1 - параметри для реєстрації

### Package com.Oyster.dao

#### Interface CRUDInterface

визначає CRUD-інтерфейс для базових операцій із постійним місцем збереження

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| <T> void | [**delete**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#delete(T))(T instance)  виконує базову операцію видалення для даної сутності |
| <T> T | [**insert**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#insert(T))(T instance)  виконує базову операцію вставлення для даної сутності |
| <T> T | [**read**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#read(java.lang.Class, java.util.UUID))(java.lang.Class entityClass, java.util.UUID id)  виконує базову операцію зчитування для даної сутності |
| <T> java.util.List<T> | [**select**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#select(java.lang.Class, com.kpi_sity.dao.DAOFilter))(java.lang.Class entityClass, [**DAOFilter**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\DAOFilter.html) filter)  виконує пошук сутностей у постійному хранилищі |
| <T> java.util.List<T> | [**select**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#select(java.lang.Class, java.lang.String))(java.lang.Class entityClass, java.lang.String SQLString)  виконує пошук сутностей у базі даних |
| <T> void | [**update**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\CRUDInterface.html#update(T))(T instance)  виконує базову операцію оновлення для даної сутності |

#### Іnsert

<T> T insert(T instance)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує базову операцію вставлення для даної сутності

**Type Parameters:**

T - тип сутності

**Parameters:**

instance - сутність

**Returns:**

саму сутність

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### read

<T> T read(java.lang.Class entityClass,

java.util.UUID id)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує базову операцію зчитування для даної сутності

**Type Parameters:**

T - тип сутності

**Parameters:**

entityClass - клас сутності

id - ключ сутності

**Returns:**

саму сутність

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### update

<T> void update(T instance)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує базову операцію оновлення для даної сутності

**Type Parameters:**

T - тип сутності

**Parameters:**

instance - сутність

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### delete

<T> void delete(T instance)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує базову операцію видалення для даної сутності

**Type Parameters:**

T - тип сутності

**Parameters:**

instance - сутність

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### select

<T> java.util.List<T> select(java.lang.Class entityClass,

[DAOFilter](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\DAOFilter.html) filter)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує пошук сутностей у постійному хранилищі

**Type Parameters:**

T - тип сутності

**Parameters:**

entityClass - клас сутності

filter - фільтр

**Returns:**

список знайдених сутностей, що задовільняють даному фільтру

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### select

<T> java.util.List<T> select(java.lang.Class entityClass,

java.lang.String SQLString)

throws [DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

виконує пошук сутностей у базі даних

**Type Parameters:**

T - тип сутності

**Parameters:**

entityClass - клас сутності

SQLString - SQL-запит до бази даних

**Returns:**

список знайдених сутностей

**Throws:**

[DAOException](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\exception\DAOException.html)

#### Interface DAOFilter

public interface **DAOFilter**

допоміжний інтерфейс для відбирання сутностей за певними параметрами

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| <T> boolean | [**accept**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\DAOFilter.html#accept(T))(T entity)  перевіряє, чи сутність задовільняє дані умови |

#### accept

<T> boolean accept(T entity)

перевіряє, чи сутність задовільняє дані умови

**Type Parameters:**

T - тип сутності

**Parameters:**

entity - сутність для перевірки

**Returns:**

true - якщо сутність задовільняє умови, false - якщо ні

#### Class DAOCRUDJdbc

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| <T> void | [**delete**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#delete(T))(T instance)  видаляє сутність із сховища даних |
| static [**DAOCRUDJdbc**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html) | [**getInstance**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#getInstance(org.springframework.context.ApplicationContext))(org.springframework.context.ApplicationContext context)  отримує екземпляр JDBC dao |
| <T> T | [**insert**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#insert(T))(T instance)  додає екземпляр нової сутності у сховище даних |
| <T> T | [**read**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#read(java.lang.Class, java.util.UUID))(java.lang.Class entityClass, java.util.UUID id)  запит сутності у сховищі даних по його UUID |
| <T> java.util.List<T> | [**select**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#select(java.lang.Class, com.kpi_sity.dao.DAOFilter))(java.lang.Class entityClass, [**DAOFilter**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\DAOFilter.html) filter)  запит для доступу до сховища даних |
| private <T> java.util.List<T> | [**select**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#select(java.lang.Class, com.kpi_sity.dao.DAOFilter, java.lang.String))(java.lang.Class entityClass, [**DAOFilter**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\DAOFilter.html) filter, java.lang.String sql)  допоміжна функція для пертворення різних типів запитів |
| <T> java.util.List<T> | [**select**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#select(java.lang.Class, java.lang.String))(java.lang.Class entityClass, java.lang.String SQLString)  запит для доступу до сховища даних |
| <T> void | [**update**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\dao\impl\DAOCRUDJdbc.html#update(T))(T instance)  оновлює сутність у сховищі даних |

### com.Oyster.ui

#### Class LoginFrame

Клас,що відображає вікно авторизації користувача у програмі

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**LoginFrame**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\LoginFrame.html#LoginFrame())()  Конструктор класу LoginFrame створює вікно, встановлює його розміри |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| void | [**actionPerformed**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\LoginFrame.html#actionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent e) |
| static void | [**centreWindow**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\LoginFrame.html#centreWindow(javax.swing.JFrame))(javax.swing.JFrame frame)  встановлює вікно в центрі екрану |
| private void | [**placeComponents**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\LoginFrame.html#placeComponents(javax.swing.JPanel))(javax.swing.JPanel panel)  ініціалізує компоненти |
| private void | [**tryToLogIn**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\LoginFrame.html#tryToLogIn())()  метод викликає команду авторизації користувача |

#### LoginFrame

public LoginFrame()

Конструктор класу LoginFrame створює вікно, встановлює його розміри

#### placeComponents

private void placeComponents(javax.swing.JPanel panel)

ініціалізує компоненти

**Parameters:**

panel - панель, на якій будуть розміщені компоненти

#### actionPerformed

public void actionPerformed(java.awt.event.ActionEvent e)

**Specified by:**

actionPerformed in interface java.awt.event.ActionListener

#### tryToLogIn

private void tryToLogIn()

метод викликає команду авторизації користувача

#### centreWindow

public static void centreWindow(javax.swing.JFrame frame)

встановлює вікно в центрі екрану

**Parameters:**

frame - вікно, яке потрібно встановити по центру

#### Class MainForm

Клас відображає головне вікно програми

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**MainForm**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#MainForm())()  створює головне вікно програми |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| void | [**actionPerformed**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#actionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent e) |
| private void | [**fillInfoFields**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#fillInfoFields(com.kpi_sity.app.model.Profile))([**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) profile)  Відображає поля, які берігаються в проіфлі |
| private void | [**fillInfoFields**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#fillInfoFields(com.kpi_sity.app.model.Student))([**Student**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html) student)  Відображає поля, які берігаються в об’єкті студент |
| void | [**fillRecipients**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#fillRecipients(java.lang.Class))(java.lang.Class c)  заповнення списку отримувачів повідомлення |
| private java.util.ArrayList<[**Group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html)> | [**getGroups**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#getGroups())() |
| private void | [**getSchedule**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#getSchedule())()  Завантаження розкладу |
| private java.util.ArrayList<[**Subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html)> | [**getSubjects**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#getSubjects())() |
| private void | [**loadHelp**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#loadHelp())() |
| private void | [**loadMarks**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#loadMarks())()  Завантаження таблиці успішності |
| private void | [**loadMessage**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#loadMessage())()  завантажує повідомлення, які надіслані поточному користувачу |
| private void | [**logOut**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#logOut())()  Завершення поточної сесії користувача |
| private void | [**placeComponents**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#placeComponents(javax.swing.JComponent))(javax.swing.JComponent component)  ініціалізує компоненти |
| private void | [**removeProfile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#removeProfile())()  видалення профілю |
| private void | [**saveButtonClick**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#saveButtonClick())()  збереження змін до облікового запису |
| private void | [**saveUpdate**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#saveUpdate(com.kpi_sity.app.model.Profile))([**Profile**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Profile.html) p)  збереження змін у профілі |
| private void | [**saveUpdate**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#saveUpdate(com.kpi_sity.app.model.Student))([**Student**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Student.html) s)  збереження змін у обєкті студент |
| private void | [**sendMessage**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#sendMessage())()  Відправлення повідомлення |
| private void | [**setReceivedMessages**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#setReceivedMessages())() |
| private void | [**updateUI**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MainForm.html#updateUI())()  оновлення графічного інтерфейсу |

#### Class MarksTable

Клас, що відповідає за створення і наповнення таблиці з оцінками

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**MarksTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#MarksTable(javax.swing.JFrame, javax.swing.JTable, com.kpi_sity.app.model.Group, com.kpi_sity.app.model.Subject))(javax.swing.JFrame frame, javax.swing.JTable table, [**Group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html) group,[**Subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html) subject)  конструкотр створює об’єкт та ініціалізує поля |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| private void | [**createScheduleTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#createScheduleTable())()  метод ініціалізує структуру даних у якій зберігатимуться дані |
| private void | [**fillTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#fillTable())()  заповнення таблиці даними про успішність |
| private int | [**findColumnByTaskId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#findColumnByTaskId(java.util.UUID))(java.util.UUID taskid)  знаходження стовпця завдання за його id |
| private int | [**findRowByStudentId**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#findRowByStudentId(java.util.UUID))(java.util.UUID studentid)  знаходження рядка студента за його id |
| private void | [**initialization**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#initialization())()  ініціалізуються списк студентів та завдань |
| private void | [**loadTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\MarksTable.html#loadTable())()  метод встановлює модель для таблиці та викликає методи для ініціалізації та заповнення даних |

#### MarksTable

public MarksTable(javax.swing.JFrame frame,

javax.swing.JTable table,

[Group](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html) group,

[Subject](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html) subject)

конструкотр створює об’єкт та ініціалізує поля

**Parameters:**

frame - вікно у якому відображатиметься таблиця

table - таблиця, у якій відображатимуться оцінки

group - група для якої потрібно відобразити успішність

subject - предмет, оцінки з якого потрібно відобразити

#### Class Registration

Клас що відображає вікно реєстрації нового користувача Created by oleg on 4/13/14.

|  |
| --- |
| **Constructor and Description** |
| [**Registration**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\Registration.html#Registration())()  Створює вікно реєстрації |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| void | [**actionPerformed**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\Registration.html#actionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent e) |
| static void | [**centreWindow**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\Registration.html#centreWindow(javax.swing.JFrame))(javax.swing.JFrame frame)  встановлює вікно в центрі екрану |
| private void | [**placeComponents**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\Registration.html#placeComponents(javax.swing.JComponent))(javax.swing.JComponent component)  Ініціалізує компоненти |
| void | [**tryToRegister**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\Registration.html#tryToRegister())()  перевіряє чи всі необхідні поля заповнені та викликає команду реєстрації |

#### Class ScheduleTbl

Клас, що відповідає за створення і наповнення таблиці з розкладом

|  |
| --- |
| **Constructors** |
| **Constructor and Description** |
| [**ScheduleTbl**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#ScheduleTbl(javax.swing.JFrame, com.kpi_sity.app.model.Group, javax.swing.JTable))(javax.swing.JFrame frame, [**Group**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Group.html) group, javax.swing.JTable table)  конструкотр створює об’єкт та ініціалізує поля |

|  |  |
| --- | --- |
| **Методи** | |
| **Модифікатор і тип** | **Метод та Пояснення** |
| private void | [**createScheduleTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#createScheduleTable())()  ініціалізує модель даних, у якій зберігатимуеться розклад |
| private void | [**fillTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#fillTable())()  заповнення таблиці даними про успішність |
| private [**Subject**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Subject.html) | [**getSubjectById**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#getSubjectById(java.util.UUID))(java.util.UUID id)  повертає предмет за його id |
| private [**Teacher**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\app\model\Teacher.html) | [**getTeacherById**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#getTeacherById(java.util.UUID))(java.util.UUID id)  повертає викладача за його id |
| private void | [**initialization**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#initialization())()  ініціалізуються списк студентів та викладачів |
| private void | [**loadTable**](file:///C:\Users\Oleh\Dropbox\javadoc\com\kpi_sity\ui\ScheduleTbl.html#loadTable())()  метод встановлює модель для таблиці та викликає методи для ініціалізації та заповнення даних |

## 

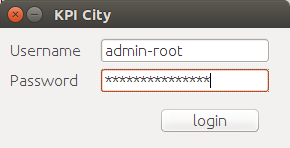
## 3.3 Інструкція для користувача

Для отримання інформації про профіль необхідно знайти його у вкладці “Профілі”. Інформація з'явиться праворуч. Для того щоб преглянути розклад, необхідно перейти у вкладку ‘Розклад” та вибрати групу та факультет. Там ж і можна змінити розклад. Для того щоб змінити налаштування програми необхідно натиснуты на пункт меню “Інструменти” -> “Налаштування”. Потім заповнити необхідні поля та натиснути на кнопку “Зберегти”. Для того щоб створити новий акаунт необхідно натиснути на кнопку “Додати акаунт” та заповнити необхідні поля. Потім треба натиснути кнопку “Створити”. Для того, щоб переглянути історію, необхідно перейти на вкладку “Історія”

# 4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

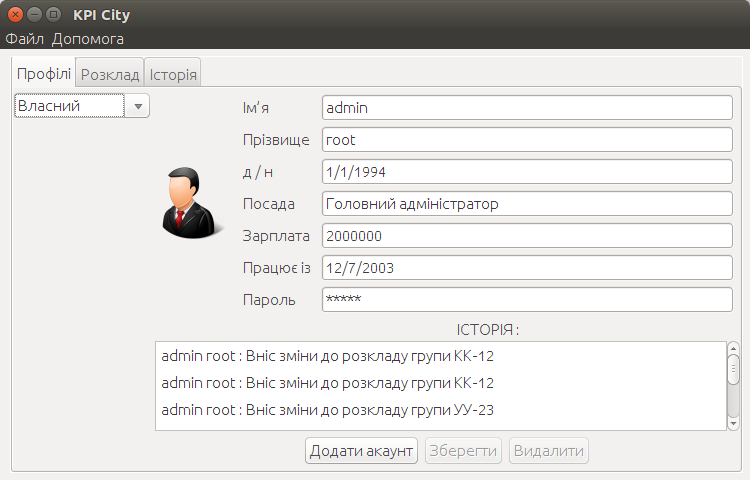
В якості тестування перевіримо усі преценденти.

На рисунку 4.1 представлено вікно авторизації (перевірка преценденту авторизації 2.1.1).



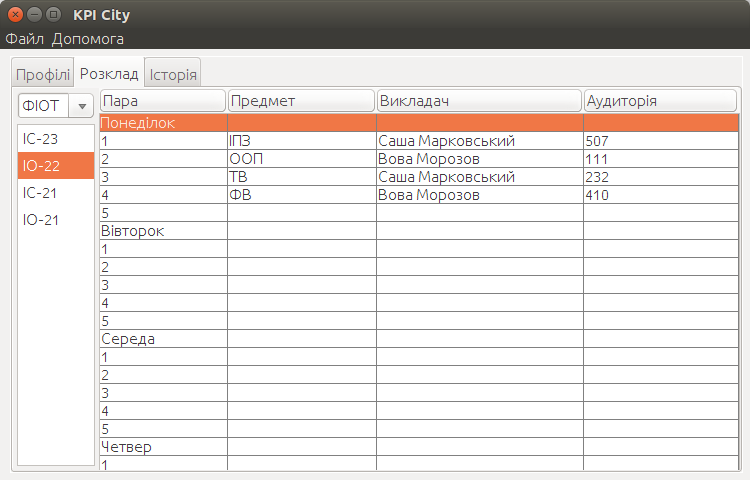
4.1 Вікно авторизації

На рисунку 4.2 представлено вікно управління профілями (перевірка преценденту редагування профілю п. 2.2.4).



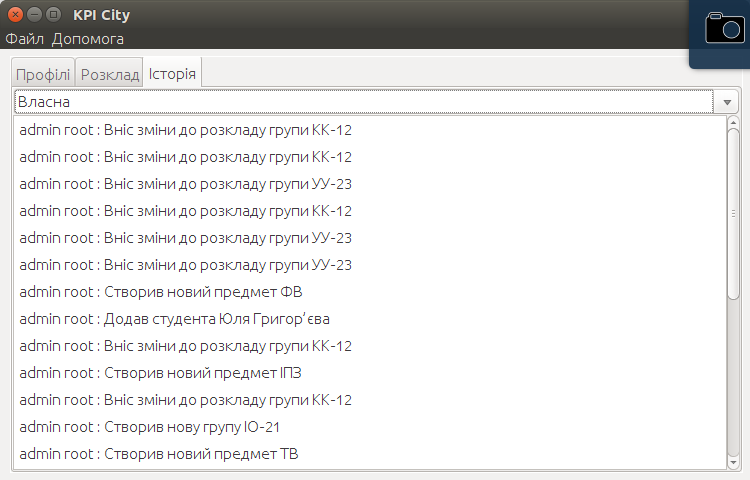
4.2 Вікно управління профілями

На рисунку 4.3 представлено вікно управління розкладом (перевірака преценденту перегляду та редагування розкладу п. 2.2.5, п. 2.2.6).



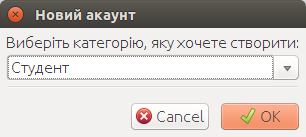
4.3 Вікно управління розкладом

На рисунку 4.3 представлено вікно перегляду історії (перевірка преценденту перегляду історії п. 2.2.7).



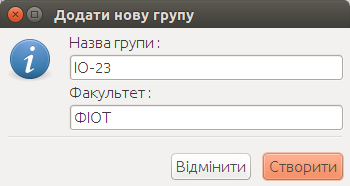
4.4 Вікно перегляду історії

На рисунку 4.5 представлено вікно вибору типу нового акаунту.



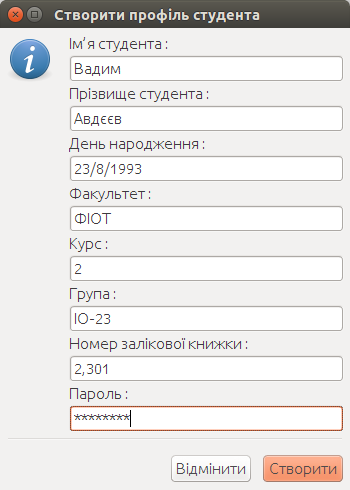
4.5 Вікно вибору типу нового акаунту

На рисунку 4.6 представлено вікно створення групи (перевірка преценденту створення нової групи п. 2.2.2).



4.6 Вікно створення групи

На рисунку 4.7 представлено вікно створення студента (перевірка преценденту створення нової групи п. 2.2.8).



4.7 Вікно створення студента

Під час проведення тестувваня усі преценденти пройшли перевіру, отже тестування проведено успішно.

# ВИСНОВКИ

Під час виконання роботи було закріплено теоретичні знання і практичні навички з проектування, моделювання, розробки та тестування програмного забезпечення набуті у курсі інженерії програмного забезпечення, вивчив шаблон проектування MVC.

Всі вимоги, які були зазначені в технічному завданні, в програмному додатку було повністю виконано. Інтерфейс адміністратора розроблено так, щоб можна було швидко і зручно оперувати профілями та розкладом.

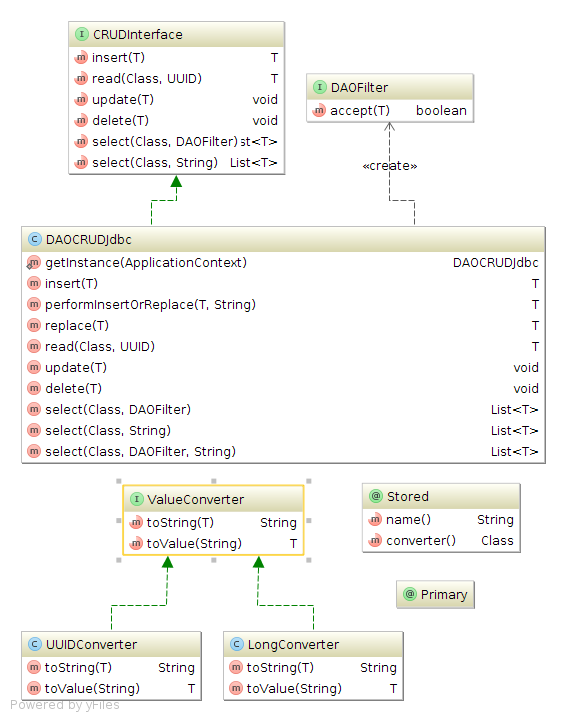
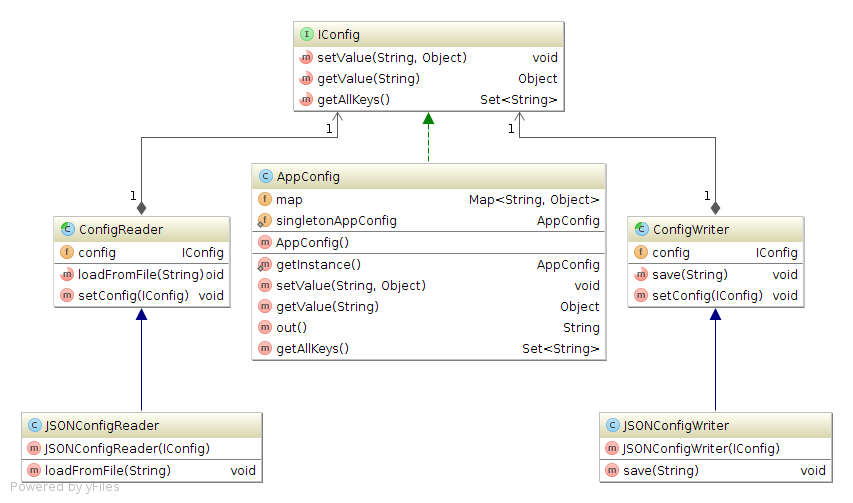
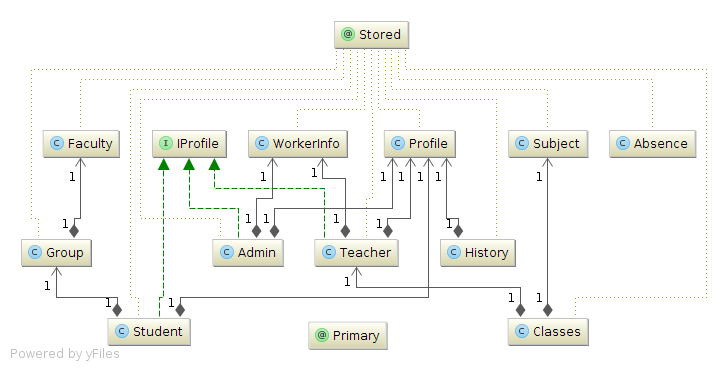
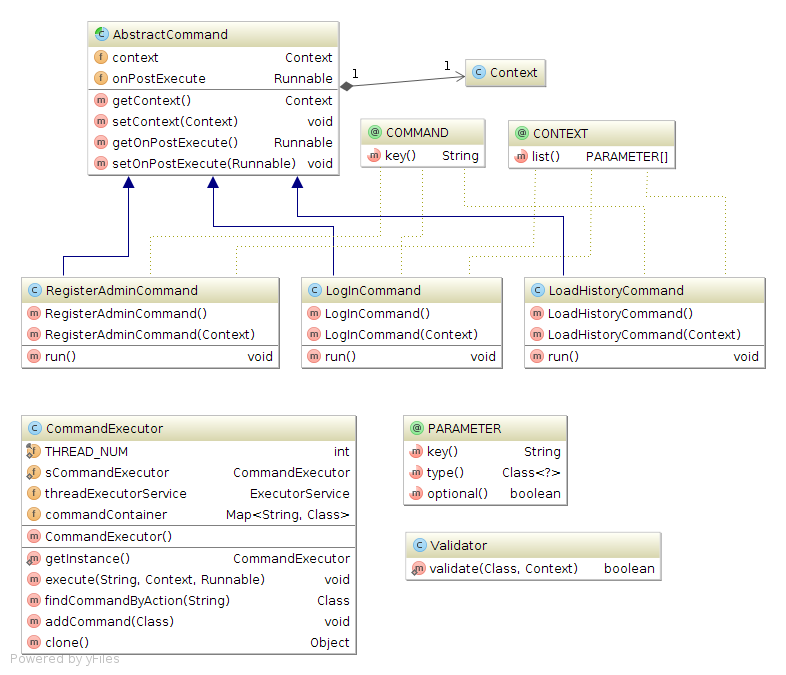
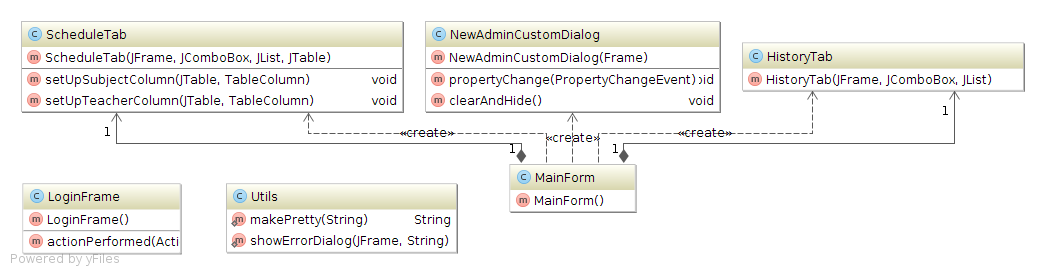
В розробленому додатку реалізовано наступний функціонал: можливість авторизації, створення нового викладача, адміністратора, студента, факультету, групи, предмету, а також їх редагування. Реалізовано можливість редагування розкладу групи, а також перегляду історії.

В процесі розробки програмного додатку, я ознайомився з основними класами пакету Swing, за допомогою яких будуються графічні додатки, а також закріпив ці знання на практиці.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Приемы объектно-ориентированого проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2011 – 368 с.: ил. –ISBN 978-5-469-01136-1.
2. Герберт Шилдт Java. Полное руководство, 8-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 20012. – 1104 с. – ISBN 978-5-8459-1759-1 (рус.)
3. Эккель Б. Философия Java / Эккель Брюс; Пер.с англ. Е.Матвеев.– 4-е изд.–СПб.: Питер, 2010. – 640с.: ил. – (Библиотека программиста). – Алф.указ.:с.631. – ISBN 978-5-388-00003-3.
4. Хорстманн Кей С. Java 2. Том 1. Основы / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 816 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1378-4 (рус.).
5. Хорстманн Кей С. Java 2. Том 2. Тонкости программирования / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 992 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1482-8 (рус.).
6. Стелтинг Стивен Применение шаблонов Java /Стелтинг Стивен, Маасен Олав; Пер. с англ. –М.: Издательский дом “Вильямс”, 2002. – 576 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). – ISBN 5-8459-0339-4 (рус.).
7. Дженифер Тидвелл Разработка пользовательских интрефейсов; Пер. с англ. –Е. Шикарева: Издательский дом “Питер”, 2008. – 416 c. – ISBN 978-5-91180-073-4 (рус.).
8. А. К. Гультяев Проектирование и дизайн пользовательского интерфейса / В. А. Машин; Пер. с англ. : Издательский дом “ Корона-Принт ”, 2010. – 350 c. – ISBN 978-5-7931-0814-0 (рус.).
9. Герберт Шилдт. Библиотека SWING для Java: руководство для начинающих - Вильямс , 2007 — C. 704 - ISBN 978-5-8459-1162-9, 0-07-2263148.
10. Марк Гранд Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560. — ISBN 5-94735-047-5

# ДОДАТОК А. ДІАГРАМИ КЛАСІВ ДОДАТКУ



# ДОДАТОК Б. ВИХІДНИЙ КОД ДОДАТКУ

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю FACULTY\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "FACULTY\_TBL")

public class Faculty {

@Primary

@Stored(name = "faculty\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "faculty\_name")

private String name;

public Faculty() {

}

public Faculty(UUID id, String name) {

this.id = id;

this.name = name;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return getName();

}

}

package com.oyster.app;

import com.oyster.app.model.Admin;

import com.oyster.config.AppConfig;

import com.oyster.dao.impl.DAOCRUDJdbc;

import org.springframework.context.ApplicationContext;

import java.text.SimpleDateFormat;

/\*\*

\* Клас містить константні поля(переважно статичні), що є спільними для

\* всього застосунку

\*/

public class AppConst {

private static Admin currentAdmin;

public static ApplicationContext CONTEXT;

public static SimpleDateFormat DATE\_FORMAT;

public static int SESSION\_TIME;

public static DAOCRUDJdbc DAO;

public static AppConfig APP\_CONFIG;

static {

DATE\_FORMAT = new SimpleDateFormat("d/M/y");

SESSION\_TIME = 0;

}

/\*\*

\* Повертає адміністратора, що останній раз авторизувався

\*

\* @return адміністратора, що залогінився, або null, якщо такого немає

\*/

public static Admin getCurrentAdmin() {

return currentAdmin;

}

/\*\*

\* Встановлює адміністратора, що розпочав сесію

\*

\* @param currentAdmin адміністратор, що здійснив авторизацію

\*/

public static void setCurrentAdmin(Admin currentAdmin) {

AppConst.currentAdmin = currentAdmin;

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю GROUP\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "GROUP\_TBL")

public class Group {

@Primary

@Stored(name = "group\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "faculty\_id", converter = UUIDConverter.class)

private UUID facultyId;

@Stored(name = "group\_name")

private String name;

private Faculty faculty;

public Group() {

}

public Group(UUID id, UUID facultyId, String name) {

this.id = id;

this.facultyId = facultyId;

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public UUID getFacultyId() {

return facultyId;

}

public void setFacultyId(UUID facultyId) {

this.facultyId = facultyId;

}

public Faculty getFaculty() {

return faculty;

}

public void setFaculty(Faculty faculty) {

this.faculty = faculty;

}

@Override

public String toString() {

return getName();

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю HISTORY\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "HISTORY\_TBL")

public class History {

@Primary

@Stored(name = "history\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "author\_id", converter = UUIDConverter.class)

private UUID authorId;

@Stored(name = "action")

private String action;

private Profile author;

public History() {

}

public History(UUID id, UUID authorId, String action) {

this.id = id;

this.authorId = authorId;

this.action = action;

}

public UUID getAuthorId() {

return authorId;

}

public void setAuthorId(UUID authorId) {

this.authorId = authorId;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getAction() {

return action;

}

public void setAction(String action) {

this.action = action;

}

@Override

public String toString() {

return String.format("%s %s : %s", getAuthor().getFirstName(), getAuthor().getSecondName(), getAction());

}

public Profile getAuthor() {

return author;

}

public void setAuthor(Profile author) {

this.author = author;

}

}

package com.oyster.app.model;

/\*\*

\* Інтерфейс для узагальнення сутностей, що містять граничний клас Profile серед полів

\*

\* @author bamboo

\*/

public interface IProfile {

/\*\*

\* повертає профіль класу

\*

\* @return граничний клас Profile

\*/

public Profile getProfile();

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.LongConverter;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю PROFILE\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "PROFILE\_TBL")

public class Profile {

@Primary

@Stored(name = "profile\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "first\_name")

private String firstName = "";

@Stored(name = "second\_name")

private String secondName = "";

@Stored(name = "password")

private String password;

@Stored(name = "birthday", converter = LongConverter.class)

private long birthday;

public Profile() {

}

public Profile(UUID id, String firstName, String secondName, String password, long birthday) {

this.id = id;

this.firstName = firstName;

this.secondName = secondName;

this.password = password;

this.birthday = birthday;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getSecondName() {

return secondName;

}

public void setSecondName(String secondName) {

this.secondName = secondName;

}

public long getBirthday() {

return birthday;

}

public void setBirthday(long birthday) {

this.birthday = birthday;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

@Override

public String toString() {

return String.format("%s %s", getFirstName(), getSecondName());

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.IntConverter;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю STUDENT\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "STUDENT\_TBL")

public class Student implements IProfile {

@Primary

@Stored(name = "student\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "profile\_id", converter = UUIDConverter.class)

private UUID profileId;

@Stored(name = "group\_id", converter = UUIDConverter.class)

private UUID groupId;

@Stored(name = "course", converter = IntConverter.class)

private int course;

@Stored(name = "book\_num", converter = IntConverter.class)

private int bookNum;

private Profile profile;

private Group group;

public Student() {

}

public Student(UUID id, UUID profileId, UUID groupId, int course, int bookNum) {

this.id = id;

this.profileId = profileId;

this.groupId = groupId;

this.course = course;

this.bookNum = bookNum;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public UUID getProfileId() {

return profileId;

}

public void setProfileId(UUID profileId) {

this.profileId = profileId;

}

public UUID getGroupId() {

return groupId;

}

public void setGroupId(UUID groupId) {

this.groupId = groupId;

}

public int getCourse() {

return course;

}

public void setCourse(int course) {

this.course = course;

}

public int getBookNum() {

return bookNum;

}

public void setBookNum(int bookNum) {

this.bookNum = bookNum;

}

public Profile getProfile() {

return profile;

}

public void setProfile(Profile profile) {

this.profile = profile;

}

public Group getGroup() {

return group;

}

public void setGroup(Group group) {

this.group = group;

}

@Override

public String toString() {

return getProfile().toString();

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю SUBJECT\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "SUBJECT\_TBL")

public class Subject {

@Primary

@Stored(name = "subject\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "subject\_name")

private String name;

public Subject() {

}

public Subject(UUID id, String name) {

this.id = id;

this.name = name;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return getName();

}

@Override

public boolean equals(Object obj) {

if (obj instanceof Subject) {

return ((Subject) obj).toString().equals(this.toString());

} else if (obj instanceof String) {

return ((String) obj).equals(this.toString());

}

return this.equals(obj);

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю TEACHER\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "TEACHER\_TBL")

public class Teacher implements IProfile {

@Primary

@Stored(name = "teacher\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "profile\_id", converter = UUIDConverter.class)

private UUID profileId;

@Stored(name = "worker\_info\_id", converter = UUIDConverter.class)

private UUID workerInfoId;

private Profile profile;

private WorkerInfo workerInfo;

public Teacher() {

}

public Teacher(UUID id, UUID profileId, UUID workerInfoId) {

this.id = id;

this.profileId = profileId;

this.workerInfoId = workerInfoId;

}

public UUID getWorkerInfoId() {

return workerInfoId;

}

public void setWorkerInfoId(UUID workerInfoId) {

this.workerInfoId = workerInfoId;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public UUID getProfileId() {

return profileId;

}

public void setProfileId(UUID profileId) {

this.profileId = profileId;

}

public Profile getProfile() {

return profile;

}

public void setProfile(Profile profile) {

this.profile = profile;

}

public WorkerInfo getWorkerInfo() {

return workerInfo;

}

public void setWorkerInfo(WorkerInfo workerInfo) {

this.workerInfo = workerInfo;

}

@Override

public String toString() {

return getProfile().toString();

}

@Override

public boolean equals(Object obj) {

if (obj instanceof Teacher) {

return ((Teacher) obj).toString().equals(this.toString());

} else if (obj instanceof String) {

return ((String) obj).equals(this.toString());

}

return this.equals(obj);

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.IntConverter;

import com.oyster.dao.annotation.utils.converter.LongConverter;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю WORKER\_INFO\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "WORKER\_INFO\_TBL")

public class WorkerInfo {

@Primary

@Stored(name = "worker\_info\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "position")

private String position;

@Stored(name = "salary", converter = IntConverter.class)

private int salary;

@Stored(name = "date\_hired", converter = LongConverter.class)

private long dateHired;

public WorkerInfo() {

}

public WorkerInfo(UUID id, String position, int salary, long dateHired) {

this.id = id;

this.position = position;

this.salary = salary;

this.dateHired = dateHired;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getPosition() {

return position;

}

public void setPosition(String position) {

this.position = position;

}

public int getSalary() {

return salary;

}

public void setSalary(int salary) {

this.salary = salary;

}

public long getDateHired() {

return dateHired;

}

public void setDateHired(long dateHired) {

this.dateHired = dateHired;

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю ABSENCE\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "ABSENCE\_TBL")

public class Absence {

@Primary

@Stored(name = "absence\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "group\_id", converter = UUIDConverter.class)

private UUID groupId;

@Stored(name = "class\_id", converter = UUIDConverter.class)

private UUID classId;

public Absence() {

}

public Absence(UUID id, UUID groupId, UUID classId) {

this.id = id;

this.groupId = groupId;

this.classId = classId;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public UUID getGroupId() {

return groupId;

}

public void setGroupId(UUID groupId) {

this.groupId = groupId;

}

public UUID getClassId() {

return classId;

}

public void setClassId(UUID classId) {

this.classId = classId;

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю ADMIN\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "ADMIN\_TBL")

public class Admin implements IProfile {

@Primary

@Stored(name = "admin\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "profile\_id", converter = UUIDConverter.class)

private UUID profileId;

@Stored(name = "worker\_info\_id", converter = UUIDConverter.class)

private UUID workerInfoId;

private Profile profile;

private WorkerInfo workerInfo;

public Admin() {

}

public Admin(UUID id, UUID profileId, UUID workerInfoId) {

this.id = id;

this.profileId = profileId;

this.workerInfoId = workerInfoId;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public UUID getWorkerInfoId() {

return workerInfoId;

}

public void setWorkerInfoId(UUID workerInfoId) {

this.workerInfoId = workerInfoId;

}

public UUID getProfileId() {

return profileId;

}

public void setProfileId(UUID profileId) {

this.profileId = profileId;

}

public Profile getProfile() {

return profile;

}

public void setProfile(Profile profile) {

this.profile = profile;

}

public WorkerInfo getWorkerInfo() {

return workerInfo;

}

public void setWorkerInfo(WorkerInfo workerInfo) {

this.workerInfo = workerInfo;

}

@Override

public String toString() {

return getProfile().toString();

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.IntConverter;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю CLASSES\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "CLASSES\_TBL")

public class Classes {

@Primary

@Stored(name = "classes\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "subject\_id", converter = UUIDConverter.class)

private UUID subjectId;

@Stored(name = "teacher\_id", converter = UUIDConverter.class)

private UUID teacherId;

@Stored(name = "audience", converter = IntConverter.class)

private int audience;

@Stored(name = "class\_date", converter = IntConverter.class)

private int time;

private Subject subject;

private Teacher teacher;

public Classes() {

}

public Classes(UUID id, UUID subjectId, UUID teacherId, int audience, int time) {

this.id = id;

this.subjectId = subjectId;

this.teacherId = teacherId;

this.audience = audience;

this.time = time;

}

public int getTime() {

return time;

}

public void setTime(int time) {

this.time = time;

}

public int getAudience() {

return audience;

}

public void setAudience(int audience) {

this.audience = audience;

}

public UUID getTeacherId() {

return teacherId;

}

public void setTeacherId(UUID teacherId) {

this.teacherId = teacherId;

}

public UUID getSubjectId() {

return subjectId;

}

public void setSubjectId(UUID subjectId) {

this.subjectId = subjectId;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public Subject getSubject() {

return subject;

}

public void setSubject(Subject subject) {

this.subject = subject;

}

public Teacher getTeacher() {

return teacher;

}

public void setTeacher(Teacher teacher) {

this.teacher = teacher;

}

}

package com.oyster.app.model;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.converter.UUIDConverter;

import java.util.UUID;

/\*\*

\* Граничний клас, що представляє таблицю FACULTY\_TBL у базі даних

\*

\* @author bamboo

\*/

@Stored(name = "FACULTY\_TBL")

public class Faculty {

@Primary

@Stored(name = "faculty\_id", converter = UUIDConverter.class)

private UUID id;

@Stored(name = "faculty\_name")

private String name;

public Faculty() {

}

public Faculty(UUID id, String name) {

this.id = id;

this.name = name;

}

public UUID getId() {

return id;

}

public void setId(UUID id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return getName();

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.config.impl;

import org.codehaus.jackson.map.ObjectMapper;

import org.codehaus.jackson.type.TypeReference;

import com.oyster.config.ConfigReader;

import com.oyster.config.IConfig;

import java.io.File;

import java.util.Map;

import java.util.Set;

/\*\*

\* клас реалізує зчитування файлу конфігурації із JSON - файлу

\*/

public class JSONConfigReader extends ConfigReader {

/\*\*

\* Конструктор

\* @param config об’єкт конфігурації

\*/

public JSONConfigReader(IConfig config) {

this.config = config;

}

/\*\*

\* завантажує зовнішній файл конфігурації

\* @param path шлях до файлу

\*/

@Override

public void loadFromFile(String path) {

try {

ObjectMapper mapper = new ObjectMapper();

// read JSON from a file

Map<String, Object> map = mapper.readValue(

new File(path),

new TypeReference<Map<String, Object>>() {

}

);

Set<String> set = map.keySet();

for (String key : set) {

config.setValue(key, map.get(key));

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.core.controller.utils;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

/\*\*

\* Допоміжний клас для роботи із аннотаціями команди

\*/

public class ControllerAnnotationUtils {

/\*\*

\* повертає ключ команди

\*

\* @param c Клас команди

\* @return ключ команди

\*/

public static String getCommandKey(Class c) {

COMMAND t = (COMMAND) c.getAnnotation(COMMAND.class);

return (t != null) ? t.key() : null;

}

/\*\*

\* повертає список усіх параметрів команди

\*

\* @param c Клас команди

\* @return список усіх параметрів команди

\*/

public static PARAMETER[] getParameterList(Class c) {

CONTEXT t = (CONTEXT) c.getAnnotation(CONTEXT.class);

if (t == null) return null;

return t.list();

}

/\*\*

\* визначає, чи є даний параметр обов’язковим

\*

\* @param paramKey ключ параметра

\* @param command команда, у якій перевіряємо

\* @return true - якщо необов’язковий, інакше - false

\*/

public static boolean paramIsOptional(String paramKey, Class command) {

PARAMETER[] pl = getParameterList(command);

for (PARAMETER p : pl) {

if (p.key().equals(paramKey)) return p.optional();

}

return true;

}

}

package com.oyster.core.controller.utils;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

/\*\*

\* Допоміжний клас для роботи із аннотаціями команди

\*/

public class ControllerAnnotationUtils {

/\*\*

\* повертає ключ команди

\*

\* @param c Клас команди

\* @return ключ команди

\*/

public static String getCommandKey(Class c) {

COMMAND t = (COMMAND) c.getAnnotation(COMMAND.class);

return (t != null) ? t.key() : null;

}

/\*\*

\* повертає список усіх параметрів команди

\*

\* @param c Клас команди

\* @return список усіх параметрів команди

\*/

public static PARAMETER[] getParameterList(Class c) {

CONTEXT t = (CONTEXT) c.getAnnotation(CONTEXT.class);

if (t == null) return null;

return t.list();

}

/\*\*

\* визначає, чи є даний параметр обов’язковим

\*

\* @param paramKey ключ параметра

\* @param command команда, у якій перевіряємо

\* @return true - якщо необов’язковий, інакше - false

\*/

public static boolean paramIsOptional(String paramKey, Class command) {

PARAMETER[] pl = getParameterList(command);

for (PARAMETER p : pl) {

if (p.key().equals(paramKey)) return p.optional();

}

return true;

}

}

package com.oyster.core.controller.utils;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

/\*\*

\* Допоміжний клас для роботи із аннотаціями команди

\*/

public class ControllerAnnotationUtils {

/\*\*

\* повертає ключ команди

\*

\* @param c Клас команди

\* @return ключ команди

\*/

public static String getCommandKey(Class c) {

COMMAND t = (COMMAND) c.getAnnotation(COMMAND.class);

return (t != null) ? t.key() : null;

}

/\*\*

\* повертає список усіх параметрів команди

\*

\* @param c Клас команди

\* @return список усіх параметрів команди

\*/

public static PARAMETER[] getParameterList(Class c) {

CONTEXT t = (CONTEXT) c.getAnnotation(CONTEXT.class);

if (t == null) return null;

return t.list();

}

/\*\*

\* визначає, чи є даний параметр обов’язковим

\*

\* @param paramKey ключ параметра

\* @param command команда, у якій перевіряємо

\* @return true - якщо необов’язковий, інакше - false

\*/

public static boolean paramIsOptional(String paramKey, Class command) {

PARAMETER[] pl = getParameterList(command);

for (PARAMETER p : pl) {

if (p.key().equals(paramKey)) return p.optional();

}

return true;

}

}

package com.oyster.core.controller.command;

import com.oyster.app.AppConst;

import com.oyster.app.model.Admin;

import com.oyster.app.model.Profile;

import com.oyster.app.model.WorkerInfo;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import com.oyster.ui.MainForm;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.List;

/\*\*

\* команда виконує авторизацію користувача із логіном та паролем, що ії передаються

\*/

@COMMAND(key = "logIn")

@CONTEXT(list = {

@PARAMETER(key = "username", type = String.class),

@PARAMETER(key = "password", type = String.class)

})

public class LogInCommand extends AbstractCommand {

public LogInCommand() {

}

/\*\*

\* Конструктор

\*

\* @param context1 контекст команди

\*/

public LogInCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

final String userName = (String) context.get("username");

final String userPassword = (String) context.get("password");

List<Profile> profiles = null;

try {

profiles = AppConst.DAO.select(Profile.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Profile p = (Profile) entity;

return userName.equals(p.getFirstName() + "-" + p.getSecondName())

&& userPassword.equals(p.getPassword());

}

});

} catch (final DAOException e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty(e.getMessage()));

}

});

return;

}

if (profiles.size() == 0) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, "Помилка авторизації : неправильні логін чи пароль !");

}

});

return;

}

final Profile profile = profiles.get(0);

WorkerInfo wi = null;

List<Admin> admins = null;

try {

admins = AppConst.DAO.select(Admin.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Admin a = (Admin) entity;

return a.getProfileId().equals(profile.getId());

}

});

wi = AppConst.DAO.read(WorkerInfo.class, admins.get(0).getWorkerInfoId());

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty(e.getMessage()));

}

});

return;

}

if (admins.size() == 0) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, "Помилка авторизації : дані логін та пароль не відповідають адміністратору !");

}

});

return;

}

final Admin admin = admins.get(0);

admin.setProfile(profile);

admin.setWorkerInfo(wi);

AppConst.setCurrentAdmin(admin);

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

MainForm form = new MainForm();

}

});

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command;

/\*\*

\* Абстпрактний клас, що реалізує базову команду,

\* містить контекст, та екземпляр інтерфейсу Runnable,

\* та методи його отримання

\*

\* @author bamboo

\*/

public abstract class AbstractCommand implements Runnable {

protected Context context;

protected Runnable onPostExecute;

/\*\*

\* повертає контекст команди

\*

\* @return контекст команди

\*/

public Context getContext() {

return context;

}

/\*\*

\* встановлює контекст команди

\*

\* @param context контекст команди

\*/

public void setContext(Context context) {

this.context = context;

}

/\*\*

\* повертає екземпляр інтерфейсу Runnable, метод якого run()

\* буде виконаний у UI потоці

\*

\* @return екземпляр інтерфейсу Runnable

\*/

public Runnable getOnPostExecute() {

return onPostExecute;

}

/\*\*

\* встановлює екземпляр інтерфейсу Runnable, метод якого run()

\* буде виконаний у UI потоці

\*

\* @param runnable екземпляр інтерфейсу Runnable

\*/

public void setOnPostExecute(Runnable runnable) {

onPostExecute = runnable;

}

}

package com.oyster.core.controller.command;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.dao.exception.DAOException;

import javax.swing.\*;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* команда виконує завантаження історії, виконуюючи SQL-запит, що їй передається

\*/

@COMMAND(key = "loadHistory")

@CONTEXT(list = {

@PARAMETER(key = "sqlQuery", type = String.class),

@PARAMETER(key = "list", type = ArrayList.class)

})

public class LoadHistoryCommand extends AbstractCommand {

public LoadHistoryCommand() {

}

/\*\*

\* Конструктор

\*

\* @param context1 контекст команди

\*/

public LoadHistoryCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

String sqlQuery = (String) context.get("sqlQuery");

List<History> histories = (List<History>) context.get("list");

try {

List<History> list = AppConst.DAO.select(History.class, sqlQuery);

for (History h : list) {

histories.add(h);

}

} catch (DAOException e) {

e.printStackTrace();

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.app.model.IProfile;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.dao.exception.DAOException;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує видалення профілю, що передається ії параметром

\*

\*/

@COMMAND(key = "deleteIProfile")

@CONTEXT(list = {

@PARAMETER(key = "profile", type = IProfile.class)

})

public class DeleteIProfileCommand extends AbstractCommand {

public DeleteIProfileCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public DeleteIProfileCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

IProfile currentPerson = (IProfile) context.get("profile");

try {

AppConst.DAO.delete(currentPerson.getProfile());

AppConst.DAO.delete(currentPerson);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Видалив користувача " + currentPerson.getProfile().toString()

);

AppConst.DAO.insert(h);

} catch (DAOException e) {

e.printStackTrace();

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command;

import com.oyster.app.AppConst;

import com.oyster.app.model.Classes;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.dao.exception.DAOException;

import javax.swing.\*;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* команда виконує завантаження розкладу, виконуюючи SQL-запит, що їй передається

\*/

@COMMAND(key = "loadSchedule")

@CONTEXT(list = {

@PARAMETER(key = "sqlQuery", type = String.class),

@PARAMETER(key = "list", type = ArrayList.class)

})

public class LoadScheduleCommand extends AbstractCommand {

public LoadScheduleCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public LoadScheduleCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

String sqlQuery = (String) context.get("sqlQuery");

List<Classes> classesList = (List<Classes>) context.get("list");

try {

List<Classes> list = AppConst.DAO.select(Classes.class, sqlQuery);

for (Classes c : list) {

classesList.add(c);

}

} catch (DAOException e) {

e.printStackTrace();

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command;

import com.oyster.app.AppConst;

import com.oyster.app.model.Admin;

import com.oyster.app.model.Profile;

import com.oyster.app.model.WorkerInfo;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import com.oyster.ui.MainForm;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.List;

/\*\*

\* команда виконує авторизацію користувача із логіном та паролем, що ії передаються

\*/

@COMMAND(key = "logIn")

@CONTEXT(list = {

@PARAMETER(key = "username", type = String.class),

@PARAMETER(key = "password", type = String.class)

})

public class LogInCommand extends AbstractCommand {

public LogInCommand() {

}

/\*\*

\* Конструктор

\*

\* @param context1 контекст команди

\*/

public LogInCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

final String userName = (String) context.get("username");

final String userPassword = (String) context.get("password");

List<Profile> profiles = null;

try {

profiles = AppConst.DAO.select(Profile.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Profile p = (Profile) entity;

return userName.equals(p.getFirstName() + "-" + p.getSecondName())

&& userPassword.equals(p.getPassword());

}

});

} catch (final DAOException e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty(e.getMessage()));

}

});

return;

}

if (profiles.size() == 0) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, "Помилка авторизації : неправильні логін чи пароль !");

}

});

return;

}

final Profile profile = profiles.get(0);

WorkerInfo wi = null;

List<Admin> admins = null;

try {

admins = AppConst.DAO.select(Admin.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Admin a = (Admin) entity;

return a.getProfileId().equals(profile.getId());

}

});

wi = AppConst.DAO.read(WorkerInfo.class, admins.get(0).getWorkerInfoId());

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty(e.getMessage()));

}

});

return;

}

if (admins.size() == 0) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, "Помилка авторизації : дані логін та пароль не відповідають адміністратору !");

}

});

return;

}

final Admin admin = admins.get(0);

admin.setProfile(profile);

admin.setWorkerInfo(wi);

AppConst.setCurrentAdmin(admin);

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

MainForm form = new MainForm();

}

});

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.app.model.Profile;

import com.oyster.app.model.Teacher;

import com.oyster.app.model.WorkerInfo;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію викладача у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerTeacher")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class),

@PARAMETER(key = "surname", type = String.class),

@PARAMETER(key = "birthday", type = Long.class, optional = true),

@PARAMETER(key = "position", type = String.class),

@PARAMETER(key = "salary", type = Integer.class),

@PARAMETER(key = "password", type = String.class),

@PARAMETER(key = "dateHired", type = Long.class, optional = true)

})

public class RegisterTeacherCommand extends AbstractCommand {

public RegisterTeacherCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterTeacherCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Profile pTeacher = new Profile(

UUID.randomUUID(),

(String) context.get("name"),

(String) context.get("surname"),

(String) context.get("password"),

(Long) context.get("birthday")

);

WorkerInfo wTeacher = new WorkerInfo(

UUID.randomUUID(),

(String) context.get("position"),

(Integer) context.get("salary"),

(Long) context.get("dateHired")

);

Teacher teacher = new Teacher(

UUID.randomUUID(),

pTeacher.getId(),

wTeacher.getId()

);

try {

AppConst.DAO.insert(pTeacher);

AppConst.DAO.insert(wTeacher);

AppConst.DAO.insert(teacher);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Додав викладача " + pTeacher.toString()

);

AppConst.DAO.insert(h);

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null,

Utils.makePretty("Помилка створення викладача : \n" + e.getMessage()));

}

});

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.Faculty;

import com.oyster.app.model.History;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію факультету у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerFaculty")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class)

})

public class RegisterFacultyCommand extends AbstractCommand {

public RegisterFacultyCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterFacultyCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Faculty fac = new Faculty(UUID.randomUUID(), ((String) context.get("name")).toUpperCase());

try {

AppConst.DAO.insert(fac);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Створив новий факультет " + fac.getName()

);

AppConst.DAO.insert(h);

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty("Помилка створення факультету : \n" + e.getMessage()));

}

});

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.\*;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію студента у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerStudent")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class),

@PARAMETER(key = "surname", type = String.class),

@PARAMETER(key = "birthday", type = Long.class, optional = true),

@PARAMETER(key = "faculty", type = String.class),

@PARAMETER(key = "group", type = String.class),

@PARAMETER(key = "course", type = Integer.class),

@PARAMETER(key = "password", type = String.class),

@PARAMETER(key = "bookNum", type = Integer.class)

})

public class RegisterStudentCommand extends AbstractCommand {

public RegisterStudentCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterStudentCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Profile pStudent = new Profile(

UUID.randomUUID(),

(String) context.get("name"),

(String) context.get("surname"),

(String) context.get("password"),

(Long) context.get("birthday")

);

final String groupName = (String) context.get("group");

java.util.List<Group> groups = null;

try {

groups = AppConst.DAO.select(Group.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Group g = (Group) entity;

return g.getName().equals(groupName);

}

});

for (Group gg : groups) {

gg.setFaculty((Faculty) AppConst.DAO.read(Faculty.class, gg.getFacultyId()));

}

} catch (final DAOException e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty("Помилка зчитування групи : \n" + e.getMessage()));

}

});

}

if (groups.size() == 0) {

Utils.showErrorDialog(null, "Немає такої групи!");

return;

}

final Group group = groups.get(0);

if (!group.getFaculty().getName().equals((String) context.get("faculty"))) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, "Немає такої групи на цьому факультеті!");

}

});

return;

}

Student student = new Student(

UUID.randomUUID(),

pStudent.getId(),

groups.get(0).getId(),

(Integer) context.get("course"),

(Integer) context.get("bookNum")

);

try {

AppConst.DAO.insert(pStudent);

AppConst.DAO.insert(student);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Додав студента " + pStudent.toString()

);

AppConst.DAO.insert(h);

} catch (final DAOException e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty("Помилка створення студента : \n" + e.getMessage()));

}

});

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.app.model.Subject;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію предмету у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerSubject")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class)

})

public class RegisterSubjectCommand extends AbstractCommand {

public RegisterSubjectCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterSubjectCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Subject subject = new Subject(UUID.randomUUID(), ((String) context.get("name")).toUpperCase());

try {

AppConst.DAO.insert(subject);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Створив новий предмет " + subject.getName()

);

AppConst.DAO.insert(h);

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null, Utils.makePretty("Помилка створення предмету : \n" + e.getMessage()));

}

});

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.Admin;

import com.oyster.app.model.History;

import com.oyster.app.model.Profile;

import com.oyster.app.model.WorkerInfo;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію адміністратора у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerAdmin")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class),

@PARAMETER(key = "surname", type = String.class),

@PARAMETER(key = "birthday", type = Long.class, optional = true),

@PARAMETER(key = "position", type = String.class),

@PARAMETER(key = "salary", type = Integer.class),

@PARAMETER(key = "password", type = String.class),

@PARAMETER(key = "dateHired", type = Long.class, optional = true)

})

public class RegisterAdminCommand extends AbstractCommand {

public RegisterAdminCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterAdminCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Profile pAdmin = new Profile(

UUID.randomUUID(),

(String) context.get("name"),

(String) context.get("surname"),

(String) context.get("password"),

(Long) context.get("birthday")

);

WorkerInfo wAdmin = new WorkerInfo(

UUID.randomUUID(),

(String) context.get("position"),

(Integer) context.get("salary"),

(Long) context.get("dateHired")

);

Admin admin = new Admin(

UUID.randomUUID(),

pAdmin.getId(),

wAdmin.getId()

);

try {

AppConst.DAO.insert(pAdmin);

AppConst.DAO.insert(wAdmin);

AppConst.DAO.insert(admin);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Додав адміністратора " + pAdmin.toString()

);

AppConst.DAO.insert(h);

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null,

Utils.makePretty("Помилка створення адміністратора : \n" + e.getMessage()));

}

});

;

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.core.controller.command.register;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.app.model.Profile;

import com.oyster.app.model.Teacher;

import com.oyster.app.model.WorkerInfo;

import com.oyster.core.controller.annotation.COMMAND;

import com.oyster.core.controller.annotation.CONTEXT;

import com.oyster.core.controller.annotation.PARAMETER;

import com.oyster.core.controller.command.AbstractCommand;

import com.oyster.core.controller.command.Context;

import com.oyster.ui.Utils;

import javax.swing.\*;

import java.util.UUID;

/\*\*

\* команда виконує реєстрацію викладача у системі, інформацію про якого їй передається у контексті

\* @author bamboo

\*/

@COMMAND(key = "registerTeacher")

@CONTEXT(list = {

@PARAMETER(key = "name", type = String.class),

@PARAMETER(key = "surname", type = String.class),

@PARAMETER(key = "birthday", type = Long.class, optional = true),

@PARAMETER(key = "position", type = String.class),

@PARAMETER(key = "salary", type = Integer.class),

@PARAMETER(key = "password", type = String.class),

@PARAMETER(key = "dateHired", type = Long.class, optional = true)

})

public class RegisterTeacherCommand extends AbstractCommand {

public RegisterTeacherCommand() {

}

/\*\*

\* Конструктор

\* @param context1 контекст команди

\*/

public RegisterTeacherCommand(Context context1) {

setContext(context1);

}

/\*\*

\* виконує роботу команди

\*/

@Override

public void run() {

Profile pTeacher = new Profile(

UUID.randomUUID(),

(String) context.get("name"),

(String) context.get("surname"),

(String) context.get("password"),

(Long) context.get("birthday")

);

WorkerInfo wTeacher = new WorkerInfo(

UUID.randomUUID(),

(String) context.get("position"),

(Integer) context.get("salary"),

(Long) context.get("dateHired")

);

Teacher teacher = new Teacher(

UUID.randomUUID(),

pTeacher.getId(),

wTeacher.getId()

);

try {

AppConst.DAO.insert(pTeacher);

AppConst.DAO.insert(wTeacher);

AppConst.DAO.insert(teacher);

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Додав викладача " + pTeacher.toString()

);

AppConst.DAO.insert(h);

} catch (final Exception e) {

e.printStackTrace();

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

Utils.showErrorDialog(null,

Utils.makePretty("Помилка створення викладача : \n" + e.getMessage()));

}

});

}

if (getOnPostExecute() != null) {

SwingUtilities.invokeLater(getOnPostExecute());

}

}

}

package com.oyster.dao.annotation.utils.converter;

import com.oyster.dao.annotation.utils.ValueConverter;

import java.util.UUID;

/\*\*

\* конвертує сутність (String <==> UUID)

\*/

public class UUIDConverter implements ValueConverter {

/\*\*

\* перетворює UUID в String

\*

\* @param value параметр для конвертації

\* @param <T> тип параметру

\* @return параметр як стрічку

\*/

@Override

public <T> String toString(T value) {

if (value == null) return null;

return "\"" + ((Object) value).toString() + "\"";

}

/\*\*

\* перетворює String в UUID

\*

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return стрічку як параметр

\*/

@Override

public <T> T toValue(String str) {

if (str == null) return null;

if (str.equals("null")) return null;

str = str.substring(1, str.length() - 1);

return (T) UUID.fromString(str);

}

}

package com.oyster.dao;

import com.oyster.dao.exception.DAOException;

import java.util.List;

import java.util.UUID;

/\*\*

\* визначає CRUD-інтерфейс для базових операцій із постійним місцем збереження

\*/

public interface CRUDInterface {

/\*\*

\* виконує базову операцію вставлення для даної сутності

\*

\* @param instance сутність

\* @param <T> тип сутності

\* @return саму сутність

\* @throws DAOException

\*/

public <T> T insert(T instance) throws DAOException;

/\*\*

\* виконує базову операцію зчитування для даної сутності

\*

\* @param entityClass клас сутності

\* @param id ключ сутності

\* @param <T> тип сутності

\* @return саму сутність

\* @throws DAOException

\*/

public <T> T read(Class entityClass, UUID id) throws DAOException;

/\*\*

\* виконує базову операцію оновлення для даної сутності

\*

\* @param instance сутність

\* @param <T> тип сутності

\* @throws DAOException

\*/

public <T> void update(T instance) throws DAOException;

/\*\*

\* виконує базову операцію видалення для даної сутності

\*

\* @param instance сутність

\* @param <T> тип сутності

\* @throws DAOException

\*/

public <T> void delete(T instance) throws DAOException;

/\*\*

\* виконує пошук сутностей у постійному хранилищі

\*

\* @param entityClass клас сутності

\* @param filter фільтр

\* @param <T> тип сутності

\* @return список знайдених сутностей, що задовільняють даному фільтру

\* @throws DAOException

\*/

public <T> List<T> select(Class entityClass, DAOFilter filter) throws DAOException;

/\*\*

\* виконує пошук сутностей у базі даних

\*

\* @param entityClass клас сутності

\* @param SQLString SQL-запит до бази даних

\* @param <T> тип сутності

\* @return список знайдених сутностей

\* @throws DAOException

\*/

public <T> List<T> select(Class entityClass, String SQLString) throws DAOException;

}

package com.oyster.dao.impl;

import com.oyster.dao.CRUDInterface;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.annotation.Stored;

import com.oyster.dao.annotation.utils.DAOAnnotationUtils;

import com.oyster.dao.exception.DAOException;

import org.springframework.context.ApplicationContext;

import org.springframework.dao.EmptyResultDataAccessException;

import org.springframework.jdbc.core.support.JdbcDaoSupport;

import java.lang.reflect.Field;

import java.util.\*;

/\*\*

\* Клас реалізує CRUDInterface для доступу до локальної бази даних на комп’ютері

\*

\* @author bamboo

\*/

public class DAOCRUDJdbc extends JdbcDaoSupport implements CRUDInterface {

/\*\*

\* отримує екземпляр JDBC dao

\*

\* @param context контекст додатку

\* @return екземпляр Jdbc dao

\*/

public static DAOCRUDJdbc getInstance(ApplicationContext context) {

return (DAOCRUDJdbc) context.getBean("DAOJdbc");

}

/\*\*

\* додає екземпляр нової сутності у сховище даних

\*

\* @param instance сутність для вставки

\* @param <T> тип сутності

\* @return саму сутність

\* @throws DAOException

\*/

@Override

public <T> T insert(T instance) throws DAOException {

return performInsertOrReplace(instance, "INSERT");

}

private <T> T performInsertOrReplace(T instance, String keyWord) {

HashMap<String, String> mapStrStr = (HashMap<String, String>)

DAOAnnotationUtils.getConvertedStoredFields(instance);

StringBuilder sb = new StringBuilder();

StringBuilder q = new StringBuilder();

for (String key : mapStrStr.keySet()) {

sb.append(", " + key);

q.append(", " + mapStrStr.get(key));

}

String sql = keyWord + " INTO " + DAOAnnotationUtils.getStorageName(instance.getClass())

+ " (" + sb.substring(1) + ")" + "VALUES" + " (" + q.substring(1) + ");";

System.out.println(sql);

getJdbcTemplate().update(sql);

return instance;

}

/\*\*

\* замінює екземпляр сутності у сховищі даних

\*

\* @param instance сутність для вставки

\* @param <T> тип сутності

\* @return саму сутність

\* @throws DAOException

\*/

public <T> T replace(T instance) throws DAOException {

return performInsertOrReplace(instance, "REPLACE");

}

/\*\*

\* запит сутності у сховищі даних по його UUID

\*

\* @param entityClass клас-обгортка сутності

\* @param id унікальний ідентифікатор сутності

\* @param <T> тип сутності

\* @return сутність, якщо така існує, інакше null

\* @throws DAOException

\*/

@Override

public <T> T read(Class entityClass, UUID id) throws DAOException {

Field primaryKeyField = DAOAnnotationUtils.getPrimaryKey(entityClass);

String sql = "SELECT \* FROM " + DAOAnnotationUtils.getStorageName(entityClass)

+ " WHERE " + primaryKeyField.getAnnotation(Stored.class).name()

+ "=\"" + id.toString() + "\";";

Map<String, Object> map = null;

try {

map = getJdbcTemplate().queryForMap(sql);

} catch (EmptyResultDataAccessException e) {

return null;

}

System.out.println(sql);

return DAOAnnotationUtils.mapToEntity(entityClass, map);

}

/\*\*

\* оновлює сутність у сховищі даних

\*

\* @param instance сутність для оновлення

\* @param <T> тип сутності

\* @throws DAOException

\*/

@Override

public <T> void update(T instance) throws DAOException {

HashMap<String, String> mapStrStr = (HashMap<String, String>)

DAOAnnotationUtils.getConvertedStoredFields(instance);

StringBuilder sb = new StringBuilder();

for (String key : mapStrStr.keySet()) {

sb.append(", " + key + "=" + mapStrStr.get(key));

}

Field primaryKeyField = DAOAnnotationUtils.getPrimaryKey(instance.getClass());

String sql = "UPDATE " + DAOAnnotationUtils.getStorageName(instance.getClass())

+ " SET " + sb.substring(1) + " WHERE "

+ primaryKeyField.getAnnotation(Stored.class).name()

+ "=" + DAOAnnotationUtils.getStringValue(instance, primaryKeyField) + ";";

System.out.println(sql);

getJdbcTemplate().update(sql);

}

/\*\*

\* видаляє сутність із сховища даних

\*

\* @param instance сутність для видалення

\* @param <T> тип сутності

\* @throws DAOException

\*/

@Override

public <T> void delete(T instance) throws DAOException {

Field primaryKeyField = DAOAnnotationUtils.getPrimaryKey(instance.getClass());

String sql = "DELETE FROM " + DAOAnnotationUtils.getStorageName(instance.getClass())

+ " WHERE "

+ primaryKeyField.getAnnotation(Stored.class).name()

+ "=" + DAOAnnotationUtils.getStringValue(instance, primaryKeyField) + ";";

getJdbcTemplate().update(sql);

}

/\*\*

\* запит для доступу до сховища даних

\*

\* @param entityClass клас-обгортка сутності

\* @param filter умови, яким повинна відповідати сутність

\* @param <T> тип сутності

\* @return список сутностей, що відповідають умовам фільтру

\* @throws DAOException

\*/

@Override

public <T> List<T> select(Class entityClass, DAOFilter filter) throws DAOException {

return select(entityClass, filter, null);

}

/\*\*

\* запит для доступу до сховища даних

\*

\* @param entityClass клас-обгортка сутності

\* @param SQLString sql select

\* @param <T> тип сутності

\* @return список сутностей, що відповідають the sql-clause

\* @throws DAOException

\*/

@Override

public <T> List<T> select(Class entityClass, String SQLString) throws DAOException {

return select(entityClass, null, SQLString);

}

/\*\*

\* допоміжна функція для пертворення різних типів запитів

\*

\* @param entityClass клас-обгортка сутності

\* @param filter умови, яким повинна відповідати сутність

\* @param sql sql select-where clause

\* @param <T> тип сутності

\* @return список сутностей, що відповідають the sql-clause і умовам фільтру

\*/

private <T> List<T> select(Class entityClass, DAOFilter filter, String sql) {

if (sql == null || sql.trim().length() == 0) {

sql = "SELECT \* FROM " + DAOAnnotationUtils.getStorageName(entityClass) + ";";

}

if (filter == null) {

filter = new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

return true;

}

};

}

List<Map<String, Object>> list = getJdbcTemplate().queryForList(sql);

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

ArrayList<T> arrayList = new ArrayList<T>();

for (int i = 0; i < list.size(); i++) {

T instance = DAOAnnotationUtils.mapToEntity(entityClass, list.get(i));

if (filter.accept(instance)) {

arrayList.add(instance);

}

}

return arrayList;

}

}

package com.oyster.dao.annotation;

import com.oyster.dao.annotation.utils.converter.StringConverter;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

/\*\*

\* анотація визначає ключ поля, за яким зберігається поле, а також ковертер для нього,

\*/

@Retention(RetentionPolicy.RUNTIME)

public @interface Stored {

/\*\*

\* @return name ключ поля, за яким параметр зберігається

\*/

public String name();

/\*\*

\* повертає ValueConverter для даної сутності

\*

\* @return ValueConverter для даної сутності

\*/

public Class converter() default StringConverter.class;

}

package com.oyster.dao.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* Анотація, що означає, що поле позначене як PRIMARY KEY у сховищі даних

\*/

@Retention(RetentionPolicy.RUNTIME)

@Target(value = ElementType.FIELD)

public @interface Primary {

}

package com.oyster.dao.annotation.utils;

import com.oyster.dao.annotation.Primary;

import com.oyster.dao.annotation.Stored;

import java.beans.IntrospectionException;

import java.beans.PropertyDescriptor;

import java.lang.reflect.Field;

import java.lang.reflect.InvocationTargetException;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

public class DAOAnnotationUtils {

/\*\*

\* @param c клас чи поле для повернення {@link com.oyster.dao.annotation.Stored} анотації

\* @return ім’я,якщо клас має анотацію Stored

\*/

public static String getStorageName(Class c) {

Stored t = (Stored) c.getAnnotation(Stored.class);

return (t != null) ? t.name() : null;

}

/\*\*

\* @param c клас, з якого потрібно повернути інформацію

\* @return primary key класу

\*/

public static Field getPrimaryKey(Class c) {

Field[] fields = c.getDeclaredFields();

for (Field field : fields) {

Primary p = field.getAnnotation(Primary.class);

if (p != null) {

return field;

}

}

return null;

}

/\*\*

\* повертає значення UUID з поля Primary Key, вважається, що його тип UUID

\*

\* @param instance екземпляр класу з якого потрібно отримати інформацію

\* @param <T> тип класу

\* @return UUID значення primary key

\*/

public static <T> UUID getPrimaryKeyValue(T instance) {

Field pk = getPrimaryKey(instance.getClass());

PropertyDescriptor p;

try {

p = new PropertyDescriptor(pk.getName(), instance.getClass());

return ((UUID) p.getReadMethod().invoke(instance, null));

} catch (IntrospectionException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* встановлює значення UUID з поля Primary Key, вважається, що його тип UUID

\*

\* @param instance екземпляр класу з якого потрібно отримати інформацію

\* @param value нове значення для встановлення primary key

\* @param <T> тип класу

\*/

public static <T> void setPrimaryKeyValue(T instance, UUID value) {

Field pk = getPrimaryKey(instance.getClass());

PropertyDescriptor p;

try {

p = new PropertyDescriptor(pk.getName(), instance.getClass());

p.getWriteMethod().invoke(instance, value);

} catch (IntrospectionException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

}

}

/\*\*

\* повертає HashMap<String, Field> для поточного класу c

\* String - Stored.name() value

\*

\* @param c клас, з якого отримується інформація про поля позначені Stored

\* @return HashMap <stored name - це поле> для поточного класу

\*/

public static HashMap<String, Field> getStoredFields(Class c) {

HashMap<String, Field> res = new HashMap<String, Field>();

Field[] fields = c.getDeclaredFields();

for (Field field : fields) {

Stored p = field.getAnnotation(Stored.class);

if (p != null) {

res.put(p.name(), field);

}

}

return res;

}

/\*\*

\* повертає значення як стрічку

\*

\* @param instance екземпляр

\* @param f поле для перетворення

\* @param <T> тип екземпляру

\* @return поле як стрічку

\*/

public static <T> String getStringValue(T instance, Field f) {

try {

PropertyDescriptor p = new PropertyDescriptor(f.getName(), instance.getClass());

ValueConverter valueConverter = getValueConverter(f);

return valueConverter.toString(p.getReadMethod().invoke(instance, null));

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (IntrospectionException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* повертає конвертер даного поля

\*

\* @param field поле

\* @param <T> тип екземпляру

\* @return конвертер поля

\*/

public static <T extends ValueConverter> T getValueConverter(Field field) {

Stored s = (Stored) field.getAnnotation(Stored.class);

Class converterClass = s.converter();

try {

ValueConverter res = (ValueConverter) converterClass.newInstance();

return (T) res;

} catch (InstantiationException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* мапа Stored.name() - значення поля

\*

\* @param instance екземпляр для конвертації

\* @param <T> тип екземпляру

\* @return мапа, яка містить збережене ім’я поля - його значення після застосування ValueConverter

\*/

public static <T> Map<String, String> getConvertedStoredFields(T instance) {

HashMap<String, Field> mapStrField = getStoredFields(instance.getClass());

HashMap<String, String> mapStrStr = new HashMap<String, String>();

for (String key : mapStrField.keySet()) {

mapStrStr.put(key, getStringValue(instance, mapStrField.get(key)));

}

return mapStrStr;

}

/\*\*

\* перетворює екземпляр класу в карту значень

\*

\* @param instance екземпляр для конвертування

\* @param <T> тип екземпляру

\* @return карту значень <ключ - поле>

\*/

public static <T> Map entityToMap(T instance) {

PropertyDescriptor p;

Map res = new HashMap<String, Object>();

HashMap<String, Field> storedFields = getStoredFields(instance.getClass());

try {

for (String storedName : storedFields.keySet()) {

Field f = storedFields.get(storedName);

p = new PropertyDescriptor(f.getName(), instance.getClass());

Class returnType = p.getReadMethod().getReturnType();

Object value = p.getReadMethod().invoke(instance, null);

ValueConverter c = getValueConverter(f);

res.put(storedName, c.toString(value));

}

return res;

} catch (IntrospectionException | IllegalAccessException | IllegalArgumentException | InvocationTargetException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* перетворює карту значень у екземпляр класу Т

\*

\* @param instanceClass клас екземпляру

\* @param map карта значень

\* @param <T> тип екземпляру

\* @return екземпляр із карти

\*/

public static <T> T mapToEntity(Class instanceClass, Map map) {

PropertyDescriptor p;

HashMap<String, Field> storedFields = getStoredFields(instanceClass);

try {

T instance = (T) instanceClass.newInstance();

for (String storedName : storedFields.keySet()) {

Object strValue = map.get(storedName);

Field f = storedFields.get(storedName);

ValueConverter c = getValueConverter(f);

p = new PropertyDescriptor(f.getName(), instanceClass);

p.getWriteMethod().invoke(instance, c.toValue(c.toString(strValue)));

}

return instance;

} catch (IntrospectionException e) {

e.printStackTrace();

} catch (InstantiationException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (InvocationTargetException e) {

e.printStackTrace();

} catch (NullPointerException e) {

e.printStackTrace();

}

return null;

}

/\*\*

\* конструктор за замовчуванням

\*/

public DAOAnnotationUtils() {

}

}

package com.oyster.dao.annotation.utils;

/\*\*

\* інтерфейс для класів, які пертворюють певні поля типу Т в String і навпаки

\*/

public interface ValueConverter {

/\*\*

\* конвертує екземпляр типу Т в String

\*

\* @param value параметр для конвертування

\* @param <T> тип параметру

\* @return параметр конвертований в String

\*/

public <T> String toString(T value);

/\*\*

\* конвертує String до екземпляру типу Т

\*

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return об’єкт типу Т

\*/

public <T> T toValue(String str);

}

package com.oyster.dao.annotation.utils.converter;

import com.oyster.dao.annotation.utils.ValueConverter;

import java.math.BigInteger;

/\*\*

\* конвертує сутність (String <==> Long)

\*/

public class LongConverter implements ValueConverter {

/\*\*

\* перетворює Long в String

\* @param value параметр для конвертації

\* @param <T> тип параметру

\* @return параметр як стрічку

\*/

@Override

public <T> String toString(T value) {

if (value == null) return "null";

// костиль (for Spring JDBC)

if (value instanceof BigInteger) {

return value.toString();

}

return Long.toString((Long) value);

}

/\*\*

\* перетворює String в Long

\*

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return стрічку як параметр

\*/

@Override

public <T> T toValue(String str) {

if (str.equals("null")) return null;

return (T) new Long(Long.parseLong(str));

}

}

package com.oyster.dao.annotation.utils.converter;

import com.oyster.dao.annotation.utils.ValueConverter;

/\*\*

\* конвертує сутність (String <==> Integer)

\*/

public class IntConverter implements ValueConverter {

/\*\*

\* перетворює Integer в String

\*

\* @param value параметр для конвертації

\* @param <T> тип параметру

\* @return параметр як стрічку

\*/

@Override

public <T> String toString(T value) {

if (value == null) return "null";

// костиль (for Spring JDBC)

if (value instanceof Long) {

return value.toString();

}

return Integer.toString((Integer) value);

}

/\*\*

\* перетворює String в Integer

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return стрічку як параметр

\*/

@Override

public <T> T toValue(String str) {

if (str.equals("null")) return null;

return (T) new Integer(Integer.parseInt(str));

}

}

package com.oyster.dao.annotation.utils.converter;

import com.oyster.dao.annotation.utils.ValueConverter;

/\*\*

\* конвертує сутність (String <==> String)

\*/

public class StringConverter implements ValueConverter {

/\*\*

\* перетворює String в String

\*

\* @param value параметр для конвертації

\* @param <T> тип параметру

\* @return параметр як стрічку

\*/

@Override

public <T> String toString(T value) {

if (value == null) return null;

return "\"" + ((Object) value).toString() + "\"";

}

/\*\*

\* перетворює String в String

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return стрічку як параметр

\*/

@Override

public <T> T toValue(String str) {

if (str == null) return null;

if (str.equals("null")) return null;

if (str.equals("\"\"")) return (T) "";

if (str.equals("")) return (T) "";

str = str.substring(1, str.length() - 1);

return (T) str;

}

}

package com.oyster.dao.annotation.utils.converter;

import com.oyster.dao.annotation.utils.ValueConverter;

import java.util.UUID;

/\*\*

\* конвертує сутність (String <==> UUID)

\*/

public class UUIDConverter implements ValueConverter {

/\*\*

\* перетворює UUID в String

\*

\* @param value параметр для конвертації

\* @param <T> тип параметру

\* @return параметр як стрічку

\*/

@Override

public <T> String toString(T value) {

if (value == null) return null;

return "\"" + ((Object) value).toString() + "\"";

}

/\*\*

\* перетворює String в UUID

\*

\* @param str String для конвертування

\* @param <T> тип параметру

\* @return стрічку як параметр

\*/

@Override

public <T> T toValue(String str) {

if (str == null) return null;

if (str.equals("null")) return null;

str = str.substring(1, str.length() - 1);

return (T) UUID.fromString(str);

}

}

package com.oyster.ui;

import com.oyster.app.AppConst;

import com.oyster.app.model.\*;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import com.oyster.ui.dialogs.\*;

import javax.swing.\*;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.text.DateFormatter;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.text.ParseException;

import java.util.ArrayList;

import java.util.Date;

/\*\*

\* Клас відповідає табові, що реалізує вікно кправління профілями

\*

\* @author bamboo

\*/

public class MainForm extends JFrame {

private JPanel rootPanel;

private JTabbedPane mTabbedPaneMain;

private JComboBox mComboBoxProfileType;

private JComboBox mComboBoxProfileStudentTypeFaculty;

private JList mListPeople;

private JComboBox mComboBoxProfileStudentTypeGroup;

private JPanel mPanelProfile;

private JPanel mPanelLeftControl;

private JButton mButtonSave;

private JButton mButtonDelete;

private JList mListHistoryProfile;

private JTextField mTextFieldInfo1;

private JTextField mTextFieldInfo2;

private JTextField mTextFieldInfo3;

private JTextField mTextFieldInfo5;

private JTextField mTextFieldInfo6;

private JTextField mTextFieldInfo7;

private JButton mButtonNewUser;

private JComboBox mComboBoxAllHistory;

private JList mListAllHistory;

private JComboBox mComboBoxScheduleFaculty;

private JScrollPane mListGroups;

private JTable mTable1;

private JLabel mLabel5;

private JLabel mLable6;

private JLabel mLable7;

private JLabel mLable1;

private JLabel mLable2;

private JLabel mLable3;

private JLabel mLablePhoto;

private JLabel mLable8;

private JPasswordField mPasswordField1;

private JLabel mLable4;

private JTextField mTextFieldInfo4;

private JScrollPane mScrollPanePeople;

private JList mListTab2Group;

private IProfile currentPerson;

private DocumentListener mDocumentListener = new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

act();

}

@Override

public void removeUpdate(DocumentEvent e) {

act();

}

@Override

public void changedUpdate(DocumentEvent e) {

act();

}

private void act() {

mButtonSave.setEnabled(true);

}

};

private HistoryTab historyTab;

private ScheduleTab scheduleTab;

/\*\*

\* конструктор, створює елементи інтерфейсу

\*/

public MainForm() {

super((String) AppConst.APP\_CONFIG.getValue("progTitle"));

add(rootPanel);

int width = (Integer) AppConst.APP\_CONFIG.getValue("mainScreenWidth");

int height = (Integer) AppConst.APP\_CONFIG.getValue("mainScreenHeight");

setPreferredSize(new Dimension(width, height));

setMinimumSize(new Dimension(width, height));

hardCoreInit();

pack();

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

}

/\*\*

\* ініціалізує елементи інтерфейсу

\*/

private void hardCoreInit() {

addJMenu();

mButtonNewUser.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

newUserAction();

}

});

comboBoxChangeAction();

mComboBoxProfileType.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

comboBoxChangeAction();

}

});

mListPeople.addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

Object o = mListPeople.getSelectedValue();

if (o instanceof Admin) {

fillInfoFields((Admin) o);

} else if (o instanceof Teacher) {

fillInfoFields((Teacher) o);

} else if (o instanceof Student) {

fillInfoFields((Student) o);

}

}

});

mComboBoxProfileStudentTypeFaculty.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

facultyChangedAction();

}

});

mComboBoxProfileStudentTypeGroup.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

groupChangedAction();

}

});

mTextFieldInfo1.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo2.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo3.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo4.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo5.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo6.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo7.getDocument().addDocumentListener(mDocumentListener);

mPasswordField1.getDocument().addDocumentListener(mDocumentListener);

mButtonSave.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

saveButtonClick();

}

});

mButtonDelete.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

deleteButtonClick();

}

});

scheduleTab = new ScheduleTab(this, mComboBoxScheduleFaculty, mListTab2Group, mTable1);

historyTab = new HistoryTab(this, mComboBoxAllHistory, mListAllHistory);

}

/\*\*

\* метод спрацьовує при натисненні на кнопку "Видалити"

\*/

private void deleteButtonClick() {

if (currentPerson == null) {

return;

}

int dialogResult = JOptionPane.showConfirmDialog(null,

"Підтвердити видалення акаунту : ",

"Захист від дурака",

JOptionPane.OK\_CANCEL\_OPTION,

JOptionPane.WARNING\_MESSAGE

);

if (dialogResult == JOptionPane.CANCEL\_OPTION) {

return;

}

Context c = new Context();

c.put("profile", currentPerson);

try {

CommandExecutor.getInstance().execute("deleteIProfile", c, null);

} catch (Exception e) {

e.printStackTrace();

}

currentPerson = null;

comboBoxChangeAction();

}

/\*\*

\* метод спрацьовує при натисненні на кнопку "Зберегти"

\*/

private void saveButtonClick() {

if (currentPerson == null) {

return;

}

if (currentPerson instanceof Admin) {

Admin a = (Admin) currentPerson;

saveUpdate(a.getProfile());

saveUpdate(a.getWorkerInfo());

} else if (currentPerson instanceof Teacher) {

Teacher t = (Teacher) currentPerson;

saveUpdate(t.getProfile());

saveUpdate(t.getWorkerInfo());

} else if (currentPerson instanceof Student) {

Student s = (Student) currentPerson;

saveUpdate(s);

}

updateUI();

}

/\*\*

\* зберігає інформацію про профіль студента

\*

\* @param s екземпляр класу Student

\*/

private void saveUpdate(Student s) {

saveUpdate(s.getProfile());

String course = mTextFieldInfo5.getText().trim();

String bookNum = mTextFieldInfo7.getText().trim();

String facultyName = mTextFieldInfo4.getText().trim();

String groupName = mTextFieldInfo6.getText().trim();

s.setCourse(Integer.parseInt(course));

s.setBookNum(Integer.parseInt(bookNum));

s.getGroup().setName(groupName);

s.getGroup().getFaculty().setName(facultyName);

try {

AppConst.DAO.update(s);

AppConst.DAO.update(s.getGroup());

AppConst.DAO.update(s.getGroup().getFaculty());

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* зберігає інформацію про профіль

\*

\* @param p екземпляр класу Profile

\*/

private void saveUpdate(Profile p) {

String firstName = mTextFieldInfo1.getText().trim();

String secondName = mTextFieldInfo2.getText().trim();

String password = mPasswordField1.getText().trim();

String birthday = mTextFieldInfo3.getText().trim();

Long birthdayLong = 0L;

try {

Date d = (Date) new DateFormatter(AppConst.DATE\_FORMAT).stringToValue(birthday);

birthdayLong = d.getTime();

} catch (ParseException e) {

e.printStackTrace();

}

p.setFirstName(firstName);

p.setSecondName(secondName);

p.setBirthday(birthdayLong);

p.setPassword(password);

try {

AppConst.DAO.update(p);

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* зберігає інформацію про роботу

\*

\* @param wi екземпляр класу WorkerInfo

\*/

private void saveUpdate(WorkerInfo wi) {

String position = mTextFieldInfo4.getText().trim();

String salary = mTextFieldInfo5.getText().trim();

String dateHiredStr = mTextFieldInfo6.getText().trim();

Long dateHired = 0L;

try {

Date d = (Date) new DateFormatter(AppConst.DATE\_FORMAT).stringToValue(dateHiredStr);

dateHired = d.getTime();

} catch (ParseException e) {

e.printStackTrace();

}

wi.setPosition(position);

wi.setSalary(Integer.parseInt(salary));

wi.setDateHired(dateHired);

try {

AppConst.DAO.update(wi);

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* оновлює інтерфейс

\*/

private void updateUI() {

mButtonSave.setEnabled(false);

validate();

repaint();

}

/\*\*

\* метод спрацьовує при зміні факультету

\*/

private void facultyChangedAction() {

final Faculty f = (Faculty) mComboBoxProfileStudentTypeFaculty.getSelectedItem();

java.util.List<Group> groups = null;

try {

groups = AppConst.DAO.select(Group.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Group g = (Group) entity;

return g.getFacultyId().equals(f.getId());

}

});

for (Group g : groups) {

g.setFaculty(f);

}

} catch (Exception e) {

e.printStackTrace();

}

mComboBoxProfileStudentTypeGroup.setModel(new DefaultComboBoxModel(groups.toArray()));

if (groups.size() > 0) {

mComboBoxProfileStudentTypeGroup.setSelectedIndex(0);

}

updateUI();

}

/\*\*

\* метод спрацьовує при зміні групи

\*/

private void groupChangedAction() {

final Group group = (Group) mComboBoxProfileStudentTypeGroup.getSelectedItem();

java.util.List<Student> students = null;

try {

students = AppConst.DAO.select(Student.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Student s = (Student) entity;

return s.getGroupId().equals(group.getId());

}

});

for (Student student : students) {

student.setProfile((Profile) AppConst.DAO.read(Profile.class, student.getProfileId()));

student.setGroup(group);

}

} catch (Exception e) {

e.printStackTrace();

}

showFilteredStudents(students);

}

/\*\*

\* метод відображає список студентів, що пройшли фільтри, встановлені користувачем

\*

\* @param students список студентів

\*/

private void showFilteredStudents(java.util.List<Student> students) {

DefaultListModel<Student> model = new DefaultListModel<Student>();

for (Student s : students) {

model.addElement(s);

}

mListPeople.setModel(model);

if (students.size() > 0) {

mListPeople.setSelectedIndex(0);

}

updateUI();

}

/\*\*

\* метод спрацьовує при зміні типу користувача

\*/

private void comboBoxChangeAction() {

mButtonSave.setEnabled(false);

int selected = mComboBoxProfileType.getSelectedIndex();

if (selected < 3) {

mComboBoxProfileStudentTypeFaculty.setVisible(false);

mComboBoxProfileStudentTypeGroup.setVisible(false);

mLable7.setVisible(false);

mTextFieldInfo7.setVisible(false);

mLable4.setText("Посада");

mLabel5.setText("Зарплата");

mLable6.setText("Працює із");

} else {

mComboBoxProfileStudentTypeFaculty.setVisible(true);

mComboBoxProfileStudentTypeGroup.setVisible(true);

mLable7.setVisible(true);

mTextFieldInfo7.setVisible(true);

mLable4.setText("Факультет");

mLabel5.setText("Курс");

mLable6.setText("Група");

mLable7.setText("НЗК");

}

switch (mComboBoxProfileType.getSelectedIndex()) {

case 0:

mScrollPanePeople.setVisible(false);

mButtonDelete.setEnabled(false);

fillInfoFields(AppConst.getCurrentAdmin());

break;

case 1:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Admin> admins = null;

try {

admins = AppConst.DAO.select(Admin.class, "");

for (Admin a : admins) {

a.setProfile((Profile) AppConst.DAO.read(Profile.class, a.getProfileId()));

a.setWorkerInfo((WorkerInfo) AppConst.DAO.read(WorkerInfo.class, a.getWorkerInfoId()));

}

} catch (DAOException e) {

e.printStackTrace();

}

DefaultListModel<Admin> model = new DefaultListModel<>();

for (Admin a : admins) {

model.addElement(a);

}

mListPeople.setModel(model);

mListPeople.setSelectedIndex(0);

break;

case 2:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Teacher> teachers = null;

try {

teachers = AppConst.DAO.select(Teacher.class, "");

for (Teacher teacher : teachers) {

teacher.setProfile((Profile) AppConst.DAO.read(Profile.class, teacher.getProfileId()));

teacher.setWorkerInfo((WorkerInfo) AppConst.DAO.read(WorkerInfo.class, teacher.getWorkerInfoId()));

}

} catch (DAOException e) {

e.printStackTrace();

}

DefaultListModel<Teacher> teacherModel = new DefaultListModel<>();

for (Teacher teacher : teachers) {

teacherModel.addElement(teacher);

}

mListPeople.setModel(teacherModel);

mListPeople.setSelectedIndex(0);

break;

case 3:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Faculty> faculties = null;

try {

faculties = AppConst.DAO.select(Faculty.class, "");

} catch (DAOException e) {

e.printStackTrace();

}

mComboBoxProfileStudentTypeFaculty.setModel(new DefaultComboBoxModel(faculties.toArray()));

mComboBoxProfileStudentTypeFaculty.setSelectedIndex(0);

break;

}

updateUI();

}

/\*\*

\* метод заповнює інформацію про профіль

\*

\* @param profile екемпляр класу Profile

\*/

private void fillInfoFields(Profile profile) {

mTextFieldInfo1.setText(profile.getFirstName());

mTextFieldInfo2.setText(profile.getSecondName());

mTextFieldInfo3.setText(AppConst.DATE\_FORMAT.format(new Date(profile.getBirthday())));

mPasswordField1.setText(profile.getPassword());

loadHistory(profile);

}

/\*\*

\* метод завантажує історію вибраного користувача

\*

\* @param profile користувач

\*/

private void loadHistory(final Profile profile) {

final java.util.List<History> histories = new ArrayList<History>();

Context c = new Context();

c.put("list", histories);

c.put("sqlQuery", "select \* from HISTORY\_TBL where author\_id = \"" +

profile.getId() + "\";");

try {

CommandExecutor.getInstance().execute("loadHistory", c, new Runnable() {

@Override

public void run() {

for (History h : histories) {

h.setAuthor(profile);

}

DefaultListModel<History> historyModel = new DefaultListModel<>();

for (History h : histories) {

historyModel.addElement(h);

}

mListHistoryProfile.setModel(historyModel);

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* метод заповнює інформацію про роботу

\*

\* @param workerInfo екемпляр класу WorkerInfo

\*/

private void fillInfoFields(WorkerInfo workerInfo) {

mTextFieldInfo4.setText(workerInfo.getPosition());

mTextFieldInfo5.setText(String.valueOf(workerInfo.getSalary()));

mTextFieldInfo6.setText(AppConst.DATE\_FORMAT.format(new Date(workerInfo.getDateHired())));

}

/\*\*

\* метод заповнює інформацію про адміністратора

\*

\* @param admin екемпляр класу Admin

\*/

private void fillInfoFields(Admin admin) {

fillInfoFields(admin.getProfile());

fillInfoFields(admin.getWorkerInfo());

currentPerson = admin;

updateUI();

}

/\*\*

\* метод заповнює інформацію про викладача

\*

\* @param teacher екемпляр класу Teacher

\*/

private void fillInfoFields(Teacher teacher) {

fillInfoFields(teacher.getProfile());

fillInfoFields(teacher.getWorkerInfo());

currentPerson = teacher;

updateUI();

}

/\*\*

\* метод заповнює інформацію про студента

\*

\* @param student екемпляр класу Student

\*/

private void fillInfoFields(Student student) {

fillInfoFields(student.getProfile());

mTextFieldInfo4.setText(student.getGroup().getFaculty().getName());

mTextFieldInfo5.setText(String.valueOf(student.getCourse()));

mTextFieldInfo6.setText(student.getGroup().getName());

mTextFieldInfo7.setText(String.valueOf(student.getBookNum()));

currentPerson = student;

updateUI();

}

/\*\*

\* метод спрацьовує при натисненні на клавішу "Додати аккаунт"

\*/

private void newUserAction() {

Object[] possibilities = {"Студент", "Виладач", "Адміністратор", "Група", "Факультет", "Предмет"};

String s = (String) JOptionPane.showInputDialog(

MainForm.this,

"Виберіть категорію, яку хочете створити:\n",

"Новий акаунт",

JOptionPane.PLAIN\_MESSAGE,

null,

possibilities,

"Студент");

if ((s != null) && (s.length() > 0)) {

JDialog dialog = null;

switch (s) {

case "Студент":

dialog = new NewStudentCustomDialog(this);

break;

case "Виладач":

dialog = new NewTeacherCustomDialog(this);

break;

case "Адміністратор":

dialog = new NewAdminCustomDialog(this);

break;

case "Група":

dialog = new NewGroupCustomDialog(this);

break;

case "Факультет":

dialog = new NewFacultyCustomDialog(this);

break;

case "Предмет":

dialog = new NewSubjectCustomDialog(this);

break;

}

dialog.pack();

dialog.setVisible(true);

}

}

/\*\*

\* ініціалізує меню у програмі

\*/

private void addJMenu() {

JMenuBar menuBar;

JMenu menu, submenu;

JMenuItem menuItem;

JRadioButtonMenuItem rbMenuItem;

JCheckBoxMenuItem cbMenuItem;

menuBar = new JMenuBar();

menu = new JMenu("Файл");

menu.setMnemonic(KeyEvent.VK\_F);

menu.getAccessibleContext().setAccessibleDescription(

"The only menu in this program that has menu items");

menuBar.add(menu);

menuItem = new JMenuItem("Додати акаунт",

KeyEvent

.VK\_N

);

menuItem.setMnemonic(KeyEvent.VK\_N);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_N, ActionEvent.ALT\_MASK));

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

newUserAction();

}

});

menu.add(menuItem);

menu.addSeparator();

menuItem = new JMenuItem("Вихід",

new ImageIcon("images/middle.gif"));

menuItem.setMnemonic(KeyEvent.VK\_W);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_W, ActionEvent.ALT\_MASK));

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

MainForm.this.dispose();

}

});

menu.add(menuItem);

menu = new JMenu("Допомога");

menu.setMnemonic(KeyEvent.VK\_H);

menuBar.add(menu);

menuItem = new JMenuItem("Посібник користувача",

KeyEvent.VK\_K);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_K, ActionEvent.ALT\_MASK));

menu.add(menuItem);

menuItem = new JMenuItem("Про програму");

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_A, ActionEvent.ALT\_MASK));

menu.add(menuItem);

setJMenuBar(menuBar);

}

}

package com.oyster.ui;

import com.oyster.app.AppConst;

import com.oyster.app.model.\*;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import javax.swing.\*;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import javax.swing.table.AbstractTableModel;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.TableCellRenderer;

import javax.swing.table.TableColumn;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.UUID;

/\*\*

\* Клас відповідає табові, що відображає розклад груп

\*/

public class ScheduleTab {

private JFrame frame;

private JComboBox comboBoxFaculty;

private JComboBox comboBoxSubject;

private JComboBox comboBoxTeacher;

private JList groupList;

private JTable table;

private java.util.List<Subject> subjects = null;

private java.util.List<Teacher> teachers = null;

private boolean DEBUG = false;

private String[] daysOfWeek = new String[]{"Понеділок", "Вівторок", "Середа", "Четвер", "П’ятниця", "Субота"};

private ArrayList<Object[]> data;

private String[] columnNames = {"Пара",

"Предмет",

"Викладач",

"Аудиторія",

};

private ArrayList<Classes> classes;

private Teacher emptyTeacher;

private Subject emptySubject;

/\*\*

\* Конструктор, що приймає елементи інтерфейсу для подальших операцій із ними

\*

\* @param frame вікно програми

\* @param comboBoxFaculty вибірка факультетів

\* @param groupList список груп

\* @param table таблиця розкладу

\*/

public ScheduleTab(JFrame frame, JComboBox comboBoxFaculty, JList groupList, JTable table) {

this.frame = frame;

this.comboBoxFaculty = comboBoxFaculty;

this.groupList = groupList;

this.table = table;

hardcoreInit();

}

/\*\*

\* ініціалізує компоненти

\*/

private void hardcoreInit() {

emptyTeacher = new Teacher();

emptyTeacher.setProfile(new Profile());

emptySubject = new Subject();

emptySubject.setName("");

comboBoxFaculty.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

facultyChangedAction();

}

});

groupList.addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

reloadTable();

}

});

reloadAll();

}

/\*\*

\* перевіряє змінений елемент рзкладу та зберігає/видаляє його за потреби

\* @param c елемент розкладу

\*/

private void checkAndSave(Classes c) {

try {

AppConst.DAO.replace(c);

Group g = (Group) groupList.getSelectedValue();

AppConst.SESSION\_TIME++;

if (AppConst.SESSION\_TIME > 20) {

History h = new History(

UUID.randomUUID(),

AppConst.getCurrentAdmin().getProfileId(),

"Вніс зміни до розкладу групи " + g.getName()

);

AppConst.DAO.insert(h);

AppConst.SESSION\_TIME = 0;

}

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* реагує на зміну розкладу

\* @param e подія

\*/

private void tableValueChangedAction(TableModelEvent e) {

if (e.getType() == TableModelEvent.UPDATE

| e.getType() == TableModelEvent.INSERT

) {

Group g = (Group) groupList.getSelectedValue();

int row = e.getFirstRow();

int column = e.getColumn();

Absence absence = null;

Classes c = classes.get(row);

if (c == null) {

c = new Classes();

c.setId(UUID.randomUUID());

absence = new Absence();

absence.setId(UUID.randomUUID());

absence.setGroupId(g.getId());

absence.setClassId(c.getId());

c.setTime(row);

try {

AppConst.DAO.insert(absence);

} catch (DAOException e1) {

e1.printStackTrace();

}

classes.set(row, c);

}

switch (column) {

case 1:

Subject s = (Subject) table.getModel().getValueAt(row,

column);

c.setSubject(s);

c.setSubjectId(s.getId());

checkAndSave(c);

break;

case 2:

Teacher t = (Teacher) table.getModel().getValueAt(row,

column);

c.setTeacher(t);

c.setTeacherId(t.getId());

checkAndSave(c);

break;

case 3:

Integer i = null;

try {

i = Integer.parseInt(table.getModel().getValueAt(row, column).toString());

if (i < 0) {

throw new IllegalArgumentException("Аудиторія повинна бути >= 0");

}

} catch (NumberFormatException ne) {

// ok, empty string then

} catch (Exception ex) {

ex.printStackTrace();

Utils.showErrorDialog(frame, Utils.makePretty(ex.getMessage()));

if (absence != null) {

try {

AppConst.DAO.delete(absence);

} catch (DAOException e1) {

e1.printStackTrace();

}

}

return;

}

c.setAudience(i);

checkAndSave(c);

break;

}

System.out.println("Cell " + e.getFirstRow() + ", "

+ e.getColumn() + " changed. The new value: "

+ table.getModel().getValueAt(e.getFirstRow(),

e.getColumn()));

}

}

/\*\*

\* заповнює таблицю розкладу

\*/

private void fillTable() {

final java.util.List<Classes> classesList = new ArrayList<>();

Group g = (Group) groupList.getSelectedValue();

if (g == null) {

table.setVisible(false);

return;

} else {

table.setVisible(true);

}

Context c = new Context();

c.put("list", classesList);

c.put("sqlQuery", "select b.\* from ABSENCE\_TBL a " +

" left join CLASSES\_TBL b on (a.class\_id = b.classes\_id and " +

" a.group\_id = \"" + g.getId() + "\")");

try {

CommandExecutor.getInstance().execute("loadSchedule", c, new Runnable() {

@Override

public void run() {

MyTableModel model = (MyTableModel) table.getModel();

for (Classes c : classesList) {

if (c != null) {

classes.set(c.getTime(), c);

int row = c.getTime();

if (c.getAudience() > 0) {

model.setValueAt(c.getAudience(), row, 3);

}

model.setValueAt(getSubjectById(c.getSubjectId()), row, 1);

model.setValueAt(getTeacherById(c.getTeacherId()), row, 2);

}

}

updateUI();

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* повертає предмет за ключем

\* @param id ключ предмету

\* @return предмет

\*/

private Subject getSubjectById(UUID id) {

for (Subject s : subjects) {

if (s != null && s.getId().equals(id)) {

return s;

}

}

return emptySubject;

}

/\*\*

\* повертає викладача за ключем

\* @param id ключ викладача

\* @return викладача

\*/

private Teacher getTeacherById(UUID id) {

for (Teacher t : teachers) {

if (t != null && t.getId().equals(id)) {

return t;

}

}

return emptyTeacher;

}

/\*\*

\* регує на зміну факультету, завантажує групи ти розклад

\*/

private void facultyChangedAction() {

final Faculty f = (Faculty) comboBoxFaculty.getSelectedItem();

java.util.List<Group> groups = null;

try {

groups = AppConst.DAO.select(Group.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Group g = (Group) entity;

return g.getFacultyId().equals(f.getId());

}

});

for (Group g : groups) {

g.setFaculty(f);

}

} catch (Exception e) {

e.printStackTrace();

}

DefaultListModel<Group> model = new DefaultListModel<Group>();

for (Group g : groups) {

model.addElement(g);

}

groupList.setModel(model);

if (groups.size() > 0) {

groupList.setSelectedIndex(0);

}

updateUI();

}

/\*\*

\* оновлює інтерфейс

\*/

private void updateUI() {

frame.validate();

frame.repaint();

}

/\*\*

\* перезавантажує таблицю розкладу

\*/

private void reloadTable() {

classes = new ArrayList<>(30);

for (int i = 0; i < 30; i++) {

classes.add(null);

}

table.setModel(new MyTableModel());

table.setPreferredScrollableViewportSize(new Dimension(500, 900));

table.setFillsViewportHeight(true);

table.getModel().addTableModelListener(new TableModelListener() {

@Override

public void tableChanged(TableModelEvent e) {

tableValueChangedAction(e);

}

});

createScheduleTable();

initColumnSizes(table);

setUpSubjectColumn(table, table.getColumnModel().getColumn(1));

setUpTeacherColumn(table, table.getColumnModel().getColumn(2));

fillTable();

}

/\*\*

\* перезавантажує всі компоненти

\*/

private void reloadAll() {

reloadTable();

java.util.List<Faculty> faculties = null;

try {

faculties = AppConst.DAO.select(Faculty.class, "");

} catch (DAOException e) {

e.printStackTrace();

}

comboBoxFaculty.setModel(new DefaultComboBoxModel(faculties.toArray()));

comboBoxFaculty.setSelectedIndex(0);

}

/\*\*

\* створює таблицю розкладу

\*/

private void createScheduleTable() {

data = new ArrayList<>(30);

for (int i = 0; i < 6; i++) {

data.add(new Object[]{daysOfWeek[i], "", "", ""});

for (int j = 1; j < 6; j++) {

data.add(new Object[]{new Integer(j), "", "", (Integer) null});

}

}

}

/\*\*

\* ініціалізує розміри колонок таблиці

\* @param table таблиця розкладу

\*/

private void initColumnSizes(JTable table) {

MyTableModel model = (MyTableModel) table.getModel();

TableColumn column = null;

Component comp = null;

int headerWidth = 0;

int cellWidth = 0;

Object[] longValues = model.longValues;

TableCellRenderer headerRenderer =

table.getTableHeader().getDefaultRenderer();

for (int i = 0; i < 4; i++) {

column = table.getColumnModel().getColumn(i);

comp = headerRenderer.getTableCellRendererComponent(

null, column.getHeaderValue(),

false, false, 0, 0);

headerWidth = comp.getPreferredSize().width;

comp = table.getDefaultRenderer(model.getColumnClass(i)).

getTableCellRendererComponent(

table, longValues[i],

false, false, 0, i);

cellWidth = comp.getPreferredSize().width;

if (DEBUG) {

System.out.println("Initializing width of column "

+ i + ". "

+ "headerWidth = " + headerWidth

+ "; cellWidth = " + cellWidth);

}

column.setPreferredWidth(Math.max(headerWidth, cellWidth));

}

}

/\*\*

\* ініціалізує колонку предметів

\* @param table таблиця розкладу

\* @param subjectColumn колонка розкладу

\*/

public void setUpSubjectColumn(JTable table,

TableColumn subjectColumn) {

try {

subjects = AppConst.DAO.select(Subject.class, "");

} catch (DAOException e) {

e.printStackTrace();

}

comboBoxSubject = new JComboBox();

comboBoxSubject.addItem(emptySubject);

for (Subject s : subjects) {

comboBoxSubject.addItem(s);

System.out.println(": : " + s.toString());

}

subjectColumn.setCellEditor(new DefaultCellEditor(comboBoxSubject));

DefaultTableCellRenderer renderer =

new DefaultTableCellRenderer();

renderer.setToolTipText("Click for combo box");

subjectColumn.setCellRenderer(renderer);

}

/\*\*

\* ініціалізує колонку викладачів

\* @param table таблиця розкладу

\* @param teacherColumn колонка розкладу

\*/

public void setUpTeacherColumn(JTable table,

TableColumn teacherColumn) {

try {

teachers = AppConst.DAO.select(Teacher.class, "");

for (Teacher t : teachers) {

t.setProfile((Profile) AppConst.DAO.read(Profile.class, t.getProfileId()));

t.setWorkerInfo((WorkerInfo) AppConst.DAO.read(WorkerInfo.class, t.getWorkerInfoId()));

}

} catch (DAOException e) {

e.printStackTrace();

}

comboBoxTeacher = new JComboBox();

comboBoxTeacher.addItem(emptyTeacher);

for (Teacher s : teachers) {

comboBoxTeacher.addItem(s);

System.out.println(": : " + s.toString());

}

teacherColumn.setCellEditor(new DefaultCellEditor(comboBoxTeacher));

DefaultTableCellRenderer renderer =

new DefaultTableCellRenderer();

renderer.setToolTipText("Click for combo box");

teacherColumn.setCellRenderer(renderer);

}

/\*\*

\* модель для таблиці

\*/

class MyTableModel extends AbstractTableModel {

public final Object[] longValues = {"Sharon", "Campione",

"None of the above",

new Integer(20), Boolean.TRUE};

public int getColumnCount() {

return columnNames.length;

}

public int getRowCount() {

return data.size();

}

public String getColumnName(int col) {

return columnNames[col];

}

public Object getValueAt(int row, int col) {

return data.get(row)[col];

}

public Class getColumnClass(int c) {

return getValueAt(0, c).getClass();

}

public boolean isCellEditable(int row, int col) {

if (col < 1 || row % 6 == 0) {

return false;

}

return true;

}

public void setValueAt(Object value, int row, int col) {

if (DEBUG) {

System.out.println("Setting value at " + row + "," + col

+ " to " + value

+ " (an instance of "

+ value.getClass() + ")");

}

data.get(row)[col] = value;

fireTableCellUpdated(row, col);

if (DEBUG) {

System.out.println("New value of data2:");

printDebugData();

}

}

private void printDebugData() {

int numRows = getRowCount();

int numCols = getColumnCount();

for (int i = 0; i < numRows; i++) {

System.out.print(" row " + i + ":");

for (int j = 0; j < numCols; j++) {

System.out.print(" " + data.get(i)[j]);

}

System.out.println();

}

}

}

}

package com.oyster.ui;

import com.oyster.app.AppConst;

import com.oyster.app.model.History;

import com.oyster.app.model.Profile;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import com.oyster.dao.exception.DAOException;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.\*;

/\*\*

\* Клас відповідає табові, що відображає історію коритсувачів

\*/

public class HistoryTab {

private JFrame frame;

private JComboBox comboBox;

private JList historyList;

java.util.List<History> histories;

private Map<UUID, Profile> profiles;

/\*\*

\* Конструктор, що приймає елементи інтерфейсу для подальших операцій із ними

\*

\* @param frame вікно програми

\* @param comboBox елемент вибору типу історії

\* @param historyList список історії

\*/

public HistoryTab(JFrame frame, JComboBox comboBox, JList historyList) {

this.frame = frame;

this.comboBox = comboBox;

this.historyList = historyList;

profiles = new HashMap<>();

hardcoreInit();

}

/\*\*

\* ініціалізує компоненти

\*/

private void hardcoreInit() {

comboBox.setModel(new DefaultComboBoxModel(new Object[]{"Власна", "Уся"}));

comboBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

comboboxChangedAction();

}

});

try {

List<Profile> list = AppConst.DAO.select(Profile.class, "");

for (Profile p : list) {

profiles.put(p.getId(), p);

}

} catch (DAOException e) {

e.printStackTrace();

}

comboBox.setSelectedIndex(0);

}

/\*\*

\* реагує на зміну опції Власна/Уся історія

\*/

private void comboboxChangedAction() {

switch (comboBox.getSelectedIndex()) {

case 0:

loadHistory("select \* from HISTORY\_TBL where author\_id = \"" +

AppConst.getCurrentAdmin().getProfileId() + "\";");

break;

case 1:

loadHistory("");

break;

}

}

/\*\*

\* Завантажує історію користувача, виконуюючи sql-запит

\*

\* @param sql SQL-запит для виконання

\*/

private void loadHistory(String sql) {

Context c = new Context();

histories = new ArrayList<>();

c.put("list", histories);

c.put("sqlQuery", sql);

try {

CommandExecutor.getInstance().execute("loadHistory", c, new Runnable() {

@Override

public void run() {

System.out.println(histories.size());

for (History h : histories) {

h.setAuthor(profiles.get(h.getAuthorId()));

}

DefaultListModel<History> model = new DefaultListModel<>();

for (History h : histories) {

model.addElement(h);

}

historyList.setModel(model);

frame.validate();

frame.repaint();

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.ui;

import com.oyster.app.AppConst;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import javax.swing.\*;

import javax.swing.text.JTextComponent;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

/\*\*

\* Клас відповідає за вікно, що здійснює авторизацію користувача

\*/

public class LoginFrame extends JFrame implements ActionListener {

private JTextField userText;

private JPasswordField passwordText;

private String userName = null;

private String userPassword = null;

/\*\*

\* Конструктор форми

\*/

public LoginFrame() {

super((String) AppConst.APP\_CONFIG.getValue("progTitle"));

JPanel panel = new JPanel();

add(panel);

placeComponents(panel);

setLocationRelativeTo(null);

}

/\*\*

\* ініціалізує компоненти

\*

\* @param panel панель, на якуі помістити компоненти

\*/

private void placeComponents(JPanel panel) {

panel.setLayout(null);

JLabel userLabel = new JLabel("Username");

userLabel.setBounds(10, 10, 80, 25);

panel.add(userLabel);

userText = new JTextField(20);

userText.setBounds(100, 10, 170, 25);

panel.add(userText);

JLabel passwordLabel = new JLabel("Password");

passwordLabel.setBounds(10, 40, 80, 25);

panel.add(passwordLabel);

passwordText = new JPasswordField(20);

passwordText.setBounds(100, 40, 170, 25);

panel.add(passwordText);

JButton loginButton = new JButton("login");

loginButton.setBounds(160, 80, 100, 25);

loginButton.addActionListener(this);

panel.add(loginButton);

}

/\*\*

\* спрацьовує на подію e

\*

\* @param e подія

\*/

@Override

public void actionPerformed(ActionEvent e) {

StringBuilder errorMsg = new StringBuilder("Введіть ");

boolean errorOccurred = false;

JTextComponent focusComponent = userText;

userName = "admin-root";//userText.getText();

userPassword = "admin";//new String(passwordText.getPassword());

/\*

if (userName.trim().length() == 0) {

errorMsg.append(" логін");

errorOccurred = true;

}

if (userPassword.trim().length() == 0) {

if (errorOccurred) {

errorMsg.append(" та");

}

errorOccurred = true;

errorMsg.append(" пароль");

focusComponent = passwordText;

}\*/

errorMsg.append("!");

if (!errorOccurred) {

tryToLogIn();

} else {

focusComponent.selectAll();

JOptionPane.showMessageDialog(

LoginFrame.this,

errorMsg.toString(),

"Спробуйте ще раз",

JOptionPane.ERROR\_MESSAGE

);

userName = null;

userPassword = null;

focusComponent.requestFocusInWindow();

}

}

/\*\*

\* викликає команду здійснення авторизації

\*/

private void tryToLogIn() {

Context c = new Context();

c.put("username", userName);

c.put("password", userPassword);

try {

CommandExecutor.getInstance().execute("logIn", c, new Runnable() {

@Override

public void run() {

LoginFrame.this.dispose();

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

}

package com.oyster.ui;

import com.oyster.app.AppConst;

import com.oyster.app.model.\*;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import com.oyster.dao.DAOFilter;

import com.oyster.dao.exception.DAOException;

import com.oyster.ui.dialogs.\*;

import javax.swing.\*;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.text.DateFormatter;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.text.ParseException;

import java.util.ArrayList;

import java.util.Date;

/\*\*

\* Клас відповідає табові, що реалізує вікно кправління профілями

\*

\* @author bamboo

\*/

public class MainForm extends JFrame {

private JPanel rootPanel;

private JTabbedPane mTabbedPaneMain;

private JComboBox mComboBoxProfileType;

private JComboBox mComboBoxProfileStudentTypeFaculty;

private JList mListPeople;

private JComboBox mComboBoxProfileStudentTypeGroup;

private JPanel mPanelProfile;

private JPanel mPanelLeftControl;

private JButton mButtonSave;

private JButton mButtonDelete;

private JList mListHistoryProfile;

private JTextField mTextFieldInfo1;

private JTextField mTextFieldInfo2;

private JTextField mTextFieldInfo3;

private JTextField mTextFieldInfo5;

private JTextField mTextFieldInfo6;

private JTextField mTextFieldInfo7;

private JButton mButtonNewUser;

private JComboBox mComboBoxAllHistory;

private JList mListAllHistory;

private JComboBox mComboBoxScheduleFaculty;

private JScrollPane mListGroups;

private JTable mTable1;

private JLabel mLabel5;

private JLabel mLable6;

private JLabel mLable7;

private JLabel mLable1;

private JLabel mLable2;

private JLabel mLable3;

private JLabel mLablePhoto;

private JLabel mLable8;

private JPasswordField mPasswordField1;

private JLabel mLable4;

private JTextField mTextFieldInfo4;

private JScrollPane mScrollPanePeople;

private JList mListTab2Group;

private IProfile currentPerson;

private DocumentListener mDocumentListener = new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

act();

}

@Override

public void removeUpdate(DocumentEvent e) {

act();

}

@Override

public void changedUpdate(DocumentEvent e) {

act();

}

private void act() {

mButtonSave.setEnabled(true);

}

};

private HistoryTab historyTab;

private ScheduleTab scheduleTab;

/\*\*

\* конструктор, створює елементи інтерфейсу

\*/

public MainForm() {

super((String) AppConst.APP\_CONFIG.getValue("progTitle"));

add(rootPanel);

int width = (Integer) AppConst.APP\_CONFIG.getValue("mainScreenWidth");

int height = (Integer) AppConst.APP\_CONFIG.getValue("mainScreenHeight");

setPreferredSize(new Dimension(width, height));

setMinimumSize(new Dimension(width, height));

hardCoreInit();

pack();

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setVisible(true);

}

/\*\*

\* ініціалізує елементи інтерфейсу

\*/

private void hardCoreInit() {

addJMenu();

mButtonNewUser.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

newUserAction();

}

});

comboBoxChangeAction();

mComboBoxProfileType.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

comboBoxChangeAction();

}

});

mListPeople.addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

Object o = mListPeople.getSelectedValue();

if (o instanceof Admin) {

fillInfoFields((Admin) o);

} else if (o instanceof Teacher) {

fillInfoFields((Teacher) o);

} else if (o instanceof Student) {

fillInfoFields((Student) o);

}

}

});

mComboBoxProfileStudentTypeFaculty.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

facultyChangedAction();

}

});

mComboBoxProfileStudentTypeGroup.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

groupChangedAction();

}

});

mTextFieldInfo1.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo2.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo3.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo4.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo5.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo6.getDocument().addDocumentListener(mDocumentListener);

mTextFieldInfo7.getDocument().addDocumentListener(mDocumentListener);

mPasswordField1.getDocument().addDocumentListener(mDocumentListener);

mButtonSave.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

saveButtonClick();

}

});

mButtonDelete.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

deleteButtonClick();

}

});

scheduleTab = new ScheduleTab(this, mComboBoxScheduleFaculty, mListTab2Group, mTable1);

historyTab = new HistoryTab(this, mComboBoxAllHistory, mListAllHistory);

}

/\*\*

\* метод спрацьовує при натисненні на кнопку "Видалити"

\*/

private void deleteButtonClick() {

if (currentPerson == null) {

return;

}

int dialogResult = JOptionPane.showConfirmDialog(null,

"Підтвердити видалення акаунту : ",

"Захист від дурака",

JOptionPane.OK\_CANCEL\_OPTION,

JOptionPane.WARNING\_MESSAGE

);

if (dialogResult == JOptionPane.CANCEL\_OPTION) {

return;

}

Context c = new Context();

c.put("profile", currentPerson);

try {

CommandExecutor.getInstance().execute("deleteIProfile", c, null);

} catch (Exception e) {

e.printStackTrace();

}

currentPerson = null;

comboBoxChangeAction();

}

/\*\*

\* метод спрацьовує при натисненні на кнопку "Зберегти"

\*/

private void saveButtonClick() {

if (currentPerson == null) {

return;

}

if (currentPerson instanceof Admin) {

Admin a = (Admin) currentPerson;

saveUpdate(a.getProfile());

saveUpdate(a.getWorkerInfo());

} else if (currentPerson instanceof Teacher) {

Teacher t = (Teacher) currentPerson;

saveUpdate(t.getProfile());

saveUpdate(t.getWorkerInfo());

} else if (currentPerson instanceof Student) {

Student s = (Student) currentPerson;

saveUpdate(s);

}

updateUI();

}

/\*\*

\* зберігає інформацію про профіль студента

\*

\* @param s екземпляр класу Student

\*/

private void saveUpdate(Student s) {

saveUpdate(s.getProfile());

String course = mTextFieldInfo5.getText().trim();

String bookNum = mTextFieldInfo7.getText().trim();

String facultyName = mTextFieldInfo4.getText().trim();

String groupName = mTextFieldInfo6.getText().trim();

s.setCourse(Integer.parseInt(course));

s.setBookNum(Integer.parseInt(bookNum));

s.getGroup().setName(groupName);

s.getGroup().getFaculty().setName(facultyName);

try {

AppConst.DAO.update(s);

AppConst.DAO.update(s.getGroup());

AppConst.DAO.update(s.getGroup().getFaculty());

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* зберігає інформацію про профіль

\*

\* @param p екземпляр класу Profile

\*/

private void saveUpdate(Profile p) {

String firstName = mTextFieldInfo1.getText().trim();

String secondName = mTextFieldInfo2.getText().trim();

String password = mPasswordField1.getText().trim();

String birthday = mTextFieldInfo3.getText().trim();

Long birthdayLong = 0L;

try {

Date d = (Date) new DateFormatter(AppConst.DATE\_FORMAT).stringToValue(birthday);

birthdayLong = d.getTime();

} catch (ParseException e) {

e.printStackTrace();

}

p.setFirstName(firstName);

p.setSecondName(secondName);

p.setBirthday(birthdayLong);

p.setPassword(password);

try {

AppConst.DAO.update(p);

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* зберігає інформацію про роботу

\*

\* @param wi екземпляр класу WorkerInfo

\*/

private void saveUpdate(WorkerInfo wi) {

String position = mTextFieldInfo4.getText().trim();

String salary = mTextFieldInfo5.getText().trim();

String dateHiredStr = mTextFieldInfo6.getText().trim();

Long dateHired = 0L;

try {

Date d = (Date) new DateFormatter(AppConst.DATE\_FORMAT).stringToValue(dateHiredStr);

dateHired = d.getTime();

} catch (ParseException e) {

e.printStackTrace();

}

wi.setPosition(position);

wi.setSalary(Integer.parseInt(salary));

wi.setDateHired(dateHired);

try {

AppConst.DAO.update(wi);

} catch (DAOException e) {

e.printStackTrace();

}

}

/\*\*

\* оновлює інтерфейс

\*/

private void updateUI() {

mButtonSave.setEnabled(false);

validate();

repaint();

}

/\*\*

\* метод спрацьовує при зміні факультету

\*/

private void facultyChangedAction() {

final Faculty f = (Faculty) mComboBoxProfileStudentTypeFaculty.getSelectedItem();

java.util.List<Group> groups = null;

try {

groups = AppConst.DAO.select(Group.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Group g = (Group) entity;

return g.getFacultyId().equals(f.getId());

}

});

for (Group g : groups) {

g.setFaculty(f);

}

} catch (Exception e) {

e.printStackTrace();

}

mComboBoxProfileStudentTypeGroup.setModel(new DefaultComboBoxModel(groups.toArray()));

if (groups.size() > 0) {

mComboBoxProfileStudentTypeGroup.setSelectedIndex(0);

}

updateUI();

}

/\*\*

\* метод спрацьовує при зміні групи

\*/

private void groupChangedAction() {

final Group group = (Group) mComboBoxProfileStudentTypeGroup.getSelectedItem();

java.util.List<Student> students = null;

try {

students = AppConst.DAO.select(Student.class, new DAOFilter() {

@Override

public <T> boolean accept(T entity) {

Student s = (Student) entity;

return s.getGroupId().equals(group.getId());

}

});

for (Student student : students) {

student.setProfile((Profile) AppConst.DAO.read(Profile.class, student.getProfileId()));

student.setGroup(group);

}

} catch (Exception e) {

e.printStackTrace();

}

showFilteredStudents(students);

}

/\*\*

\* метод відображає список студентів, що пройшли фільтри, встановлені користувачем

\*

\* @param students список студентів

\*/

private void showFilteredStudents(java.util.List<Student> students) {

DefaultListModel<Student> model = new DefaultListModel<Student>();

for (Student s : students) {

model.addElement(s);

}

mListPeople.setModel(model);

if (students.size() > 0) {

mListPeople.setSelectedIndex(0);

}

updateUI();

}

/\*\*

\* метод спрацьовує при зміні типу користувача

\*/

private void comboBoxChangeAction() {

mButtonSave.setEnabled(false);

int selected = mComboBoxProfileType.getSelectedIndex();

if (selected < 3) {

mComboBoxProfileStudentTypeFaculty.setVisible(false);

mComboBoxProfileStudentTypeGroup.setVisible(false);

mLable7.setVisible(false);

mTextFieldInfo7.setVisible(false);

mLable4.setText("Посада");

mLabel5.setText("Зарплата");

mLable6.setText("Працює із");

} else {

mComboBoxProfileStudentTypeFaculty.setVisible(true);

mComboBoxProfileStudentTypeGroup.setVisible(true);

mLable7.setVisible(true);

mTextFieldInfo7.setVisible(true);

mLable4.setText("Факультет");

mLabel5.setText("Курс");

mLable6.setText("Група");

mLable7.setText("НЗК");

}

switch (mComboBoxProfileType.getSelectedIndex()) {

case 0:

mScrollPanePeople.setVisible(false);

mButtonDelete.setEnabled(false);

fillInfoFields(AppConst.getCurrentAdmin());

break;

case 1:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Admin> admins = null;

try {

admins = AppConst.DAO.select(Admin.class, "");

for (Admin a : admins) {

a.setProfile((Profile) AppConst.DAO.read(Profile.class, a.getProfileId()));

a.setWorkerInfo((WorkerInfo) AppConst.DAO.read(WorkerInfo.class, a.getWorkerInfoId()));

}

} catch (DAOException e) {

e.printStackTrace();

}

DefaultListModel<Admin> model = new DefaultListModel<>();

for (Admin a : admins) {

model.addElement(a);

}

mListPeople.setModel(model);

mListPeople.setSelectedIndex(0);

break;

case 2:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Teacher> teachers = null;

try {

teachers = AppConst.DAO.select(Teacher.class, "");

for (Teacher teacher : teachers) {

teacher.setProfile((Profile) AppConst.DAO.read(Profile.class, teacher.getProfileId()));

teacher.setWorkerInfo((WorkerInfo) AppConst.DAO.read(WorkerInfo.class, teacher.getWorkerInfoId()));

}

} catch (DAOException e) {

e.printStackTrace();

}

DefaultListModel<Teacher> teacherModel = new DefaultListModel<>();

for (Teacher teacher : teachers) {

teacherModel.addElement(teacher);

}

mListPeople.setModel(teacherModel);

mListPeople.setSelectedIndex(0);

break;

case 3:

mScrollPanePeople.setVisible(true);

mButtonDelete.setEnabled(true);

java.util.List<Faculty> faculties = null;

try {

faculties = AppConst.DAO.select(Faculty.class, "");

} catch (DAOException e) {

e.printStackTrace();

}

mComboBoxProfileStudentTypeFaculty.setModel(new DefaultComboBoxModel(faculties.toArray()));

mComboBoxProfileStudentTypeFaculty.setSelectedIndex(0);

break;

}

updateUI();

}

/\*\*

\* метод заповнює інформацію про профіль

\*

\* @param profile екемпляр класу Profile

\*/

private void fillInfoFields(Profile profile) {

mTextFieldInfo1.setText(profile.getFirstName());

mTextFieldInfo2.setText(profile.getSecondName());

mTextFieldInfo3.setText(AppConst.DATE\_FORMAT.format(new Date(profile.getBirthday())));

mPasswordField1.setText(profile.getPassword());

loadHistory(profile);

}

/\*\*

\* метод завантажує історію вибраного користувача

\*

\* @param profile користувач

\*/

private void loadHistory(final Profile profile) {

final java.util.List<History> histories = new ArrayList<History>();

Context c = new Context();

c.put("list", histories);

c.put("sqlQuery", "select \* from HISTORY\_TBL where author\_id = \"" +

profile.getId() + "\";");

try {

CommandExecutor.getInstance().execute("loadHistory", c, new Runnable() {

@Override

public void run() {

for (History h : histories) {

h.setAuthor(profile);

}

DefaultListModel<History> historyModel = new DefaultListModel<>();

for (History h : histories) {

historyModel.addElement(h);

}

mListHistoryProfile.setModel(historyModel);

}

});

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* метод заповнює інформацію про роботу

\*

\* @param workerInfo екемпляр класу WorkerInfo

\*/

private void fillInfoFields(WorkerInfo workerInfo) {

mTextFieldInfo4.setText(workerInfo.getPosition());

mTextFieldInfo5.setText(String.valueOf(workerInfo.getSalary()));

mTextFieldInfo6.setText(AppConst.DATE\_FORMAT.format(new Date(workerInfo.getDateHired())));

}

/\*\*

\* метод заповнює інформацію про адміністратора

\*

\* @param admin екемпляр класу Admin

\*/

private void fillInfoFields(Admin admin) {

fillInfoFields(admin.getProfile());

fillInfoFields(admin.getWorkerInfo());

currentPerson = admin;

updateUI();

}

/\*\*

\* метод заповнює інформацію про викладача

\*

\* @param teacher екемпляр класу Teacher

\*/

private void fillInfoFields(Teacher teacher) {

fillInfoFields(teacher.getProfile());

fillInfoFields(teacher.getWorkerInfo());

currentPerson = teacher;

updateUI();

}

/\*\*

\* метод заповнює інформацію про студента

\*

\* @param student екемпляр класу Student

\*/

private void fillInfoFields(Student student) {

fillInfoFields(student.getProfile());

mTextFieldInfo4.setText(student.getGroup().getFaculty().getName());

mTextFieldInfo5.setText(String.valueOf(student.getCourse()));

mTextFieldInfo6.setText(student.getGroup().getName());

mTextFieldInfo7.setText(String.valueOf(student.getBookNum()));

currentPerson = student;

updateUI();

}

/\*\*

\* метод спрацьовує при натисненні на клавішу "Додати аккаунт"

\*/

private void newUserAction() {

Object[] possibilities = {"Студент", "Виладач", "Адміністратор", "Група", "Факультет", "Предмет"};

String s = (String) JOptionPane.showInputDialog(

MainForm.this,

"Виберіть категорію, яку хочете створити:\n",

"Новий акаунт",

JOptionPane.PLAIN\_MESSAGE,

null,

possibilities,

"Студент");

if ((s != null) && (s.length() > 0)) {

JDialog dialog = null;

switch (s) {

case "Студент":

dialog = new NewStudentCustomDialog(this);

break;

case "Виладач":

dialog = new NewTeacherCustomDialog(this);

break;

case "Адміністратор":

dialog = new NewAdminCustomDialog(this);

break;

case "Група":

dialog = new NewGroupCustomDialog(this);

break;

case "Факультет":

dialog = new NewFacultyCustomDialog(this);

break;

case "Предмет":

dialog = new NewSubjectCustomDialog(this);

break;

}

dialog.pack();

dialog.setVisible(true);

}

}

/\*\*

\* ініціалізує меню у програмі

\*/

private void addJMenu() {

JMenuBar menuBar;

JMenu menu, submenu;

JMenuItem menuItem;

JRadioButtonMenuItem rbMenuItem;

JCheckBoxMenuItem cbMenuItem;

menuBar = new JMenuBar();

menu = new JMenu("Файл");

menu.setMnemonic(KeyEvent.VK\_F);

menu.getAccessibleContext().setAccessibleDescription(

"The only menu in this program that has menu items");

menuBar.add(menu);

menuItem = new JMenuItem("Додати акаунт",

KeyEvent

.VK\_N

);

menuItem.setMnemonic(KeyEvent.VK\_N);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_N, ActionEvent.ALT\_MASK));

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

newUserAction();

}

});

menu.add(menuItem);

menu.addSeparator();

menuItem = new JMenuItem("Вихід",

new ImageIcon("images/middle.gif"));

menuItem.setMnemonic(KeyEvent.VK\_W);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_W, ActionEvent.ALT\_MASK));

menuItem.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

MainForm.this.dispose();

}

});

menu.add(menuItem);

menu = new JMenu("Допомога");

menu.setMnemonic(KeyEvent.VK\_H);

menuBar.add(menu);

menuItem = new JMenuItem("Посібник користувача",

KeyEvent.VK\_K);

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_K, ActionEvent.ALT\_MASK));

menu.add(menuItem);

menuItem = new JMenuItem("Про програму");

menuItem.setAccelerator(KeyStroke.getKeyStroke(

KeyEvent.VK\_A, ActionEvent.ALT\_MASK));

menu.add(menuItem);

setJMenuBar(menuBar);

}

}

package com.oyster.ui.dialogs;

import com.oyster.app.AppConst;

import com.oyster.core.controller.CommandExecutor;

import com.oyster.core.controller.command.Context;

import javax.swing.\*;

import javax.swing.text.JTextComponent;

import javax.swing.text.NumberFormatter;

import java.awt.\*;

import java.awt.event.ComponentAdapter;

import java.awt.event.ComponentEvent;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

import java.text.NumberFormat;

import java.text.ParseException;

import java.util.Date;

/\*\*

\* Клас відповідає за діалог, який збирає усі дані необхідні для створення нового адміністратора

\*/

public class NewAdminCustomDialog extends JDialog

implements PropertyChangeListener {

private String userName = null;

private String userSurmane = null;

private String userBirthDateStr = null;

private Date userBirthDate = null;

private String adminPosition;

private String adminSalary;

private Integer adminSalaryInt;

private String adminDateHiredStr;

private Date adminDateHired;

private String userPassword;

private JTextField textField1;

private JTextField textField2;

private JTextField textField3;

private JTextField textField4;

private JTextField textField5;

private JTextField textField6;

private JPasswordField mJPasswordField;

private JOptionPane optionPane;

private String btnString1 = "Створити";

private String btnString2 = "Відмінити";

/\*\*

\* Створює нове ділогове вікно

\*

\* @param aFrame вкно, що викликало діалог

\*/

public NewAdminCustomDialog(Frame aFrame) {

super(aFrame, true);

super.setLocationRelativeTo(null);

setTitle("Створити профіль адміністратора");

textField1 = new JTextField(10);

textField2 = new JTextField(15);

textField3 = new JFormattedTextField(AppConst.DATE\_FORMAT);

textField4 = new JTextField(15);

NumberFormat format = NumberFormat.getInstance();

NumberFormatter formatter = new NumberFormatter(format);

formatter.setValueClass(Integer.class);

formatter.setMinimum(1);

formatter.setMaximum(Integer.MAX\_VALUE);

formatter.setCommitsOnValidEdit(true);

textField5 = new JFormattedTextField(formatter);

textField6 = new JFormattedTextField(AppConst.DATE\_FORMAT);

mJPasswordField = new JPasswordField(15);

String msgString1 = "Ім’я адміністратора: ";

String msgString2 = "Прізвище адміністратора : ";

String msgString3 = "День народження : ";

String msgString4 = "Посада : ";

String msgString5 = "Зарплата / рік : ";

String msgString6 = "Дата прийняття на роботу : ";

String msgString7 = "Пароль : ";

Object[] array = {msgString1, textField1,

msgString2, textField2,

msgString3, textField3,

msgString4, textField4,

msgString5, textField5,

msgString6, textField6,

msgString7, mJPasswordField};

Object[] options = {btnString1, btnString2};

optionPane = new JOptionPane(array,

JOptionPane.INFORMATION\_MESSAGE,

JOptionPane.YES\_NO\_OPTION,

null,

options,

options[0]);

setContentPane(optionPane);

setDefaultCloseOperation(DO\_NOTHING\_ON\_CLOSE);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent we) {

optionPane.setValue(new Integer(

JOptionPane.CLOSED\_OPTION));

}

});

addComponentListener(new ComponentAdapter() {

public void componentShown(ComponentEvent ce) {

textField1.requestFocusInWindow();

}

});

optionPane.addPropertyChangeListener(this);

setPreferredSize(new Dimension(350, 450));

setMinimumSize(new Dimension(350, 450));

}

/\*\*

\* Метод спрацьовує на зміну властивостей у JOptionsPane

\*

\* @param e дані про подію

\*/

public void propertyChange(PropertyChangeEvent e) {

String prop = e.getPropertyName();

if (isVisible()

&& (e.getSource() == optionPane)

&& (JOptionPane.VALUE\_PROPERTY.equals(prop) ||

JOptionPane.INPUT\_VALUE\_PROPERTY.equals(prop))) {

Object value = optionPane.getValue();

if (value == JOptionPane.UNINITIALIZED\_VALUE) {

return;

}

optionPane.setValue(

JOptionPane.UNINITIALIZED\_VALUE);

if (btnString1.equals(value)) {

userName = textField1.getText().trim();

userSurmane = textField2.getText().trim();

userBirthDateStr = textField3.getText().trim();

adminPosition = textField4.getText().trim();

adminSalary = textField5.getText().trim();

adminDateHiredStr = textField6.getText().trim();

userPassword = new String(mJPasswordField.getPassword()).trim();

boolean errorOccured = false;

JTextComponent focusComponent = textField1;

StringBuilder errorMsg = new StringBuilder("Введіть ");

if (userName.length() == 0) {

errorMsg.append(" ім’я адміністратора");

errorOccured = true;

}

if (userSurmane.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" прізвище адміністратора");

focusComponent = textField2;

}

if (userBirthDateStr.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" дату народження у фрматі день/місяць/рік");

focusComponent = textField3;

}

if (adminPosition.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" посаду");

focusComponent = textField4;

}

if (adminSalary.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" зарплату");

focusComponent = textField5;

}

if (adminDateHiredStr.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" дату прийняття у фрматі день/місяць/рік");

focusComponent = textField6;

}

if (userPassword.length() == 0) {

if (errorOccured) {

errorMsg.append(", та");

}

errorOccured = true;

errorMsg.append(" пароль");

focusComponent = mJPasswordField;

}

errorMsg.append("!");

if (!errorOccured) {

try {

adminSalaryInt = (Integer) ((JFormattedTextField) textField5)

.getFormatter().stringToValue(adminSalary);

userBirthDate = (Date) ((JFormattedTextField) textField3)

.getFormatter().stringToValue(userBirthDateStr);

adminDateHired = (Date) ((JFormattedTextField) textField6)

.getFormatter().stringToValue(adminDateHiredStr);

} catch (ParseException e1) {

e1.printStackTrace();

}

Context c = new Context();

c.put("name", userName);

c.put("surname", userSurmane);

c.put("birthday", userBirthDate.getTime());

c.put("position", adminPosition);

c.put("salary", adminSalaryInt);

c.put("dateHired", adminDateHired.getTime());

c.put("password", userPassword);

try {

CommandExecutor.getInstance().execute("registerAdmin", c, null);

} catch (Exception ex) {

ex.printStackTrace();

}

clearAndHide();

} else {

focusComponent.selectAll();

JOptionPane.showMessageDialog(

NewAdminCustomDialog.this,

errorMsg.toString(),

"Спробуйте ще раз",

JOptionPane.ERROR\_MESSAGE

);

userName = null;

userSurmane = null;

userBirthDateStr = null;

adminPosition = null;

adminSalary = null;

adminDateHiredStr = null;

userPassword = null;

focusComponent.requestFocusInWindow();

}

} else {

userName = null;

userSurmane = null;

userBirthDateStr = null;

adminPosition = null;

adminSalary = null;

adminDateHiredStr = null;

userPassword = null;

clearAndHide();

}

}

}

/\*\*

\* Метод очищує всі поля діалогу

\*/

public void clearAndHide() {

textField1.setText(null);

textField2.setText(null);

textField3.setText(null);

textField4.setText(null);

textField5.setText(null);

textField6.setText(null);

mJPasswordField.setText(null);

setVisible(false);

dispose();

}

}