Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

КУРСОВА РОБОТА

з дисципліни: «Інженерія програмного забезпечення»

на тему: програмний додаток «Система інтернет – магазину. Функціональність користувача»

Студента 2-го курсу групи ІО-21

Напряму підготовки: 6.050102 Комп’ютерна інженерія

Спеціальності: 7.8.05010201 Комп’ютерні системи та мережі

Кахерського Олега Ігоровича

Керівник к.т.н., доцент Абу Усбах О. Н.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

Київ - 2014 рік

ЗМІСТ

ЗМІСТ …………………………………………………………………………………. 2

ВСТУП …………………………………………………………………………………3

1. ОГЛЯД МVC ..……………………………………………………………………. .. 4

1.1 Загальна характеристика ...………………………………………………… 4

1.2 Історія MVC .……..………………………………………………………… 4

1.3 Концепція …..…….………………………………………………………… 4

1.4 Реалізація .…..…….………………………………………………………… 5

1.5 Переваги MVC ……………………………………………………………... 6

1.6 Варіанти MVC ………………………………………………………………7

1.6.1 Model – View – Presenter (MVP) …………………………………7

1.6.2 Мodel–View–ViewModel (MVVM) ……………………………….9

1.7 Реалізація MVC у програмному додатку …………………………………10

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ ……………………………….11

2.1 Прецеденти ………………………………………………………………..11

2.2 Сценарії прецедентів ……………………………………………………..13

2.3 Діаграми граничних класів ………………………………………………16

2.4 Проектування графічного інтерфейсу користувача ………………….....20

2.5 Проектування структури програмного додатку ………………………..25

3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ……………………………………… 26

3.1 Структура програмного додатку ………………………………………..26

3.2 Специфікації класів ………………………………………………………30

3.3 Інструкції для користувача ………………………………………………58

4. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ…………………………………...60

ВИСНОВОК ………………………………………………………………………….68

5. ДОДАТКИ …………………………………………………………………………69

3.1 Додаток А. Таблиця відповідності елементів дизайну …………………69

3.2 Додаток Б. Лістинг коду програмного додатку………………...………..72

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ …………………………………………...120

ВСТУП

Метою курсової роботи є закріплення знань з проектування, моделювання, тестування програмного забезпечення та отримання практичних навичок у проектуванні та розробці програмного додатку з графічним інтерфейсом користувача (за допомогою графічних бібліотек AWT та SWING), взаємодією з реляційними базами даних (з використанням бібліотеки JDBC). Для цього розроблятиметься програмний додаток для взаємодії користувача із системою інтернет – магазину, який повинен мати зручний інтерфейс, налагоджений зв’язок з реляційною базою даних.

1. ОГЛЯД MVC
   1. Загальна характеристика

MVC - схема використання декількох шаблонів проектування з метою відокремлення даних моделі, представлення та взаємодію з користувачем в різні компоненти таким чином, щоб зміна в кожному з них мала мінімальний вплив на інші.

* 1. Історія MVС

Термін Модель – Представлення – Контролер використовується з кінця 1970-x та виник у проекті Smalltalk в Хerox PARC, де був вигаданий як спосіб організації деяких додатків з інтерфейсом користувача. [1]

* 1. Концепція

MVC розділяє програмний додаток на такі три частини:

* Модель, яка містить дані, з якими взаємодіє користувач.

Виділяють два типи Моделі:

* 1. Пасивна Модель - не має жодних засобів для впливу на вигляд чи контролер. Використовується ними в якості джерела даних для відображення. Будь – яка зміна моделі відстежується контролером, який, у свою чергу, може відповідати за оновлення вигляду. Така модель дуже часто застосовується в структурному програмуванні, так як у цьому випадку модель є просто структурою даних без методів, що їх обробляють. [6]
  2. Активна Модель – модель сповіщає вигляд про свої зміни, а вигляди, що зацікавлені в оновленнях у моделі, підписуються на такі повідомлення. Такий підхід дозволяє зберегти незалежність моделі як від контролера, так і від вигляду.[6]
* Представлення, яке використовується для того, щоб обробляти деякі частини моделі в якості інтерфейсу користувача.
* Контролер, який обробляє вхідні запити, виконує операції над моделлю та обирає представлення для того, щоб показати користувачеві.

Модель є визначенням оточення, в якому працює програмний додаток . У банківському додатку, наприклад, моделями є рахунки, головна книга та кредитні обмеження для клієнтів, а також операції, які можуть бути використані для маніпулювання даними в моделі. Модель також несе відповідальність за збереження загального стану та узгодженості даних, например, вона гарантує, що всі угоди будуть додані в книгу, і що клієнт не зможе зняти більше грошей, ніж він має на рахунку, або більше, ніж має банк. [2]

Представлення містять у собі логіку, необхідну для відображення елементів моделі користувачеві. Контролери можуть бути мостом між відображеннями та моделлю: запити надходять від клієнта та обробляються контролером, який обирає відповідний вигляд для відображення користувачеві й, за необхідності, відповідні дії, які необхідно виконати з моделлю. [2]

Кожен елемент MVC архітектури є чітко визначеним та автономним, але в той же час вигляд та контролер залежать від моделі, яка незалежна від останніх. Логіка, що маніпулює даними в моделі, міститься виключно в моделі; логіка, що відображує дані, знаходиться тільки у вигляді, а код, який обробляє запити користувачів і введені дані, міститься лише в контролері. Це дозволяє будувати модель незалежно від візуального представлення, а також створювати кілька різних виглядів для однієї моделі. З чітким розподілом обов’язків між кожною з частин програмний додаток легше супроводжувати, незалежно від того, наскільки об’ємним він є. [2]

* 1. Реалізація

Для реалізації схеми використовується достатньо велика кількість шаблонів проектування (в залежності від складності архітектурного рішення), основними з яких є ,,Компонувальник’’, ,,Наглядач’’, ,,Стратегія’’. Найбільш типова реалізація схеми MVC відділяє представлення від моделі за допомогою рішення, представленого в шаблоні проектування ,,Наглядач’’. При обробці дій користувача вигляд обирає потрібний контролер в залежності від потрібної поведінки для взаємодії з моделлю. Для цього використовується шаблон проектування ,,Стратегія’’ або ,,Команда’’. А для однотипної взаємодії з компонентами ієрархічної структури представлень використовуєтсья ,,Компонувальник’’. [6]

1.5 Переваги MVC [5 ст. 7 – 8 , 10 – 11]

Переваги MVC поділяються на декілька категорій:

1. Розподілення логіки відношень між компонентами

Відокремлення моделі, вигляду та контролеру:

1. сприяє повторному використанню бізнес – логіки додатку;
2. сприяє розробці декількох графічних інтерфейсів без використання коду бізнес – логіки додатку;
3. запобігає дублюванню коду,спрощує модернізацію та супроводження додатку.

Дивлячись з іншої сторони, об’єднання коду моделі, контролеру та вигляду призводить до:

1. збільшення об’єму, утворення гірших в плані супроводження класів;
2. дублювання коду в класах;

2. Спеціалізація розробника та напрям розробки:

1.) розробники графічного інтерфейсу користувача можуть зосередитися лише на розробці графічного інтерфейсу, не вникаючи в правила бізнес – логіки коду;

2.) Розробники бізнес – логіки додатку можуть зосередитися лише на реалізації останньої з відсутністю утомлюючої роботи з багатьма віджетами.

3. Паралельна розробка додатку різними командами розробників;

1.) Розробники бізнес – логіки можуть створити класи – заглушки, що дозволяють почати працювати розробникам графічних інтерфейсів до того часу, як повністю буде реалізована бізнес – логіка;

2.) графічний інтерфейс може змінюватися безліч разів за вимогою замовника без перекомпіляції додатку та внесення змін в код бізнес – логіки;

3.) біснес методи менше впливають на зміну в графічному інтерфейсі (тому що бізнес методи не завжди порушують структури даних, що відображаються користувачеві);

Розглянувши категорії більш детально, стає зрозумілим, що друга та третя категорії є наслідками першої.

* 1. Варіанти MVC

1.6.1 Model – View – Presenter (MVP) [3]

Model – View – Presenter (Модель – Представлення – Пред’явник) (MVP) – шаблон проектування, який вперше з’явився в IBM, а потім використовувався в Taligent в 1990-х. MVP є похідним шаблоном від MVC, при цьому має дещо інший підхід. В MVP вигляд не тісно зв’язане з моделлю, як це спостерігається в MVC.

Пред'явник зайняв місце контролера та відповідає за переміщення даних, введених користувачем, а також за оновлення вигляду при змінах, що виникли в моделі. Пред'явник контактує з виглядом через интерфейс, який дозволяє збільшити здатність до тестування, так як модель може бути замінена на спеціальний макет для модульних тестів.

MVP є великим кроком вперед в порівнянні з MVC в декількох напрямках:

* MVP забезпечує перевірку стану та логіки відображення, поміщаючи їх у Пред'явник.
* Вигляд жорстко відокремлюється від моделі, при цьому зв’язок організовується через Пред'явник. На відміну від MVC, MVP допускає повторне використання бізнес – логіки без безпосереднього видозміни моделі (за рахунок використання интерфейсу вигляду). Наприклад, якщо необхідно створити новий інтерфейс програмного додатку, то просто створюється новий Вигляд, що реалізує інтерфейс вигляду, а Пред'явник та Модель не змінюється.

Різновиди MVP [7]:

* 1. Пасивне Представлення

Представлення майже не містить логіки. Пред'явник — посередник між Представленням та Моделлю. Крім того, у Представлення та Моделі повністю закритий доступ один до одного. Модель може викликати події, але Пред'явник підписується на них для оновлення Представлення.

У "Пасивному Представленні" немає прямої прив'язки даних, натомість, Представлення надає set – властивості, які Пред'явник використовує для наданнят значень данним. Всі стани керуються Пред'явником, а не Представленням.

Плюси: легкість тестування, прозорий поділ Представлення та Моделі.

Мінуси: витрати часу на самостійну прив'язку даних.

2.) Supervising Controller

Вданному випадку Пред'явник обробляє події користувача. Представлення зв'язується з Моделлю безпосередньо через прив'язку даних. У цьому випадку, завдання Пред'явника "пройти" від Моделі до Представлення, таким чином, щоб могла відбутися прив'язка даних. Пред'явник буде також містити логіку для таких подій як натискання клавіш, навігації і т.д.

Плюсом є  зменшення обсягу коду за рахунок використання прив'язки даних.

Мінуси:

1.) більш складне тестування;

2.) зменшення інкапсуляції у Представленні через прямий зв'язок з Моделлю.

1.6.2 Model-View-ViewModel (MVVM) [4]

Model-View-ViewModel ( модель – вигляд – модель вигляду) (MVVM) - це шаблон, який з’явився з метою подолання обмеженнь шаблонів MVC та MVP та об’єднання деяких їхніх сильних сторін. Ця модель вперше з’явилась у складі фреймворку Small Talk у 80 – х, та пізніше була покращена з урахуванням оновлень MVP.

Шаблон MVVM складається з трьох основних компонентів:

* 1. Модель (Model) - як і в класичному шаблоні MVC, Модель представляє

собою фундаментальні дані, що необхідні для роботи застосунку.

* 1. Вигляд (View) - як і в класичному шаблоні MVC, Вигляд — це графічний інтерфейс, тобто вікно, кнопки тощо.
  2. Модель вигляду (ViewModel, що означає «Model of View») з одного боку є абстракцією Вигляду, а з іншого надає обгортку даних з Моделі, які мають зв'язуватись. Тобто вона містить Модель, дані якої відображає Вигляд, а також містить у собі команди, якими може скористатися Вигляд для впливу на Модель. Фактично ViewModel призначена для того, щоб здійснювати зв'язок між моделлю та вікном, відслідковувати зміни в даних, що зроблені користувачем, відпрацьовувати логіку роботи View (механізм команд).

Переваги MVVM:

1. Тестування MVVM – додатків.

Програмні додатки, в основі яких лежить MVVM, мають схильність до проведення модульного тестування з ціллю перевірки працездатності окремих класів та методів;

1. Менша кількість програмного коду.

Об’єм коду, необхідного для керування Виглядом дещо зменшується при використанні MVVM, а це означає, що знижується ризик допустити помилки та зменшується програмний код модульних тестів;

1. Покращене проектування додатків.

Розробники та дизайнери можуть самостійно працювати над різними частинами додатку: дизайнери – над зовнішнім виглядом, розробники – над бізнес – логікою додатку.

1. Легкість у розумінні логіки Вигляду

MVVM передбачає гарно організовану та лекгу для розуміння конструкцію побудови графічного інтерфейсу за рахунок використання команд та шаблонів даних.

* 1. Реалізація MVC у програмному додатку

У програмному додатку використовуватиметься класичний MVC з пасивною моделлю.

1. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Прецеденти

Система інтернет – магазину надає можливість неавторизованому користувачеві авторизуватись в системі, зареєструватись у тому випадку, якщо користувач не має облікового запису, та відновити пароль від облікового запису (Рис 2.1).

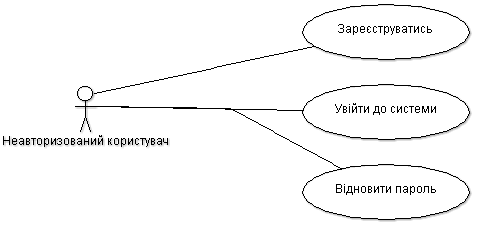


Рис. 2.1 Діаграма прецедентів операції авторизації користувача в системі

Користувач може редагувати дані власного облікового запису (Рис. 2.2).

Рис. 2.2 Діаграма прецедентів операції редагування даних облікового запису

Користувач може повернутись на головну вкладку, якщо він її закрив, переглянути історію замовлень та інформацію про програму (Рис 2.3).

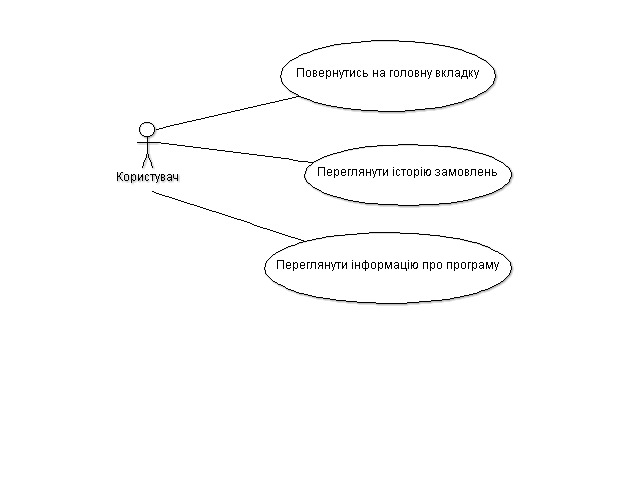


Рис. 2.3 Діаграма прецедентів операцій повернення на головну вкладку,

перегляду історії замовлень та інформації про програму

Користувач може замовити товар в інтернет – магазині (Рис. 2.4).



Рис. 2.4 Діаграма прецедентів здійснення замовлення користувачем

Користувач може переглянути каталог товарів, запропонованих інтернет – магазином (Рис 2.5).



Рис. 2.5 Діаграма прецедентів операції перегляду каталогу товарів користувачем

2.2 Сценарії прецедентів

* Сценарій прецеденту авторизації користувача в системі (Рис. 2.1)

1. Неавторизований користувач запускає програму;
2. Система створює вкладку ,,Авторизація’’ та відображає панель входу в систему;
3. Користувач виконує одну з дій:
   1. натискає кнопку ,,Зареєструватись’’ :
      1. система відображає панель реєстрації та ховає панель входу в систему;
      2. користувач вводить дані в поля та натискає кнопку ,,Реєстрація’’;
      3. система перевіряє дані на коректність та повідомляє про результат реєстрації:
         1. якщо результат реєстрації позитивний – система відображає вкладку ,,Головна’’ додатку, повідомляє про завершення реєстрації та активізує всі можливості користувача;
         2. Якщо результат негативний – п. 3.1.1;
      4. реєстрація завершена;
   2. натискає кнопку ,,Відновити пароль’’:
      1. система відображає панель для відновлення паролю та ховає панель входу в систему;
      2. користувач вводить дані в поля та натискає кнопку ,,Відновити пароль’’;
      3. система перевіряє введені дані на коректність та повідомляє про результат відновлення.
      4. якщо результат відновлення паролю позитивний – система відображає панель входу в систему та ховає панель відновлення паролю; в іншому випадку – п.3.2.1.
      5. операція з відновлення паролю завершена;
   3. вводить дані в поля панелі входу та натискає кнопку ,,Увійти’’:
      1. система перевіряє введені дані на коректність;
      2. якщо дані – коректні, система відображає вкладку ,,Головна’’ додатку та активізує всі можливості користувача; в іншому випадку – п. 3.3;
4. Прецедент завершується входом користувача в систему.

* Сценарій прецеденту редагування даних облікового запису (Рис. 2.2).

Передумова: користувач має авторизуватись у системі.

1. Користувач натискає кнопку ,,Меню’’ та обирає підпункт ,, Редагувати дані’’;
2. система створює нову вкладку з ім’ям ,,Редагування’’ , де відображає системну та контактну інформацію облікового запису;
3. користувач змінює дані та натискає кнопку ,, Зберегти’’;
4. система перевіряє дані на правильність та вносить корективи в базу даних; система вносить корективи в базу даних;
5. сценарій завершується відображенням інформаційного вікна, яке сповіщає користувача про результат операції.

* Сценарій прецеденту повернення на головну вкладку (Рис. 2.3).

1. Користувач натискає кнопку ,, Меню’’ та обирає пункт ,, На головну’’;
2. система створює вкладку ,,Головна’’ або передає фокус головній вкладці, якщо вона не закрита;
3. сценарій завершується відображенням вкладки ,,Головна’’.

* Сценарій прецеденту перегляду історії замовлень (Рис. 2.3).

1. Виконання сценарію прецеденту ,,Авторизація в системі’’ або ,,Повернутись на головну вкладку’’;
2. сценарій завершується відображенням вкладки ,,Головна’’ з таблицею, що містить історію замовлень;

* Сценарій прецеденту перегляду інформації про програму (Рис. 2.3).

1. Користувач натискає кнопку ,,Меню’’ та обирає підпункт ,,Про програму’’
2. Система виводить інформаційне діалогове вікно з версією програми та інформацією про автора та його контактну інформацію;
3. Користувач натискає кнопку ,,Ок’’ для завершення ознайомлення з інформацією про програму.

* Сценарій прецеденту здійснення замовлення (Рис. 2.4)

Передумова: кошик користувача має один або декілька товарів.

1. Користувач натискає кнопку ,,Меню’’ -> ,, Кошик ’’;
2. система створює нову вкладку з ім’ям ,,Кошик’’, де відображає інформацію про товари, що знаходяться в кошику;
3. користувач натискає кнопку ,,Замовити’’ ;
4. система створює нову вкладку з ім’ям ,,Замовлення’’ , де відображає панелі оформлення замовлення;
5. користувач вводить інформацію в поля панелі та натискає кнопку ,,Завершити’’;
6. система додає інформацію про замовлення в базу даних та до історії замовлень користувача, видаляє інформацію про замовлені товари з кошику;
7. сценарій завершується закриттям вкладки оформелення замовлення та відображенням інформаційного повідомлення.

* Сценарій перегляду каталогу товарів (Рис. 2.5).

Передумова: користувач має авторизуватись у системі.

1. Користувач натискає кнопку ,,Меню’’ та обирає підпункт ,, До каталогу ’’;
2. система створює нову вкладку з ім’ям ,, Каталог товарів’’ , де відображає каталог товарів та дерево каталогу;
3. користувач може виконати одну з дій :
   1. відобразити вміст каталогу:
      1. користувач клікає на назві каталогу, вміст якого хоче переглянути, в дереві каталогу;
      2. система відображає вміст каталогу в панелі каталогу.
   2. перейти на сторінку з інформацією про конкретний товар, натиснувши на його відображення в дереві каталогу чи в робочій області вкладки, з метою перегляду інформації про нього:
      1. система створює нову вкладку з назвою товару , відображає детальну інформацію про нього та кнопку ,,Додати до кошика’’.
      2. якщо користувач натискає кнопку ,,Додати до кошика’’, то він може замовити цей товар через кошик.
4. сценарій завершується, коли користувач закриває вкладку ,,Каталог товарів’’ , усі вкладки, що відображають детальну інформацію про конкретний товар, та переходить на вкладку ,, Головна’’.
   1. Діаграми граничних класів

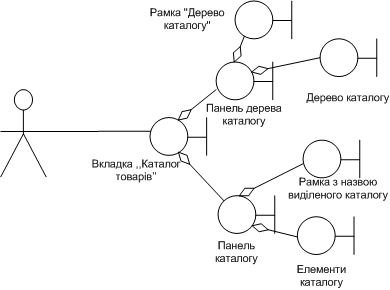


Рис. 2.6 Діаграма граничних класів вкладки ,,Каталог товарів’’

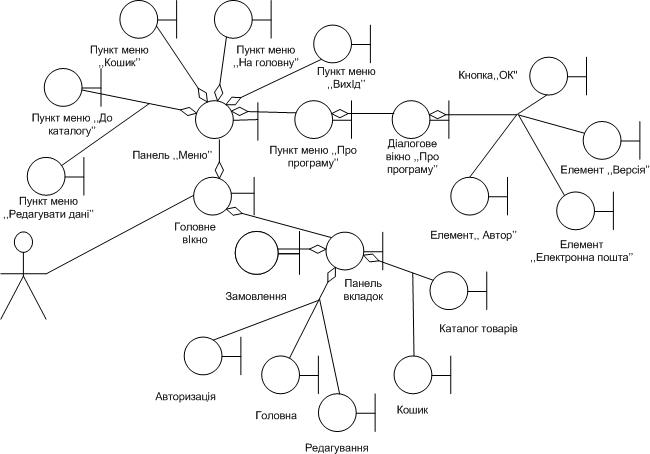


Рис. 2.7 Загальна діаграма граничних класів програмного додатку

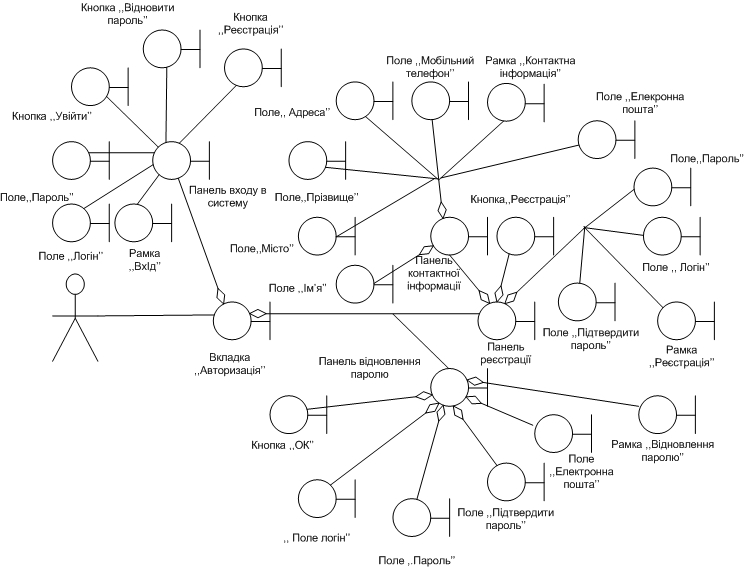


Рис. 2.8 Діаграма граничних класів вкладки ,,Авторизація’’

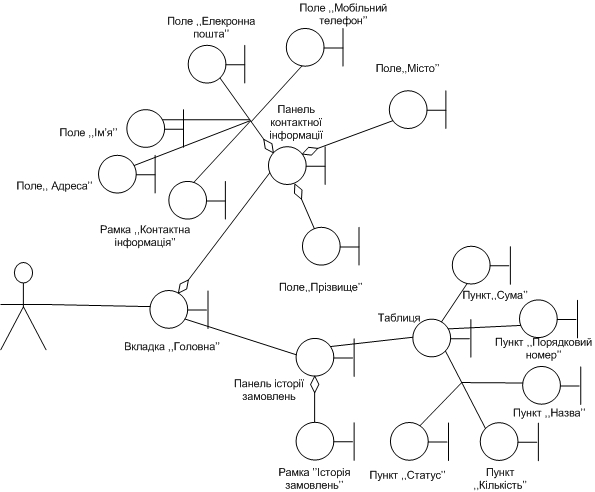


Рис. 2.9 Діаграма граничних класів вкладки ,, Головна’’

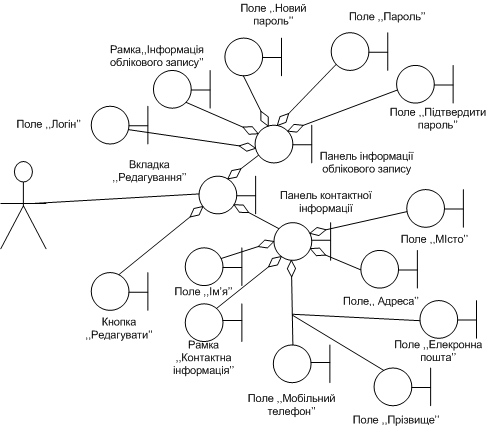


Рис. 2.10 Діаграма граничних класів вкладки ,,Редагування’’

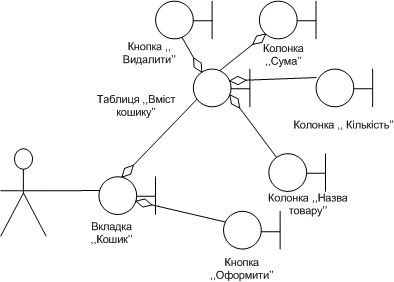


Рис. 2.11 Діаграма граничних класів вкладки ,,Кошик’’

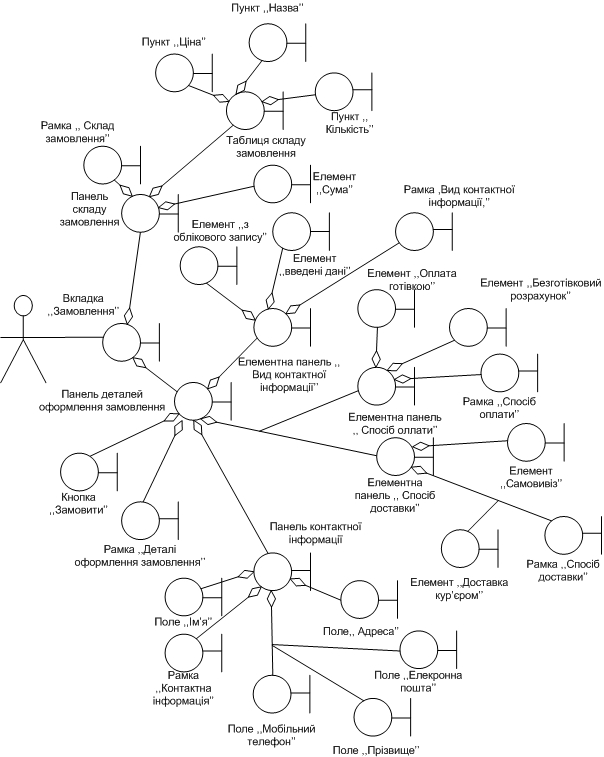


Рис. 2.12 Діаграма граничних класів вкладки ,,Замовлення’’

2.4 Проектування графічного інтерфейсу користувача

На рисунках 2.13 – 2.19 зображено ескіз графічного інтерфейсу користувача програмного додатку <<Інтернет – магазин ,,TechnoUA’’>> . На них виділені області розташування всіх елементів.

Графічний інтерфейс користувача складається з головного вікна, декількох вкладок та діалогового вікна. Він розроблявся таким чином, щоб був легким у використанні та ,,інтуїтивно зрозумілим’’ користувачеві.

При розробці інтерфейсу користувача бралась до уваги праця [5], де автор детально описуює усі аспекти проеткування складних і, водночас, привабливих та простих у сприйнятті графічних інтерфейсів користувача програмних додатків.

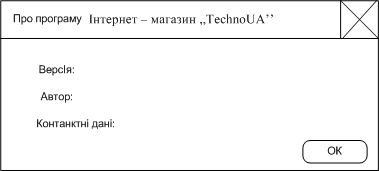


Рис. 2.13 Ескіз діалогового вікна ,,Про програму’’

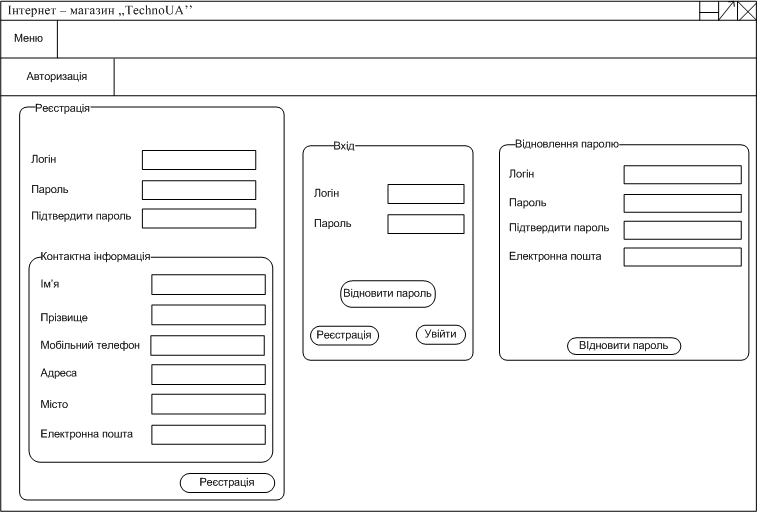


Рис. 2.14 Ескіз графічного інтерфейсу вкладки ,,Авторизація’’

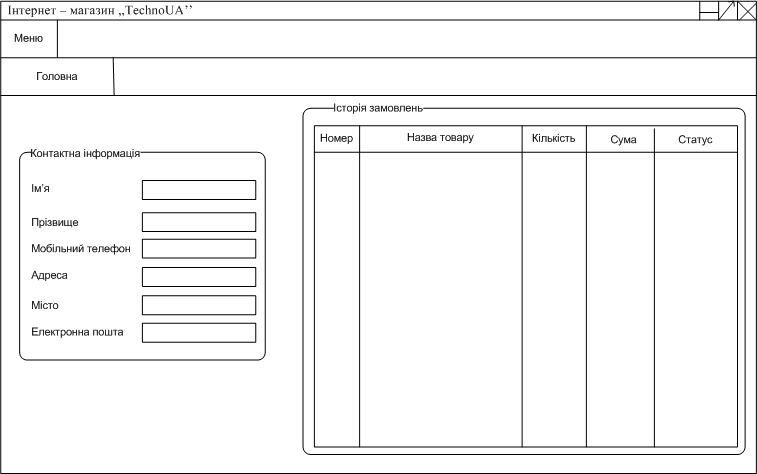


Рис. 2.15 Ескіз графічного інтерфейсу вкладки ,,Головна’’

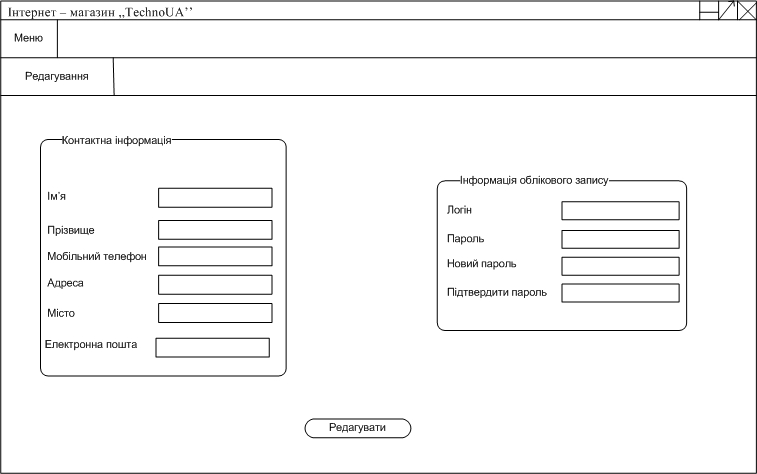


Рис. 2.16 Ескіз графічного інтерфейсу вкладки ,,Редагування’’

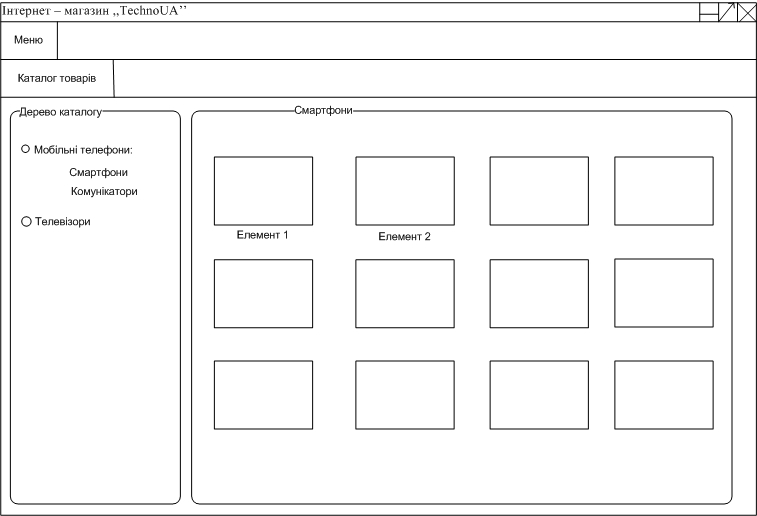


Рис. 2.17 Ескіз графічного інтерфейсу вкладки ,,Каталог товарів’’

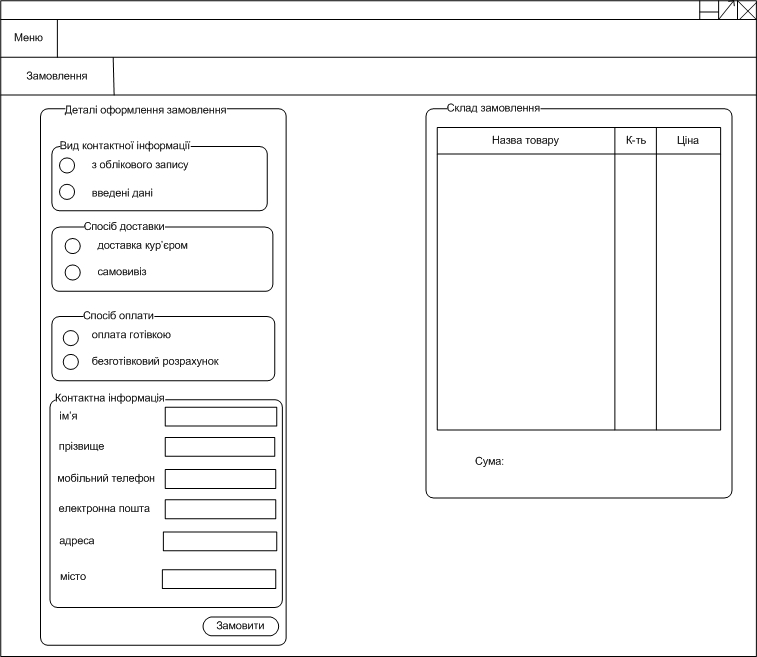


Рис. 2.18 Ескіз графічного інтерфейсу вкладки ,,Замовлення’’

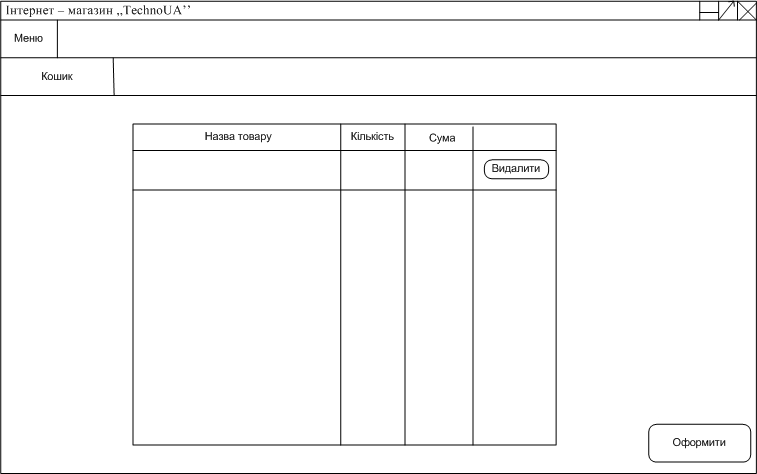


Рис. 2.19 Ескіз графічного інтерфейсу вкладки ,,Кошик’’

Графічний інтерфейс користувача був розроблений за допомогою стандартних засобів Java, а саме графічної бібліотеки SWING. Класи, які були використані для розробки інтерфейсу, наведені в табл. 2. 1(див. додаток Б).

1. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Структура програмного додатку.

Діаграми класів програмного додатку “Інтернет – магазин ,,TechnoUA’’” створені для розуміння структури програми вцілому. На них зображені структурні зв’язки між класами.

Програмний додаток складається з чотирьох підсистем: конфігурування, відображення, контролеру та об’єктно-реляційного відображення.

Підсистема конфігурування зображена на рисунку 3.1. В основі закладено структурний шаблон проектування ‘Міст’, де кореневим елементом ієрархії абстракцій є клас AbstractConfigurator, а ієрархії реалізації – IConfigurator.

Підсистема працює з зовнішніми xml - файлами. Логіка такої взаємодії інкапсульована в абстрактному класі FileConfigurator та розширена у класі XMLConfigurator. Клас SAXConfigurator призначений для обробки даних, поданих у xml-файлах.

Також у даній підсистемі використаний шаблон проектування ‘Фасад’, у ролі якого виступає клас ConfigFacade.

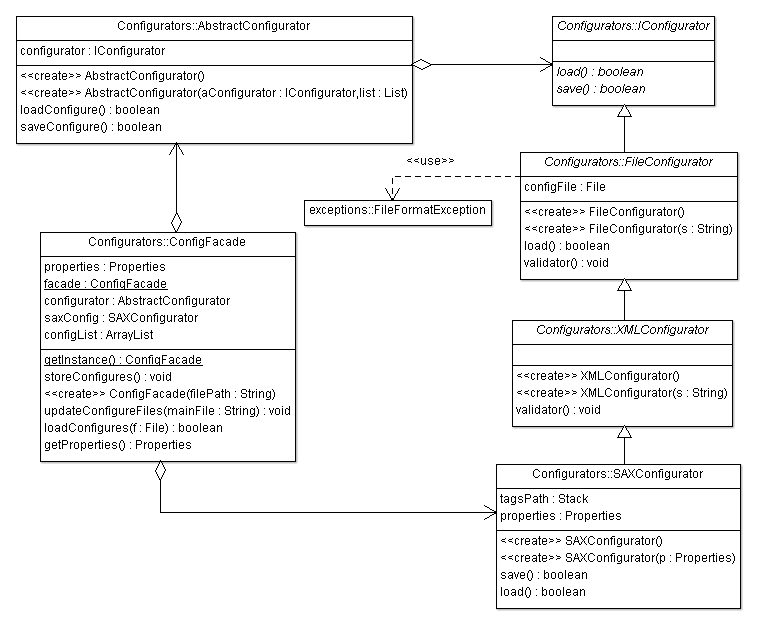


Рис.3.1 Підсистема конфігурування програмного додатку

Підсистема об’єктно-реляційного відображення зображена на рис.3.2. Основу системи складають інтерфейс CRUDInterface, що інкапсулює логіку взаємодії програмного додатку зі сховищем даних, та його конкретними класами -реалізаціями – TransparendCRUD (має виключно методи – “заглушки”, створений задля зменшення кількості програмного коду при розширенні програмного додатку) та ApplicationCRUD, що наділений логікою взаємодії програмного додатку з базою даних. Ця ієрархія надає основну функціональність взаємодії з будь-яким сховищем даних, а саме відкрити/закрити з'єднання, видалити/ змінити/ оновити/ вибрати дані зі сховища.

Для встановлення з’єднання використовується клас DBConfigurator. Низькорівневу логіку реалізації об’єктно-реляційного відображення реалізує утілітний клас DAOAnnotationUtility, який використовує класи, що реалізують інтерфейс ValueConverter та може взаємодіяти лише з тими класами підсистеми, що розмічені анотаціями Primary та Stored

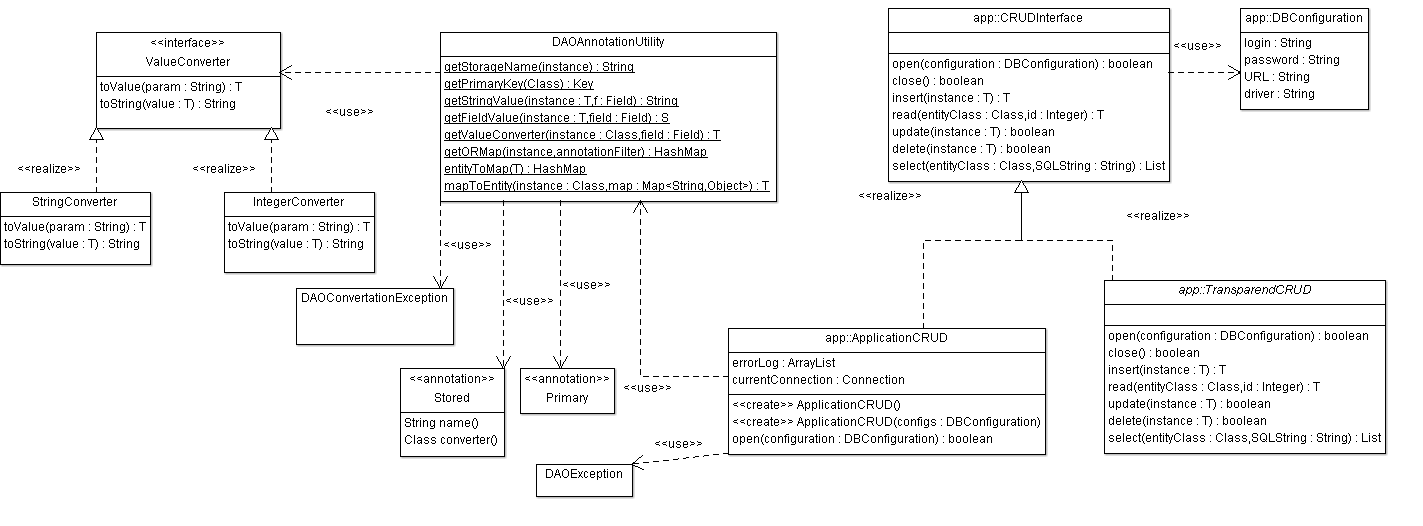


Рис.3.2 Підсистема об’єктно-реляційного відображення

Підсистема контролеру зображена на рисунку 3.3. Основою підсистеми є класи Controler, що реалізує логіку реєстрування запиту користувача системою. Цей клас, прийнявши запит від зовнішнього середовища, звертається до класу CommandFactory з запитом на пошук необхідного об’єкту-команди типу AbstractCommand, після чого передає об’єкту типу Validator отриману команду та вхідні дані з метою перевірки працездатності команди з такими даними, та, після підтвердження працездатності команди, ставить її в чергу на виконання.

Конкретні класи-команди мають бути розміченими анотаціями Command, що ідентифікує команду та вказує, чи може команда працювати без вхідних даних; View, що вказує ідентифікатори тих об’єктів-відображень, які мають бути зміненими в результаті виконання команди; ContextAnnotation та Parameter, які інкапсулюють мета-дані щодо вхідних даних команди.

Вхідні дані для об’єктів – команд містить в собі об’єкт типу Context.

Класи Controller, CommandFactory виконані згідно шаблону проектування ‘Одинак’.

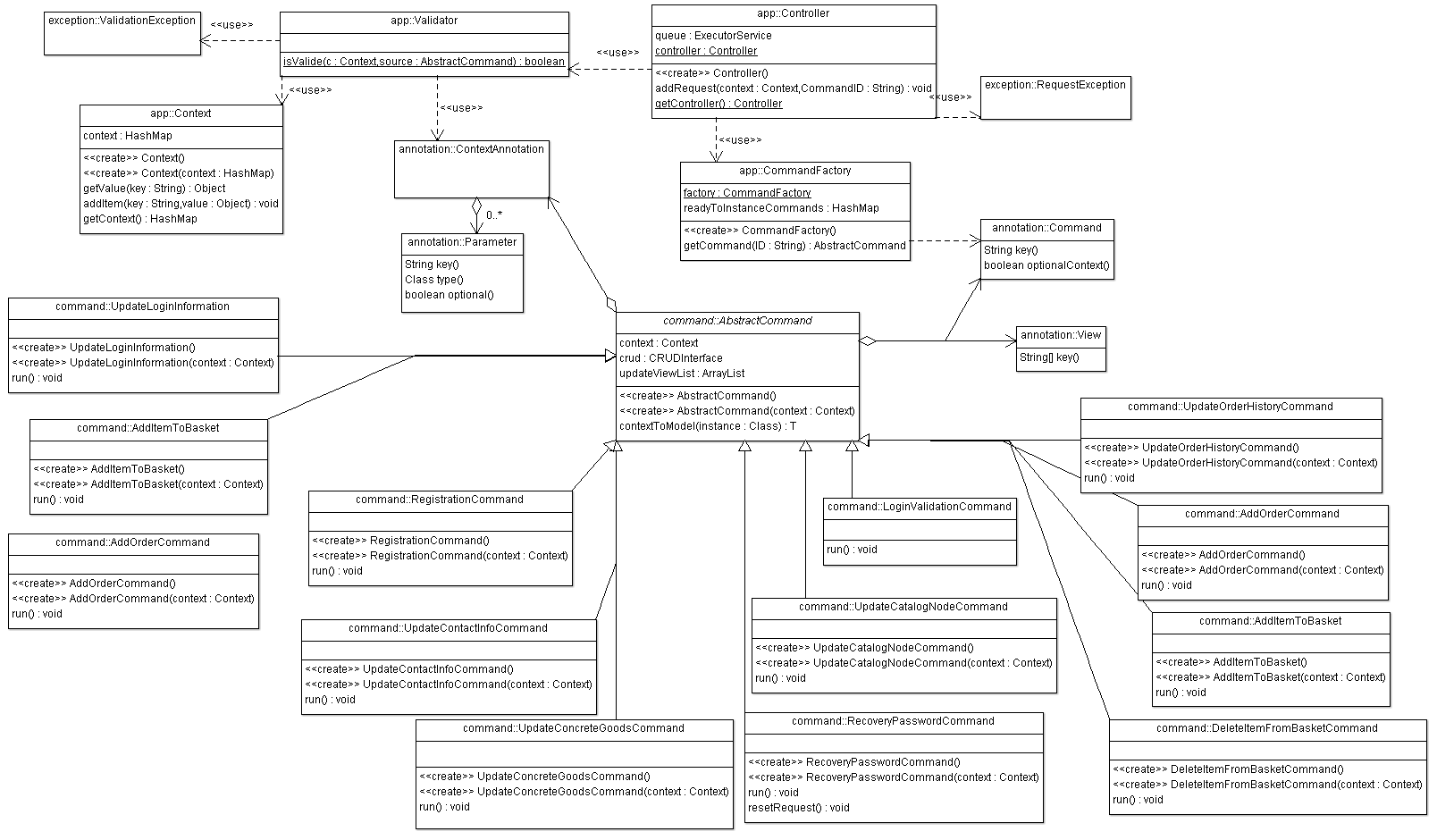


Рис.3.3 Підсистема контролеру

Підсистема відображення зображена на рисунку 3.4. Усі класи-відображення реалізують інтерфейс AbstractView з метою послаблення зв’язку між класами-командами та відображеннями. Ті класи-відображення, що формують об’єкт типу Context, агрегують об’єкт типу ApplicationValidator.

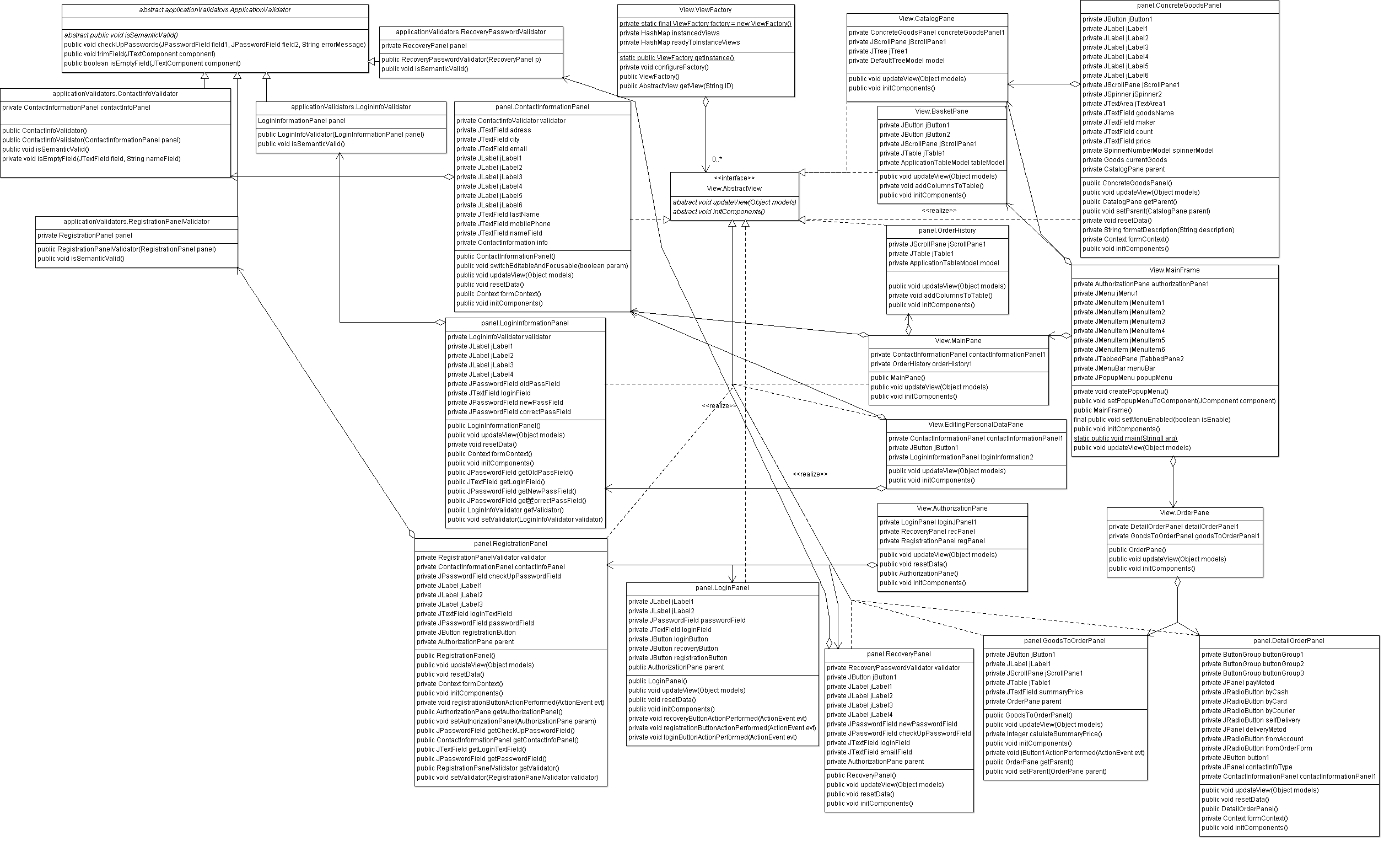


Рис.3.4 Підсистема відображень

Загальна структура програмного додатку зображена на рисунку 3.5.

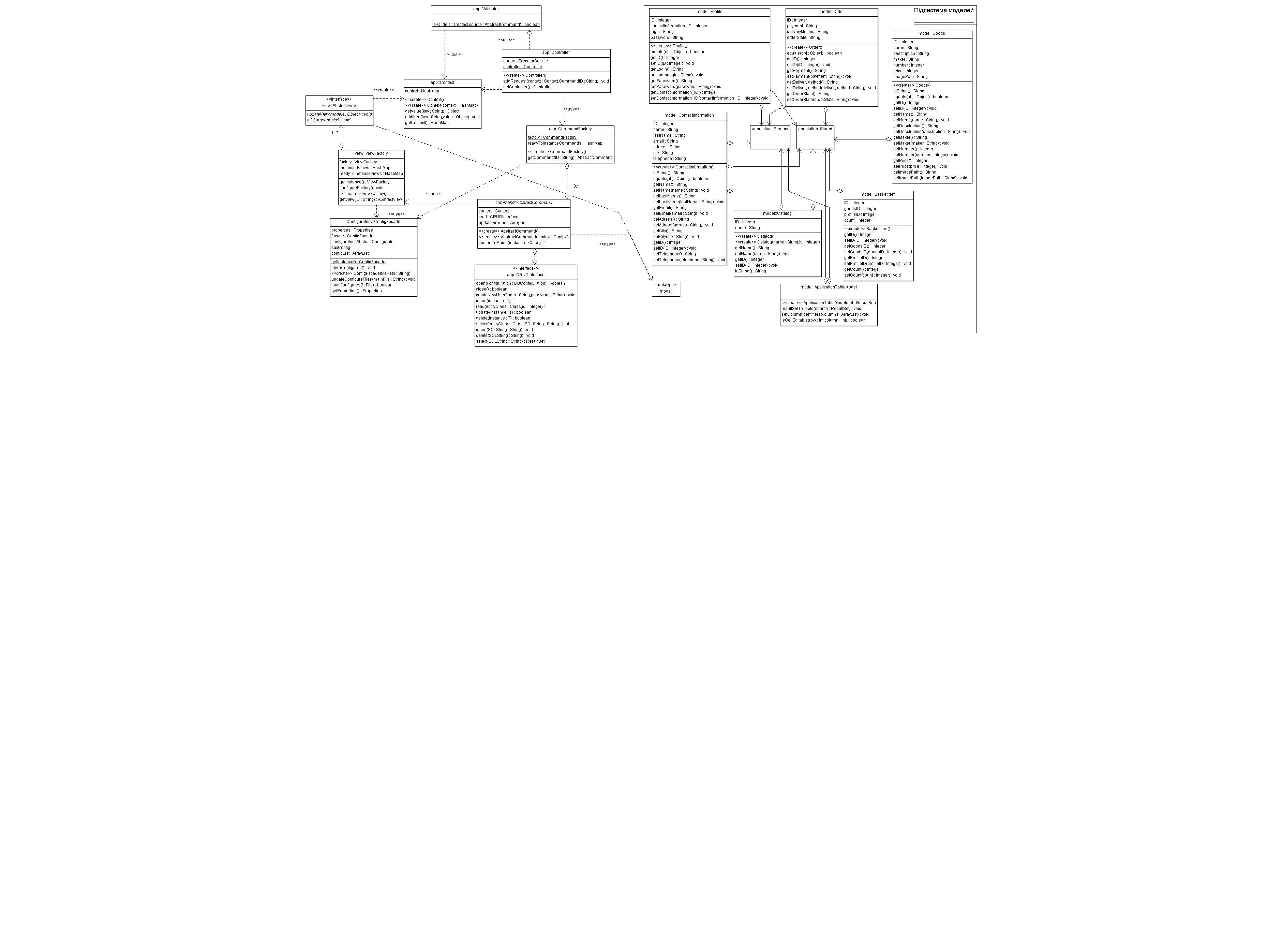


Рис.3.5 Загальна структура програмного додатку

* 1. Специфікації класів

**A**

[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Клас представляє кореневий елемент ієрархії команд програмного додатку.

[AbstractCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#AbstractCommand()) - Конструктор класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Конструктор класу без параметрів, що ініціалізує значення полів [AbstractCommand.crud](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#crud), [AbstractCommand.updateViewList](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#updateViewList).

[AbstractCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#AbstractCommand(app.Context)) - Конструктор класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Конструктор класу, що додатково ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

Клас інкапсулює загальну логіку роботи конфігураційної системи, в основі якої лежить шаблон проектування'' Міст''.

[AbstractConfigurator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html#AbstractConfigurator()) - Конструктор класу Configurators.[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

Конструктор класу без параметрів

[AbstractConfigurator(IConfigurator, List<String>)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html#AbstractConfigurator(Configurators.IConfigurator, java.util.List)) - Конструктор класу Configurators.[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

Конструктор класу.

[AbstractView](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AbstractView.html) - Interface in [View](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\package-summary.html)

Клас є коренем ієрархії об'єктів відображень, задає загальний протокол, який має бути реалізований з метою правильної роботи програмного додатку.

[addItem(String, Object)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html#addItem(java.lang.String, java.lang.Object)) - Метод класу app.[Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html)

Метод додає пару "ключ-значення" до мапи.

[AddItemToBasket](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddItemToBasket.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда додає рядок до таблиці, що відповідає об'єктному представленню кошика.

[AddItemToBasket()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddItemToBasket.html#AddItemToBasket()) - Конструктор класу command.[AddItemToBasket](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddItemToBasket.html)

Конструктор класу без параметрів.

[AddItemToBasket(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddItemToBasket.html#AddItemToBasket(app.Context)) - Конструктор класу command.[AddItemToBasket](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddItemToBasket.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[AddOrderCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddOrderCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда додає замовлення до бази даних, після чого видаляє дані з таблиці, що відповідає об'єктній моделі кошика.

[AddOrderCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddOrderCommand.html#AddOrderCommand()) - Конструктор класу command.[AddOrderCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddOrderCommand.html)

Конструктор класу без параметрів.

[AddOrderCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddOrderCommand.html#AddOrderCommand(app.Context)) - Конструктор класу command.[AddOrderCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AddOrderCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[addRequest(Context, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html#addRequest(app.Context, java.lang.String)) - Метод класу app.[Controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html)

Метод додає команду в поток виконання запитів.

[adress](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#adress) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля ADRESS в реляційній таблиці.

[adress](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html#adress) - Змінна класу panel.[ContactInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html)

[annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html) - пакет annotation

[app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html) - пакет app

[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас інкапсулює в собі логіку взаємодії програмного додатку з реляційною базою даних та реалізує об'єктно - реляційну модель представлення даних.

[ApplicationCRUD()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#ApplicationCRUD()) - Конструктор класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Конструктор класу без параметрів.

[ApplicationCRUD(DBConfiguration)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#ApplicationCRUD(app.DBConfiguration)) - Конструктор класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Конструктор класу, що відразу створює з'єднання з базою даних.

[ApplicationTableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

[ApplicationTableModel(ResultSet)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html#ApplicationTableModel(java.sql.ResultSet)) - Конструктор класу model.[ApplicationTableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html)

[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html) - Клас в пакеті [applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html)

Клас призначений для семантичної перевірки введених користувачем даних та їх корегування.

[ApplicationValidator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html#ApplicationValidator()) - Конструктор класу applicationValidators.[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html)

[applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html) - пакет applicationValidators

[AuthorizationPane](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AuthorizationPane.html) - Клас в пакеті [View](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\package-summary.html)

Вкладка створена, щоб реалізувати прецедент "Авторизація користувача".

[AuthorizationPane()](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AuthorizationPane.html#AuthorizationPane()) - Конструктор класу View.[AuthorizationPane](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AuthorizationPane.html)

Конструктор класу без параметрів.

[authorizationPane1](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainFrame.html#authorizationPane1) - Змінна класу View.[MainFrame](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainFrame.html)

**B**

[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас є об'єктним відображенням рядку таблиці, що містить інформацію про кошик інтернет-магазину.

[BasketItem()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html#BasketItem()) - Конструктор класу model.[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html)

Конструктор класу без параметрів.

[buildDocument(Set<Object>, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#buildDocument(java.util.Set, java.lang.String)) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Метод будує об'єкт типу Document, вхідними данними якого є множина конфігураційних даних та назва кореневого елементу.

**C**

[Catalog](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html) - Клас в пакеті  [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас є об'єктним відображенням рядку таблиці, що містить інформацію про конкретний каталог.

[Catalog()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html#Catalog()) - Конструктор класу model.[Catalog](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html)

Конструктор класу без параметрів.

[Catalog(String, Integer)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html#Catalog(java.lang.String, java.lang.Integer)) - Конструктор класу model.[Catalog](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html)

[checkUpPasswords(JPasswordField, JPasswordField, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html#checkUpPasswords(javax.swing.JPasswordField, javax.swing.JPasswordField, java.lang.String)) - Метод класу applicationValidators.[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html)

Метод перевіряє паролі, введені до двох полів, на рівність.

[chooseConfigures(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#chooseConfigures(java.lang.String)) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Метод вибирає з об'єкту Properties ті дані, в ключі яких присутній параметр.

[city](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#city) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля CITY в реляційній таблиці.

[city](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html#city) - Змінна класу panel.[ContactInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html)

[close()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#close()) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує закриття діючого підключення до бази даних.

[close()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#close()) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує закриття діючого підключення до бази даних.

[close()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#close()) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує закриття діючого підключення до бази даних.

[command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html) - пакет command

[Command](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\Command.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Даною анотацією помічається об'єкт - команда типу [AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html).

[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Основне завдання даного класу є створення нових об'єктів-команд згідно з переданим ключем у якості параметру.

[CommandFactory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#CommandFactory()) - Конструктор класу app.[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html)

Конструктор класу, конфігурує поле [CommandFactory.readyToInstanceCommands](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#readyToInstanceCommands)

[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

Клас надає єдиний уніфікований інтерфейс до підсистеми конфігурування, тим самим зменшуючи залежності між даною підсистемою та підсистемами-користувачами та полегшуючи її використання.

[ConfigFacade(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#ConfigFacade(java.lang.String)) - Конструктор класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Конструктор, який ініціалізує значення полів класу та завантажує конфігураційні дані.

[configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile) - Змінна класу Configurators.[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html)

[configList](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#configList) - Змінна класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Список конфігураційних файлів.

[configurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html#configurator) - Змінна класу Configurators.[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

Клас ієрархії реалізацій конфігураційної системи, що реалізує інтерфейс [IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html).

[configurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#configurator) - Змінна класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

об'єкт класу [AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

[Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html) - пакет Configurators

[configureFactory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#configureFactory()) - Метод класу app.[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html)

Метод конфігурує фабрику команд відповідними класами команд та їх ключами.

[configureFactory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#configureFactory()) - Метод класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

Метод конфігурує фабрику конфігураційними даними.

[contactInfoPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#contactInfoPanel) - Змінна класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас інкапсулює контактну інформацію про користувача.

[ContactInformation()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#ContactInformation()) - Конструктор класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Конструктор класу без параметрів.

[ContactInformation(Integer, String, String, String, String, String, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#ContactInformation(java.lang.Integer, java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String)) - Конструктор класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Конструктор класу, дані для якого взяті з реляційної таблиці

[ContactInformation(String, String, String, String, String, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#ContactInformation(java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String, java.lang.String)) - Конструктор класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Конструктор класу, дані для якого взяті з програмного додатку

[contactInformation\_ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#contactInformation_ID) - Змінна класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Поле зберігає значення ID контактної інформації даного облікового запису

[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html) - Клас в пакеті [applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html)

Клас перевіряє дані, введені в об'єкт типу [ContactInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html) на семантичну правильність (наприклад, поле 'Ім'я' не може містити цифри, поле 'Телефон' може містити виключно цифри)

[ContactInfoValidator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#ContactInfoValidator()) - Конструктор класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

Конструктор класу без параметрів.

[ContactInfoValidator(ContactInformationPanel)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#ContactInfoValidator(panel.ContactInformationPanel)) - Конструктор класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

Конструктор класу, що ініціалізує значення поля [ContactInfoValidator.contactInfoPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#contactInfoPanel)

[Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас інкапсулює в собі інформацію, необхідну для правильної роботи об'єктів-команд [AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html).

[Context()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html#Context()) - Конструктор класу app.[Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html)

Конструктор класу без параметрів.

[Context(HashMap<String, Object>)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html#Context(java.util.HashMap)) - Конструктор класу app.[Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html)

Конструктор класу, що ініціалізує значення поля [Context.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html#context) об'єктом-параметром типу HashMap.

[context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html#context) - Змінна класу app.[Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html)

Мапа, що має інформацію, необхідну для правильного функціонування команд.

[context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context) - Змінна класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Об'єкт-контектс, що необхідний для реалізації функцій команд.

[ContextAnnotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\ContextAnnotation.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Анотація містить в собі масив анотацій типу [Parameter](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\Parameter.html).

[contextToModel(Class<T>)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#contextToModel(java.lang.Class)) - Метод класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Метод відображає дані контексту на тип моделі, що вказана у параметрі.

[Controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас є контролером в архітектурі MVC.

[Controller()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html#Controller()) - Конструктор класу app.[Controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html)

Конструктор класу, що ініціалізує значення поля [Controller.queue](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html#queue), задавши логіку реалізації однопотокового виконання запитів.

[controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html#controller) - Static Змінна класу app.[Controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html)

Реалізація шаблону 'Singleton'.

[ConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\ConvertationException.html) - Клас виключення в  [exceptions](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\package-summary.html)

[count](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html#count) - Змінна класу model.[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html)

Кількість товару, що міститься в даному рядку таблиці кошика.

[count](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html#count) - Змінна класу panel.[ConcreteGoodsPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html)

[crud](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#crud) - Змінна класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Об'єкт типу [ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html), що інкапсулює логіку взаємодії з базою даних.

[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html) - Interface in [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Інтерфейс забезпечує взаємодію з персистентним сховищем, в т.ч. з реляційною базою даних.

[currentConnection](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#currentConnection) - Змінна класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Поточне з'єднання з реляційною базою даних.

[currentGoods](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html#currentGoods) - Змінна класу panel.[ConcreteGoodsPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html)

**D**

[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html) - Клас в пакеті [utils](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\package-summary.html)

Утілітний клас, що інкапсулює в собі логіку реалізацції об'єктно - реляційної моделі представлення даних програмного додатку.

[DAOAnnotationUtility()](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#DAOAnnotationUtility()) - Конструктор класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html) - Клас виключення в  [exceptions](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\package-summary.html)

Клас помилки, що викликається при виконанні операцій, пов'язаних з об'єктно-реляційним відображенням.

[DAOConvertationException(String, Throwable, boolean, boolean)](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html#DAOConvertationException(java.lang.String, java.lang.Throwable, boolean, boolean)) - Constructor for exception exceptions.[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html)

[DAOConvertationException(String, Throwable)](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html#DAOConvertationException(java.lang.String, java.lang.Throwable)) - Constructor for exception exceptions.[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html)

[DAOConvertationException(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html#DAOConvertationException(java.lang.String)) - Constructor for exception exceptions.[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html)

[DAOConvertationException(Throwable)](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html#DAOConvertationException(java.lang.Throwable)) - Constructor for exception exceptions.[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html)

[DAOConvertationException()](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html#DAOConvertationException()) - Constructor for exception exceptions.[DAOConvertationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOConvertationException.html)

[DAOException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\DAOException.html) - Клас виключення в  [exceptions](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\package-summary.html)

Клас помилки, що викликається при виникненні помилки будь-якого характеру, що пов'язана з базою даних.

[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас призначений для інкапсуляції даних,за допомогою яких відбувається підключення до бази даних.

[DBConfiguration()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#DBConfiguration()) - Конструктор класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Конструктор класу без параметрів.

[DBConfiguration(String, String, String, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#DBConfiguration(java.lang.String, java.lang.String, java.lang.String, java.lang.String)) - Конструктор класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Конструктор класу, що ініціалізує всі поля класу.

[delete(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#delete(T)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод виконує видалення рядку з таблиці бази даних, об'єктне представлення якого відповідає вхідному параметру.

[delete(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#delete(java.lang.String)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує видалення строки з бази даних

[delete(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#delete(T)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує видалення запису зі сховища, що відповідає об'єктному представленню вхідного параметру.

[delete(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#delete(java.lang.String)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує видалення строки з бази даних

[delete(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#delete(T)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує видалення запису зі сховища, що відповідає об'єктному представленню вхідного параметру.

[DeleteItemFromBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\DeleteItemFromBasketCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда видаляє дані з бази даних, що відповідають об'єктній моделі кошика.

[DeleteItemFromBasketCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\DeleteItemFromBasketCommand.html#DeleteItemFromBasketCommand()) - Конструктор класу command.[DeleteItemFromBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\DeleteItemFromBasketCommand.html)

Конструктор класу без параметрів.

[DeleteItemFromBasketCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\DeleteItemFromBasketCommand.html#DeleteItemFromBasketCommand(app.Context)) - Конструктор класу command.[DeleteItemFromBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\DeleteItemFromBasketCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[deliveryMethod](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#deliveryMethod) - Змінна класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Поле зберігає спосіб доставки замовлення.

[desctription](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#desctription) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає опис товару.

[driver](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#driver) - Змінна класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Назва класу - драйверу, який використовується JDBC для взаємодії з базою даних стороннього виробника.

**E**

[email](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#email) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля EMAIL в реляційній таблиці.

[entityToMap(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#entityToMap(T)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод створює реляційну таблицю, представлену у вигляді об'єкту класу HashMap, з об'єкту - параметру.

**F**

[facade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#facade) - Static Змінна класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Реалізація шаблону 'Singleton'

[factory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#factory) - Static Змінна класу app.[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html)

Реалізація шаблону проектування 'Singleton'

[factory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#factory) - Static Змінна класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

Реалізація шаблону 'Singleton'.

[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

Клас розширює можливості [IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html) та інкапсулює логіку реалізації конфігурування через зовнішні файли з різними розширеннями.

[FileConfigurator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#FileConfigurator()) - Конструктор класу Configurators.[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html)

Конструктор без параметрів, викликає конструктор суперкласу.

[FileConfigurator(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#FileConfigurator(java.lang.String)) - Конструктор класу Configurators.[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html)

Конструктор ініціалізує поле [FileConfigurator.configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile) за допомогою параметру.

[FileFormatException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\FileFormatException.html) - Клас виключення в  [exceptions](file:///C:\Users\Wh15p3r\workspace\Course\doc\exceptions\package-summary.html)

Клас, створений з метою опису помилок, пов'язаних з невідповідністю об'єкта класа File специфічним вимогам.

[fillUpdateViewList()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#fillUpdateViewList()) - Метод класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Метод заповнює поле [AbstractCommand.updateViewList](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#updateViewList), звертаючись із запитами до [ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html) з ID об'єктів-відображень.

**G**

[getFactory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#getFactory()) - Static Метод класу app.[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html)

[getFieldValue(T, Field)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getFieldValue(T, java.lang.reflect.Field)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає значення параметру типу Field об'єкту-параметру, що має задекларований в полі-параметрі тип.

[getInstance()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#getInstance()) - Static Метод класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Повертає значення поля [ConfigFacade.facade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#facade)

[getInstance()](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#getInstance()) - Static Метод класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

Повертає значення поля [ViewFactory.factory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#factory)

[getORMap(Class, Class)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getORMap(java.lang.Class, java.lang.Class)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод виконує відображення параметру типу Class на реляційну таблицю бази даних.

[getPrimaryKey(Class)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getPrimaryKey(java.lang.Class)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає об'єкт типу Field, який зберігає значення первинного ключа.

[getPrimaryKeyValue(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getPrimaryKeyValue(T)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає значення первинного ключа(який може бути будь - яким об'єктом), що зберігається в об'єкті - параметрі.

[getProperties()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#getProperties()) - Метод класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Повертає значення поля [ConfigFacade.properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#properties)

[getStorageName(Class)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getStorageName(java.lang.Class)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає значення таблиці, об'єктне представлення якої інкапсульоване в об'єкті - параметрі типу Class

[getStringValue(T, Field)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getStringValue(T, java.lang.reflect.Field)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає строкове значення параметру типу Field об'єкту-параметру.

[getValueConverter(Field)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#getValueConverter(java.lang.reflect.Field)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод повертає об'єкт класу, що реалізує інтерфейс [ValueConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html).

[getView(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#getView(java.lang.String)) - Метод класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

Метод повертає об'єкт класу [AbstractView](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AbstractView.html) згідно з вхідним параметром.

[global](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\package-summary.html) - пакет global

[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html) - Клас в пакеті [global](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\package-summary.html)

[GlobalContext()](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html#GlobalContext()) - Конструктор класу global.[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html)

[globalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html#globalContext) - Static Змінна класу global.[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html)

[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас є об'єктним відображенням рядку таблиці, що містить в собі інформацію про конкретний товар.

[Goods()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#Goods()) - Конструктор класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Конструктор класу без параметрів.

[Goods(Integer, String, String, String, Integer, Integer)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#Goods(java.lang.Integer, java.lang.String, java.lang.String, java.lang.String, java.lang.Integer, java.lang.Integer)) - Конструктор класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Конструктор класу, що ініціалізує значення всіх полів класу

[goodsID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html#goodsID) - Змінна класу model.[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html)

ID товару, що міститься в даному рядку таблиці кошика.

**I**

[IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

У шаблоні проектування "Міст" даний клас відіграє базовий інтерфейс реалізації функцій системи конфігурування.

[IConfigurator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html#IConfigurator()) - Конструктор класу Configurators.[IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html)

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html#ID) - Змінна класу model.[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html)

ID рядку кошика

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html#ID) - Змінна класу model.[Catalog](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html)

ID каталогу

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#ID) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення первинного ключа типу в реляційній таблиці.

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#ID) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає ID замовлення.

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#ID) - Змінна класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Поле зберігає ID замовлення.

[ID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#ID) - Змінна класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Поле зберігає значення ID облікового запису.

[imagePath](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#imagePath) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає шлях до зображення товару.

[info](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html#info) - Змінна класу panel.[ContactInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html)

[initComponents()](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AbstractView.html#initComponents()) - Method in interface View.[AbstractView](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AbstractView.html)

Метод інкапсулює логіку розміщення елементів графічного інтерфейсу.

[insert(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#insert(T)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод дозволяє додати нову строку даних, що інкапсульовані в об'єкті-параметрі, в реляційну таблицю.

[insert(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#insert(java.lang.String)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує вставку строки в базу даних

[insert(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#insert(T)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує вставлення нового запису, інкапсульованого у вхідному параметрі.

[insert(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#insert(java.lang.String)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує вставку строки в базу даних

[insert(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#insert(T)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує вставлення нового запису, інкапсульованого у вхідному параметрі.

[instancedViews](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#instancedViews) - Змінна класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

завантажені об'єкти-відображення

[IntegerConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html) - Клас в пакеті [utils](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\package-summary.html)

Клас реалізує об'єктно - реляційну конвертацію даних типу Integer.

[IntegerConverter()](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html#IntegerConverter()) - Конструктор класу utils.[IntegerConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html)

[isCellEditable(int, int)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html#isCellEditable(int, int)) - Метод класу model.[ApplicationTableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html)

[isEmptyField(JTextComponent)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html#isEmptyField(javax.swing.text.JTextComponent)) - Метод класу applicationValidators.[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html)

Метод перевіряє поле на наявність даних в ньому.

[isEmptyField(JTextField, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#isEmptyField(javax.swing.JTextField, java.lang.String)) - Метод класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

Перевіряє поле на наявність символів.

[isSemanticValid()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html#isSemanticValid()) - Метод класу applicationValidators.[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html)

Метод, який перевіряє введені дані в об'єкт-відображення на семантичну вірність

[isSemanticValid()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#isSemanticValid()) - Метод класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

Метод перевіряє поля форми на наявність символів та їх корректність (приклад корректності даних наведено в описі класу)

[isSemanticValid()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html#isSemanticValid()) - Метод класу applicationValidators.[LoginInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html)

Метод, який перевіряє введені дані в об'єкт-відображення на семантичну вірність

[isSemanticValid()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html#isSemanticValid()) - Метод класу applicationValidators.[RecoveryPasswordValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html)

Метод, який перевіряє введені дані в об'єкт-відображення на семантичну вірність

[isSemanticValid()](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html#isSemanticValid()) - Метод класу applicationValidators.[RegistrationPanelValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html)

Метод, який перевіряє введені дані в об'єкт-відображення на семантичну вірність

[isValide(Context, AbstractCommand)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Validator.html#isValide(app.Context, command.AbstractCommand)) - Static Метод класу app.[Validator](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Validator.html)

Статичний метод, що перевіряє відповідність даних, інкапсульованих в об'єкті контексті, вимогам об'єкту команди, що інкапсульовані в анотації [ContextAnnotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\ContextAnnotation.html).

**J**

[jdomConfig](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#jdomConfig) - Змінна класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

об'єкт класу [JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

Клас реалізує конфігурування системи за допомогою інструменту JDOM Принцип полягає в тому, що клас використовує даний інструмент для зчитування даних з файлу - конфігурації та запису пари "ключ - значення" в об'єкт класу Properties.

[JDOMConfigurator(Properties)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#JDOMConfigurator(java.util.Properties)) - Конструктор класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Конструктор классу, що ініціалізує значення поля [JDOMConfigurator.properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#properties).

[JDOMConfigurator(String, Properties)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#JDOMConfigurator(java.lang.String, java.util.Properties)) - Конструктор класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Конструктор класу, що ініціалізує поле [JDOMConfigurator.properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#properties) та викликає конструктор суперкласу [XMLConfigurator.XMLConfigurator(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html#XMLConfigurator(java.lang.String)) з метою ініціалізації поля[FileConfigurator.configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile).

**L**

[lastName](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#lastName) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля LASTNAME в реляційній таблиці.

[load()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#load()) - Метод класу Configurators.[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html)

Метод створений для конфігурування програмного додатку.

[load()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html#load()) - Метод класу Configurators.[IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html)

Метод створений для конфігурування програмного додатку.

[load()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#load()) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Метод створений для конфігурування програмного додатку.

[loadConfigure()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html#loadConfigure()) - Метод класу Configurators.[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

Метод виконує завантаження конфігурацій.

[loadConfigures(File)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#loadConfigures(java.io.File)) - Метод класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Метод завантажує конфігураційні дані з файлу.

[loadFromConfigs()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#loadFromConfigs()) - Метод класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

[login](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#login) - Змінна класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Строкове значення логіну облікового запису.

[login](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#login) - Змінна класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Поле зберігає значення логіна облікового запису

[LoginInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html) - Клас в пакеті [applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html)

Клас перевіряє дані, введені в об'єкт типу [LoginInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\LoginInformationPanel.html) на семантичну правильність.

[LoginInfoValidator(LoginInformationPanel)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html#LoginInfoValidator(panel.LoginInformationPanel)) - Конструктор класу applicationValidators.[LoginInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html)

Конструктор класу, що ініціалізує значення поля [LoginInfoValidator.panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html#panel)

[loginTextField](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RegistrationPanel.html#loginTextField) - Змінна класу panel.[RegistrationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RegistrationPanel.html)

[LoginValidationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\LoginValidationCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Клас здійснює перевірку на наявність даного пароля та логіна в базі даних.

[LoginValidationCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\LoginValidationCommand.html#LoginValidationCommand()) - Конструктор класу command.[LoginValidationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\LoginValidationCommand.html)

**M**

[main](file:///C:\Users\Wh15p3r\workspace\Course\doc\main\package-summary.html) - пакет main

[maker](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#maker) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає виробника товару.

[maker](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html#maker) - Змінна класу panel.[ConcreteGoodsPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html)

[mapToEntity(Class, Map<String, Object>)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html#mapToEntity(java.lang.Class, java.util.Map)) - Static Метод класу utils.[DAOAnnotationUtility](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\DAOAnnotationUtility.html)

Метод створює об'єкт класу, вказанного у параметрі, використовуючи реляційну таблицю, що представлена об'єктом класу Map.

[menuBar](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainFrame.html#menuBar) - Змінна класу View.[MainFrame](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainFrame.html)

[model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html) - пакет model

[model](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\CatalogPane.html#model) - Змінна класу View.[CatalogPane](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\CatalogPane.html)

**N**

[name](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html#name) - Змінна класу model.[Catalog](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Catalog.html)

Ім'я каталогу

[name](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#name) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля NAME в реляційній таблиці.

[name](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#name) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає назву товару.

[NewMain](file:///C:\Users\Wh15p3r\workspace\Course\doc\main\NewMain.html) - Клас в пакеті [main](file:///C:\Users\Wh15p3r\workspace\Course\doc\main\package-summary.html)

Клас містить головний метод програми.

[NewMain()](file:///C:\Users\Wh15p3r\workspace\Course\doc\main\NewMain.html#NewMain()) - Конструктор класу main.[NewMain](file:///C:\Users\Wh15p3r\workspace\Course\doc\main\NewMain.html)

[number](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#number) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає кількість товарів.

**O**

[open(DBConfiguration)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#open(app.DBConfiguration)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує підключення до бази даних.

[open(DBConfiguration)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#open(app.DBConfiguration)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує підключення до бази даних.

[open(DBConfiguration)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#open(app.DBConfiguration)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує підключення до бази даних.

[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас є об'єктним відображенням таблиці бази даних, що містить інформацію про замовлення.

[Order()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#Order()) - Конструктор класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Конструктор класу без параметрів.

[Order(Integer, String, String, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#Order(java.lang.Integer, java.lang.String, java.lang.String, java.lang.String)) - Конструктор класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Конструктор класу, що ініціалізує значення всіх полів класу

[OrderHistory](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\OrderHistory.html) - Клас в пакеті [panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\package-summary.html)

Клас створений з метою виводу інформації про замовлення користувача.

[OrderHistory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\OrderHistory.html#OrderHistory()) - Конструктор класу panel.[OrderHistory](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\OrderHistory.html)

Конструктор класу без параметрів.

[orderHistory1](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainPane.html#orderHistory1) - Змінна класу View.[MainPane](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\MainPane.html)

[orderState](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#orderState) - Змінна класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Поле зберігає статус замовлення.

**P**

[panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\package-summary.html) - пакет panel

[panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html#panel) - Змінна класу applicationValidators.[LoginInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\LoginInfoValidator.html)

Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

[panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html#panel) - Змінна класу applicationValidators.[RecoveryPasswordValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html)

Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

[panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html#panel) - Змінна класу applicationValidators.[RegistrationPanelValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html)

Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

[Parameter](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\Parameter.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Анотація інкапсулює мета-дані про зміну, що відбулась в результаті діяльності користувача.

[parseElement(Element, StringBuilder)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#parseElement(org.jdom2.Element, java.lang.StringBuilder)) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Метод заповнює об'экт, що зберігається в полі [JDOMConfigurator.properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#properties), парами "ключ - значення", де ключ - шлях, який включає в себе назви тегів та атрибутів XML.

[password](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#password) - Змінна класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Строкове значення паролю облікового запису.

[password](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#password) - Змінна класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Поле зберігає значення паролю облікового запису

[payment](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html#payment) - Змінна класу model.[Order](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Order.html)

Поле зберігає спосіб оплати замовлення.

[price](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html#price) - Змінна класу model.[Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html)

Поле зберігає ціну товару.

[Primary](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\Primary.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Анотація - мітка, якою позначається поле класу, що містить у собі первинний ключ.

[processError(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#processError(java.lang.String)) - Метод класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Метод виводить на екран інформаційне повідомлення з системною помилкою, опис якої вказаний у параметрі, після чого аварійно завершує програму.

[processError(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\LoginValidationCommand.html#processError(java.lang.String)) - Метод класу command.[LoginValidationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\LoginValidationCommand.html)

У разі виникнення помилки виводиться на екран інформаційне повідомлення з описом помилки, а потім видаляються дані з форми.

[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html) - Клас в пакеті [model](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\package-summary.html)

Клас є об'єктним відображенням таблиці в базі даних, що містить інформацію облікового запису.

[Profile()](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#Profile()) - Конструктор класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Конструктор класу без параметрів

[Profile(Integer, Integer, String, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html#Profile(java.lang.Integer, java.lang.Integer, java.lang.String, java.lang.String)) - Конструктор класу model.[Profile](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Profile.html)

Конструктор класу, що ініціалізує значення всіх полів класу

[profileID](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html#profileID) - Змінна класу model.[BasketItem](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\BasketItem.html)

ID облікового запису, що міститься в даному рядку таблиці кошика.

[properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#properties) - Змінна класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

[properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#properties) - Змінна класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

поле є сховищем даних типу "ключ - значення"

**Q**

[queue](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html#queue) - Змінна класу app.[Controller](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Controller.html)

об'єкт типу Executors, що інкапсулює в собі логіку реалізації схеми мультипоточності програмного додатку.

**R**

[read(Class<T>, Integer)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#read(java.lang.Class, java.lang.Integer)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод виконує зчитування рядку з бази даних, об'єктне представлення якої вказане у параметрі.

[read(Class<T>, Integer)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#read(java.lang.Class, java.lang.Integer)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує зчитування одного запису зі сховища (згідно з первинним ключем) та його відображення в об'єктному представленні.

[read(Class<T>, Integer)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#read(java.lang.Class, java.lang.Integer)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує зчитування одного запису зі сховища (згідно з первинним ключем) та його відображення в об'єктному представленні.

[readyToInstanceCommands](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html#readyToInstanceCommands) - Змінна класу app.[CommandFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CommandFactory.html)

Мапа, що містить в собі назви команд, готових до створення, та їх строкові значення

[readyToInstanceViews](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#readyToInstanceViews) - Змінна класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

готові до завантаження об'єкти-відображення.

[RecoveryPasswordCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда оновлює системні дані користувача, завантажуючи новий пароль до бази даних, якщо пройдена перевірка(логін та електронна пошта співпадають з даними у базі даних).

[RecoveryPasswordCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html#RecoveryPasswordCommand()) - Конструктор класу command.[RecoveryPasswordCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html)

Конструктор класу без параметрів.

[RecoveryPasswordCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html#RecoveryPasswordCommand(app.Context)) - Конструктор класу command.[RecoveryPasswordCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[RecoveryPasswordValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html) - Клас в пакеті [applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html)

Клас перевіряє дані, введені в об'єкт типу [RecoveryPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RecoveryPanel.html) на семантичну правильність.

[RecoveryPasswordValidator(RecoveryPanel)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html#RecoveryPasswordValidator(panel.RecoveryPanel)) - Конструктор класу applicationValidators.[RecoveryPasswordValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html)

Конструктор класу, що ініціалізує значення поля [RecoveryPasswordValidator.panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html#panel)

[RegistrationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RegistrationCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда реєструє нового користувача у системі, попередньо перевіривши його системні дані (логін, пароль) на унікальність у системі.

[RegistrationCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RegistrationCommand.html#RegistrationCommand()) - Конструктор класу command.[RegistrationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RegistrationCommand.html)

Конструктор класу без параметрів.

[RegistrationCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RegistrationCommand.html#RegistrationCommand(app.Context)) - Конструктор класу command.[RegistrationCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RegistrationCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[RegistrationPanelValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html) - Клас в пакеті [applicationValidators](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\package-summary.html)

Клас перевіряє дані, введені в об'єкти типів [RegistrationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RegistrationPanel.html) та [ContactInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ContactInformationPanel.html), що є агрегатом останнього, на семантичну правильність.

[RegistrationPanelValidator(RegistrationPanel)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html#RegistrationPanelValidator(panel.RegistrationPanel)) - Конструктор класу applicationValidators.[RegistrationPanelValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html)

Конструктор класу, що ініціалізує значення поля [RegistrationPanelValidator.panel](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html#panel)

[RequestException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exception\RequestException.html) - Клас виключення в  [exception](file:///C:\Users\Wh15p3r\workspace\Course\doc\exception\package-summary.html)

Викликається при системній помилці в результаті виконання запиту.

[resetRequest()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html#resetRequest()) - Метод класу command.[RecoveryPasswordCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\RecoveryPasswordCommand.html)

Викликається, якщо перевірка логіна те електронної пошти не пройшла успішно.

[resultSetToTable(ResultSet)](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html#resultSetToTable(java.sql.ResultSet)) - Метод класу model.[ApplicationTableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html)

**S**

[save()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html#save()) - Метод класу Configurators.[IConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\IConfigurator.html)

Метод створений для збереження конфігураційних даних програмного додатку.

[save()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#save()) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Метод створений для збереження конфігураційних даних програмного додатку.

[saveConfigure()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html#saveConfigure()) - Метод класу Configurators.[AbstractConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\AbstractConfigurator.html)

Метод виконує запис конфігураційних даних.

[select(Class<T>, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#select(java.lang.Class, java.lang.String)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує вибірку масиву даних з таблиці за допомогою строкової команди, що записана у вхідному параметрі, та перетворення цих даних в об'єктне представлення, тип якого вказаний у вхідному параметрі.

[select(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#select(java.lang.String)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод забезпечує вибірку масиву даних з бази даних.

[select(Class<T>, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#select(java.lang.Class, java.lang.String)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує вибірку масиву даних з таблиці за допомогою строкової команди, що записана у вхідному параметрі, та перетворення цих даних в об'єктне представлення, тип якого вказаний у вхідному параметрі.

[select(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#select(java.lang.String)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод забезпечує вибірку масиву даних з бази даних.

[select(Class<T>, String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#select(java.lang.Class, java.lang.String)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод забезпечує вибірку масиву даних з таблиці за допомогою строкової команди, що записана у вхідному параметрі, та перетворення цих даних в об'єктне представлення, тип якого вказаний у вхідному параметрі.

[setContactInfoPanel(ContactInformationPanel)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html#setContactInfoPanel(panel.ContactInformationPanel)) - Метод класу applicationValidators.[ContactInfoValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ContactInfoValidator.html)

[setProperties(Properties)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#setProperties(java.util.Properties)) - Метод класу Configurators.[JDOMConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html)

Встановлює значення поля [JDOMConfigurator.properties](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\JDOMConfigurator.html#properties)

[showOKMessage(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html#showOKMessage(java.lang.String)) - Static Метод класу global.[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html)

[showSystemErrorMessage(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html#showSystemErrorMessage(java.lang.String)) - Static Метод класу global.[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html)

[showWarningmessage(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html#showWarningmessage(java.lang.String)) - Static Метод класу global.[GlobalContext](file:///C:\Users\Wh15p3r\workspace\Course\doc\global\GlobalContext.html)

[spinnerModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html#spinnerModel) - Змінна класу panel.[ConcreteGoodsPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\ConcreteGoodsPanel.html)

[storeConfigures()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#storeConfigures()) - Метод класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Метод зберігає конфігураційні дані в зовнішній файл.

[Stored](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\Stored.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Дана анотація використовується для реалізації об'єктно - реляційної моделі даних.

[StringConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html) - Клас в пакеті [utils](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\package-summary.html)

Клас реалізує об'єктно - реляційну конвертацію строкових даних.

[StringConverter()](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html#StringConverter()) - Конструктор класу utils.[StringConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html)

**T**

[tableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\BasketPane.html#tableModel) - Змінна класу View.[BasketPane](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\BasketPane.html)

Зберігає значення [ApplicationTableModel](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ApplicationTableModel.html)

[telephone](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html#telephone) - Змінна класу model.[ContactInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\ContactInformation.html)

Значення поля TELEPHONE в реляційній таблиці.

[toString(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html#toString(T)) - Метод класу utils.[IntegerConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html)

Метод забезпечує конвертацію об'єктних даних в строкові.

[toString(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html#toString(T)) - Метод класу utils.[StringConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html)

Метод забезпечує конвертацію об'єктних даних в строкові.

[toString(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html#toString(T)) - Method in interface utils.[ValueConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html)

Метод забезпечує конвертацію об'єктних даних в строкові.

[toValue(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html#toValue(java.lang.String)) - Метод класу utils.[IntegerConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\IntegerConverter.html)

Метод забезпечує конвертацію строкових даних в об'єктні.

[toValue(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html#toValue(java.lang.String)) - Метод класу utils.[StringConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\StringConverter.html)

Метод забезпечує конвертацію строкових даних в об'єктні.

[toValue(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html#toValue(java.lang.String)) - Method in interface utils.[ValueConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html)

Метод забезпечує конвертацію строкових даних в об'єктні.

[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас є реалізацією - заглушкою інтерфейсу [CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html), створений з метою зменшення дублювання коду та надає можливість дочірнім класам перевизначати лише ті методи, які необхідні для специфічних реалізацій функціональностей взаємодії з персистентними сховищами.

[TransparendCRUD()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#TransparendCRUD()) - Конструктор класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

[trimField(JTextComponent)](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html#trimField(javax.swing.text.JTextComponent)) - Метод класу applicationValidators.[ApplicationValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\ApplicationValidator.html)

Метод обрізає пусті символи спочатку та кінця поля.

**U**

[update(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html#update(T)) - Метод класу app.[ApplicationCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\ApplicationCRUD.html)

Метод виконує оновлення існуючих даних в рядку бази даних, що відповідає первинному ключу об'єкту-параметру.

[update(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html#update(T)) - Method in interface app.[CRUDInterface](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\CRUDInterface.html)

Метод запезпечує оновлення запису в сховищі, що відповідає об'єктному представленню вхідного параметру.

[update(T)](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html#update(T)) - Метод класу app.[TransparendCRUD](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\TransparendCRUD.html)

Метод запезпечує оновлення запису в сховищі, що відповідає об'єктному представленню вхідного параметру.

[UpdateBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateBasketCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда оновлює вкладку 'Кошик', зчитавши необхідні дані з бази даних.

[UpdateBasketCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateBasketCommand.html#UpdateBasketCommand()) - Конструктор класу command.[UpdateBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateBasketCommand.html)

Конструктор класу без параметрів.

[UpdateBasketCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateBasketCommand.html#UpdateBasketCommand(app.Context)) - Конструктор класу command.[UpdateBasketCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateBasketCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[UpdateCatalogNodeCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateCatalogNodeCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда оновлює вузол дерева каталогу, зчитавши необхідні дані з бази даних

[UpdateCatalogNodeCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateCatalogNodeCommand.html#UpdateCatalogNodeCommand()) - Конструктор класу command.[UpdateCatalogNodeCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateCatalogNodeCommand.html)

Конструктор класу без параметрів.

[UpdateCatalogNodeCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateCatalogNodeCommand.html#UpdateCatalogNodeCommand(app.Context)) - Конструктор класу command.[UpdateCatalogNodeCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateCatalogNodeCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[UpdateConcreteGoodsCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateConcreteGoodsCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда зчитує дані з бази даних про конкретний об'єкт типу [Goods](file:///C:\Users\Wh15p3r\workspace\Course\doc\model\Goods.html).

[UpdateConcreteGoodsCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateConcreteGoodsCommand.html#UpdateConcreteGoodsCommand()) - Конструктор класу command.[UpdateConcreteGoodsCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateConcreteGoodsCommand.html)

Конструктор класу без параметрів.

[UpdateConcreteGoodsCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateConcreteGoodsCommand.html#UpdateConcreteGoodsCommand(app.Context)) - Конструктор класу command.[UpdateConcreteGoodsCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateConcreteGoodsCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[updateConfigureFiles(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html#updateConfigureFiles(java.lang.String)) - Метод класу Configurators.[ConfigFacade](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\ConfigFacade.html)

Метод завантажує список конфігураційних файлів.

[UpdateContactInfoCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateContactInfoCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда оновлює в базі даних контактну інформацію користувача згідно з даними, переданими в об'єкті типу [Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html)

[UpdateContactInfoCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateContactInfoCommand.html#UpdateContactInfoCommand()) - Конструктор класу command.[UpdateContactInfoCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateContactInfoCommand.html)

Конструктор класу без параметрів.

[UpdateContactInfoCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateContactInfoCommand.html#UpdateContactInfoCommand(app.Context)) - Конструктор класу command.[UpdateContactInfoCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateContactInfoCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[UpdateLoginInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateLoginInformation.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда оновлює в базі даних інформацію облікового запису.

[UpdateLoginInformation()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateLoginInformation.html#UpdateLoginInformation()) - Конструктор класу command.[UpdateLoginInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateLoginInformation.html)

Конструктор класу без параметрів.

[UpdateLoginInformation(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateLoginInformation.html#UpdateLoginInformation(app.Context)) - Конструктор класу command.[UpdateLoginInformation](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateLoginInformation.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[UpdateOrderHistoryCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateOrderHistoryCommand.html) - Клас в пакеті [command](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\package-summary.html)

Команда завантажує історію замовлень користувача, після чого оновлює об'єкти-відображення, задекларовані в анотації [View](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\View.html) та передає ним в якості параметру об'єкт ResultSet

[UpdateOrderHistoryCommand()](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateOrderHistoryCommand.html#UpdateOrderHistoryCommand()) - Конструктор класу command.[UpdateOrderHistoryCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateOrderHistoryCommand.html)

Конструктор класу без параметрів.

[UpdateOrderHistoryCommand(Context)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateOrderHistoryCommand.html#UpdateOrderHistoryCommand(app.Context)) - Конструктор класу command.[UpdateOrderHistoryCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\UpdateOrderHistoryCommand.html)

Конструктор класу, що ініціалізує значення поля [AbstractCommand.context](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#context)

[updateViewList](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#updateViewList) - Змінна класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Список об'єктів-відображень, що мають бути сповіщені про зміни в моделі

[updateViews(Object...)](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html#updateViews(java.lang.Object...)) - Метод класу command.[AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html)

Метод посилає запит на оновлення до об'єктів-відображень.

[URL](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html#URL) - Змінна класу app.[DBConfiguration](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\DBConfiguration.html)

Строкове значення URL бази даних.

[utils](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\package-summary.html) - пакет utils

**V**

[ValidationException](file:///C:\Users\Wh15p3r\workspace\Course\doc\exception\ValidationException.html) - Клас виключення в  [exception](file:///C:\Users\Wh15p3r\workspace\Course\doc\exception\package-summary.html)

Помилка при валідації даних.

[Validator](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Validator.html) - Клас в пакеті [app](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\package-summary.html)

Клас виконує перевірку на правильність конфігурування об'єкту-команди [AbstractCommand](file:///C:\Users\Wh15p3r\workspace\Course\doc\command\AbstractCommand.html) об'єктом-контекстом [Context](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Context.html).

[Validator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Validator.html#Validator()) - Конструктор класу app.[Validator](file:///C:\Users\Wh15p3r\workspace\Course\doc\app\Validator.html)

[validator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#validator()) - Метод класу Configurators.[FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html)

Метод перевіряє файл, записаний в поле [FileConfigurator.configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile), на специфічні умови конкретних реалізацій системи конфігурацій.

[validator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html#validator()) - Метод класу Configurators.[XMLConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html)

Метод перевіряє файл, записаний в поле [FileConfigurator.configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile), на специфічні умови конкретних реалізацій системи конфігурацій.

[validator](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RecoveryPanel.html#validator) - Змінна класу panel.[RecoveryPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RecoveryPanel.html)

об'єкт типу [RecoveryPasswordValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RecoveryPasswordValidator.html).

[validator](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RegistrationPanel.html#validator) - Змінна класу panel.[RegistrationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\RegistrationPanel.html)

об'єкт типу [RegistrationPanelValidator](file:///C:\Users\Wh15p3r\workspace\Course\doc\applicationValidators\RegistrationPanelValidator.html)

[ValueConverter](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\ValueConverter.html) - Interface in [utils](file:///C:\Users\Wh15p3r\workspace\Course\doc\utils\package-summary.html)

Інтерфейс задає правила для конвертації об'єкту довільного типу зі строкового представлення в об'єктне.

[View](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\package-summary.html) - пакет View

[View](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\View.html) - Annotation Type in [annotation](file:///C:\Users\Wh15p3r\workspace\Course\doc\annotation\package-summary.html)

Анотація надає мета-дані об'єктам-командам про об'єкти-відображення, які мають бути оновлені у результаті діяльності об'єкту-команди.

[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html) - Клас в пакеті [View](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\package-summary.html)

Основне завдання даного класу є повернення об'єкту типу [AbstractView](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\AbstractView.html) за ключем, що передається як параметр.

[ViewFactory()](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html#ViewFactory()) - Конструктор класу View.[ViewFactory](file:///C:\Users\Wh15p3r\workspace\Course\doc\View\ViewFactory.html)

Клас, що ініціалізує значення полів та завантажує конфігураційні дані для нього.

**X**

[XMLConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html) - Клас в пакеті [Configurators](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\package-summary.html)

Клас розширює можливості [FileConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html) та інкапсулює логіку реалізації конфігурування за допомогою мови XML.Згідно з шаблоном проектування "Міст" клас виконує роль елемента ієрархії реалізації абстракції.

[XMLConfigurator()](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html#XMLConfigurator()) - Конструктор класу Configurators.[XMLConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html)

Конструктор без параметрів, викликає конструктор без параметрів суперкласу.

[XMLConfigurator(String)](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html#XMLConfigurator(java.lang.String)) - Конструктор класу Configurators.[XMLConfigurator](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\XMLConfigurator.html)

Конструктор, який ініціалізує значення поля [FileConfigurator.configFile](file:///C:\Users\Wh15p3r\workspace\Course\doc\Configurators\FileConfigurator.html#configFile), використовуючи строковий параметр

**С**

[СorrectPassField](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\LoginInformationPanel.html#СorrectPassField) - Змінна класу panel.[LoginInformationPanel](file:///C:\Users\Wh15p3r\workspace\Course\doc\panel\LoginInformationPanel.html)

* 1. Інструкції для користувача

Щоб використовувати функціонал програмного додатку, необхідно авторизуватись у системі. Це можна зробити декількома шляхами:

1. Увійшовши до системи під своїм логіном та паролем. Необхідно ввести дані в поля панелі ,,Вхід’’ та натиснути кнопку ,,Увійти’’.
2. Зареєструвавшись у системі. Після запуску програмного додатку натиснути кнопку ,,Реєстрація’’ ввести дані до реєстраційної форми та підтвердити запит, натиснувши кнопку ,,Реєстрація’’.

Для того, щоб відновити пароль від облікового запису, необхідно після запуску програмного додатку натиснути кнопку ,,Відновити пароль’’, заповнити форму відновлення паролю та підтвердити запит, натиснувши кнопку ,,Відновити пароль’’. Після успішного відновлення паролю буде відображена панель входу до ситеми.

Після авторизації у системі на головній вкладці буде відображена контактна інформація власника облікового запису та історія його замовлень з поточним етапом замовлення.

Якщо необхідно редагувати контактну інформацію та інформацію облікового запису, необхідно зайти в меню та натиснути пункт меню ,,Редагувати дані’’. Після цього необхідно ввести дані в текстові поля та натиснути кнопку ,,Редагувати’’. У тому випадку, якщо дані, введені в поля, не коректні (наприклад, якщо поля ,,Новий пароль’’ та ,,Підтвердження паролю’’ не співпадають) система відобразить інформаційне повідомлення, у якому буде зазначено причину помилки – необхідно виправити дані та повторити запит.

Для перегляду вмісту каталогу товарів необхідно зайти в меню та натиснути пункт меню ,,До каталогу’’. Після цього буде відкрита вкладка ,,Каталог’’ , на якій можна переглянути вміст каталогу, що буде відображений зліва у вигляді дерева даних. Дійшовши до листкового вузла і клацнувши на ньому, буде відображена інформація про даний товар та можливість додати до кошика певну його кількість (кількість товару можна налаштувати за допомогою лічильника).

Щоб повернутися на головну сторінку та оновити дані на ній, необхідно зайти в меню та натиснути пункт меню ,,На головну’’.

Для перегляду вмісту кошика з метою його корегування чи подальшого оформлення замовлення, необхідно зайти в меню та натиснути пункт меню ,,Кошик’’. Після цього будуть відображені вкладка з таблицею, вмістом якої є основна інформація про доданий товар, та кнопки, що дозволяють видалити товар з кошика чи перейти до оформлення замовлення.

Щоб оформити замовлення, необхідно натиснути кнопку ,,Оформити замовлення’’ вкладки ,,Замовлення’’ (кнопка стане активною лише коли кошик має хоча б один товар) та заповнити форму, що відображає деталі оформлення замовлення. Після заповнення форми необхідно натиснути кнопку ,,Оформити’’, після чого буде відображене інформаційне вікно з результатом оформлення замовлення.

Для перегляду інформації про програму та автора необхідно зайти в меню та натиснути пункт меню ,,Про програму’’. У цьому випадку буде відображене вікно з інформацією про автора та програму.

Для виходу із системи необхідно зайти в меню та натиснути пункт меню ,,Вихід’’. У цьому випадку будуть закриті усі вкладки та відображена вкладка ,,Авторизація’’ з панеллю ,,Вхід’’.

4.ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

Реалізація прецеденту реєстрації в системі зображено на рисунках 4.1,4.2.

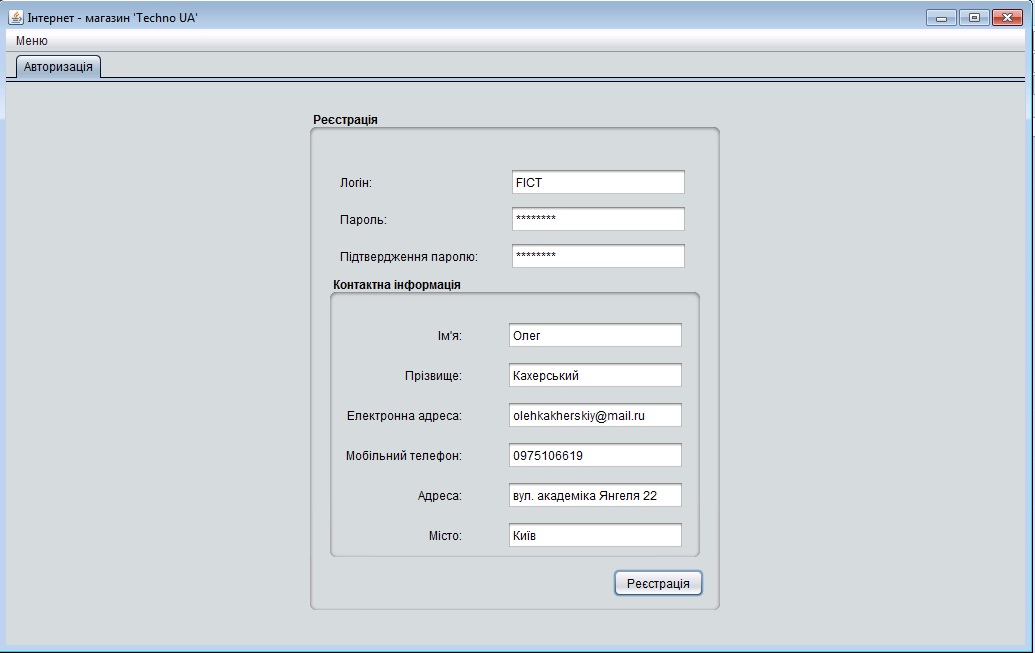


Рис.4.1 Заповнення реєстраційної форми

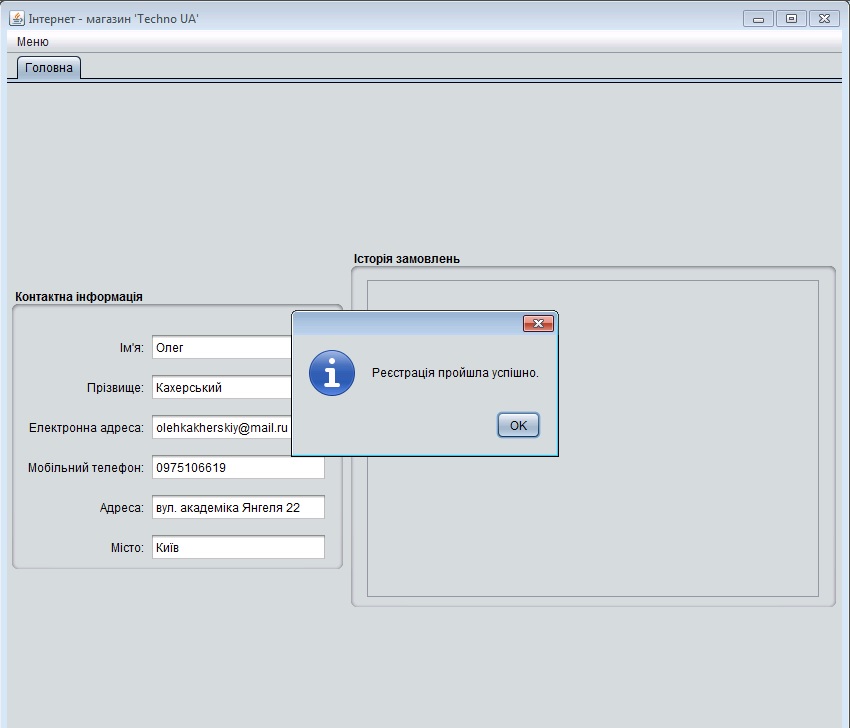


Рис.4.2 Кінець прецеденту реєстрації

Реалізація прецеденту відновлення паролю облікового запису продемонстровано на рисунках 4.3, 4.4.

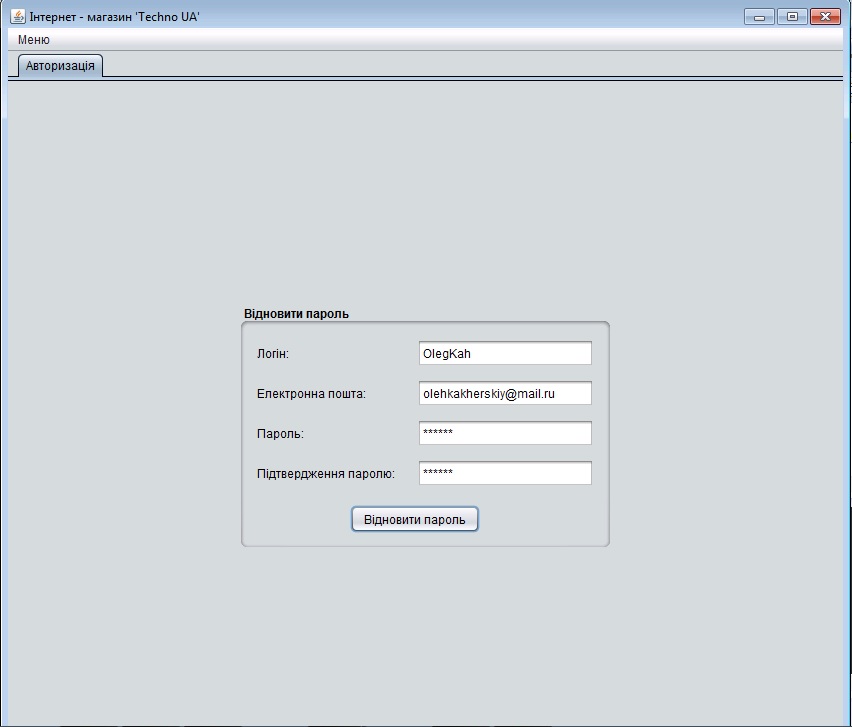


Рис. 4.3 Заповнення форми відновлення паролю

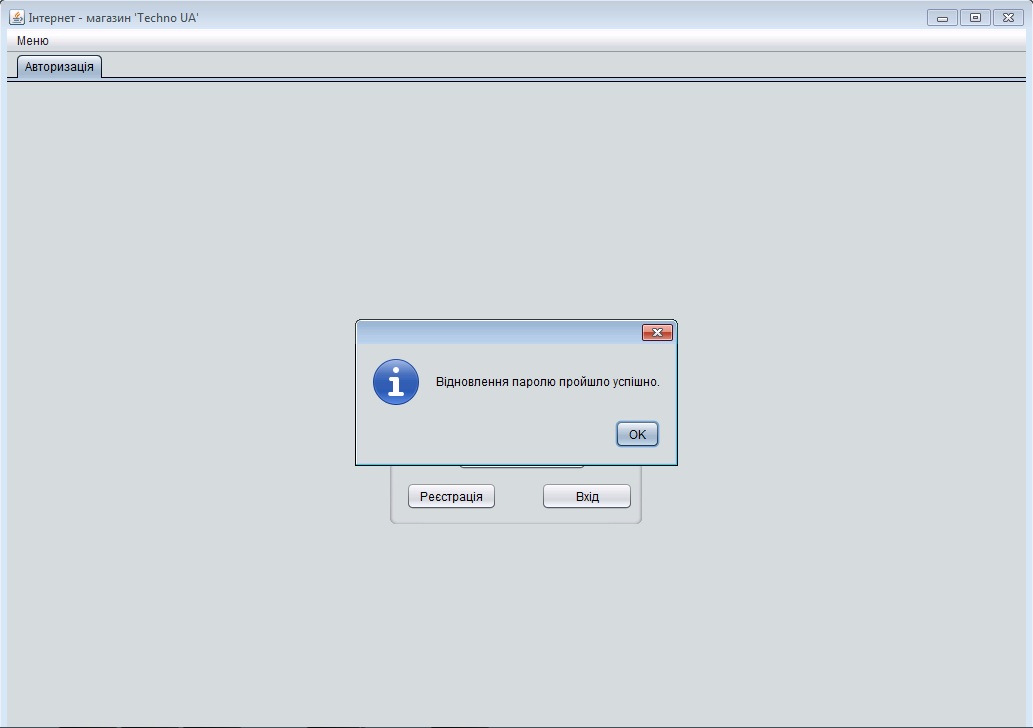


Рис.4.4 Кінець прецеденту відновлення паролю

Прецедент входу в систему зображено на рисунках 4.5, 4.6

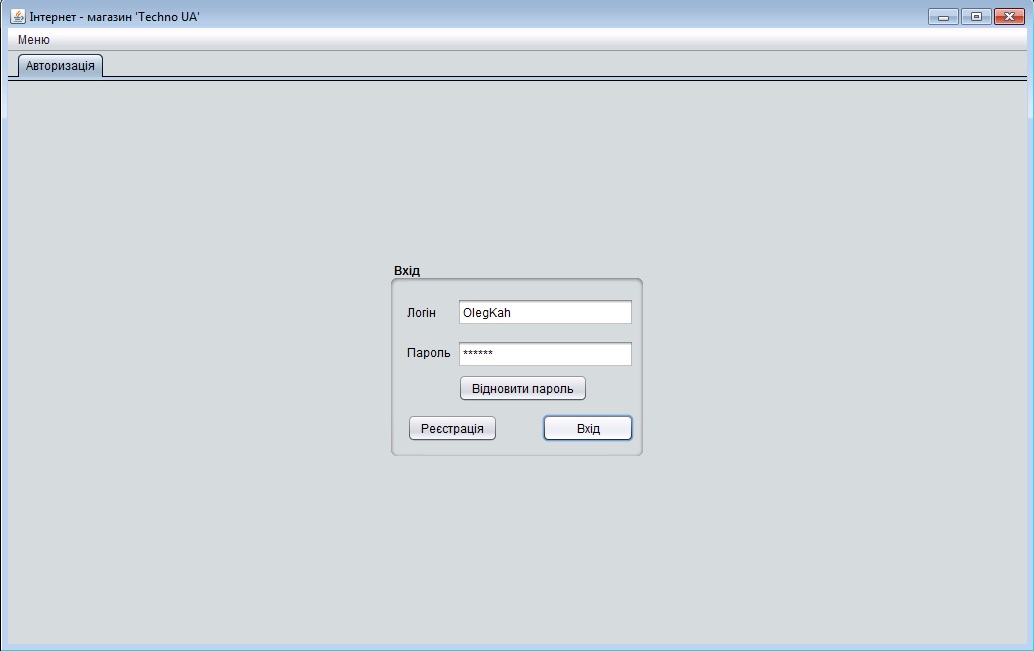


Рис.4.5 Заповненя форми входу в систему

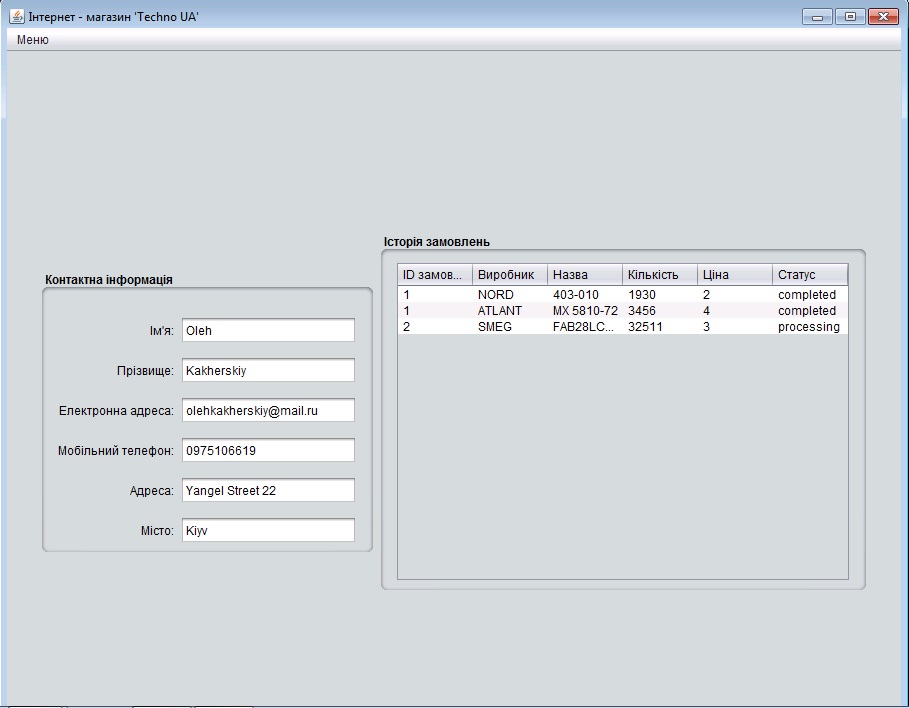


Рис.4.6 Результат завершення прецеденту входу в систему

Прецедент перегляду інформації про програму зображено на рисунку 4.7

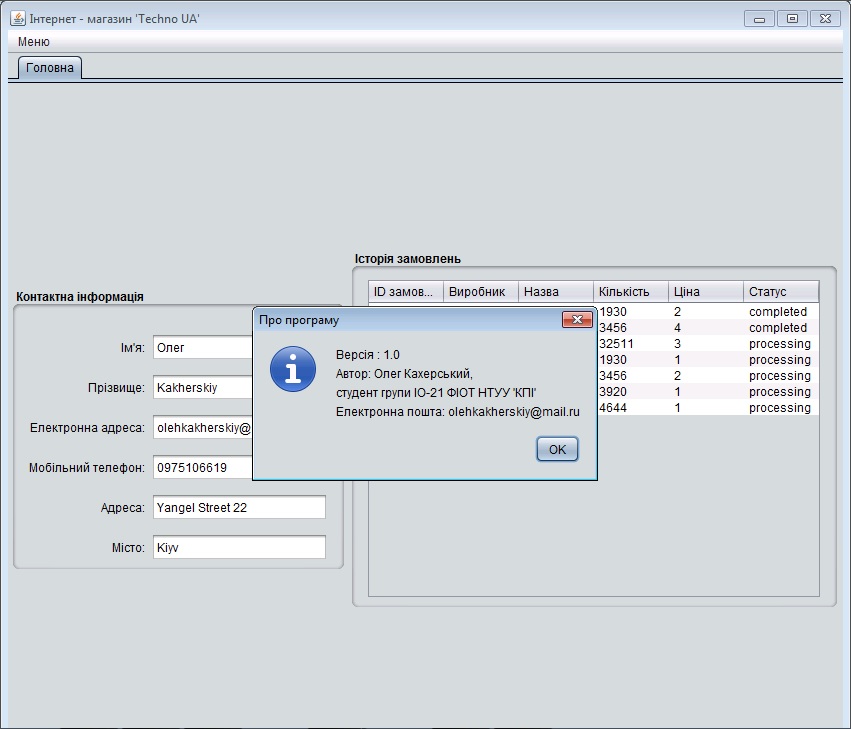


Рис.4.7 Відображення інформації про програму

Реалізацію прецеденту перегляду вмісту каталогу продемонстровано на рисунку 4.8.

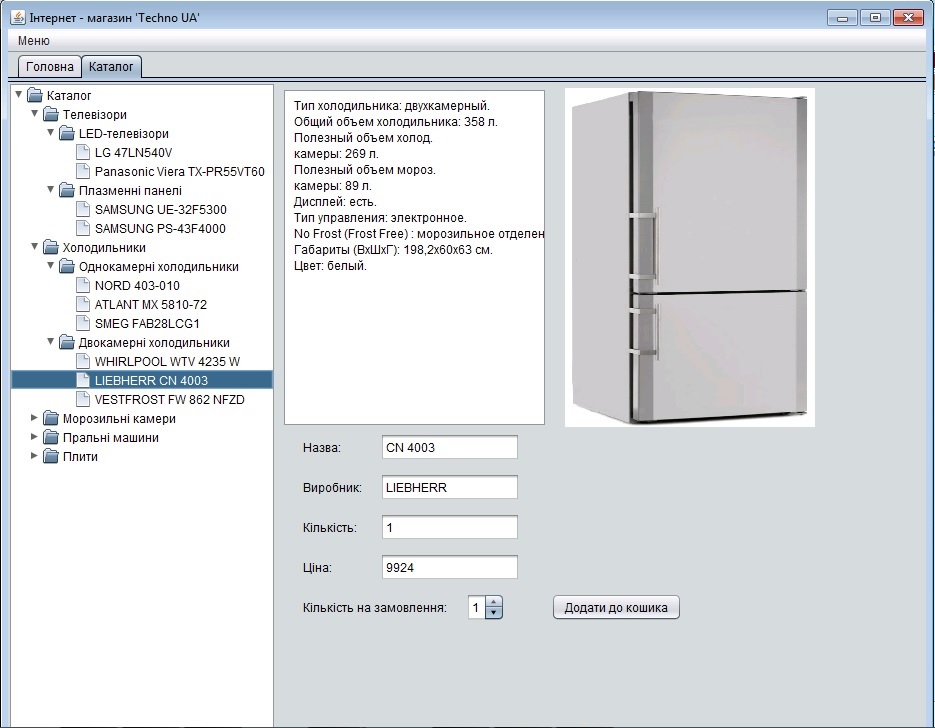


Рис.4.8 Вміст каталогу та інформація про конкретний товар

Прецедент редагування даних відображено на рисунках 4.9, 4.10

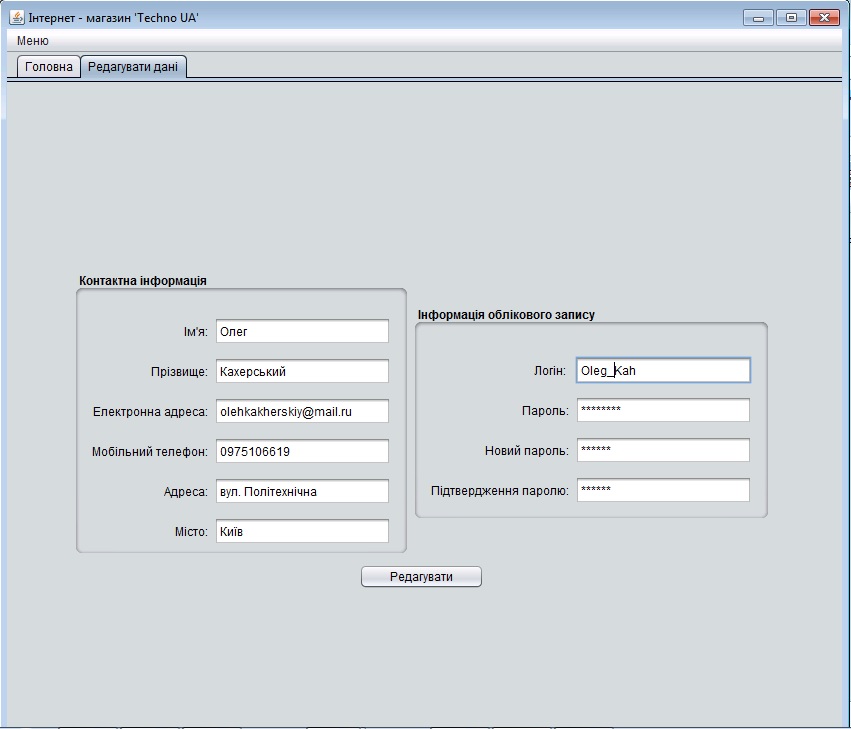


Рис.4.9 Заповнення форм редагування даних

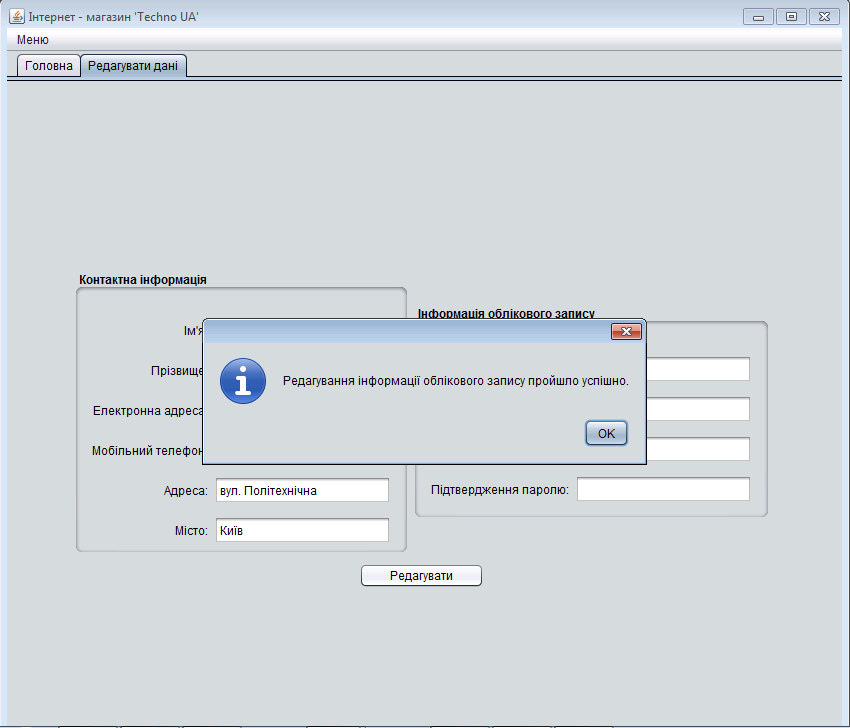


Рис.4.10 Завершення прецеденту редагування даних

Реалізацію прецедентів повернення на головну сторінку, перегляду історії замовлень зображено на рисунках 4.11, 4.12.

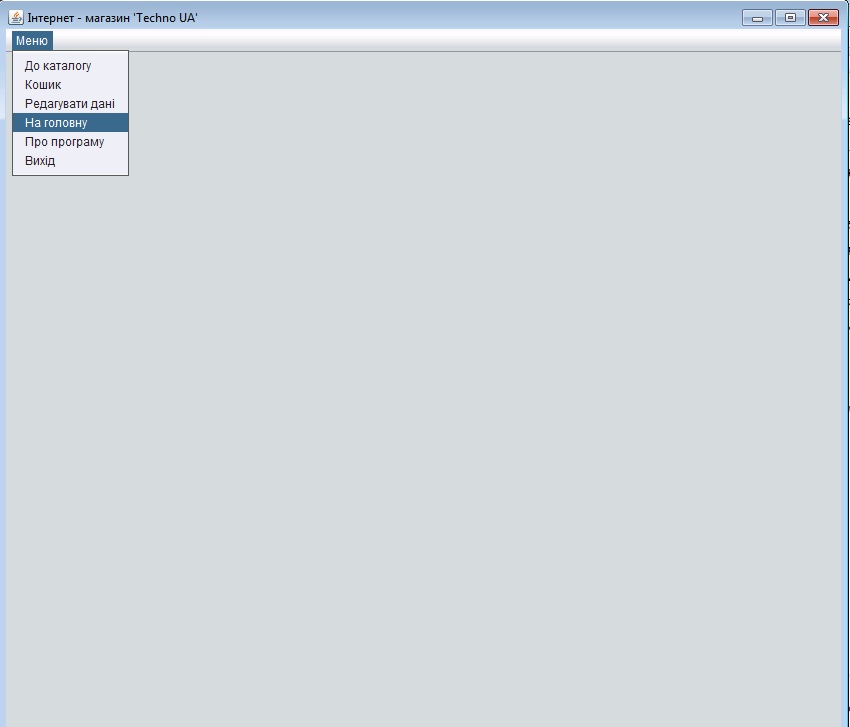


Рис. 4.11 Початок прецедентів

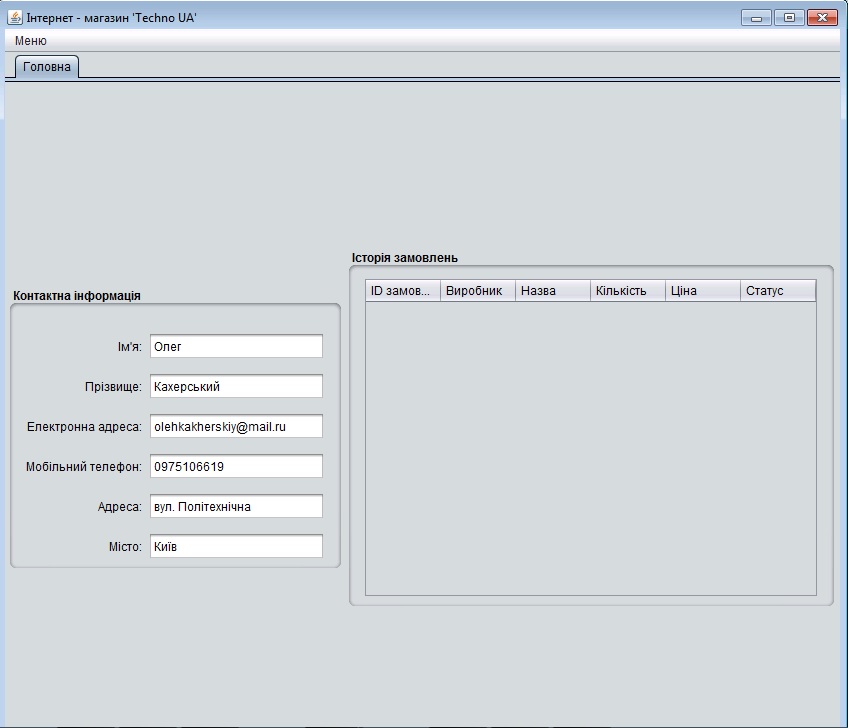


Рис.4.12 Завершення прецедентів повернення на головну сторінку та перегляду історії замовлень

Прецедент оформлення замовлення відображено на рисунках 4.13-4.15.

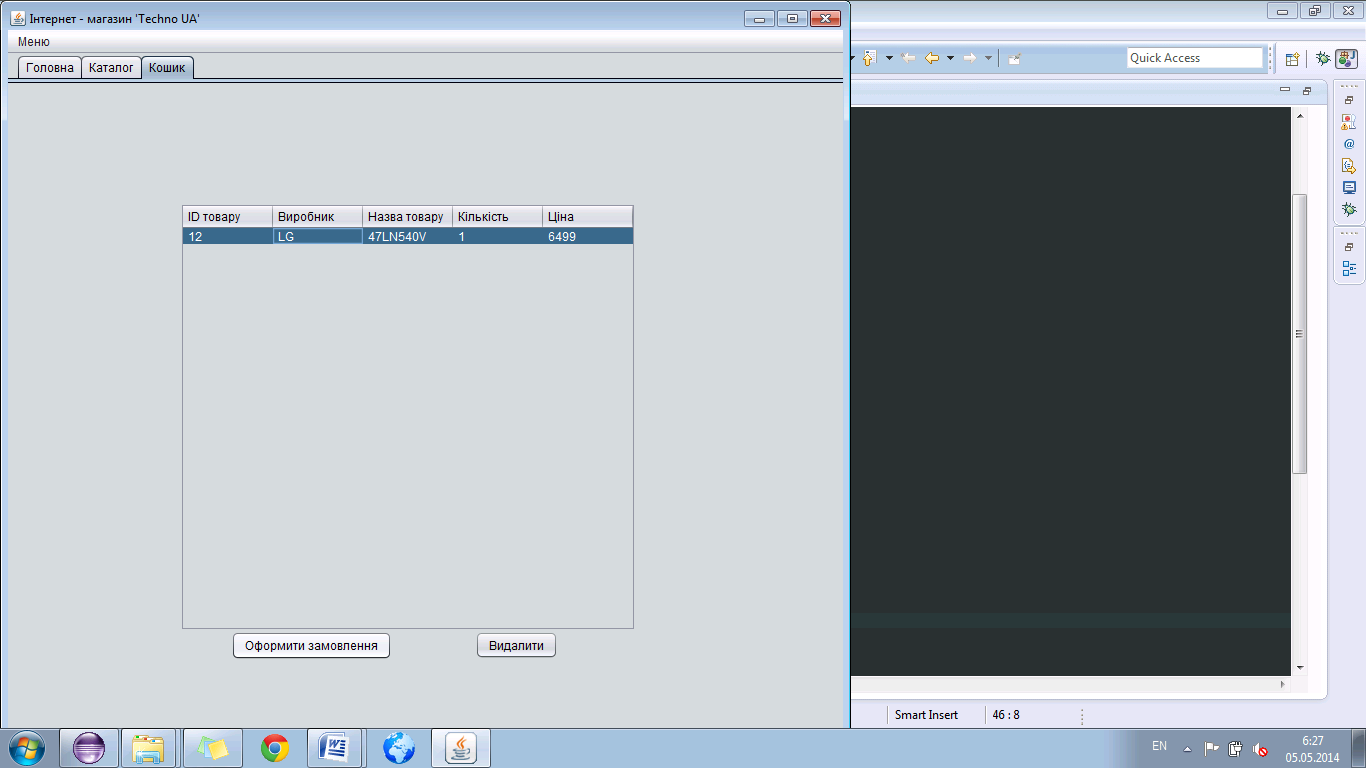


Рис.4.13 Перегляд вмісту кошика

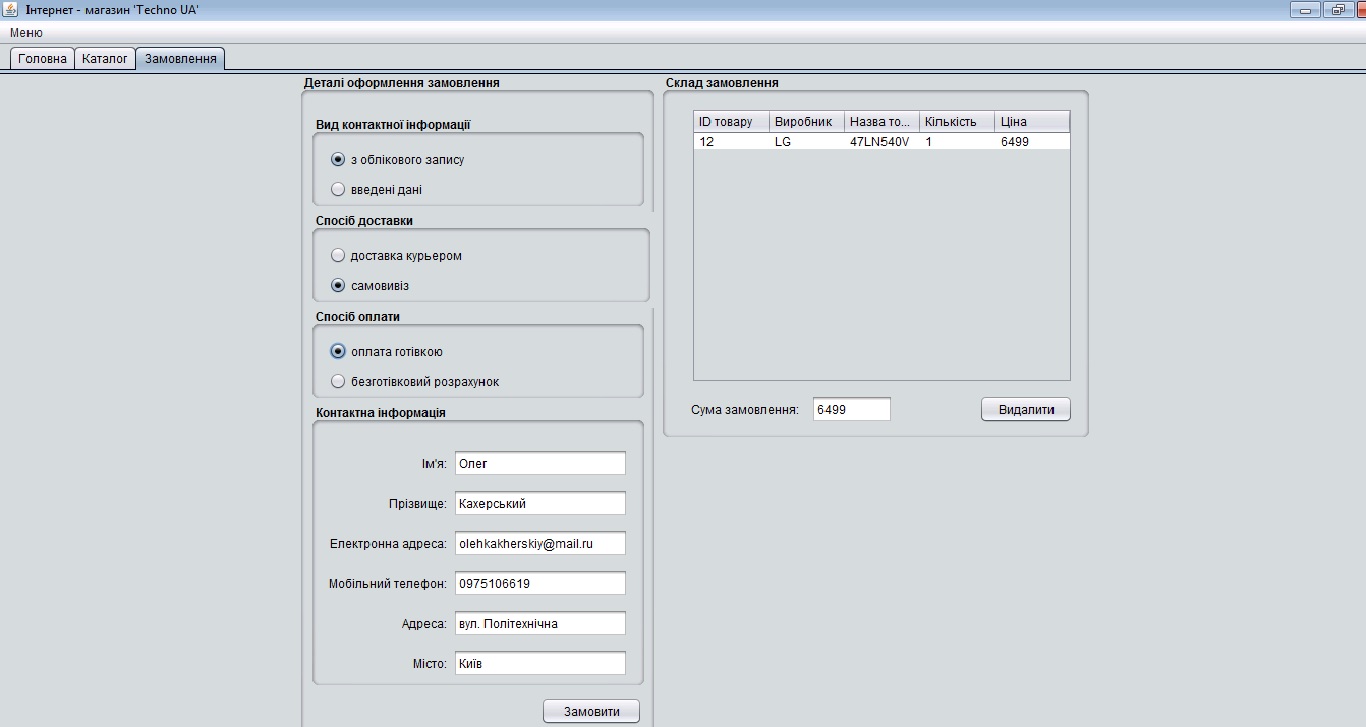


Рис.4.14 Заповнення форми оформлення замовлення

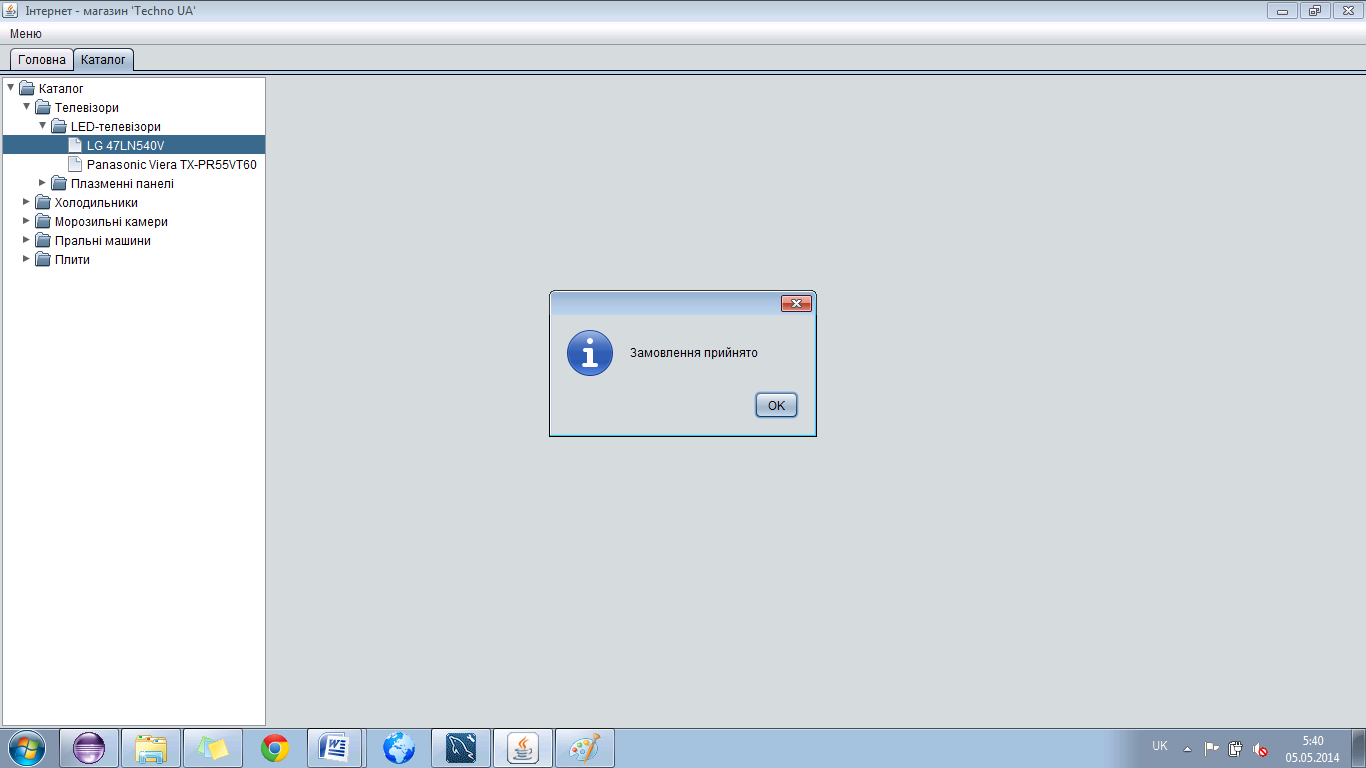


Рис.4.15 Завершення прецеденту оформлення замовлення

Під час тестування програмного додатку <<Інтернет – магазин ,,TechnoUA’’>> не виявлено помилок чи зависань. Програма працює коректно на всіх етапах роботи. Всі прецеденти повністю реалізовано. Тестування програмного додатку пройдено успішно.

ВИСНОВОК

Додаток А. Таблиця відповідності елементів дизайну

Таблиця 2.1. Відповідність елементів інтерфейсу до класів

|  |  |
| --- | --- |
| Елемент інтерфейсу | Класи пакету java.swing |
| Головне вікно | java.swing .JFrame |
| Діалогове вікно ,,Про програму’’ | java.swing .JDialog |
| Панель вкладок | java.swing .JTabbedPane |
| Вкладка ,, Авторизація’’ | java.swing .JPanel |
| Вкладка ,, Головна’’ | java.swing .JPanel |
| Вкладка ,,Кошик’’ | java.swing .JPanel |
| Вкладка ,,Каталог товарів’’ | java.swing .JPanel |
| Вкладка ,,Замовлення’’ | java.swing .JPanel |
| Вкладка ,,Редагування’’ | java.swing .JPanel |
| Панель ,,Меню | java.swing .JMenuBar |
| Пункт меню ,,До каталогу’’ | java.swing .JMenuItem |
| Пункт меню ,,На головну’’ | java.swing .JMenuItem |
| Пункт меню ,,Кошик’’ | java.swing .JMenuItem |
| Пункт меню ,,Редагувати дані’’ | java.swing .JMenuItem |
| Пункт меню ,,Про програму’’ | java.swing .JMenuItem |
| Пункт меню ,,Вихід’’ | java.swing .JMenuItem |
| Панель реєстрації | java.swing.JPanel |
| Панель входу в систему | java.swing.JPanel |
| Панель відновлення паролю | java.swing.JPanel |
| Панель дерева каталогу | java.swing.JPanel |
| Панель каталогу | java.swing.JPanel |
| Панель інформації облікового запису | java.swing.JPanel |
| Панель історії замовлень | java.swing.JPanel |
| Панель складу замовлення | java.swing.JPanel |
| Панель контактної інформації | java.swing.JPanel |
| Елемент ,,Версія’’ | java.swing.JLabel |
| Елемент ,,Автор’’ | java.swing.JLabel |
| Елемент ,,Електронна пошта’’ | java.swing.JLabel |
| Елемент ,,Cума’’ | java.swing.JLabel |
| Кнопка ,,ОК’’ | java.swing.JButton |
| Кнопка ,,Відновити пароль’’ | java.swing.JButton |
| Кнопка ,,Реєстрація’’ | java.swing.JButton |
| Кнопка ,,Зареєструватися’’ | java.swing.JButton |
| Кнопка ,,Редагувати’’ | java.swing.JButton |
| Кнопка ,,Увійти’’ | java.swing.JButton |
| Кнопка ,, Оформити’’ | java.swing.JButton |
| Кнопка ,, Замовити’’ | java.swing.JButton |
| Кнопка ,, Видалити’’ | java.swing.JButton |
| Поле ,, Логін’’ | java.swing.JTextField |
| Поле ,,Пароль’’ | java.swing. JPasswordField, JTextField |
| Поле ,,Ім’я ’’ | java.swing.JTextField |
| Поле ,,Підтвердження паролю ’’ | java.swing.JTextField |
| Поле ,,Новий пароль ’’ | java.swing.JTextField |
| Поле ,,Електронна пошта’’ | java.swing.JTextField |
| Поле ,,Прізвище’’ | java.swing.JTextField |
| Поле ,, Мобільний телефон’’ | java.swing.JTextField |
| Поле ,, Адреса’’ | java.swing.JTextField |
| Поле ,, Місто’’ | java.swing.JTextField |
| Елемент ,,Доставка кур'єром’’ | java.swing.JRadioButton |
| Елемент ,,Самовивіз’’ | java.swing.JRadioButton |
| Елемент ,,Безготівковий розрахунок’’ | java.swing.JRadioButton |
| Елемент ,,Оплата готівкою’’ | java.swing.JRadioButton |
| Елемент ,,Введені дані’’ | java.swing.JRadioButton |
| Елемент ,,З облікового запису’’ | java.swing.JRadioButton |
| Рамка ,,Вхід’’ | javax.swing.border.TitledBorder |
| Рамка ,,Контактна інформація’’ | javax.swing.border.TitledBorder |
| Рамка ,,Реєстрація’’ | javax.swing.border.TitledBorder |
| Рамка ,,Відновлення паролю’’ | javax.swing.border.TitledBorder |
| Рамка ,,Склад замовлення’’ | javax.swing.border.TitledBorder |
| Рамка ,,Вид контактної інформації’’ | javax.swing.border.TitledBorder |
| Рамка ,,Спосіб оплати’’ | javax.swing.border.TitledBorder |
| Рамка ,,Спосіб доставки’’ | javax.swing.border.TitledBorder |
| Рамка ,,Деталі оформлення замовлення’’ | javax.swing.border.TitledBorder |
| Рамка з назвою виділеного каталогу | javax.swing.border.TitledBorder |
| Рамка ,,Дерево каталогу’’ | javax.swing.border.TitledBorder |
| Рамка,,Інформація облікового запису’’ | javax.swing.border.TitledBorder |
| Рамка ,,Історія замовлень’’ | javax.swing.border.TitledBorder |
| Таблиця ,,Склад замовлення’’ | java.swing.JTable |
| Таблиця ,,Вміст кошику’’ | java.swing.JTable |
| Елементна панель ,,Вид контактної інформації’’ | javax.swing.ButtonGroup |
| Елементна панель ,,Спосіб оплати’’ | javax.swing.ButtonGroup |
| Елементна панель ,,Спосіб доставки’’ | javax.swing.ButtonGroup |
| Дерево каталогу | java.swing.JTree |

Додаток Б. Лістинг коду програмного додатку

package main;

import exceptions.DAOException;

import global.GlobalContext;

import View.MainFrame;

import View.ViewFactory;

import app.ApplicationCRUD;

import app.DBConfiguration;

/\*\*

\* Клас містить головний метод програми.

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ"

\* @version 1.0

\*/

public class NewMain {

/\*\*

\* головний метод програми.

\* @param args

\*/

public static void main(String[] args) {

DBConfiguration configs = new DBConfiguration();

GlobalContext.globalContext.put("dbConfigs", configs);

ApplicationCRUD crud = null;

try {

crud = new ApplicationCRUD(configs);

} catch (DAOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

GlobalContext.globalContext.put("crud", crud);

GlobalContext.globalContext.put("ViewFactory", ViewFactory.getInstance());

MainFrame.main(args);

}

}

package model;

import utils.IntegerConverter;

import annotation.Primary;

import annotation.Stored;

/\*\*

\* Клас є об'єктним відображенням таблиці в базі даних, що містить інформацію

\* облікового запису.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Stored(name = "PROFILE")

public class Profile{

/\*\*

\* Поле зберігає значення ID облікового запису.

\*/

@Primary

@Stored(name = "ID", converter = IntegerConverter.class)

private Integer ID;

/\*\*

\* Поле зберігає значення ID контактної інформації даного облікового запису

\*/

@Stored(name = "ContactInformation\_ID", converter = IntegerConverter.class)

private Integer contactInformation\_ID;

/\*\*

\* Поле зберігає значення логіна облікового запису

\*/

@Stored(name = "LOGIN")

private String login;

/\*\*

\* Поле зберігає значення паролю облікового запису

\*/

@Stored(name = "PASSWORD")

private String password;

/\*\*

\* Конструктор класу без параметрів

\*/

public Profile(){

}

/\*\*

\* Конструктор класу, що ініціалізує значення всіх полів класу

\*

\* @param iD ID облікового запису

\* @param contactInformation\_ID ID контакної інформації даного облікового запису

\* @param login логін

\* @param password пароль

\*/

public Profile(Integer iD, Integer contactInformation\_ID, String login, String password) {

super();

ID = iD;

this.contactInformation\_ID = contactInformation\_ID;

this.login = login;

this.password = password;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean equals(Object obj){

if(obj == null)

return false;

if(!this.getClass().equals(obj.getClass()))

return false;

if(!this.ID.equals(((Profile)obj).getID()))

return false;

return true;

}

/\*\*

\* Метод повертає ID запису.

\*

\* @return ID запису.

\*/

public Integer getID() {

return ID;

}

/\*\*

\* Метод встановлює ID запису.

\*

\* @param iD ID запису.

\*/

public void setID(Integer iD) {

ID = iD;

}

/\*\*

\* Метод повертає логін облікового запису

\*

\* @return логін облікового запису

\*/

public String getLogin() {

return login;

}

/\*\*

\* Метод встановлює логін облікового запису

\*

\* @param login логін облікового запису

\*/

public void setLogin(String login) {

this.login = login;

}

/\*\*

\* Метод повертає пароль облікового запису

\*

\* @return пароль облікового запису

\*/

public String getPassword() {

return password;

}

/\*\*

\* Метод встановлює пароль облікового запису

\*

\* @param password пароль облікового запису

\*/

public void setPassword(String password) {

this.password = password;

}

/\*\*

\* Метод повертає ID контактної інформації.

\*

\* @return ID контактної інформації.

\*/

public Integer getContactInformation\_ID() {

return contactInformation\_ID;

}

/\*\*

\* Метод встановлює ID контактної інформації.

\* @param contactInformation\_ID ID контактної інформації.

\*/

public void setContactInformation\_ID(Integer contactInformation\_ID) {

this.contactInformation\_ID = contactInformation\_ID;

}

}

package model;

import utils.IntegerConverter;

import annotation.Primary;

import annotation.Stored;

/\*\*

\* Клас є об'єктним відображенням таблиці бази даних, що містить інформацію про замовлення.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Stored(name = "database.ORDER")

public class Order{

/\*\*

\* Поле зберігає ID замовлення.

\*/

@Primary

@Stored(name = "ID", converter = IntegerConverter.class)

private Integer ID;

/\*\*

\* Поле зберігає спосіб оплати замовлення.

\*/

@Stored(name = "Payment")

private String payment;

/\*\*

\* Поле зберігає спосіб доставки замовлення.

\*/

@Stored(name = "DeliveryMethod")

private String deliveryMethod;

/\*\*

\* Поле зберігає статус замовлення.

\*/

@Stored(name = "OrderState")

private String orderState;

/\*\*

\* Конструктор класу без параметрів.

\*/

public Order(){

}

/\*\*

\* Конструктор класу, що ініціалізує значення всіх полів класу

\* @param iD ID замовлення.

\* @param payment спосіб оплати замовлення.

\* @param deliveryMethod спосіб доставки замовлення.

\* @param orderState статус замовлення.

\*/

public Order(Integer iD, String payment, String deliveryMethod, String orderState) {

ID = iD;

this.payment = payment;

this.deliveryMethod = deliveryMethod;

this.orderState = orderState;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean equals(Object obj){

if(obj == null)

return false;

if(!this.getClass().equals(obj.getClass()))

return false;

if(!this.ID.equals(((Order)obj).getID()))

return false;

return true;

}

/\*\*

\* Метод повертає ID замовлення

\*

\* @return ID замовлення

\*/

public Integer getID() {

return ID;

}

/\*\*

\* Метод встановлює ID замовлення

\*

\* @param iD ID замовлення

\*/

public void setID(Integer iD) {

ID = iD;

}

/\*\*

\* Метод повертає спосіб оплати

\*

\* @return спосіб оплати

\*/

public String getPayment() {

return payment;

}

/\*\*

\* Метод встановлює спосіб оплати

\*

\* @param payment спосіб оплати

\*/

public void setPayment(String payment) {

this.payment = payment;

}

/\*\*

\* Метод повертає спосіб доставки

\*

\* @return спосіб доставки

\*/

public String getDeliveryMethod() {

return deliveryMethod;

}

/\*\*

\* Метод встановлює спосіб доставки

\*

\* @param deliveryMethod спосіб доставки

\*/

public void setDeliveryMethod(String deliveryMethod) {

this.deliveryMethod = deliveryMethod;

}

/\*\*

\* Метод повертає статус замовлення

\*

\* @return статус замовлення

\*/

public String getOrderState() {

return orderState;

}

/\*\*

\* Метод встановлює статус замовлення

\*

\* @param orderState статус замовлення

\*/

public void setOrderState(String orderState) {

this.orderState = orderState;

}

}

package model;

import utils.IntegerConverter;

import annotation.Primary;

import annotation.Stored;

/\*\*

\* Клас є об'єктним відображенням рядку таблиці, що містить в собі інформацію про конкретний

\* товар.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Stored(name = "GOODS")

public class Goods{

/\*\*

\* Поле зберігає ID замовлення.

\*/

@Stored(name = "ID", converter = IntegerConverter.class)

@Primary

private Integer ID;

/\*\*

\* Поле зберігає назву товару.

\*/

@Stored(name = "Name")

private String name;

/\*\*

\* Поле зберігає опис товару.

\*/

@Stored(name = "COMMENT")

private String desctription;

/\*\*

\* Поле зберігає виробника товару.

\*/

@Stored(name = "Maker")

private String maker;

/\*\*

\* Поле зберігає кількість товарів.

\*/

@Stored(name = "NUMBEROFGOODS", converter = IntegerConverter.class)

private Integer number;

/\*\*

\* Поле зберігає ціну товару.

\*/

@Stored(name = "PRICE", converter = IntegerConverter.class)

private Integer price;

/\*\*

\* Поле зберігає шлях до зображення товару.

\*/

@Stored(name = "ImagePath")

private String imagePath;

/\*\*

\* Конструктор класу без параметрів.

\*/

public Goods(){

}

/\*\*

\* Конструктор класу, що ініціалізує значення всіх полів класу

\* @param iD ID замовлення.

\* @param name назва товару.

\* @param desctription опис товару.

\* @param maker виробник товару.

\* @param number кількість товарів.

\* @param price ціна товару.

\*/

public Goods(Integer iD, String name, String desctription, String maker,

Integer number, Integer price) {

ID = iD;

this.name = name;

this.desctription = desctription;

this.maker = maker;

this.number = number;

this.price = price;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public String toString(){

return maker +" "+ name;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean equals(Object obj){

if(obj == null)

return false;

if(!this.getClass().equals(obj.getClass()))

return false;

if(!this.ID.equals(((Goods)obj).getID()))

return false;

return true;

}

/\*\*

\* Метод повертає ID товару.

\*

\* @return ID товару.

\*/

public Integer getID() {

return ID;

}

/\*\*

\* Метод встановлює ID товару

\*

\* @param iD ID товару

\*/

public void setID(Integer iD) {

ID = iD;

}

/\*\*

\* Метод повертає назву товару

\*

\* @return назва товару

\*/

public String getName() {

return name;

}

/\*\*

\* Метод встановлює назву товару

\*

\* @param name назва товару

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Метод повертає опис товару

\*

\* @return опис товару

\*/

public String getDesctription() {

return desctription;

}

/\*\*

\* Метод встановлює опис товару

\*

\* @param desctription опис товару

\*/

public void setDesctription(String desctription) {

this.desctription = desctription;

}

/\*\*

\* Метод повертає виробника товару

\*

\* @return виробник товару

\*/

public String getMaker() {

return maker;

}

/\*\*

\* Метод встановлює виробника товару

\*

\* @param maker виробник товару

\*/

public void setMaker(String maker) {

this.maker = maker;

}

/\*\*

\* Метод повертає кількість товару

\*

\* @return кількість товару

\*/

public Integer getNumber() {

return number;

}

/\*\*

\* Метод встановлює кількість товару

\*

\* @param number кількість товару

\*/

public void setNumber(Integer number) {

this.number = number;

}

/\*\*

\* Метод повертає ціну товару

\*

\* @return ціна товару

\*/

public Integer getPrice() {

return price;

}

/\*\*

\* Метод встановлює ціну товару

\*

\* @param price ціна товару

\*/

public void setPrice(Integer price) {

this.price = price;

}

/\*\*

\* Метод повертає шлях до зображення

\*

\* @return шлях до зображення

\*/

public String getImagePath() {

return imagePath;

}

/\*\*

\* Метод встановлює шлях до зображення

\*

\* @param imagePath шлях до зображення

\*/

public void setImagePath(String imagePath) {

this.imagePath = imagePath;

}

}

package model;

import utils.IntegerConverter;

import annotation.Primary;

import annotation.Stored;

/\*\*

\* Клас інкапсулює контактну інформацію про користувача.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Stored(name = "CONTACTINFORMATION")

public class ContactInformation{

/\*\*

\* Значення первинного ключа типу в реляційній таблиці.

\*/

@Stored(name = "ID",converter = IntegerConverter.class)

@Primary

private Integer ID;

/\*\*

\* Значення поля <code>NAME</code> в реляційній таблиці.

\*/

@Stored(name = "NAME")

private String name;

/\*\*

\* Значення поля <code>LASTNAME</code> в реляційній таблиці.

\*/

@Stored(name = "LASTNAME")

private String lastName;

/\*\*

\* Значення поля <code>EMAIL</code> в реляційній таблиці.

\*/

@Stored(name = "EMAIL")

private String email;

/\*\*

\* Значення поля <code>ADRESS</code> в реляційній таблиці.

\*/

@Stored(name = "ADRESS")

private String adress;

/\*\*

\* Значення поля <code>CITY</code> в реляційній таблиці.

\*/

@Stored(name = "CITY")

private String city;

/\*\*

\* Значення поля <code>TELEPHONE</code> в реляційній таблиці.

\*/

@Stored(name = "TELEPHONE")

private String telephone;

/\*\*

\* Конструктор класу без параметрів.

\*/

public ContactInformation(){

}

/\*\*

\* Конструктор класу, дані для якого взяті з реляційної таблиці

\* @param ID {@link #ID}

\* @param name {@link #name}

\* @param lastName {@link #lastName}

\* @param email {@link #email}

\* @param adress {@link #adress}

\* @param city {@link #city}

\* @param telephone {@link #telephone}

\*/

public ContactInformation(Integer ID,String name,String lastName,String email, String adress, String city, String telephone){

this.ID = ID;

this.name = name;

this.lastName = lastName;

this.email = email;

this.adress = adress;

this.city = city;

this.telephone = telephone;

}

/\*\*

\* Конструктор класу, дані для якого взяті з програмного додатку

\* @param name {@link #name}

\* @param lastName {@link #lastName}

\* @param email {@link #email}

\* @param adress {@link #adress}

\* @param city {@link #city}

\* @param telephone {@link #telephone}

\*/

public ContactInformation(String name,String lastName,String email, String adress, String city, String telephone){

this.name = name;

this.lastName = lastName;

this.email = email;

this.adress = adress;

this.city = city;

this.telephone = telephone;

}

/\*\*

\* Перевизначений метод суперкласу, що повертає рядок з назвами та значеннями всіх полів класу.

\* формат виводу: <code>назва\_поля = параметр</code>.

\*

\* @return рядок, що містить назви та значення всіх полів класу.

\*/

@Override

public String toString(){

return "ID = "+ID+" name = "+ name +" lastName = "+lastName+"Email = "+ email+" city = "+city+"adress = "+adress+"telephone = "+ telephone;

}

/\*\*

\* Перевизначений метод суперкласу, який порівнює об'єкти за приналежністю до одного типу та за

\* значенням первинного ключа.

\* @param obj об'єкт, з яким необхідно порівняти об'єкт даного класу.

\* @return true, якщо об'єкти належать до одного типу та мають рівні первинні ключі, в іншому випадку - false.

\* Якщо параметр - <code>null</code>, метод повертає - <code>false</code>.

\*/

@Override

public boolean equals(Object obj){

ContactInformation object = (ContactInformation)obj;

if(obj == null)

return false;

if(!this.getClass().equals(obj.getClass()))

return false;

if(name.equals(object.name) == false || lastName.equals(object.lastName) == false || adress.equals(object.adress) == false ||

city.equals(object.city) == false || email.equals(object.email) == false || telephone.equals(object.telephone) == false )

return false;

else return true;

}

/\*\*

\* Метод повертає ім'я

\*

\* @return ім'я

\*/

public String getName() {

return name;

}

/\*\*

\* Метод встановлює ім'я

\*

\* @param name ім'я

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Метод повертає прізвище

\*

\* @return прізвище

\*/

public String getLastName() {

return lastName;

}

/\*\*

\* Метод встановлює прізвище

\*

\* @param lastName прізвище

\*/

public void setLastName(String lastName) {

this.lastName = lastName;

}

/\*\*

\* Метод повертає адресу електронної пошти

\*

\* @return адреса електронної пошти

\*/

public String getEmail() {

return email;

}

/\*\*

\* Метод встановлює адресу електронної пошти

\*

\* @param email адреса електронної пошти

\*/

public void setEmail(String email) {

this.email = email;

}

/\*\*

\* Метод повертає адресу

\*

\* @return адреса

\*/

public String getAdress() {

return adress;

}

/\*\*

\* Метод встановлює адресу

\*

\* @param adress адреса

\*/

public void setAdress(String adress) {

this.adress = adress;

}

/\*\*

\* Метод повертає назву міста

\*

\* @return назва міста

\*/

public String getCity() {

return city;

}

/\*\*

\* Метод встановлює назву міста

\*

\* @param city назва міста

\*/

public void setCity(String city) {

this.city = city;

}

/\*\*

\* Метод повертає ID контактної інформації

\*

\* @return ID контактної інформації

\*/

public Integer getID() {

return ID;

}

/\*\*

\* Метод встановлює ID контактної інформації

\*

\* @param iD ID контактної інформації

\*/

public void setID(Integer iD) {

ID = iD;

}

/\*\*

\* Метод повертає номер мобільниго телефону

\*

\* @return номер мобільниго телефону

\*/

public String getTelephone() {

return telephone;

}

/\*\*

\* Метод встановлює номер мобільниго телефону

\*

\* @param telephone номер мобільниго телефону

\*/

public void setTelephone(String telephone) {

this.telephone = telephone;

}

}

package model;

import utils.IntegerConverter;

import annotation.Primary;

import annotation.Stored;

/\*\*

\* Клас є об'єктним відображенням рядку таблиці, що містить інформацію про конкретний

\* каталог.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Stored(name = "CATALOG")

public class Catalog{

/\*\*

\* ID каталогу

\*/

@Stored(name = "ID",converter = IntegerConverter.class)

@Primary

private Integer ID;

/\*\*

\* Ім'я каталогу

\*/

@Stored(name = "NAME")

private String name;

/\*\*

\* Конструктор класу без параметрів.

\*/

public Catalog(){

}

/\*\*

\* @param name конструктор класу, що ініціалізує значення усіх полів класу.

\*/

public Catalog(String name, Integer id) {

this.name = name;

ID = id;

}

/\*\*

\* Метод повертає назву каталогу

\*

\* @return назва каталогу

\*/

public String getName() {

return name;

}

/\*\*

\* Метод встановлює назву каталогу

\*

\* @param name назва каталогу

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* Метод повертає ID каталогу

\*

\* @return ID каталогу

\*/

public Integer getID() {

return ID;

}

/\*\*

\* Метод встановлює ID каталогу

\*

\* @param iD ID каталогу

\*/

public void setID(Integer iD) {

ID = iD;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public String toString(){

return name;

}

}

**package** model;

**import** utils.IntegerConverter;

**import** annotation.Primary;

**import** annotation.Stored;

/\*\*

\* Клас є об'єктним відображенням рядку таблиці, що містить інформацію про кошик інтернет-магазину.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Stored(name = "BASKET")

**public** **class** BasketItem {

/\*\*

\* ID рядку кошика

\*/

@Primary

@Stored(name = "ID", converter = IntegerConverter.**class**)

**private** Integer ID;

/\*\*

\* ID товару, що міститься в даному рядку таблиці кошика.

\*/

@Stored(name = "goods\_ID", converter = IntegerConverter.**class**)

**private** Integer goodsID;

/\*\*

\* ID облікового запису, що міститься в даному рядку таблиці кошика.

\*/

@Stored(name = "profile\_ID", converter = IntegerConverter.**class**)

**private** Integer profileID;

/\*\*

\* Кількість товару, що міститься в даному рядку таблиці кошика.

\*/

@Stored(name = "Count", converter = IntegerConverter.**class**)

**private** Integer count;

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **BasketItem**(){

}

/\*\*

\* Метод повертає ID даного запису

\* **@return** ID даного запису

\*/

**public** Integer **getID**() {

**return** ID;

}

/\*\*

\* Метод встановлює ID даного запису

\* **@param** iD ID даного запису

\*/

**public** **void** **setID**(Integer iD) {

ID = iD;

}

/\*\*

\* Метод повертає ID товару.

\* **@return** ID товару.

\*/

**public** Integer **getGoodsID**() {

**return** goodsID;

}

/\*\*

\* Метод встановлює ID товару.

\* **@param** goodsID ID товару.

\*/

**public** **void** **setGoodsID**(Integer goodsID) {

**this**.goodsID = goodsID;

}

/\*\*

\* Метод повертає ID облікового запису

\* **@return** ID облікового запису

\*/

**public** Integer **getProfileID**() {

**return** profileID;

}

/\*\*

\* Метод встановлює ID облікового запису

\* **@param** profileID ID облікового запису

\*/

**public** **void** **setProfileID**(Integer profileID) {

**this**.profileID = profileID;

}

/\*\*

\* Метод повертає кількість товарів.

\* **@return** кількість товарів.

\*/

**public** Integer **getCount**() {

**return** count;

}

/\*\*

\* Метод встановлює кількість товарів.

\* **@param** count кількість товарів.

\*/

**public** **void** **setCount**(Integer count) {

**this**.count = count;

}

}

package model;

import java.sql.ResultSet;

import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.util.Vector;

import javax.swing.table.DefaultTableModel;

/\*\*

\*

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@SuppressWarnings("serial")

public class ApplicationTableModel extends DefaultTableModel {

public ApplicationTableModel(ResultSet set){

dataVector = new Vector<>();

columnIdentifiers = new Vector<>();

resultSetToTable(set);

}

@SuppressWarnings("unchecked")

protected void resultSetToTable(ResultSet source){

try {

ResultSetMetaData metadata = source.getMetaData();

for (int i = 1; i <= metadata.getColumnCount(); i++) {

columnIdentifiers.add(metadata.getColumnName(i));

}

while(source.next()){

Vector<Object> current = new Vector<>();

for (int i = 1; i <= metadata.getColumnCount(); i++) {

current.add(source.getObject(i));

}

dataVector.add(current);

}

} catch (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean isCellEditable(int row, int column){

return false;

}

}

package app;

import java.util.List;

import exceptions.DAOException;

/\*\*

\* Клас є реалізацією - заглушкою інтерфейсу {@link CRUDInterface}, створений з метою

\* зменшення дублювання коду та надає можливість дочірнім класам перевизначати лише ті

\* методи, які необхідні для специфічних реалізацій функціональностей взаємодії з персистентними сховищами.

\*

\* @see CRUDInterface

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

abstract class TransparendCRUD implements CRUDInterface{

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>false</code> за умовчуванням та якщо вхідний параметр - <code>null</code>.

\* </p>

\*/

@Override

public boolean open(DBConfiguration configuration) throws DAOException {

if(configuration == null)

return false;

return false;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>false</code> за умовчуванням.

\* </p>

\*/

@Override

public boolean close() throws DAOException {

return false;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>false</code> за умовчуванням та якщо вхідний параметр - <code>null</code>.

\* </p>

\*/

@Override

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

if(instance == null)

return null;

return null;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>null</code> за умовчуванням та якщо один із вхідних параметрів -

\* <code>null</code>.

\* </p>

\*/

@Override

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

if(entityClass == null || id == null)

return null;

return null;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>false</code> за умовчуванням та якщо вхідний параметр - <code>null</code>.

\* </p>

\*/

@Override

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

if(instance == null)

return false;

return false;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>false</code> за умовчуванням та якщо вхідний параметр - <code>null</code>.

\* </p>

\*/

@Override

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

if(instance == null)

return false;

return false;

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод повертає значення <code>null</code> за умовчуванням та якщо один із вхідних параметрів -

\* <code>null</code>.

\* </p>

\*/

@Override

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

return null;

}

}

package app;

import java.util.Properties;

import Configurators.ConfigFacade;

/\*\*

\* Клас призначений для інкапсуляції даних,за допомогою яких відбувається підключення

\* до бази даних. Об'єкт класу передається в якості параметру методу {@link CRUDInterface#open(DBConfiguration)}.

\*

\* @see CRUDInterface

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

public class DBConfiguration {

/\*\*

\* Строкове значення логіну облікового запису.

\*/

private String login;

/\*\*

\* Строкове значення паролю облікового запису.

\*/

private String password;

/\*\*

\* Строкове значення URL бази даних.

\*/

private String URL;

/\*\*

\* Назва класу - драйверу, який використовується JDBC для взаємодії з базою даних стороннього виробника.

\*/

private String driver;

/\*\*

\* Конструктор класу без параметрів.

\*/

public DBConfiguration(){

loadFromConfigs();

}

/\*\*

\* Конструктор класу, що ініціалізує всі поля класу.

\*

\* @param login {@link #login}

\* @param password {@link #password}

\* @param uRL {@link #URL}

\* @param driver {@link #driver}

\*/

public DBConfiguration(String login, String password, String uRL, String driver) {

this.login = login;

this.password = password;

URL = uRL;

this.driver = driver;

}

private void loadFromConfigs(){

String start = new String("ApplicationConfigurations.mysql.");

Properties p = ConfigFacade.getInstance().getProperties();

login = p.getProperty(start + "login");

password = p.getProperty(start + "password");

URL = p.getProperty(start+"URL");

driver = p.getProperty(start +"driver");

p = null;

start = null;

}

/\*\*

\* Метод повертає логін

\*

\* @return логін

\*/

public String getLogin() {

return login;

}

/\*\*

\* Метод встановлює логін

\*

\* @param login логін

\*/

public void setLogin(String login) {

this.login = login;

}

/\*\*

\* Метод повертає пароль

\*

\* @return пароль

\*/

public String getPassword() {

return password;

}

/\*\*

\* Метод встановлює пароль

\*

\* @param password пароль

\*/

public void setPassword(String password) {

this.password = password;

}

/\*\*

\* Метод повертає URL

\*

\* @return URL

\*/

public String getURL() {

return URL;

}

/\*\*

\* Метод встановлює URL

\* @param uRL

\*/

public void setURL(String uRL) {

URL = uRL;

}

/\*\*

\* Метод повертає драйвер БД

\*

\* @return драйвер БД

\*/

public String getDriver() {

return driver;

}

/\*\*

\* Метод встановлює драйвер БД

\*

\* @param driver драйвер БД

\*/

public void setDriver(String driver) {

this.driver = driver;

}

}

**package** app;

**import** java.sql.ResultSet;

**import** java.util.List;

**import** exceptions.DAOConvertationException;

**import** exceptions.DAOException;

/\*\*

\* Інтерфейс забезпечує взаємодію з персистентним сховищем, в т.ч. з реляційною базою даних. Також встановлює правила

\* для створення об'єктно - реляційної моделі представлення даних((англ.) Object - Relational Model (ORM)),яка реалізується

\* в програмному додатку за допомогою:

\* <p>

\* 1.CRUD - базових методів для заємодії зі сховищем даних, а саме створення підключення, закриття існуючого підключення,

\* створення запису, оновлення існуючого запису, видалення запису, створення запиту з багатьма та одним результатом;

\* </p>

\* <p>

\* 2.механізму інтроспекції та технології анотування об'єктів моделей.

\* </p>

\* <p>

\* Сховище в програмному додатку є базою даних. Для підключення до сховища необхідно викликати метод

\* {@link #open(DBConfiguration сonfiguration)}, а після завершення роботи зі сховищем обов'язково викликати метод

\* {@link #close()}. В якості параметрів до решти методів можна передавати об'єкти будь - яких класів,

\* які повинні бути розміченими за допомогою анотацій {@link annotation.Stored} та {@link annotation.Primary}.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **interface** CRUDInterface{

/\*\*

\* Метод забезпечує підключення до бази даних.

\* Конфігурується об'єктом класу {@link DBConfiguration}.

\*

\* **@param** configuration об'єкт класу {@link DBConfiguration}

\* **@return** булеве значення, що означає результат операції підключення.

\* **@throws** DAOException

\*

\*/

**public** **boolean** **open**(DBConfiguration configuration) **throws** DAOException;

/\*\*

\* Метод забезпечує закриття діючого підключення до бази даних. Викликається лише після

\* {@link #open(DBConfiguration)}.

\*

\* **@return** булеве значення, що означає результат відключення до сховища.

\* **@throws** DAOException

\*/

**public** **boolean** **close**() **throws** DAOException;

/\*\*

\* Метод забезпечує вставлення нового запису, інкапсульованого у вхідному параметрі.

\*

\* **@param** instance об'єкт, що інкапсулює в собі дані, які мають бути зберігатися в базі даних.

\* **@return** об'єкт того ж типу, що і параметр, з оновленим первинним ключем.

\* **@throws** DAOException

\*/

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

/\*\*

\* Метод забезпечує зчитування одного запису зі сховища (згідно з первинним ключем) та його відображення в

\* об'єктному представленні.

\*

\* **@param** entityClass тип, до якого мають бути приведений результат запиту.

\* **@param** id первинний ключ даних, що зберігаються в базі даних.

\* **@return** об'єкт типу, що вказаний у вхідних параметрах.

\* **@throws** DAOException

\* **@throws** DAOConvertationException

\*/

**public** <**T**> **T** **read** (Class<**T**> entityClass, Integer id) **throws** DAOException, DAOConvertationException;

/\*\*

\* Метод запезпечує оновлення запису в сховищі, що відповідає об'єктному представленню вхідного

\* параметру. Повертає значення <code>true<code>, якщо в результаті виконання операції відбулася

\* зміна даних в сховищі.

\*

\* **@param** instance об'єкт, що інкапсулює дані, які мають бути занесені в сховище.

\* **@return** <code>true</code>, якщо дані в сховищі оновлені.

\* **@throws** DAOException

\*/

**public** <**T**> **boolean** **update** (**T** instance) **throws** DAOException;

/\*\*

\* Метод забезпечує видалення запису зі сховища, що відповідає об'єктному представленню вхідного

\* параметру. Повертає значення <code>true</code>, якщо в результаті виконання операції відбулося

\* видалення запису зі сховища.

\*

\* **@param** instance об'єкт, що інкапсулює дані, які мають бути видалені зі сховища.

\* **@return** <code>true</code>, якщо дані видалені зі сховища.

\* **@throws** DAOException

\*/

**public** <**T**> **boolean** **delete** (**T** instance) **throws** DAOException;

/\*\*

\* Метод забезпечує вибірку масиву даних з таблиці за допомогою строкової команди, що записана

\* у вхідному параметрі, та перетворення цих даних в об'єктне представлення, тип якого вказаний

\* у вхідному параметрі.

\*

\* **@param** entityClass тип, до якого мають бути приведений кожен запис, що задовільнив запит.

\* **@param** SQLString селектор, що створений згідно із синтаксисом SQL.

\* **@return** об'єкт типу {@link List}, що зберігає об'єкти, реляційне представлення яких задовольнив

\* запит

\* **@throws** DAOException

\*

\* **@see** List

\*/

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

/\*\*

\* Метод забезпечує вставку строки в базу даних

\*

\* **@param** SQLString SQL - запит до бази даних

\* **@throws** DAOException якщо виникла помилка при запиті до бази даних

\*/

**public** **void** **insert**(String SQLString) **throws** DAOException;

/\*\*

\* Метод забезпечує видалення строки з бази даних

\*

\* **@param** SQLString SQL - запит до бази даних

\* **@throws** DAOException якщо виникла помилка при запиті до бази даних

\*/

**public** **void** **delete**(String SQLString) **throws** DAOException;

/\*\*

\* Метод забезпечує вибірку масиву даних з бази даних.

\*

\* **@param** SQLString SQL - запит до бази даних

\* **@return** об'єкт типу {@link ResultSet}

\* **@throws** DAOException якщо виникла помилка при запиті до бази даних

\*/

**public** ResultSet **select**(String SQLString) **throws** DAOException;

}

**package** app;

**import** java.beans.IntrospectionException;

**import** java.beans.PropertyDescriptor;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.InvocationTargetException;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** exceptions.DAOConvertationException;

**import** exceptions.DAOException;

**import** annotation.Primary;

**import** annotation.Stored;

**import** utils.DAOAnnotationUtility;

/\*\*

\* Клас інкапсулює в собі логіку взаємодії програмного додатку з реляційною базою даних та

\* реалізує об'єктно - реляційну модель представлення даних.

\* <p>

\* Клас реалізує шаблон DAO, основним призначенням якого є інкапсуляція логіки взаємодії програмного

\* додатку з реляційною базою даних.

\* </p>

\* <p>

\* Вхідними параметрами методів є будь - які об'єкти класів, що розмічені за допомогою анотацій

\* {@link Stored} та {@link Primary}, та самі класи. Таким чином послаблюється взаємодія цього класу

\* з іншими, адже ніде не фігурують конкретні типи даних, що використовуються в програмному додатку.

\* Такий підхід дає можливість динамічній зміні структури програмного додатку та його розширенню.

\* </p>

\* <p>

\* Клас є реалізацією інтерфейсу {@link CRUDInterface}.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\* **@see** CRUDInterface

\* **@see** TransparendCRUD

\* **@see** DBConfiguration

\* **@see** Connection

\*/

**public** **class** ApplicationCRUD **implements** CRUDInterface{

/\*\*

\* Поточне з'єднання з реляційною базою даних.

\*/

**private** Connection currentConnection;

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **ApplicationCRUD**() {

}

/\*\*

\* Конструктор класу, що відразу створює з'єднання з базою даних.

\*

\* **@param** configs об'єкт класу {@link DBConfiguration}

\* **@throws** DAOException

\*/

**public** **ApplicationCRUD**(DBConfiguration configs) **throws** DAOException{

open(configs);

}

/\*\*

\* {@inheritDoc}

\*

\* **@return** true, якщо операція підключення пройшла успішно.

\* **@throws** DAOException

\*/

@Override

**public** **boolean** **open**(DBConfiguration configuration) **throws** DAOException{

**try** {

Class.*forName*(configuration.getDriver()).newInstance();

currentConnection = DriverManager.*getConnection*(configuration.getURL(), configuration.getLogin(), configuration.getPassword());

**return** **true**;

} **catch** (InstantiationException | IllegalAccessException | ClassNotFoundException | SQLException e) {

**throw** **new** DAOException("Системна помилка при встановленні з'єднання з базою даних. " +

"Перевірте правильність даних для з'єднання з базою даних.");

}

}

/\*\*

\* {@inheritDoc}

\* **@return** true, якщо операція відключення пройшла успішно.

\* **@throws** DAOException

\*/

@Override

**public** **boolean** **close**() **throws** DAOException{

**if**(currentConnection != **null**)

**try** {

currentConnection.close();

**return** **true**;

} **catch** (SQLException e) {

**throw** **new** DAOException("Системна помилка при відключенні від бази даних");

}

**return** **false**;

}

/\*\*

\* Метод дозволяє додати нову строку даних, що інкапсульовані в об'єкті-параметрі, в реляційну таблицю.

\* Також після оновлення даних бази у методі виконується додатковий запит з метою встановлення значення

\* поля первинного ключа об'єкту-параметру.

\*

\*

\* **@param** instance об'єкт, дані якого повинні бути поміщені в базу даних. Має бути розміченим за допомогою

\* анотацій {@link Stored} та {@link Primary}, інакше SQL-запит буде згенерований не коректно

\* **@return** об'єкт-параметр з оновленним первинним ключем. Якщо вхідний параметр - <code>null</code>, то

\* повертає <code>nul</code>.

\* **@throws** DAOException помилка при

\*/

@Override

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

//--------------------------------------------Перевірка вхідних параметрів---------------------------------------------------------------//

**if**(instance == **null**)

**return** **null**;

//---------------------------------------------Ініціалізація параметрів------------------------------------------------------------------//

HashMap<String, Object> map;

**try** {

map = DAOAnnotationUtility.*entityToMap*(instance);

} **catch** (DAOConvertationException e1) {

**throw** **new** DAOException(e1.getMessage());

}

String storageName = DAOAnnotationUtility.*getStorageName*(instance.getClass());

Statement statement = **null**;

ResultSet set = **null**;

//----------------------------------------------Генерування SQL - запиту-----------------------------------------------------------------//

StringBuilder SQLBuilder = **new** StringBuilder("INSERT INTO "+storageName+" ()");

StringBuilder values = **new** StringBuilder("()");

StringBuilder selectString = **new** StringBuilder("SELECT ID FROM "+storageName+" WHERE ");

**for**(String column: map.keySet()){

SQLBuilder.insert(SQLBuilder.indexOf(")"),","+column);

values.insert(values.lastIndexOf(")"),","+ map.get(column));

selectString.append(column).append(" = ").append(map.get(column)).append("AND ");

}

SQLBuilder.deleteCharAt(SQLBuilder.indexOf(","));

values.deleteCharAt(values.indexOf(","));

SQLBuilder.append(" VALUES ").append(values);

values.deleteCharAt(values.indexOf(","));

String select = selectString.toString().substring(0, selectString.lastIndexOf("AND"));

//-----------------------------------------------Виконання запиту------------------------------------------------------------------------//

**try** {

statement = currentConnection.createStatement();

statement.executeUpdate(SQLBuilder.toString());

set = statement.executeQuery(select);

//-----------------------------------------------Встановлення значення первинного ключа--------------------------------------------------//

PropertyDescriptor pd = **new** PropertyDescriptor(DAOAnnotationUtility.*getPrimaryKey*(instance.getClass()).getName(), instance.getClass());

**while**(set.next())

pd.getWriteMethod().invoke(instance, set.getInt("ID"));

**return** instance;

} **catch** (SQLException | IntrospectionException |IllegalAccessException | IllegalArgumentException

| InvocationTargetException e) {

e.printStackTrace();

**throw** **new** DAOException("Системна помилка при додаванні нового рядка до бази даних. Перевірте коректність введених даних." + e.getMessage());

}

//-----------------------------------------------Звільнення системних ресурсів----------------------------------------------------------//

**finally**{

**try** {

statement.close();

map = **null**;

SQLBuilder = **null**;

values = **null**;

selectString = **null**;

select = **null**;

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* Метод виконує зчитування рядку з бази даних, об'єктне представлення якої вказане у параметрі.

\* Результат виконання SQL-запиту є параметром реляційно - об'єктного відображення.

\*

\* **@param** entityClass об'єктний тип, до якого має бути приведений результат запиту.

\* Має бути розміченим за допомогою анотацій {@link Stored}, {@link Primary}, інакше операція відображення

\* не матиме очікуваного результату.

\* **@param** id значення первинного ключа рядку, який має бути вхідним параметром відображення.

\* **@return** об'єкт типу вхідного параметру. Якщо цей параметр - <code>null</code> або не розмічений за допомогою

\* анотацій, то результат операції - <code>null</code>.Якщо значення первинного ключа - <code>null</code>,

\* вхідним параметром відображенням буде рядок з найбільшим значенням первинного ключа.

\* **@throws** DAOException системна помилка при зчитуванні або конвертації даних.

\*/

@Override

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

//--------------------------------------------Перевірка вхідних параметрів---------------------------------------------------------------//

**if**(entityClass == **null** || DAOAnnotationUtility.*getStorageName*(entityClass) == **null**)

**return** **null**;

//---------------------------------------------Ініціалізація параметрів------------------------------------------------------------------//

PreparedStatement s = **null**;

ResultSet set = **null**;

HashMap<String,Field> ORMap = DAOAnnotationUtility.*getORMap*(entityClass,**null**);

HashMap<String,Object> resultMap = **new** HashMap<>();

//----------------------------------------------Генерація SQL - запиту------------------------------------------------------------------//

**try** {

s = currentConnection.prepareStatement("SELECT \* FROM " + DAOAnnotationUtility.*getStorageName*(entityClass) + " WHERE ID = ?");

s.setInt(1, id);

set = s.executeQuery();

//----------------------------------------------Підготовка до реляційно-об'єктного відображення-----------------------------------------//

**while**(set.next())

**for**(**int** i = 1; i <= set.getMetaData().getColumnCount(); i++){

String column = set.getMetaData().getColumnName(i);

resultMap.put(column,DAOAnnotationUtility.*getValueConverter*(ORMap.get(column)).toValue(set.getString(i)));

}

//---------------------------------------------Реляційно-об'єктне відображення---------------------------------------------------------//

**return** DAOAnnotationUtility.*mapToEntity*(entityClass, resultMap);

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** DAOException("Cистемна помилка при виконанні запиту на зчитування рядку з бази даних.");

} **catch** (DAOConvertationException e) {

e.printStackTrace();

**throw** **new** DAOException(e.getMessage());

}

//-----------------------------------------------Звільнення системних ресурсів----------------------------------------------------------//

**finally**{

**try** {

s.close();

set.close();

} **catch** (SQLException e) {

**throw** **new** DAOException();

}

}

}

/\*\*

\* Метод виконує оновлення існуючих даних в рядку бази даних, що відповідає первинному ключу об'єкту-параметру.

\*

\* **@param** instance об'єктне представлення даних, які мають бути оновлені в таблиці. Об'єкт-параметр має бути

\* розміченим за допомогою анотацій {@link Stored}, та {@link Primary}, інакше оновлення даних не відбудеться.

\* **@return** <code>true</code>, якщо в результаті виконання запиту відбулося оновлення рядку в базі даних. В іншому

\* випадку - <code>false</code>.

\* **@throws** DAOException системна помилка при виконанні запиту або конвертації даних.

\*/

@Override

**public** <**T**> **boolean** **update**(**T** instance) **throws** DAOException{

//--------------------------------------------Перевірка вхідних параметрів---------------------------------------------------------------//

**if**(instance == **null** || DAOAnnotationUtility.*getStorageName*(instance.getClass()) == **null**)

**return** **false**;

Statement s = **null**;

**try**{

HashMap<String, Object> table = DAOAnnotationUtility.*entityToMap*(instance);

//--------------------------------------------Генерування SQL - запиту-------------------------------------------------------------------//

StringBuilder sqlRequest = **new** StringBuilder("UPDATE " + DAOAnnotationUtility.*getStorageName*(instance.getClass()) + " SET ");

**for**(String key : table.keySet())

sqlRequest.append(key).append(" = ").append(table.get(key)).append(", ");

sqlRequest.deleteCharAt(sqlRequest.lastIndexOf(","));

sqlRequest.append(" WHERE ID = ").append(String.*valueOf*((DAOAnnotationUtility.*getPrimaryKeyValue*(instance))));

s = currentConnection.createStatement();

**return** s.executeUpdate(sqlRequest.toString()) != 0;

//--------------------------------------------Виконання запиту та перевірка результату---------------------------------------------------//

}**catch**(DAOConvertationException e){

e.printStackTrace();

**throw** **new** DAOException(e.getMessage());

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** DAOException("Cистемна помилка при виконанні запиту на оновлення даних в рядку в базі даних.");

}

//-----------------------------------------------Звільнення системних ресурсів----------------------------------------------------------//

**finally**{

**try** {

s.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* Метод виконує видалення рядку з таблиці бази даних, об'єктне представлення якого відповідає вхідному параметру.

\*

\* **@param** instance вхідний параметр, реляційне відображення якого має бути видалене з таблиці.

\* **@return** true, якщо видалення виконане. Якщо вхідний параметр - <code>null</code> або не розмічений за допомогою

\* анотацій {@link Stored} та {@link Primary}, то результат - <code>false</code>.

\* **@throws** DAOException системна помилка при зчитуванні або конвертації даних.

\*/

@Override

**public** <**T**> **boolean** **delete**(**T** instance) **throws** DAOException{

//--------------------------------------------Перевірка вхідних параметрів---------------------------------------------------------------//

**if**(instance == **null** || DAOAnnotationUtility.*getStorageName*(instance.getClass()) == **null**)

**return** **false**;

//--------------------------------------------Генерація SQL - запиту---------------------------------------------------------------------//

**try** {

PreparedStatement s = currentConnection.prepareStatement("DELETE FROM "+ DAOAnnotationUtility.*getStorageName*(instance.getClass())+" WHERE ID = ?");

s.setInt(1, (Integer) DAOAnnotationUtility.*getPrimaryKeyValue*(instance));

//--------------------------------------------Виконання SQL - запиту та перевірка результату---------------------------------------------//

**boolean** result = s.executeUpdate() != 0 ;

s.close();

**return** result;

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** DAOException("Cистемна помилка при виконанні запиту на видалення рядка з бази даних.");

} **catch** (DAOConvertationException e) {

e.printStackTrace();

**throw** **new** DAOException(e.getMessage());

}

}

/\*\*

\* {@inheritDoc}

\* **@return** об'єкт типу {@link List}, що містить у собі об'єктний результат виконання запиту, що вказаний у параметрі.

\* Якщо параметр-клас - <code>null</code> або не розмічений за допомогою анотацій {@link Stored}, {@link Primary} -

\* метод повертає <code>null</code>. Якщо параметр-запит - <code>null</code>, метод повертає <code>null</code>.

\* **@throws** DAOException системна помилка при зчитуванні або конвертації даних.

\*/

@SuppressWarnings("unchecked")

@Override

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

**if**(entityClass == **null** || SQLString == **null** )

**return** **null**;

Statement s;

ResultSet set = **null**;

**try** {

s = currentConnection.createStatement();

set = s.executeQuery(SQLString);

} **catch** (SQLException e) {

**throw** **new** DAOException();

}

HashMap<String,Object> resultMap = **new** HashMap<>();

ArrayList<**T**> resultList = **new** ArrayList<>();

**try** {

**while**(set.next() == **true**){

**for**(**int** i = 1; i <= set.getMetaData().getColumnCount(); i++){

String column = set.getMetaData().getColumnName(i);

resultMap.put(column, set.getString(i));

}

**if**(!resultMap.isEmpty())

resultList.add((**T**) DAOAnnotationUtility.*mapToEntity*(entityClass, resultMap));

}

} **catch** (DAOConvertationException e) {

e.printStackTrace();

**throw** **new** DAOException(e.getMessage());

} **catch** (SQLException e) {

e.printStackTrace();

**throw** **new** DAOException("Помилка при виконанні запиту до бази даних.");

}

**finally**{

**try** {

s.close();

set.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

**return** resultList;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **insert**(String SQLString) {

Statement s;

**if**(SQLString.compareTo("") != 0)

**try** {

s = currentConnection.createStatement();

s.executeUpdate(SQLString);

s.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **delete**(String SQLString) **throws** DAOException{

**if**(SQLString == **null** || SQLString == "")

**throw** **new** DAOException("Системна помилка. SQL запит на видалення даних не може бути пустим.");

**try** {

Statement s = currentConnection.createStatement();

s.execute(SQLString);

s.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** ResultSet **select**(String SQLString){

**if**(SQLString == **null** || SQLString == "")

**return** **null**;

**try** {

Statement s = currentConnection.createStatement();

ResultSet result = s.executeQuery(SQLString);

**return** result;

} **catch** (SQLException e) {

e.printStackTrace();

**return** **null**;

}

}

}

**package** annotation;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** utils.DAOAnnotationUtility;

**import** utils.ValueConverter;

**import** app.CRUDInterface;

/\*\*

\* Дана анотація використовується для реалізації об'єктно - реляційної моделі даних. Детальніше: {@link CRUDInterface},

\* {@link DAOAnnotationUtility}.

\* <p>

\* Анотацією позначається клас, що має відповідну таблицю в базі даних, та поля, що мають відповідні колонки

\* в таблиці бази даних. Анотація зберігається в runtime.

\* </p>

\* <p>

\* Анотація має два поля, одне з яких зберігає рядкове значення ім'я елементу в базі даних(назви таблиці чи її колонки),

\* інше - конкретний тип {@link ValueConverter}.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\* **@see** CRUDInterface

\* **@see** DAOAnnotationUtility

\*/

@Retention(RetentionPolicy.RUNTIME)

**public** **@interface** Stored {

/\*\*

\* Поле зберігає рядкове значення ім'я елементу в базі даних(назви таблиці чи її колонки).

\*

\* **@return** строкове значення ім'я елементу в базі даних.

\*/

**public** String **name**();

/\*\*

\* Поле зберігає значення конкретного типу {@link ValueConverter}.

\*

\* **@return** значення типу {@link ValueConverter}.

\*/

@SuppressWarnings("rawtypes")

**public** Class **converter**() **default** utils.StringConverter.**class**;

}

**package** annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

/\*\*

\* Анотація - мітка, якою позначається поле класу, що містить у собі первинний ключ. Анотація зберігається в runtime.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Retention(RetentionPolicy.RUNTIME)

@Target(value = ElementType.FIELD)

**public** **@interface** Primary {

}

**package** utils;

/\*\*

\* Інтерфейс задає правила для конвертації об'єкту довільного типу зі строкового представлення

\* в об'єктне. Разом з класом {@link DAOAnnotationUtility} забезпечує реалізацію об'єктно - реляційної

\* моделі представлення даних моделей, що використовуються в програмному додатку.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** DAOAnnotationUtility

\* **@see** DAOAnnotationUtility#getFieldValue(Object, java.lang.reflect.Field)

\* **@see** DAOAnnotationUtility#getStringValue(Object, java.lang.reflect.Field)

\* **@see** DAOAnnotationUtility#getValueConverter(java.lang.reflect.Field)

\* **@see** DAOAnnotationUtility#mapToEntity(Class, java.util.Map)

\* **@see** DAOAnnotationUtility#entityToMap(Object)

\*/

**public** **interface** ValueConverter {

/\*\*

\* Метод забезпечує конвертацію строкових даних в об'єктні.

\* Наприклад, якщо на вхід подати строкове значення типу {@link Integer}, то

\* на виході буде отриманий об'єкт даного типу.

\* <p>

\* Даний метод в програмному додатку використовується методами класів, що призначені

\* для конвертування реляційної моделі представлення даних в об'єктну модель та навпаки.

\* </p>

\*

\* **@param** param вхідний параметр, строкове значення якого необхідно конвертувати в об'єктне.

\* **@return** об'єктне значення строкового вхідного параметру.

\*/

**public** <**T**> **T** **toValue**(String param);

/\*\*

\* Метод забезпечує конвертацію об'єктних даних в строкові.

\* Наприклад, якщо в якості параметру подати значення типу {@link Integer}, то на

\* виході буде отримане число, представлене типом {@link String} та заключене в одинарні лапки -

\* це відповідає синтаксису SQL.

\*

\* **@param** value об'єктне представлення даних для конвертації.

\* **@return** строкове значення об'єктного вхідного параметру.

\*/

**public** <**T**> String **toString**(**T** value);

}

**package** utils;

/\*\*

\* Клас реалізує об'єктно - реляційну конвертацію строкових даних.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\* **@see** ValueConverter

\* **@see** java.lang.String

\*/

**public** **class** StringConverter **implements** ValueConverter {

/\*\*

\* {@inheritDoc}

\* **@return** <code>null</code>, якщо вхідний параметр - <code>null</code>.

\* В іншому випадку - об'єкт класу {@link String} з одинарними кавичками.

\*/

@Override

**public** <**T**> String **toString**(**T** value) {

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

**return** "\'"+(String)value+"\'";

}

/\*\*

\* {@inheritDoc}

\* **@return** <code>null</code>, якщо вхідний параметр - <code>null</code> чи строка <code>null</code>,

\* пусте значення, якщо вхідний параметр - пустий рядок.

\* В іншому випадку - об'єкт класу {@link String} без одинарних кавичок.

\*/

@SuppressWarnings("unchecked")

@Override

**public** <**T**> **T** **toValue**(String str) {

**if** (str == **null**) **return** **null**;

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

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

**return** (**T**)(str.replaceAll("'", "").trim());

}

}

**package** utils;

/\*\*

\* Клас реалізує об'єктно - реляційну конвертацію даних типу {@link Integer}.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\* **@see** ValueConverter

\* **@see** java.lang.Integer

\*/

**public** **class** IntegerConverter **implements** ValueConverter {

/\*\*

\* {@inheritDoc}

\* **@return** <code>null</code>, якщо вхідний параметр - <code>null</code>.

\* В іншому випадку - об'єкт класу {@link String} з одинарними кавичками.

\*/

@Override

**public** <**T**> String **toString**(**T** value) {

**if**(value == **null**)

**return** **null**;

**return** "\'"+Integer.*toString*((Integer)value)+"\'";

}

/\*\*

\* {@inheritDoc}

\* **@return** <code>null</code>, якщо вхідний параметр - <code>null</code>.

\* В іншому випадку - об'єкт класу {@link Integer} без одинарних кавичок.

\*/

@SuppressWarnings("unchecked")

@Override

**public** <**T**> **T** **toValue**(String param) {

**if**(param == **null**)

**return** **null**;

**return** (**T**) **new** Integer(param.replaceAll("'", "").trim());

}

}

**package** utils;

**import** java.beans.IntrospectionException;

**import** java.beans.PropertyDescriptor;

**import** java.lang.annotation.Annotation;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.InvocationTargetException;

**import** java.util.HashMap;

**import** java.util.Map;

**import** exceptions.DAOConvertationException;

**import** annotation.Primary;

**import** annotation.Stored;

**import** app.CRUDInterface;

/\*\*

\* Утілітний клас, що інкапсулює в собі логіку реалізацції об'єктно - реляційної моделі представлення

\* даних програмного додатку. В якості параметрів методів приймає об'єкти будь - яких типів та об'єкти

\* типу {@link Class}, що дозволяє використовувати його функціональність при реалізації

\* {@link CRUDInterface}. Перевагою такого підходу є відділення логіки реалізації об'єктно - реляційної моделі,

\* яка може бути застосована до різних типів сховищ даних, та реалізації взаємодії зі самим сховищем даних, що,

\* у свою чергу, покращує розуміння логіки роботи додатку та супроводження його.

\* <p>

\* Класс надає можливість отримати : назву таблиці в реляційній базі даних, об'єктне представлення якої

\* подане в якості параметру; об'єкту типу {@link Field}, щро містить в собі значення первинного ключа, значення

\* цього ключа, строкове та об'єктне значення поля, що подане в якості параметру, об'єкт типу {@link ValueConverter} для

\* поля, поданого в якості параметру, об'єктно - реляційної карти, ключем якої є назва колонки в реляційній базі даних,

\* а значенням - об'єкт типу {@link Field}, який зберігає об'єктне представлення інформації, що міститься в базі даних.

\* Також клас має методи для конвертації об'єкту - параметру в реляційне представлення та конвертації реляційної таблиці

\* в об'єктне представлення.

\* </p>

\* <p>

\* У якості параметрів методів клас може приймати об'єкти будь - яких типів, що розмічені за допомогою анотацій

\* {@link Stored} та {@link Primary}.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** CRUDInterface

\* **@see** ValueConverter

\* **@see** Primary

\* **@see** Stored

\* **@see** java.util.Map

\* **@see** java.util.HashMap

\* **@see** java.lang.Class

\* **@see** java.lang.reflect.Field

\*/

@SuppressWarnings({"unchecked", "rawtypes"})

**public** **class** DAOAnnotationUtility {

/\*\*

\* Метод повертає значення таблиці, об'єктне представлення якої інкапсульоване в

\* об'єкті - параметрі типу {@link Class}

\*

\* **@param** instance об'єктне представлення реляційної таблиці.

\* **@return** строкове значення назви реляційної таблиці. Якщо параметр - <code>null</code>, то повертає <code>null</code>.

\*/

**public** **static** String **getStorageName**(Class instance){

**if**(instance == **null**)

**return** **null**;

**return** ((Stored)instance.getAnnotation(Stored.**class**)).name();

}

/\*\*

\* Метод повертає об'єкт типу {@link Field}, який зберігає значення первинного ключа.

\*

\* **@param** instance об'єктне представлення реляційної таблиці.

\* **@return** об'єкт типу {@link Field}, який зберігає значення первинного ключа. Якщо параметр - <code>null</code>,

\* то повертає - <code>null</code>.

\*/

**public** **static** Field **getPrimaryKey**(Class instance){

**if**(instance == **null**)

**return** **null**;

Field[] fields = instance.getDeclaredFields();

**for**(Field f : fields){

Annotation current = f.getAnnotation(Primary.**class**);

**if**(current != **null**)

**return** f;

}

**return** **null**;

}

/\*\*

\* Метод повертає значення первинного ключа(який може бути будь - яким об'єктом), що зберігається

\* в об'єкті - параметрі.

\*

\* **@param** instance об'єкт, значення первинного ключа якого необхідно повернути.

\* **@return** об'єктне представлення первинного ключа об'єкту - параметру. Якщо параметр - <code>null</code>

\* або не містить первинного ключа - метод повертає <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**T**,**S**> **S** **getPrimaryKeyValue**(**T** instance) **throws** DAOConvertationException{

**if**(instance == **null**)

**return** **null**;

Field primaryField = *getPrimaryKey*(instance.getClass());

**try** {

PropertyDescriptor pd = **new** PropertyDescriptor(primaryField.getName(), instance.getClass());

ValueConverter converter = *getValueConverter*(primaryField);

**S** c = converter.toValue(pd.getReadMethod().invoke(instance).toString());

**return** (**S**)c;

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

**throw** **new** DAOConvertationException("помилка при конвертації даних:"+e.getMessage());

}

}

/\*\*

\* Метод повертає строкове значення параметру типу {@link Field} об'єкту-параметру.

\*

\* **@param** instance об'єкт, строкове значення поля-параметру якого необідно повернути.

\* **@param** value поле об'єкту-параметру, строкове значення якого треба повернути.

\* **@return** строкове значення поля-параметру об'єкту-параметру. Якщо один з параметрів або

\* результат методу {@link #getValueConverter(Field)} - <code>null</code>, то повертає <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**T**> String **getStringValue**(**T** instance, Field value) **throws** DAOConvertationException{

**try** {

**if**(instance == **null** || value == **null**)

**return** **null**;

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

ValueConverter v = *getValueConverter*(value);

**if**(v == **null**)

**return** **null**;

**return** v.toString(p.getReadMethod().invoke(instance));

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

**throw** **new** DAOConvertationException("Cистемна помилка при конвертації даних."+e.getMessage());

}

}

/\*\*

\* Метод повертає значення параметру типу {@link Field} об'єкту-параметру, що має задекларований в полі-параметрі тип.

\*

\* **@param** instance об'єкт, значення поля-параметру якого необідно повернути.

\* **@param** field поле об'єкту-параметру, значення якого треба повернути.

\* **@return** значення поля-параметру об'єкту-параметру. Якщо один з параметрів або

\* результат методу {@link #getValueConverter(Field)} - <code>null</code>, то повертає <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**S**,**T**> **S** **getFieldValue**(**T** instance, Field field) **throws** DAOConvertationException{

**if**(instance == **null** || field == **null**)

**return** **null**;

**try** {

PropertyDescriptor pd = **new** PropertyDescriptor(field.getName(), instance.getClass());

String stringResult = (pd.getReadMethod().invoke(instance)).toString();

ValueConverter v = *getValueConverter*(field);

**if**(v == **null**)

**return** **null**;

**return** (**S**)(v.toValue(stringResult));

} **catch** (IllegalAccessException | IllegalArgumentException

| InvocationTargetException | IntrospectionException e) {

**throw** **new** DAOConvertationException("Cистемна помилка при конвертації даних."+e.getMessage());

}

}

/\*\*

\* Метод повертає об'єкт класу, що реалізує інтерфейс {@link ValueConverter}.

\*

\* **@param** f поле, об'єкт-конвертер значення якого необхідно повернути.

\* **@return** об'єкт класу, що реалізує інтерфейс {@link ValueConverter}. Якщо параметр - <code>null</code> або

\* не розмічений за допомогою анотації {@link Stored}, то метод повертає значення <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**T** **extends** ValueConverter> **T** **getValueConverter**(Field f) **throws** DAOConvertationException{

**if**(f == **null**)

**return** **null**;

**try** {

Stored s = f.getAnnotation(Stored.**class**);

**if**(s == **null**)

**return** **null**;

ValueConverter c = (ValueConverter) s.converter().newInstance();

**return** (**T**) c;

} **catch** (InstantiationException | IllegalAccessException e) {

**throw** **new** DAOConvertationException("Cистемна помилка при конвертації даних."+e.getMessage());

}

}

/\*\*

\* Метод виконує відображення параметру типу {@link java.lang.Class} на реляційну таблицю бази даних.

\* Повертає об'єкт класу {@link HashMap}, ключем якого є назва колонки реляційної таблиці, а іншим параметром -

\* об'єкт {@link Field} класу-параметру.

\*

\* **@param** parameter клас-параметр, поля якого мають бути у відображенні.

\* **@param** annotationFilter клас-анотація, помічені якою поля не мають увійти до відображення.

\* **@return** об'єкт класу {@link HashMap}, що містить у собі об'єктно - реляційне відображення. Якщо перший вхідний

\* параметр - <code>null</code>, то метод повертає <code>null</code>.

\*/

**public** **static** HashMap <String,Field> **getORMap**(Class parameter, Class annotationFilter){

**if**(parameter == **null**)

**return** **null**;

Field[] fields = parameter.getDeclaredFields();

HashMap<String, Field> specifiedAnnotatedFields = **new** HashMap();

**for**(Field f: fields){

**if**(f.getAnnotation(Stored.**class**) != **null**){

**boolean** flag = **true**;

**for**(Annotation n: f.getDeclaredAnnotations())

**if**(n.annotationType() == annotationFilter){

flag = **false**;

**break**;

}

**if**(flag == **true**){

specifiedAnnotatedFields.put(f.getAnnotation(Stored.**class**).name(), f);

}

}

}

**return** specifiedAnnotatedFields;

}

/\*\*

\* Метод створює реляційну таблицю, представлену у вигляді об'єкту класу {@link HashMap}, з об'єкту - параметру.

\*

\* **@param** instance об'єкт, з якого необхідно побудувати реляційну таблицю.

\* **@return** об'єкт типу {@link HashMap}, який є представленням реляційної таблиці. Об'єкт може не містити даних у випадку, коли

\* вхідний параметр не розмічений за допомогою анотації {@link Stored}. Якщо вхідний параметр - <code>null</code>, то метод

\* повертає <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**T**> HashMap<String,Object> **entityToMap**(**T** instance) **throws** DAOConvertationException{

**if**(instance == **null**)

**return** **null**;

HashMap<String, Object> resultMap = **new** HashMap<>();

HashMap<String,Field> storedFields = *getORMap*(instance.getClass(),Primary.**class**);

**for**(Object key : storedFields.keySet()){

String tableColumn = ((Stored)storedFields.get(key).getAnnotation(Stored.**class**)).name();

resultMap.put(tableColumn, *getStringValue*(instance, storedFields.get(key)));

}

**return** resultMap;

}

/\*\*

\* Метод створює об'єкт класу, вказанного у параметрі, використовуючи реляційну таблицю, що представлена об'єктом класу

\* {@link Map}.

\*

\* **@param** instanceClass об'єктний тип, до якого необхідно привести дані з реляційної таблиці.

\* **@param** map представлення реляційної таблиці.

\* **@return** об'єкт типу, зазначеного у вхідному параметрі. Якщо цей параметр - <code>null</code>, то

\* метод повертає <code>null</code>.

\* **@throws** DAOConvertationException помилка при конвертації даних

\*/

**public** **static** <**T**> **T** **mapToEntity**(Class instanceClass, Map<String,Object> map) **throws** DAOConvertationException{

**if**(map == **null** || instanceClass == **null** || map.isEmpty())

**return** **null**;

HashMap<String,Field> ORMap = *getORMap*(instanceClass,**null**);

**T** instance = **null**;

**try** {

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

**for**(String column : map.keySet()){

Field currentField = ORMap.get(column);

PropertyDescriptor pd = **new** PropertyDescriptor(currentField.getName(),instanceClass);

pd.getWriteMethod().invoke(instance,*getValueConverter*(currentField).toValue(map.get(column).toString()));

}

} **catch** (InstantiationException | IllegalAccessException | IntrospectionException | IllegalArgumentException |

InvocationTargetException | SecurityException e) {

**throw** **new** DAOConvertationException("Cистемна помилка при конвертації даних."+e.getMessage());

}

**return** instance;

}

} **package** exceptions;

/\*\*

\* Клас помилки, що викликається при виникненні помилки будь-якого характеру, що пов'язана

\* з базою даних.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** DAOException **extends** Exception {

**private** **static** **final** **long** serialVersionUID = -5756606380897370471L;

**public** **DAOException**() {

**super**();

}

**public** **DAOException**(String message, Throwable cause,

**boolean** enableSuppression, **boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

}

**public** **DAOException**(String message, Throwable cause) {

**super**(message, cause);

}

**public** **DAOException**(String message) {

**super**(message);

}

**public** **DAOException**(Throwable cause) {

**super**(cause);

}

}

**package** exceptions;

/\*\*

\* Клас помилки, що викликається при виконанні операцій, пов'язаних з об'єктно-реляційним

\* відображенням.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** DAOConvertationException **extends** Exception {

**private** **static** **final** **long** serialVersionUID = 4594290276973619301L;

**public** **DAOConvertationException**(String message, Throwable cause,

**boolean** enableSuppression, **boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

// **TODO** Auto-generated constructor stub

}

**public** **DAOConvertationException**(String message, Throwable cause) {

**super**(message, cause);

// **TODO** Auto-generated constructor stub

}

**public** **DAOConvertationException**(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

**public** **DAOConvertationException**(Throwable cause) {

**super**(cause);

// **TODO** Auto-generated constructor stub

}

**public** **DAOConvertationException**() {

// **TODO** Auto-generated constructor stub

}

}

**package** exceptions;

**import** java.io.IOException;

/\*\*

\* Клас, створений з метою опису помилок, пов'язаних з невідповідністю об'єкта класа {@link java.io.File}

\* специфічним вимогам.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ"

\* **@version** 1.0

\*/

**public** **class** FileFormatException **extends** IOException {

**private** **static** **final** **long** serialVersionUID = -1474890536790265900L;

/\*\*

\* Конструктор класу без параметрів. Викликає конструктор суперкласу.

\*/

**public** **FileFormatException**() {

**super**();

}

/\*\*

\* Конструктор класу з параметрами, що створює новий об'єкт типу {@code FileFormatException} зі спеціальним

\* рядком - описом помилки.

\*

\* **@param** message рядок - опис помилки

\*/

**public** **FileFormatException**(String message) {

**super**(message);

}

/\*\*

\*

\*/

**public** **FileFormatException**(Throwable cause) {

**super**(cause);

}

**public** **FileFormatException**(String message, Throwable cause) {

**super**(message, cause);

}

}

/\*\*

\*

\*/

package exceptions;

@SuppressWarnings("serial")

public class ConvertationException extends Exception {

/\*\*

\*

\*/

public ConvertationException() {

// TODO Auto-generated constructor stub

}

/\*\*

\* @param message

\*/

public ConvertationException(String message) {

super(message);

// TODO Auto-generated constructor stub

}

/\*\*

\* @param cause

\*/

public ConvertationException(Throwable cause) {

super(cause);

// TODO Auto-generated constructor stub

}

/\*\*

\* @param message

\* @param cause

\*/

public ConvertationException(String message, Throwable cause) {

super(message, cause);

// TODO Auto-generated constructor stub

}

/\*\*

\* @param message

\* @param cause

\* @param enableSuppression

\* @param writableStackTrace

\*/

public ConvertationException(String message, Throwable cause,

boolean enableSuppression, boolean writableStackTrace) {

super(message, cause, enableSuppression, writableStackTrace);

// TODO Auto-generated constructor stub

}

}

**package** Configurators;

**import** exceptions.FileFormatException;

/\*\*

\* Клас розширює можливості {@link FileConfigurator} та інкапсулює логіку реалізації конфігурування

\* за допомогою мови XML.Згідно з шаблоном проектування "Міст" клас виконує роль елемента

\* ієрархії реалізації абстракції.

\*

\* **@see** FileConfigurator

\* **@see** FileFormatException

\* **@see** java.io.File

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ"

\* **@version** 1.0

\*/

**abstract** **class** XMLConfigurator **extends** FileConfigurator {

/\*\*

\* Конструктор без параметрів, викликає конструктор без параметрів суперкласу.

\*/

**protected** **XMLConfigurator**(){

**super**();

}

/\*\*

\* Конструктор, який ініціалізує значення поля {@link FileConfigurator#configFile}, використовуючи строковий параметр

\*

\* **@param** s шлях до файлу конфігурацій

\*/

**protected** **XMLConfigurator**(String s) {

**super**(s);

}

/\*\*

\* {@inheritDoc}

\* Додатковою умовою є розширення файлу - мусить мати розширення <code>.xml</code>.

\*/

@Override

**protected** **void** **validator**() **throws** FileFormatException{

**super**.validator();

**if**(!(configFile.getName().endsWith(".xml")))

**throw** **new** FileFormatException("Ошибка. Расширение файла - не .xml");

}

}

**package Configurators;**

**import java.io.IOException;**

**import java.util.Properties;**

**import java.util.Stack;**

**import javax.naming.ConfigurationException;**

**import javax.xml.parsers.ParserConfigurationException;**

**import javax.xml.parsers.SAXParser;**

**import javax.xml.parsers.SAXParserFactory;**

**import org.xml.sax.Attributes;**

**import org.xml.sax.SAXException;**

**import org.xml.sax.helpers.DefaultHandler;**

**import exceptions.FileFormatException;**

**public class SAXConfigurator extends XMLConfigurator{**

**private Stack<String> tagsPath;**

**private Properties properties;**

**public SAXConfigurator(){**

**}**

**public SAXConfigurator(Properties p){**

**properties = p;**

**}**

**private class SAXHandler extends DefaultHandler{**

**SAXHandler() {**

**tagsPath = new Stack<>();**

**}**

**@Override**

**public void startElement(String uri, String localName, String qName,**

**Attributes atts) throws SAXException {**

**tagsPath.push(qName);**

**String path = makeTagsPath();**

**if(atts.getLength() != 0){**

**for (int i = 0; i < atts.getLength(); i++) {**

**properties.put(path+"#"+atts.getQName(i), atts.getValue(i));**

**}**

**}**

**}**

**@Override**

**public void endElement(String uri, String localName, String qName)**

**throws SAXException {**

**tagsPath.pop();**

**}**

**@Override**

**public void characters(char[] ch, int start, int length)**

**throws SAXException {**

**String data = new String(ch, start, length);**

**properties.put(makeTagsPath(), data);**

**data = null;**

**}**

**private String makeTagsPath(){**

**String result = "";**

**String[] tags = (String[]) tagsPath.toArray(new String[0]);**

**for (int i = 0; i < tags.length; i++) {**

**result += tags[i] +".";**

**}**

**return result.substring(0, result.length() - 1);**

**}**

**}**

**@Override**

**boolean save() throws ConfigurationException {**

**// TODO Auto-generated method stub**

**return false;**

**}**

**@Override**

**protected boolean load() throws FileFormatException,ConfigurationException{**

**try {**

**SAXParserFactory factory = SAXParserFactory.*newInstance*();**

**SAXParser parser = factory.newSAXParser();**

**SAXHandler handler = new SAXHandler();**

**parser.parse(configFile, handler);**

**return true;**

**} catch (ParserConfigurationException | SAXException | IOException e) {**

**e.printStackTrace();**

**throw new ConfigurationException("Системна помилка при конфігуруванні SAX-парсеру");**

**}**

**}**

**public Properties getProperties() {**

**return properties;**

**}**

**public void setProperties(Properties properties) {**

**this.properties = properties;**

**}**

**}**

**package** Configurators;

**import** javax.naming.ConfigurationException;

**import** exceptions.FileFormatException;

/\*\*

\* У шаблоні проектування "Міст" даний клас відіграє базовий інтерфейс реалізації функцій системи конфігурування.

\* Він агрегується {@link AbstractConfigurator} та реалізує запити на завантаження та збереження конфігурацій

\* згідно з реалізацією шаблону "Міст". Також клас об'явлений з модифікатором доступу за умовчуванням з метою

\* інкапсуляції складної логіки конфігураційної системи від зовнішнього середовища згідно шаблону проектування

\* "Фасад", у якому клас виконуэ роль элемента підсистеми.

\* <p>

\* Клас об'явлений як абстрактний з метою заборони створювати об'єкти від нього, хоча він не містить абстрактних методів,

\* які, у свою чергу, реалізують досить базові реалізації для ієрархії реалізацій підсистеми конфігурування.

\* </p>

\*

\* **@see** AbstractConfigurator

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**abstract** **class** IConfigurator {

/\*\*

\* Метод створений для конфігурування програмного додатку. Має бути перевизначенним з метою розширення

\* функціональності для завантаження конфігураційних даних з різних джерел даних: зовнішніх файлів з

\* різними розширеннями, баз даних, і т.д.

\*

\* **@return** В підкласах має повертати <code>true</code>, якщо операція завантаження

\* конфігураційних даних виконана успішно.

\*/

**abstract** **boolean** **load**() **throws** FileFormatException, ConfigurationException;

/\*\*

\* Метод створений для збереження конфігураційних даних програмного додатку. Має бути перевизначенним

\* з метою розширення функціональності для збереження даних у різні джерела даних: зовнішні файли з

\* різними розширеннями, бази даних, і т.д.

\*

\* **@return** В підкласах має повертати <code>true</code>, якщо операція збереження

\* конфігураційних даних виконана успішно.

\* **@throws** ConfigurationException

\*/

**abstract** **boolean** **save**() **throws** ConfigurationException;

}

**package** Configurators;

**import** java.io.File;

**import** javax.naming.ConfigurationException;

**import** exceptions.FileFormatException;

/\*\*

\* Клас розширює можливості {@link IConfigurator} та інкапсулює логіку реалізації конфігурування через

\* зовнішні файли з різними розширеннями. Згідно з шаблоном проектування "Міст" клас виконує роль елемента

\* ієрархії реалізації абстракції.

\*

\* **@see** IConfigurator

\* **@see** FileFormatException

\* **@see** File

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**abstract** **class** FileConfigurator **extends** IConfigurator {

/\*

\* Поле містить посилання на зовнішній файл, з якого відбуватиметься конфігурування системи.

\*/

**protected** File configFile;

/\*\*

\* Конструктор без параметрів, викликає конструктор суперкласу.

\*/

**protected** **FileConfigurator**(){

**super**();

}

/\*\*

\* Конструктор ініціалізує поле {@link #configFile} за допомогою параметру.

\*

\* **@param** s шлях до файлу конфігурацій.

\*/

**protected** **FileConfigurator**(String s) {

**super**();

configFile = **new** File(s);

}

/\*\*

\* {@inheritDoc}

\* <p>

\* Метод виконує завантаження конфігурацій із зовнішнього файлу, зазначеного в {@link #configFile}.

\* Спочатку відбувається виклик методу суперкласу, а потім викликаєтсья метод {@link #validator()}.

\* </p>

\*

\* **@throws** FileFormatException

\* **@throws** ConfigurationException

\* **@see** FileFormatException

\*/

@Override

**protected** **boolean** **load**() **throws** FileFormatException, ConfigurationException{

validator();

**return** **true**;

}

/\*\*

\* Метод перевіряє файл, записаний в поле {@link #configFile}, на специфічні умови конкретних реалізацій

\* системи конфігурацій. Якщо файл не пройшов перевірку, метод ініціює помилку {@link FileFormatException}.

\* <p>

\* Метод перевіряє, чи існує об'єкт класу {@link File} в системі та чи не є він директорією.

\* </p>

\*

\* **@throws** FileFormatException якщо значення поля {@link #configFile} не відповідає специфічним умовам

\* конфігуратора.

\*

\* **@see** FileFormatException

\*/

**protected** **void** **validator**() **throws** FileFormatException{

**if**((!configFile.exists() || configFile.isDirectory())){

**throw** **new** FileFormatException("Ошибка. Файл не существует или является директорией");

}

}

}

**package** Configurators;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Properties;

**import** java.util.Scanner;

**import** javax.naming.ConfigurationException;

**import** exceptions.FileFormatException;

/\*\*

\* Клас надає єдиний уніфікований інтерфейс до підсистеми конфігурування, тим самим зменшуючи залежності між даною

\* підсистемою та підсистемами-користувачами та полегшуючи її використання. Клас може існувати в одному екземплярі.

\* <p>

\* Клас має завантажувати конфігураційні дані при запуску програмного додатку. Головний файл конфігурацій має бути

\* прописаним у системних змінних з назвою <tt>PROJECT\_HOME<tt>, клас конфігуратору, що має завантажити конфігураційні

\* дані, має бути записаним у системних змінних з назвою <tt>PROJECT\_CONFIGURATOR</tt>.

\* Конфігураційні дані зберігаються в об'єкті класу {@link Properties}.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** Properties

\* **@see** AbstractConfigurator

\*

\*/

**public** **class** ConfigFacade {

**private** **final** Properties properties;

/\*\*

\* Реалізація шаблону 'Singleton'

\*/

**private** **static** **final** ConfigFacade facade = **new** ConfigFacade(System.*getenv*("PROJECT\_HOME"));

/\*\*

\* об'єкт класу {@link AbstractConfigurator}

\*/

**private** AbstractConfigurator configurator;

/\*\*

\* об'єкт класу {@link JDOMConfigurator}

\*/

**private** JDOMConfigurator jdomConfig;

/\*\*

\* Список конфігураційних файлів.

\*/

**private** ArrayList<File> configList = **new** ArrayList<>();

/\*\*

\* Повертає значення поля {@link #facade}

\*

\* **@return** значення поля {@link #facade}

\*/

**public** **static** ConfigFacade **getInstance**(){

**return** facade;

}

/\*\*

\* Метод зберігає конфігураційні дані в зовнішній файл.

\*/

**public** **void** **storeConfigures**(){

**try** {

configurator.saveConfigure();

} **catch** (ConfigurationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Конструктор, який ініціалізує значення полів класу та завантажує конфігураційні дані.

\*

\* **@param** filePath головний файл, який містить список інших конфігураційних файлів.

\*/

**private** **ConfigFacade**(String filePath){

properties = **new** Properties();

jdomConfig = **new** JDOMConfigurator(properties);

configurator = **new** AbstractConfigurator();

configurator.configurator = jdomConfig;

updateConfigureFiles(filePath);

**for**(File f: configList){

**try** {

loadConfigures(f);

} **catch** (FileFormatException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (ConfigurationException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

/\*\*

\* Метод завантажує список конфігураційних файлів.

\*

\* **@param** mainFile файл, у якому зберігається список конфігураційних файлів.

\*/

**private** **void** **updateConfigureFiles**(String mainFile){

Scanner scanner = **null**;

**try** {

scanner = **new** Scanner(**new** File(mainFile));

**while**(scanner.hasNext())

configList.add(**new** File(scanner.nextLine()));

}**catch** (FileNotFoundException e) {

System.out.println("Ошибка. Файл не найден.");

}**finally**{

scanner.close();

}

}

/\*\*

\* Метод завантажує конфігураційні дані з файлу.

\*

\* **@param** f файл, що містить конфігураційні дані.

\* **@return** булеве значення, що означає результат завантаження даних.

\* **@throws** FileFormatException

\* **@throws** ConfigurationException

\*/

**private** **boolean** **loadConfigures**(File f) **throws** FileFormatException, ConfigurationException {

jdomConfig.configFile = f;

**return** configurator.loadConfigure();

}

/\*\*

\* Повертає значення поля {@link #properties}

\*

\* **@return** значення поля {@link #properties}

\*/

**public** Properties **getProperties**() {

**return** properties;

}

}

**package** Configurators;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.naming.ConfigurationException;

**import** exceptions.FileFormatException;

/\*\*

\* Клас інкапсулює загальну логіку роботи конфігураційної системи, в основі якої лежить шаблон проектування'' Міст''.

\* Згідно з шаблоном цей клас є початковим в ієрархії абстракцій та агрегує інтерфейс(яким є абстрактний клас

\* {@link IConfigurator})ієрархії реалізацій конфігураційної системи та делегує йому запити на завантаження чи

\* збереження конфігураційних даних. Клас об'явлений з модифікатором доступу за умовчуванням з метою інкапсуляції

\* логіки конфігурування системи у пакеті та згідно з реалізацією шаблона проектування "Фасад", у якому даний клас

\* відіграє роль компонента підсистеми.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** IConfigurator

\* **@see** List

\* **@see** ArrayList

\*/

**class** AbstractConfigurator {

/\*\*

\* Клас ієрархії реалізацій конфігураційної системи, що реалізує інтерфейс {@link IConfigurator}.

\*/

**protected** IConfigurator configurator;

/\*\*

\* Конструктор класу без параметрів. \*

\*/

**AbstractConfigurator**(){

**super**();

}

/\*\*

\* Конструктор класу. Ініціалізує значення поля {@link #configurator}.

\*

\* **@param** aConfigurator об'єкт типу {@link IConfigurator}.

\* **@param** list об'єкт типу {@link List}.

\*/

**AbstractConfigurator**(IConfigurator aConfigurator, List<String> list){

configurator = aConfigurator;

}

/\*\*

\* Метод виконує завантаження конфігурацій. У методі викликається метод {@link IConfigurator#load()},

\* результат виконання якого записується в локальну булеву змінну.

\*

\* **@return** булеве значення результату операції завантаження конфігурацій.

\* **@throws** ConfigurationException

\* **@throws** FileFormatException

\*/

**protected** **boolean** **loadConfigure**() **throws** FileFormatException, ConfigurationException{

**return** configurator.load();

}

/\*\*

\* Метод виконує запис конфігураційних даних.

\*

\* **@return** булеве значення результату операції запису конфігураційних даних.

\* **@throws** ConfigurationException

\*/

**protected** **boolean** **saveConfigure**() **throws** ConfigurationException{

**return** configurator.save();

}

}

**package** app;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** command.AbstractCommand;

**import** exception.RequestException;

**import** exception.ValidationException;

/\*\*

\* Клас виконує перевірку на правильність конфігурування об'єкту-команди {@link AbstractCommand}

\* об'єктом-контекстом {@link Context}. У разі не проходження перевірки програмний додаток може

\* аварійно завершити роботу, тому що користувач не в змозі виправити помилку конфігурування команди.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** AbstractCommand

\* **@see** Context

\*/

**public** **class** Validator {

/\*\*

\* Статичний метод, що перевіряє відповідність даних, інкапсульованих в об'єкті контексті, вимогам

\* об'єкту команди, що інкапсульовані в анотації {@link ContextAnnotation}.

\*

\* **@param** c об'єкт типу {@link Context}, що має пройти валідацію.

\* **@param** source об'єкт типу {@link AbstractCommand}, що встановлює умови,

\* які мають бути виконані в об'єкті контексті

\* **@return** <code>true</code>, якщо валідація пройшла успішно.

\* **@throws** ValidationException якщо данних, що мають бути обов'язково у контексті, не існує або вони

\* не відповідають вимогам, задекларованим у команді.

\* **@throws** RequestException якщо команда-параметр або контекст - <code>null</code> або команда не

\* розмічена анотаціями {@link annotation.Command} , {@link ContextAnnotation}.

\*/

**public** **static** **boolean** **isValide**(Context c, AbstractCommand source) **throws** ValidationException, RequestException {

**if**(source == **null**)

**throw** **new** RequestException("Системна помилка. Об'єкт-команда має буди переданий як параметр. Робота програми буде завершена.");

**if**(c == **null** && source.getClass().getAnnotation(Command.**class**).optionalContext() == **false**)

**throw** **new** RequestException("Системна помилка. Команда має обов'язково містити контекст. Робота програми буде завершена.");

**else** **if**(c == **null**)

**return** **true**;

Parameter[] valideParams = source.getClass().getAnnotation(ContextAnnotation.**class**).list();

**for**(Parameter p : valideParams){

Object realParam = c.getValue(p.key());

**if**((realParam == **null** || realParam.getClass() != p.type() ) && p.optional() == **false**)

**throw** **new** ValidationException("Помилка при підготовці даних до запиту. Робота програми буде завершена.");

}

**return** **true**;

}

}

**package** app;

**import** command.AbstractCommand;

**import** exception.RequestException;

**import** exception.ValidationException;

**import** global.GlobalContext;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

/\*\*

\* Клас є контролером в архітектурі MVC. Його завдання полягає реалізації життєвого циклу

\* запиту від об'єкту-відображення до об'єкту-команди. Контролер задає однопотокову схему

\* виконання об'єктів-команд типу FIFO.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** Executors

\* **@see** ExecutorService

\*/

**public** **class** Controller {

/\*\*

\* об'єкт типу {@link Executors}, що інкапсулює в собі логіку реалізації

\* схеми мультипоточності програмного додатку.

\*/

**protected** **final** ExecutorService queue;

/\*\*

\* Реалізація шаблону 'Singleton'.

\*/

**private** **static** **final** Controller controller = **new** Controller();

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #queue}, задавши логіку

\* реалізації однопотокового виконання запитів.

\*/

**private** **Controller**(){

queue = Executors.*newSingleThreadExecutor*();

}

/\*\*

\* Метод додає команду в поток виконання запитів. Відбувається пошук команди в

\* фабриці команд {@link CommandFactory} та валідація контексту в об'єкті-валідаторі

\* {@link Validator}. Якщо валідація пройшла успішно, команда відправляється до потоку

\* виконання запитів.

\*

\* **@param** context об'єкт типу {@link Context}

\* **@param** CommandID ідентифікатор команди

\*/

**public** **void** **addRequest**(Context context,String CommandID){

**try** {

AbstractCommand c = CommandFactory.*getFactory*().getCommand(CommandID);

**if**(Validator.*isValide*(context, c)){

c.setContext(context);

queue.execute(c);

}

} **catch** (RequestException e) {

GlobalContext.*showSystemErrorMessage*(e.getMessage());

e.printStackTrace();

} **catch** (ValidationException e) {

GlobalContext.*showSystemErrorMessage*(e.getMessage());

e.printStackTrace();

}

}

**public** **static** Controller **getController**() {

**return** controller;

}

}

**package** app;

**import** java.util.HashMap;

**import** View.AbstractView;

/\*\*

\* Клас інкапсулює в собі інформацію, необхідну для правильної роботи об'єктів-команд

\* {@link command.AbstractCommand}. Формується об'єктами-відображеннями {@link AbstractView}.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** AbstractView

\* **@see** command.AbstractCommand

\*/

**public** **class** Context {

/\*\*

\* Мапа, що має інформацію, необхідну для правильного функціонування команд.

\*/

**private** HashMap<String,Object> context;

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **Context**(){

context = **new** HashMap<>();

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #context} об'єктом-параметром

\* типу {@link HashMap}.

\*

\* **@param** context об'єкт типу {@link HashMap}.

\*/

**public** **Context**(HashMap<String, Object> context) {

**this**.context = context;

}

/\*\*

\* Метод повертає значення з об'єкту-мапи, що має ключ,

\* указаний у параметрі.

\*

\* **@param** key ключ, згідно з яким буде повернене значення з {@link #context}.

\*

\* **@return** значення в мапі, що відповідає ключу-параметру. <code>null</code>,

\* якщо ключ-параметр - <code>null</code>.

\*/

**public** Object **getValue**(String key){

**if**(key == **null**)

**return** **null**;

**return** context.get(key);

}

/\*\*

\* Метод додає пару "ключ-значення" до мапи.

\* **@param** key ключ

\* **@param** value значення, що відповідає заданому ключу.

\*/

**public** **void** **addItem**(String key, Object value){

context.put(key, value);

}

**public** HashMap<String, Object> **getContext**(){

**return** context;

}

}

**package** app;

**import** java.util.HashMap;

**import** java.util.Properties;

**import** annotation.Command;

**import** Configurators.ConfigFacade;

**import** command.AbstractCommand;

**import** exception.RequestException;

**import** global.GlobalContext;

/\*\*

\* Основне завдання даного класу є створення нових об'єктів-команд згідно з переданим ключем

\* у якості параметру.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** CommandFactory {

/\*\*

\* Реалізація шаблону проектування 'Singleton'

\*/

**private** **static** **final** CommandFactory factory = **new** CommandFactory();

/\*\*

\* Мапа, що містить в собі назви команд, готових до створення, та їх

\* строкові значення

\*/

**private** HashMap<String, String> readyToInstanceCommands;

/\*\*

\* Конструктор класу, конфігурує поле {@link #readyToInstanceCommands}

\*/

**private** **CommandFactory**(){

readyToInstanceCommands = **new** HashMap<>();

**try** {

configureFactory();

} **catch** (RequestException e) {

GlobalContext.*showSystemErrorMessage*(e.getMessage());

e.printStackTrace();

}

}

/\*\*

\* Метод повертає об'єкт-команду згідно з переданим значенням ключа.

\*

\* **@param** ID ключ - ID об'єкту-команди.

\* **@return** об'єкт-команду типу {@link AbstractCommand}

\* **@throws** RequestException системна помилка при створенні об'єкту-команди чи відсутності команди за

\* заданим ключем.

\*/

**public** AbstractCommand **getCommand**(String ID) **throws** RequestException{

**try** {

AbstractCommand command = (AbstractCommand) Class.*forName*(readyToInstanceCommands.get(ID)).newInstance();

**if**(command == **null**)

**throw** **new** RequestException("Системна помилка. Команди з даним ID не існує.");

**return** command;

} **catch** (InstantiationException | IllegalAccessException | ClassNotFoundException e) {

**throw** **new** RequestException("Cистена помилка при створенні об'єкту команди.");

}

}

/\*\*

\* Метод конфігурує фабрику команд відповідними класами команд та їх ключами.

\*

\* **@throws** RequestException якщо класу команди не існує порівняно з даними

\* конфігураційного файлу або команда не розмічена анотацією {@link Command}.

\* **@see** ConfigFacade

\*/

**private** **void** **configureFactory**() **throws** RequestException{

Properties props = ConfigFacade.*getInstance*().getProperties();

String start = **new** String("ApplicationConfigurations.Commands.command\_");

String end = **new** String("#class");

**boolean** flag = **true**;

**int** i = 1;

String commandClass = **null**;

String commandKey = **null**;

**while**(flag == **true**){

commandClass = props.getProperty(start + i + end);

**if**(commandClass != **null**){

**try** {

commandKey = Class.*forName*(commandClass).getAnnotation(Command.**class**).key();

**if**(commandKey == **null**)

**throw** **new** RequestException("Системна помилка. Об'єкт команди має містити анотацію типу"+ Command.**class**);

**else** readyToInstanceCommands.put(commandKey, commandClass);

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

**throw** **new** RequestException("Системна помилка. Такого типу команди не існує.");

}

}

**else** flag = **false**;

i++;

}

start = **null**;

end = **null**;

props.clear();

commandClass = **null**;

commandKey = **null**;

}

**public** **static** CommandFactory **getFactory**() {

**return** factory;

}

}

**package** annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** View.AbstractView;

**import** command.AbstractCommand;

/\*\*

\* Анотація надає мета-дані об'єктам-командам про об'єкти-відображення, які мають

\* бути оновлені у результаті діяльності об'єкту-команди.

\* <p>

\* Даною анотацією розмічаються об'єкти - команди типу {@link AbstractCommand}.

\* Містить у собі масив рядків, що зберігають значення ID об'єктів відображень

\* {@link AbstractView}.

\* </p>

\* <p>

\* Зберігається у runtime.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** AbstractCommand

\* **@see** AbstractView

\*/

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

**public** **@interface** View {

/\*\*

\* масив рядків-ключів об'єктів-відображень.

\*

\* **@return** масив рядків.

\*/

String[] **key**();

}

**package** annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** View.AbstractView;

**import** command.AbstractCommand;

/\*\*

\* Анотація інкапсулює мета-дані про зміну, що відбулась в результаті діяльності користувача.

\* <p>

\* Використовується, щоб об'єкт-команда могла внести зміни до бази даних чи іншого сховища,

\* опираючись на дані, що містить в собі дана анотація, а об'єкт-валідатор типу {@link app.Validator}

\* міг перевірити дані, інкапсульовані в об'єкті-контексті {@link app.Context}, сформовані

\* об'єктом-відображенням {@link AbstractView} в результаті діяльності користувача.

\* </p>

\* <p>

\* Зберігає в собі мета-дані про те, як має бути внесений один запис в об'єкт-контекст.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** AbstractCommand

\* **@see** AbstractView

\* **@see** app.Context

\* **@see** app.Validator

\*/

@Retention (value = RetentionPolicy.RUNTIME)

@Target (value = ElementType.TYPE)

**public** **@interface** Parameter{

/\*\*

\* Ключ, під яким має зберігатися дані в контексті.

\* **@return** рядок-ключ.

\*/

**public** String **key** ();

/\*\*

\* Тип об'єкту, що інкапсулює дані в контексті.

\*

\* **@return** тип даних.

\*/

**public** Class<?> **type**() **default** Object.**class**;

/\*\*

\* Булеве значення, яке вказує, чи обо'язково даний запис має бути включеним до контексту.

\*

\* **@return** булеве значення.

\*/

**public** **boolean** **optional**() **default** **false**;

}

**package** annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** command.AbstractCommand;

/\*\*

\* Анотація містить в собі масив анотацій типу {@link Parameter}. Розмічає об'єкти типу

\* {@link AbstractCommand}.Зберігається у runtime.

\* <p>

\* Якщо об'єкт-контекст не буде мати даних, що описані згідно з вимог даною анотацією,

\* об'єкт-команда не коректно виконає запит, що призведе до аварійної зупинки програмного

\* додатку.

\* </p>

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** AbstractCommand

\* **@see** Parameter

\*/

@Retention (value = RetentionPolicy.RUNTIME)

@Target (value = ElementType.TYPE)

**public** **@interface** ContextAnnotation {

/\*\*

\* Масив анотацій {@link Parameter}

\*

\* **@return** масив анотацій {@link Parameter}

\*/

Parameter[] **list**();

}

**package** annotation;

**import** java.lang.annotation.ElementType;

**import** java.lang.annotation.Retention;

**import** java.lang.annotation.RetentionPolicy;

**import** java.lang.annotation.Target;

**import** View.AbstractView;

**import** app.CommandFactory;

/\*\*

\* Даною анотацією помічається об'єкт - команда типу {@link command.AbstractCommand}. Ця команда надає мета-дані команди, що

\* дозволяє неявно поєднувати об'єкти - відображення та необхідну реакцію на дії користувача, що інкапсульована в

\* об'єктах типу {@link command.AbstractCommand}.

\* <p>

\* Анотація дозволяє з'ясувати ID команди та чи обов'язково потребує вона об'єкт типу {@link app.Context} для виконання

\* операцій.

\* </p>

\* Також анотація дозволяє об'єкту типу {@link CommandFactory} побудувати мапу, ключем якої є ID команди, а значенням

\* клас команди.

\* <p>

\* </p>

\* <p>

\* Дана анотація зберігається в runtime.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** command.AbstractCommand

\* **@see** app.Context

\* **@see** CommandFactory

\* **@see** AbstractView

\*/

@Retention (value = RetentionPolicy.RUNTIME)

@Target (value = ElementType.TYPE)

**public** **@interface** Command {

/\*\*

\* Повертає строкове значення ID об'єкту {@link command.AbstractCommand}. Використовується об'єктами - відображеннями

\* при запиті до контролера та об'єктом - фабрикою команд.

\*

\* **@return** строкове значення ID об'єкту {@link command.AbstractCommand}

\*/

**public** String **key**();

/\*\*

\* Вказує, чи обов'язково команда потребує вхідні дані для виконання операції.

\*

\* **@return** булеве значення.

\*/

**public** **boolean** **optionalContext**() **default** **true**;

}

**package** exception;

/\*\*

\* Помилка при валідації даних. Викликається, коли дані, підготовлені до запиту, не відповідають

\* специфічним вимогам об'єктів-команд або логічним вимогам (помилка користувача). У більшості

\* випадків носять попереджувальний характер, тому, після виводу інформації про зміст помилки,

\* керування передається користувачеві.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** ValidationException **extends** Exception {

**private** **static** **final** **long** serialVersionUID = 1L;

**public** **ValidationException**() {

// **TODO** Auto-generated constructor stub

}

**public** **ValidationException**(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

**public** **ValidationException**(Throwable cause) {

**super**(cause);

// **TODO** Auto-generated constructor stub

}

**public** **ValidationException**(String message, Throwable cause) {

**super**(message, cause);

// **TODO** Auto-generated constructor stub

}

**public** **ValidationException**(String message, Throwable cause,

**boolean** enableSuppression, **boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

// **TODO** Auto-generated constructor stub

}

}

**package** exception;

/\*\*

\* Викликається при системній помилці в результаті виконання запиту. У більшості випадків

\* результатом цієї помилки є аварійне завершення програми, тому що користувач не в змозі

\* самостійно виправити помилку.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** RequestException **extends** Exception{

/\*\*

\* **@param** message

\* **@param** cause

\* **@param** enableSuppression

\* **@param** writableStackTrace

\*/

**public** **RequestException**(String message, Throwable cause,

**boolean** enableSuppression, **boolean** writableStackTrace) {

**super**(message, cause, enableSuppression, writableStackTrace);

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@param** message

\* **@param** cause

\*/

**public** **RequestException**(String message, Throwable cause) {

**super**(message, cause);

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@param** message

\*/

**public** **RequestException**(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@param** cause

\*/

**public** **RequestException**(Throwable cause) {

**super**(cause);

// **TODO** Auto-generated constructor stub

}

/\*\*

\*

\*/

**private** **static** **final** **long** serialVersionUID = 785719621861511113L;

**public** **RequestException**() {

// **TODO** Auto-generated constructor stub

}

}

**package** command;

**import** global.GlobalContext;

**import** java.sql.ResultSet;

**import** model.Profile;

**import** annotation.Command;

**import** annotation.View;

**import** app.Context;

/\*\*

\* Команда завантажує історію замовлень користувача, після чого оновлює об'єкти-відображення,

\* задекларовані в анотації {@link View} та передає ним в якості параметру об'єкт {@link ResultSet}

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** View

\* **@see** ResultSet

\*/

@Command(key = "updateOrderHistory", optionalContext = **true**)

@View(key = { "OrderHistory" })

**public** **class** UpdateOrderHistoryCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **UpdateOrderHistoryCommand**() {

**super**();

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **UpdateOrderHistoryCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

Profile profile = (Profile) GlobalContext.*globalContext*.get("profile");

ResultSet result = crud.select("select maker,name,price,count, orderState from Goods, position,database.Order where Goods.ID in"+

"(select goods\_ID from position where Position.Order\_ID IN (select Order\_ID from OrderList where Profile\_ID = '"+profile.getID()+"'))"+

"and goods.id = position.goods\_id AND order.ID = position.order\_ID");

updateViews(result);

}

}

**package** command;

**import** global.GlobalContext;

**import** model.ContactInformation;

**import** model.Profile;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** annotation.View;

**import** app.Context;

**import** exception.RequestException;

**import** exceptions.DAOException;

/\*\*

\* Команда оновлює в базі даних контактну інформацію користувача згідно з даними, переданими

\* в об'єкті типу {@link Context}

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "updateContactInfo", optionalContext = **true**)

@View(key = {"ContactInformationPanel"})

@ContextAnnotation(list = {

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

@Parameter(key = "lastName", type = String.**class**, optional = **true**),

@Parameter(key = "city", type = String.**class**),

@Parameter(key = "adress", type = String.**class**),

@Parameter(key = "telephone", type = String.**class**),

@Parameter(key = "email", type = String.**class**)})

**public** **class** UpdateContactInfoCommand **extends** AbstractCommand{

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **UpdateContactInfoCommand**() {

**super**();

// **TODO** Auto-generated constructor stub

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **UpdateContactInfoCommand**(Context context) {

**super**(context);

// **TODO** Auto-generated constructor stub

}

@Override

**public** **void** **run**() {

**try** {

ContactInformation cInfo = **null**;

**if**(context == **null**){

cInfo = crud.read(ContactInformation.**class**,((Profile)GlobalContext.*globalContext*.get("profile")).getContactInformation\_ID());

}

**else**{

cInfo = contextToModel(ContactInformation.**class**);

cInfo.setID(((Profile)GlobalContext.*globalContext*.get("profile")).getContactInformation\_ID());

crud.update(cInfo);

GlobalContext.*showOKMessage*("Редагування контактної інформації пройшло успішно.");

}

updateViews(cInfo);

} **catch** (DAOException | RequestException e) {

processError(e.getMessage());

}

}

}

**package** command;

**import** java.util.ArrayList;

**import** exceptions.DAOException;

**import** model.Catalog;

**import** model.Goods;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** annotation.View;

**import** app.Context;

/\*\*

\* Команда оновлює вузол дерева каталогу, зчитавши необхідні дані з бази даних

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "updateCatalogNode",optionalContext = **true**)

@View(key = "CatalogPane")

@ContextAnnotation(list = { @Parameter(key = "ID",type = Integer.**class**) })

**public** **class** UpdateCatalogNodeCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **UpdateCatalogNodeCommand**() {

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **UpdateCatalogNodeCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

**try** {

ArrayList<Goods> goodsList = **null**;

ArrayList<Catalog> resultList = (ArrayList<Catalog>) crud.select(Catalog.**class**,

"Select \* from catalog where catalog.id IN (Select catalog\_ID1 from catalog\_has\_catalog where catalog\_ID = '"+context.getValue("ID")+"')");

**if**(resultList.isEmpty()){

goodsList = (ArrayList<Goods>) crud.select(Goods.**class**, "Select \* from Goods where Goods.ID IN (Select Goods\_ID from catalog\_has\_goods where Catalog\_ID = '"+ context.getValue("ID")+"')");

updateViews(goodsList.toArray());

}

**else**{

updateViews(resultList.toArray());

}

resultList.clear();

} **catch** (DAOException e) {

processError(e.getMessage());

}

}

}

**package** command;

**import** global.GlobalContext;

**import** java.sql.ResultSet;

**import** model.Profile;

**import** annotation.Command;

**import** annotation.View;

**import** app.Context;

/\*\*

\* Команда оновлює вкладку 'Кошик', зчитавши необхідні дані з бази даних.

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "updateBasket")

@View(key = {"BasketPane"})

**public** **class** UpdateBasketCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **UpdateBasketCommand**() {

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **UpdateBasketCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

ResultSet result = crud.select("select goods.ID,Maker,Name,Count,Price from basket,Goods where Profile\_ID = '"+

((Profile)GlobalContext.*globalContext*.get("profile")).getID()+"' and Basket.goods\_ID = Goods.ID");

updateViews(result);

}

}

package command;

import java.util.ArrayList;

import exception.RequestException;

import exceptions.DAOException;

import global.GlobalContext;

import model.ContactInformation;

import model.Profile;

import annotation.Command;

import annotation.ContextAnnotation;

import annotation.Parameter;

import annotation.View;

import app.Context;

/\*\*

\* Команда реєструє нового користувача у системі, попередньо перевіривши його системні дані

\* (логін, пароль) на унікальність у системі.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Command(key = "registrate", optionalContext = false)

@ContextAnnotation(list = {

@Parameter(key = "login", type = String.class, optional = false),

@Parameter(key = "password", type = String.class, optional = false),

@Parameter(key = "name", type = String.class, optional = false),

@Parameter(key = "lastName", type = String.class, optional = false),

@Parameter(key = "city", type = String.class, optional = false),

@Parameter(key = "adress", type = String.class, optional = false),

@Parameter(key = "telephone", type = String.class, optional = false),

@Parameter(key = "email", type = String.class, optional = false)})

@View(key = { "MainPane" })

public class RegistrationCommand extends AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

public RegistrationCommand() {

super();

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* @param context об'єкт, яким ініціалізується значення поля.

\*/

public RegistrationCommand(Context context) {

super(context);

}

@Override

public void run() {

try {

ArrayList<Profile> profiles = (ArrayList<Profile>) crud.select(Profile.class, "Select \* from profile where login = '"+

context.getValue("login")+"' AND password = '"+context.getValue("password")+"'");

if(!profiles.isEmpty()){

GlobalContext.showWarningmessage("Обліковий запис з даним логіном та паролем вже існує.");

return;

}

ContactInformation c = contextToModel(ContactInformation.class);

Profile p = contextToModel(Profile.class);

c = crud.insert(c);

p.setContactInformation\_ID(c.getID());

p = crud.insert(p);

GlobalContext.globalContext.put("profile", p);

updateViews(c);

GlobalContext.showOKMessage("Реєстрація пройшла успішно.");

} catch (RequestException | DAOException e) {

e.printStackTrace();

processError(e.getMessage());

}

}

}

**package** command;

**import** java.util.ArrayList;

**import** panel.RecoveryPanel;

**import** exceptions.DAOException;

**import** global.GlobalContext;

**import** model.Profile;

**import** View.AuthorizationPane;

**import** View.ViewFactory;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** annotation.View;

**import** app.Context;

/\*\*

\* Команда оновлює системні дані користувача, завантажуючи новий пароль до бази даних,

\* якщо пройдена перевірка(логін та електронна пошта співпадають з даними у базі даних).

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "recoveryPassword", optionalContext = **false**)

@ContextAnnotation(list = {

@Parameter(key = "login",type = String.**class**),

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

@Parameter(key = "email", type = String.**class**)

})

@View(key = {"LoginPanel"})

**public** **class** RecoveryPasswordCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **RecoveryPasswordCommand**() {

**super**();

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **RecoveryPasswordCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

String SQLRequest = **new** String("select profile.ID,Login,Password,contactinformation\_ID from Profile, ContactInformation Where LOGIN = '"

+context.getValue("login")+"' AND EMAIL = '"+context.getValue("email") + "'");

**try** {

ArrayList<Profile> p = (ArrayList<Profile>) crud.select(Profile.**class**, SQLRequest);

**if**(p.isEmpty()){

GlobalContext.*showWarningmessage*("Облікового запису з даним логіном або електронною поштою не існує. " +

"Перевірте правильність введених даних та повторіть запит знову.");

resetRequest();

**return**;

}

Profile current = p.get(0);

p = **null**;

current.setPassword((String) context.getValue("password"));

crud.update(current);

GlobalContext.*globalContext*.put("profile", current);

updateViews();

GlobalContext.*showOKMessage*("Відновлення паролю пройшло успішно.");

} **catch** (DAOException e) {

processError(e.getMessage()+". Програма буде закрита примусово.");

}

}

/\*\*

\* Викликається, якщо перевірка логіна те електронної пошти не пройшла успішно. Дані видаляються з форми.

\*/

**private** **void** **resetRequest**(){

AuthorizationPane pane = (AuthorizationPane) ViewFactory.*getInstance*().getView("AuthorizationPane");

RecoveryPanel panel = pane.getRecPanel();

panel.updateView();

pane.add(panel);

pane.repaint();

pane = **null**;

panel = **null**;

}

}

**package** command;

**import** java.util.ArrayList;

**import** View.AbstractView;

**import** View.ViewFactory;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** annotation.View;

**import** exception.RequestException;

**import** exceptions.DAOException;

**import** global.GlobalContext;

**import** model.Profile;

/\*\*

\* Клас здійснює перевірку на наявність даного пароля та логіна в базі даних. Якщо перевірка

\* пройдена успішно, відбувається перехід на головну вкладку програмного додатку та активізується

\* весь функціонал користувача. Об'єкт {@link Profile}, що утворюється в результаті діяльності

\* команди, заноситься в глобальний контекст {@link GlobalContext}.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** Profile

\* **@see** GlobalContext

\*/

@Command(key = "loginValidation", optionalContext = **false**)

@ContextAnnotation(list = {

@Parameter(key = "login", type = String.**class**),

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

})

@View(key = "MainPane")

**public** **class** LoginValidationCommand **extends** AbstractCommand{

@Override

**public** **void** **run**() {

**try** {

String sql = **new** String("SELECT \* FROM PROFILE WHERE LOGIN = '" + context.getValue("login")+"' AND PASSWORD = '"

+ context.getValue("password")+"'");

ArrayList<Profile> profiles = (ArrayList<Profile>) crud.select(Profile.**class**, sql);

**if**(profiles.isEmpty())

**throw** **new** RequestException("Логін чи пароль не вірний. Повторіть знову");

**if**(profiles.size() > 1)

**throw** **new** RequestException("Системна помилка. Існує декілька облікових записів з даним логіном та паролем.");

GlobalContext.*globalContext*.put("profile", profiles.get(0));

updateViews();

} **catch** (DAOException | RequestException e) {

processError(e.getMessage());

}

}

/\*\*

\* У разі виникнення помилки виводиться на екран інформаційне повідомлення з описом помилки, а потім

\* видаляються дані з форми.

\*/

@Override

**public** **void** **processError**(String message){

GlobalContext.*showWarningmessage*(message);

AbstractView view = ((ViewFactory) GlobalContext.*globalContext*.get("ViewFactory")).getView("AuthorizationPane");

view.updateView();

}

}

**package** command;

**import** java.util.ArrayList;

**import** View.AbstractView;

**import** View.ViewFactory;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** annotation.View;

**import** exception.RequestException;

**import** exceptions.DAOException;

**import** global.GlobalContext;

**import** model.Profile;

/\*\*

\* Клас здійснює перевірку на наявність даного пароля та логіна в базі даних. Якщо перевірка

\* пройдена успішно, відбувається перехід на головну вкладку програмного додатку та активізується

\* весь функціонал користувача. Об'єкт {@link Profile}, що утворюється в результаті діяльності

\* команди, заноситься в глобальний контекст {@link GlobalContext}.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** Profile

\* **@see** GlobalContext

\*/

@Command(key = "loginValidation", optionalContext = **false**)

@ContextAnnotation(list = {

@Parameter(key = "login", type = String.**class**),

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

})

@View(key = "MainPane")

**public** **class** LoginValidationCommand **extends** AbstractCommand{

@Override

**public** **void** **run**() {

**try** {

String sql = **new** String("SELECT \* FROM PROFILE WHERE LOGIN = '" + context.getValue("login")+"' AND PASSWORD = '"

+ context.getValue("password")+"'");

ArrayList<Profile> profiles = (ArrayList<Profile>) crud.select(Profile.**class**, sql);

**if**(profiles.isEmpty())

**throw** **new** RequestException("Логін чи пароль не вірний. Повторіть знову");

**if**(profiles.size() > 1)

**throw** **new** RequestException("Системна помилка. Існує декілька облікових записів з даним логіном та паролем.");

GlobalContext.*globalContext*.put("profile", profiles.get(0));

updateViews();

} **catch** (DAOException | RequestException e) {

processError(e.getMessage());

}

}

/\*\*

\* У разі виникнення помилки виводиться на екран інформаційне повідомлення з описом помилки, а потім

\* видаляються дані з форми.

\*/

@Override

**public** **void** **processError**(String message){

GlobalContext.*showWarningmessage*(message);

AbstractView view = ((ViewFactory) GlobalContext.*globalContext*.get("ViewFactory")).getView("AuthorizationPane");

view.updateView();

}

}

**package** command;

**import** exceptions.DAOException;

**import** model.Profile;

**import** global.GlobalContext;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** app.Context;

/\*\*

\* Команда видаляє дані з бази даних, що відповідають об'єктній моделі кошика.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "deleteItemFromBasket", optionalContext = **false**)

@ContextAnnotation(list = {

@Parameter(key = "GoodsName", type = String.**class**, optional = **false**),

@Parameter(key = "Price", type = Integer.**class**, optional = **false**)})

**public** **class** DeleteItemFromBasketCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **DeleteItemFromBasketCommand**() {

**super**();

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **DeleteItemFromBasketCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

String name = (String) context.getValue("GoodsName");

Integer price = (Integer) context.getValue("Price");

Profile p = (Profile) GlobalContext.*globalContext*.get("profile");

String SQL = **new** String("DELETE FROM BASKET WHERE goods\_ID IN (SELECT ID FROM Goods Where name = '"+

name+"' AND price = '"+price+"') AND Profile\_ID = '" + p.getID()+"'");

**try** {

crud.delete(SQL);

} **catch** (DAOException e) {

processError(e.getMessage());

}

name = **null**;

p = **null**;

price = **null**;

SQL = **null**;

}

}

**package** command;

**import** java.util.ArrayList;

**import** exception.RequestException;

**import** exceptions.DAOException;

**import** global.GlobalContext;

**import** model.BasketItem;

**import** model.ContactInformation;

**import** model.Goods;

**import** model.Order;

**import** model.Profile;

**import** annotation.Command;

**import** annotation.ContextAnnotation;

**import** annotation.Parameter;

**import** app.Context;

/\*\*

\* Команда додає замовлення до бази даних, після чого видаляє дані з таблиці, що відповідає

\* об'єктній моделі кошика.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@Command(key = "addOrder")

@ContextAnnotation(list = {

@Parameter(key = "payment", type = String.**class**),

@Parameter(key = "deliveryMethod", type = String.**class**),

@Parameter(key = "name", type = String.**class**, optional = **true**),

@Parameter(key = "lastName", type = String.**class**, optional = **true**),

@Parameter(key = "city", type = String.**class**, optional = **true**),

@Parameter(key = "adress", type = String.**class**, optional = **true**),

@Parameter(key = "telephone", type = String.**class**, optional = **true**),

@Parameter(key = "email", type = String.**class**, optional = **true**)

})

**public** **class** AddOrderCommand **extends** AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **AddOrderCommand**() {

// **TODO** Auto-generated constructor stub

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* **@param** context об'єкт, яким ініціалізується значення поля.

\*/

**public** **AddOrderCommand**(Context context) {

**super**(context);

}

@Override

**public** **void** **run**() {

**try** {

Profile p = (Profile) GlobalContext.*globalContext*.get("profile");

Order currentOrder = contextToModel(Order.**class**);

ContactInformation cInfo = contextToModel(ContactInformation.**class**);

ContactInformation accountContactInfo = crud.read(ContactInformation.**class**, p.getContactInformation\_ID());

**if**(cInfo != **null** && cInfo.equals(accountContactInfo) == **false**){

cInfo = crud.insert(cInfo);

context.addItem("contactInformation\_ID", cInfo.getID());

}

**else** context.addItem("contactInformation\_ID", p.getContactInformation\_ID());

ArrayList<BasketItem> basketItems = (ArrayList<BasketItem>) crud.select(BasketItem.**class**, "SELECT \* From Basket WHERE Profile\_ID = '"+p.getID()+"'");

currentOrder.setOrderState("processing");

crud.insert(currentOrder);

Goods currentGoods;

**for** (BasketItem basketItem : basketItems){

crud.insert("INSERT INTO position (count, Order\_ID, goods\_ID) VALUES ('"+basketItem.getCount()+"','"+currentOrder.getID()+"','"+basketItem.getGoodsID()+"')");

crud.insert("INSERT INTO OrderList (Profile\_ID, Order\_ID, contactInformation\_ID) VALUES ('"+p.getID()+"','"+currentOrder.getID()+"','"+context.getValue("contactInformation\_ID")+"')");

currentGoods = crud.read(Goods.**class**, basketItem.getGoodsID());

currentGoods.setNumber(currentGoods.getNumber().intValue() - basketItem.getCount().intValue());

crud.update(currentGoods);

currentGoods = **null**;

}

crud.delete("DELETE FROM Basket Where Profile\_ID = "+ p.getID());

GlobalContext.*showOKMessage*("Замовлення прийнято");

p = **null**;

currentOrder = **null**;

cInfo = **null**;

accountContactInfo = **null**;

basketItems.clear();

} **catch** (RequestException | DAOException e){

processError(e.getMessage());

}

}

}

/\*\*

\*

\*/

package command;

import exception.RequestException;

import exceptions.DAOException;

import model.BasketItem;

import annotation.Command;

import annotation.ContextAnnotation;

import annotation.Parameter;

import app.Context;

/\*\*

\* Команда додає рядок до таблиці, що відповідає об'єктному представленню кошика.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@Command(key = "addItemToBasket")

@ContextAnnotation(list = {

@Parameter(key = "goodsID",type = Integer.class),

@Parameter(key = "profileID",type = Integer.class),

@Parameter(key = "count",type = Integer.class)})

public class AddItemToBasket extends AbstractCommand {

/\*\*

\* Конструктор класу без параметрів.

\*/

public AddItemToBasket() {

// TODO Auto-generated constructor stub

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link AbstractCommand#context}

\*

\* @param context об'єкт, яким ініціалізується значення поля.

\*/

public AddItemToBasket(Context context) {

super(context);

// TODO Auto-generated constructor stub

}

@Override

public void run() {

try {

BasketItem item = contextToModel(BasketItem.class);

crud.insert(item);

} catch (RequestException | DAOException e) {

processError(e.getMessage());

}

}

}

**package** command;

**import** View.AbstractView;

**import** View.ViewFactory;

**import** annotation.Stored;

**import** annotation.View;

**import** app.ApplicationCRUD;

**import** app.Context;

**import** exception.RequestException;

**import** global.GlobalContext;

**import** java.beans.IntrospectionException;

**import** java.beans.PropertyDescriptor;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.InvocationTargetException;

**import** java.util.ArrayList;

/\*\*

\* Клас представляє кореневий елемент ієрархії команд програмного додатку. Об'єкти

\* даного типу інкапсулюють логіку відповідей на запити користувача в результаті його взаємодії

\* з програмним додатком.

\* <p>

\* У даному класі реалізовані базові методи, необхідні для реалізації функціональностей

\* об'єктів-команд, такі як: визначення об'єктів-відображень, які мають бути сповіщени

\* про зміни в конкретній моделі, перетворення об'єкту-контексту в об'єкт конкретної моделі.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** Context

\*/

**public** **abstract** **class** AbstractCommand **implements** Runnable {

/\*\*

\* Об'єкт-контектс, що необхідний для реалізації функцій команд.

\*/

**protected** Context context;

/\*\*

\* Об'єкт типу {@link ApplicationCRUD}, що інкапсулює логіку

\* взаємодії з базою даних.

\*/

**protected** ApplicationCRUD crud;;

/\*\*

\* Список об'єктів-відображень, що мають бути сповіщені про зміни в моделі

\*/

**protected** ArrayList<AbstractView> updateViewList;

/\*\*

\* Конструктор класу без параметрів, що ініціалізує значення полів {@link #crud},

\* {@link #updateViewList}.

\*/

**public** **AbstractCommand**(){

crud = (ApplicationCRUD) GlobalContext.*globalContext*.get("crud");

updateViewList = **new** ArrayList<>();

fillUpdateViewList();

}

/\*\*

\* Конструктор класу, що додатково ініціалізує значення поля {@link #context}

\*/

**public** **AbstractCommand**(Context context) {

crud = (ApplicationCRUD) GlobalContext.*globalContext*.get("crud");

updateViewList = **new** ArrayList<>();

fillUpdateViewList();

**this**.context = context;

}

/\*\*

\* Метод відображає дані контексту на тип моделі, що вказана у параметрі.

\* Ключ в контексті має відповідати назві поля у моделі.

\*

\* **@param** instance тип об'єкту, до якого мають бути приведені дані контексту

\* **@return** об'єкт моделі типу, вказаному у параметрах, заповнений даними, що

\* вказані в контексті

\* **@throws** RequestException якщо параметр - <code>null</code> чи модель

\* не розмічена анотацією {@link Stored}

\*/

**public** <**T**> **T** **contextToModel**(Class<**T**> instance) **throws** RequestException{

**if**(instance == **null** || instance.getAnnotation(Stored.**class**) == **null**)

**throw** **new** RequestException("Системна помилка. Команда не має зв'язку з жодною моделлю. Програма буде закрита примусово.");

**T** resultObject;

Field[] storedFields = instance.getDeclaredFields();

**try** {

resultObject = instance.newInstance();

**boolean** updateFields = **false**;

**for**(Field f: storedFields){

**if**(context.getValue(f.getName()) != **null**){

PropertyDescriptor pd = **new** PropertyDescriptor(f.getName(), instance);

pd.getWriteMethod().invoke(resultObject, context.getValue(f.getName()));

updateFields = **true**;

}

}

**if**(updateFields == **false**)

**return** **null**;

} **catch** (InstantiationException | IllegalAccessException | IntrospectionException

| IllegalArgumentException | InvocationTargetException e) {

**throw** **new** RequestException("Системна помилка при завантаженні об'єкту команди. Програма буде закрита примусово");

}

**return** resultObject;

}

/\*\*

\* Метод заповнює поле {@link #updateViewList}, звертаючись із запитами

\* до {@link ViewFactory} з ID об'єктів-відображень. ID взяті з анотації

\* {@link View}

\*/

**protected** **void** **fillUpdateViewList**(){

View updateViews = **this**.getClass().getAnnotation(View.**class**);

**if**(updateViews != **null**)

**for** (**int** i = 0; i < updateViews.key().length; i++) {

ViewFactory factory = ViewFactory.*getInstance*();

String key = updateViews.key()[i];

updateViewList.add(factory.getView(key));

}

}

/\*\*

\* Метод посилає запит на оновлення до об'єктів-відображень.

\*

\* **@param** m конкретні моделі, що змінилися в результаті дії команди

\*/

**public** **void** **updateViews**(Object ... m){

**for**(AbstractView view: updateViewList)

view.updateView(m);

}

/\*\*

\* Метод виводить на екран інформаційне повідомлення з системною помилкою, опис

\* якої вказаний у параметрі, після чого аварійно завершує програму.

\*

\* **@param** message опис помилки

\*/

**public** **void** **processError**(String message){

GlobalContext.*showSystemErrorMessage*(message);

System.*exit*(0);

}

**public** Context **getContext**() {

**return** context;

}

**public** **void** **setContext**(Context context) {

**this**.context = context;

}

}

**package** applicationValidators;

**import** panel.ContactInformationPanel;

**import** panel.RegistrationPanel;

**import** exception.ValidationException;

/\*\*

\* Клас перевіряє дані, введені в об'єкти типів {@link RegistrationPanel} та {@link ContactInformationPanel},

\* що є агрегатом останнього, на семантичну правильність.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** RegistrationPanel

\* **@see** ContactInformationPanel

\*/

**public** **class** RegistrationPanelValidator **extends** ApplicationValidator {

/\*\*

\* Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

\*/

**private** RegistrationPanel panel;

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #panel}

\*

\* **@param** panel об'єкт, який ініціалізує значення поля {@link #panel}

\*/

**public** **RegistrationPanelValidator**(RegistrationPanel panel) {

**this**.panel = panel;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **isSemanticValid**() **throws** ValidationException {

**if**(isEmptyField(panel.getLoginTextField()))

**throw** **new** ValidationException("Поле 'Логін' не може бути порожнім. Виправте помилку та повторіть запит знову.");

trimField(panel.getLoginTextField());

**if**(panel.getPasswordField().getPassword().length == 0)

**throw** **new** ValidationException("Поле 'Пароль' не може бути порожнім. Виправте помилку та повторіть запит знову.");

checkUpPasswords(panel.getPasswordField(), panel.getCheckUpPasswordField(), "Поля 'Пароль' та 'Підтвердження паролю' не співпадають. " +

"Виправте помилку та повторіть запит знову.");

panel.getContactInfoPanel().getValidator().isSemanticValid();

}

}

**package** applicationValidators;

**import** panel.RecoveryPanel;

**import** exception.ValidationException;

/\*\*

\* Клас перевіряє дані, введені в об'єкт типу {@link RecoveryPanel} на семантичну правильність.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** RecoveryPanel

\*/

**public** **class** RecoveryPasswordValidator **extends** ApplicationValidator {

/\*\*

\* Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

\*/

**private** RecoveryPanel panel;

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #panel}

\*

\* **@param** p об'єкт, який ініціалізує значення поля {@link #panel}

\*/

**public** **RecoveryPasswordValidator**(RecoveryPanel p){

panel = p;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **isSemanticValid**() **throws** ValidationException {

**if**(isEmptyField(panel.getLoginField()) || isEmptyField(panel.getEmailField()))

**throw** **new** ValidationException("Поле 'Логін' або 'Електронна пошта' не повинно буди порожнім. Виправте помилку та повторіть запит знову.");

trimField(panel.getLoginField());

trimField(panel.getEmailField());

**if**(isEmptyField(panel.getNewPasswordField()))

**throw** **new** ValidationException("Поле 'Пароль' не має бути порожнім. Виправте помилку та повторіть запит знову.");

checkUpPasswords(panel.getNewPasswordField(), panel.getCheckUpPasswordField(), "Поля 'Пароль' та 'Підтвердження паролю' мають співпадати. " +

"Виправте помилку та повторіть запит знову.");

}

}

**package** applicationValidators;

**import** model.Profile;

**import** panel.LoginInformationPanel;

**import** exception.ValidationException;

**import** global.GlobalContext;

/\*\*

\* Клас перевіряє дані, введені в об'єкт типу {@link LoginInformationPanel} на семантичну правильність.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** LoginInformationPanel

\*/

**public** **class** LoginInfoValidator **extends** ApplicationValidator {

/\*\*

\* Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

\*/

LoginInformationPanel panel;

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #panel}

\*

\* **@param** panel об'єкт, який ініціалізує значення поля {@link #panel}

\*/

**public** **LoginInfoValidator**(LoginInformationPanel panel){

**this**.panel = panel;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **isSemanticValid**() **throws** ValidationException {

**if**(!isEmptyField(panel.getLoginField()))

trimField(panel.getLoginField());

**if**(isEmptyField(panel.getOldPassField()) && !isEmptyField(panel.getNewPassField()))

**throw** **new** ValidationException("Для зміни паролю необхідно ввести старий пароль в поле 'Пароль'. " +

"Виправте помилку та повторіть запит знову.");

**if**(!isEmptyField(panel.getOldPassField())){

@SuppressWarnings("deprecation")

String password = panel.getOldPassField().getText();

**if**(!password.equals(((Profile)GlobalContext.*globalContext*.get("profile")).getPassword()))

**throw** **new** ValidationException("Пароль не вірний. Введіть дані в поле 'Пароль' знову.");

password = **null**;

}

checkUpPasswords(panel.getNewPassField(), panel.getСorrectPassField(), "Поля 'Новий пароль' та 'Підтвердження паролю' не співпадають." +

"Введіть дані в ці поля знову та повторіть запит.");

}

}

**package** applicationValidators;

**import** java.util.HashMap;

**import** javax.swing.JTextField;

**import** exception.ValidationException;

**import** panel.ContactInformationPanel;

/\*\*

\* Клас перевіряє дані, введені в об'єкт типу {@link ContactInformationPanel} на семантичну

\* правильність (наприклад, поле 'Ім'я' не може містити цифри, поле 'Телефон' може містити

\* виключно цифри)

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\* **@see** ContactInformationPanel

\*/

**public** **class** ContactInfoValidator **extends** ApplicationValidator{

/\*\*

\* Об'єкт-відображення, дані якого необхідно перевірити на семантичну правильність.

\*/

**private** ContactInformationPanel contactInfoPanel;

/\*\*

\* Конструктор класу без параметрів.

\*/

**public** **ContactInfoValidator**() {

}

/\*\*

\* Конструктор класу, що ініціалізує значення поля {@link #contactInfoPanel}

\*

\* **@param** panel об'єкт, який ініціалізує значення поля {@link #contactInfoPanel}

\*/

**public** **ContactInfoValidator**(ContactInformationPanel panel){

contactInfoPanel = panel;

}

/\*\*

\* Метод перевіряє поля форми на наявність символів та їх корректність (приклад корректності

\* даних наведено в описі класу)

\*

\* **@throws** ValidationException якщо поле не має символів або введені не допустимі символи.

\*/

**public** **void** **isSemanticValid**() **throws** ValidationException{

HashMap<String,JTextField> stringOnlyChars = **new** HashMap<>();

stringOnlyChars.put("Ім'я", contactInfoPanel.getNameField());

stringOnlyChars.put("Прізвище", contactInfoPanel.getLastName());

stringOnlyChars.put("Місто", contactInfoPanel.getCity());

**for**(String key: stringOnlyChars.keySet()){

isEmptyField(stringOnlyChars.get(key), key);

trimField(stringOnlyChars.get(key));

StringBuilder currentValue = **new** StringBuilder(stringOnlyChars.get(key).getText());

**for** (**int** i = 0; i < currentValue.length(); i++) {

**int** c = (**int**)currentValue.charAt(i);

**if**((c > 65 && c < 122) || (c > 1040 && c < 1103) || c == 45 || c == 1105 || c == 1108 || c == 1110 || c == 1111 || c == 1030)

**continue**;

**else** **throw** **new** ValidationException("Поле '"+ key + "' має недопустимі символи. Виправте помилку та повторіть запит знову. Символ");

}

currentValue = **null**;

}

isEmptyField(contactInfoPanel.getMobilePhone(), "Номер телефону");

StringBuilder telephone = **new** StringBuilder(contactInfoPanel.getMobilePhone().getText());

**for** (**int** i = 0; i < telephone.length(); i++) {

**int** c = (**int**)telephone.charAt(i);

**if**(c >= 48 && c <= 57 )

**continue**;

**else** **throw** **new** ValidationException("Поле 'Номер телефону' не може містити символи, відмінні від цифр.");

}

isEmptyField(contactInfoPanel.getAdress(), "Адреса");

trimField(contactInfoPanel.getAdress());

isEmptyField(contactInfoPanel.getEmail(), "Електронна пошта");

trimField(contactInfoPanel.getEmail());

}

/\*\*

\* Перевіряє поле на наявність символів. Якщо поле - пусте, генерує об'єкт помилки

\* типу {@link ValidationException}

\*

\* **@param** field поле, наявність символів у якого необхідно перевірити

\* **@param** nameField назва поля

\* **@throws** ValidationException якщо поле - пусте

\*/

**private** **void** **isEmptyField**(JTextField field, String nameField) **throws** ValidationException{

**if**(**super**.isEmptyField(field) == **true**)

**throw** **new** ValidationException("Поле '"+ nameField +"' не може буди пустим. Виправте помилку та повторіть знову.");

}

**public** ContactInformationPanel **getContactInfoPanel**() {

**return** contactInfoPanel;

}

**public** **void** **setContactInfoPanel**(ContactInformationPanel contactInfoPanel) {

**this**.contactInfoPanel = contactInfoPanel;

}

}

**package** applicationValidators;

**import** java.util.Arrays;

**import** javax.swing.JPasswordField;

**import** javax.swing.text.JTextComponent;

**import** exception.ValidationException;

/\*\*

\* Клас призначений для семантичної перевірки введених користувачем даних та їх корегування.

\* Містить реалізації перевірки даних, записаних у поля для паролей, обрізання полів від порожніх

\* символів та перевірки поля на відсутність даних у ньому.

\* <p>

\* Використовується в об'єктах-відображеннях перед формуванням контексту.

\* </p>

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **abstract** **class** ApplicationValidator {

/\*\*

\* Метод, який перевіряє введені дані в об'єкт-відображення

\* на семантичну вірність

\*

\* **@throws** ValidationException якщо дані, введені до форми, семантично не правильні.

\*/

**public** **abstract** **void** **isSemanticValid**() **throws** ValidationException;

/\*\*

\* Метод перевіряє паролі, введені до двох полів, на рівність. Якщо значення полів не однакове -

\* метод генерує помилку.

\*

\* **@param** field1 об'єкт поля, що зберігає пароль

\* **@param** field2 об'єкт поля, що зберігає пароль

\* **@param** errorMessage повідомлення, яке буде поміщене в об'єкт-помилку

\* **@throws** ValidationException якщо значення полів не співпадають

\*/

**public** **void** **checkUpPasswords**(JPasswordField field1, JPasswordField field2, String errorMessage) **throws** ValidationException{

**char**[] password1 = field1.getPassword();

**char**[] password2 = field2.getPassword();

**if**(Arrays.*equals*(password1, password2) == **false**)

**throw** **new** ValidationException(errorMessage);

}

/\*\*

\* Метод обрізає пусті символи спочатку та кінця поля.

\*

\* **@param** component текстове поле, що містить в собі дані

\*/

**public** **void** **trimField**(JTextComponent component){

component.setText(component.getText().trim());

}

/\*\*

\* Метод перевіряє поле на наявність даних в ньому.

\*

\* **@param** component текстове поле, наявність даних в якому необхідно перевірити.

\* **@return** <code>true</code>, якщо поле не містить даних.

\*/

**public** **boolean** **isEmptyField**(JTextComponent component){

**return** component.getText().isEmpty();

}

}

**package** panel;

**import** java.util.Arrays;

**import** View.AbstractView;

**import** View.AuthorizationPane;

**import** View.MainFrame;

**import** View.ViewFactory;

**import** javax.swing.JTabbedPane;

**import** exception.ValidationException;

**import** global.GlobalContext;

**import** app.Context;

**import** app.Controller;

**import** applicationValidators.RegistrationPanelValidator;

/\*\*

\* Клас створений для відображення реєстраційної форми з метою реєстрації нового користувача

\* у системі.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*

\*/

@SuppressWarnings("serial")

**public** **class** RegistrationPanel **extends** javax.swing.JPanel **implements** AbstractView{

/\*\*

\* об'єкт типу {@link RegistrationPanelValidator}

\*/

**private** RegistrationPanelValidator validator;

/\*\*

\* об'єкт типу {@link ContactInformationPanel}

\*/

**private** panel.ContactInformationPanel contactInfoPanel;

**private** javax.swing.JPasswordField checkUpPasswordField;

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JLabel jLabel2;

**private** javax.swing.JLabel jLabel3;

**private** javax.swing.JTextField loginTextField;

**private** javax.swing.JPasswordField passwordField;

**private** javax.swing.JButton registrationButton;

**private** AuthorizationPane parent;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Встановлює значення поля {@link #validator}.

\*/

**public** **RegistrationPanel**() {

initComponents();

validator = **new** RegistrationPanelValidator(**this**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

resetData();

contactInfoPanel.switchEditableAndFocusable(**true**);

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**public** **void** **resetData**() {

contactInfoPanel.resetData();

loginTextField.setText(**null**);

passwordField.setText(**null**);

checkUpPasswordField.setText(**null**);

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* **@return** створений об'єкт типу {@link Context}

\*/

**private** Context **formContext**(){

Context c = contactInfoPanel.formContext();

c.addItem("login", loginTextField.getText());

StringBuilder password = **new** StringBuilder();

**char**[] passwordChars = passwordField.getPassword();

**for** (**int** i = 0; i < passwordChars.length; i++) {

password.append(passwordChars[i]);

}

Arrays.*fill*(passwordChars, '0');

c.addItem("password",password.toString());

password = **null**;

**return** c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jLabel1 = **new** javax.swing.JLabel();

loginTextField = **new** javax.swing.JTextField();

jLabel2 = **new** javax.swing.JLabel();

jLabel3 = **new** javax.swing.JLabel();

registrationButton = **new** javax.swing.JButton();

contactInfoPanel = (ContactInformationPanel) ViewFactory.*getInstance*().getView("ContactInformationPanel");

passwordField = **new** javax.swing.JPasswordField();

checkUpPasswordField = **new** javax.swing.JPasswordField();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Реєстрація"));

jLabel1.setText("Логін:");

loginTextField.setColumns(15);

jLabel2.setText("Пароль:");

jLabel3.setText("Підтвердження паролю:");

registrationButton.setText("Реєстрація");

registrationButton.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

registrationButtonActionPerformed(evt);

}

});

passwordField.setColumns(15);

checkUpPasswordField.setColumns(15);

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(registrationButton))

.addGroup(layout.createSequentialGroup()

.addGap(20, 20, 20)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel3)

.addComponent(jLabel2)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 40, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 32, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(loginTextField, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(passwordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(checkUpPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(17, 17, 17))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addContainerGap()

.addComponent(contactInfoPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(29, 29, 29)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(loginTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(46, 46, 46)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel3)

.addComponent(checkUpPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addComponent(jLabel2)

.addComponent(passwordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(contactInfoPanel, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(registrationButton)

.addGap(5, 5, 5))

);

}// </editor-fold>//GEN-END:initComponents

**private** **void** **registrationButtonActionPerformed**(java.awt.event.ActionEvent evt) {

**try** {

validator.isSemanticValid();

} **catch** (ValidationException e) {

GlobalContext.*showWarningmessage*(e.getMessage());

**return**;

}

Controller.*getController*().addRequest(formContext(), "registrate");

MainFrame frame = (MainFrame)parent.getRootPane().getParent();

frame.setMenuEnabled(**true**);

JTabbedPane pane = frame.getjTabbedPane2();

pane.remove(pane.indexOfTab("Авторизація"));

frame.setAuthorizationPane2(**null**);

frame = **null**;

pane = **null**;

}

**public** AuthorizationPane **getAuthorizationPanel**(){

**return** parent;

}

**public** **void** **setAuthorizationPanel**(AuthorizationPane param){

parent = param;

}

/\*\*

\* **@return** the checkUpPasswordField

\*/

**public** javax.swing.JPasswordField **getCheckUpPasswordField**() {

**return** checkUpPasswordField;

}

/\*\*

\* **@return** the contactInfoPanel

\*/

**public** panel.ContactInformationPanel **getContactInfoPanel**() {

**return** contactInfoPanel;

}

/\*\*

\* **@return** the loginTextField

\*/

**public** javax.swing.JTextField **getLoginTextField**() {

**return** loginTextField;

}

/\*\*

\* **@return** the passwordField

\*/

**public** javax.swing.JPasswordField **getPasswordField**() {

**return** passwordField;

}

**public** RegistrationPanelValidator **getValidator**() {

**return** validator;

}

**public** **void** **setValidator**(RegistrationPanelValidator validator) {

**this**.validator = validator;

}

}

**package** panel;

**import** java.util.Arrays;

**import** exception.ValidationException;

**import** global.GlobalContext;

**import** app.Context;

**import** app.Controller;

**import** applicationValidators.RecoveryPasswordValidator;

**import** View.AbstractView;

**import** View.AuthorizationPane;

/\*\*

\* Клас створений з метою надання можливості користувачеві відновити пароль.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** RecoveryPanel **extends** javax.swing.JPanel **implements** AbstractView {

/\*\*

\* об'єкт типу {@link RecoveryPasswordValidator}.

\*/

**private** RecoveryPasswordValidator validator;

**private** javax.swing.JButton jButton1;

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JLabel jLabel2;

**private** javax.swing.JLabel jLabel3;

**private** javax.swing.JLabel jLabel4;

**private** javax.swing.JPasswordField newPasswordField;

**private** javax.swing.JPasswordField checkUpPasswordField;

**private** javax.swing.JTextField loginField;

**private** javax.swing.JTextField emailField;

**private** AuthorizationPane parent;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Встановлює значення поля {@link #validator}.

\*/

**public** **RecoveryPanel**() {

setValidator(**new** RecoveryPasswordValidator(**this**));

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

resetData();

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**public** **void** **resetData**() {

loginField.setText(**null**);

emailField.setText(**null**);

newPasswordField.setText(**null**);

checkUpPasswordField.setText(**null**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jLabel1 = **new** javax.swing.JLabel();

loginField = **new** javax.swing.JTextField();

jLabel2 = **new** javax.swing.JLabel();

emailField = **new** javax.swing.JTextField();

jLabel3 = **new** javax.swing.JLabel();

newPasswordField = **new** javax.swing.JPasswordField();

jLabel4 = **new** javax.swing.JLabel();

checkUpPasswordField = **new** javax.swing.JPasswordField();

jButton1 = **new** javax.swing.JButton();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Відновити пароль"));

jLabel1.setText("Логін:");

loginField.setColumns(15);

jLabel2.setText("Електронна пошта:");

emailField.setColumns(15);

jLabel3.setText("Пароль:");

newPasswordField.setColumns(15);

jLabel4.setText("Підтвердження паролю:");

checkUpPasswordField.setColumns(15);

jButton1.setText("Відновити пароль");

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1)

.addComponent(jLabel2)

.addComponent(jLabel3)

.addComponent(jLabel4))

.addGap(22, 22, 22)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(checkUpPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(newPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(emailField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(loginField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createSequentialGroup()

.addGap(99, 99, 99)

.addComponent(jButton1)))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(loginField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel2)

.addComponent(emailField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel3)

.addComponent(newPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel4)

.addComponent(checkUpPasswordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 18, Short.MAX\_VALUE)

.addComponent(jButton1)

.addContainerGap())

);

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

**try** {

validator.isSemanticValid();

} **catch** (ValidationException e) {

GlobalContext.*showWarningmessage*(e.getMessage());

**return**;

}

Context c = **new** Context();

c.addItem("login", loginField.getText());

StringBuilder password = **new** StringBuilder();

**char**[] passwordChars = newPasswordField.getPassword();

**for** (**int** i = 0; i < passwordChars.length; i++) {

password.append(passwordChars[i]);

}

Arrays.*fill*(passwordChars, '0');

c.addItem("password",password.toString());

password = **null**;

c.addItem("email", emailField.getText());

Controller.*getController*().addRequest(c,"recoveryPassword");

parent.remove(**this**);

}

/\*\*

\* **@return** the newPasswordField

\*/

**public** javax.swing.JPasswordField **getNewPasswordField**() {

**return** newPasswordField;

}

/\*\*

\* **@return** the checkUpPasswordField

\*/

**public** javax.swing.JPasswordField **getCheckUpPasswordField**() {

**return** checkUpPasswordField;

}

/\*\*

\* **@return** the loginField

\*/

**public** javax.swing.JTextField **getLoginField**() {

**return** loginField;

}

/\*\*

\* **@return** the emailField

\*/

**public** javax.swing.JTextField **getEmailField**() {

**return** emailField;

}

**public** AuthorizationPane **getAuthorizationPanel**(){

**return** parent;

}

**public** **void** **setAuthorizationPanel**(AuthorizationPane param){

parent = param;

}

**public** RecoveryPasswordValidator **getValidator**() {

**return** validator;

}

**public** **void** **setValidator**(RecoveryPasswordValidator validator) {

**this**.validator = validator;

}

}

**package** panel;

**import** java.sql.ResultSet;

**import** app.Controller;

**import** model.ApplicationTableModel;

**import** View.AbstractView;

/\*\*

\* Клас створений з метою виводу інформації про замовлення користувача.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** OrderHistory **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.JScrollPane jScrollPane1;

**private** javax.swing.JTable jTable1;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **OrderHistory**() {

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

**if**(models.length == 0)

Controller.*getController*().addRequest(**null**, "updateOrderHistory");

**if**(models.length != 0){

ApplicationTableModel model = **new** ApplicationTableModel((ResultSet)models[0]);

jTable1.setModel(model);

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jScrollPane1 = **new** javax.swing.JScrollPane();

jTable1 = **new** javax.swing.JTable();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Історія замовлень"));

jScrollPane1.setViewportView(jTable1);

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 465, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(0, 0, Short.MAX\_VALUE)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 456, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE)))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGap(0, 321, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(0, 0, Short.MAX\_VALUE)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 321, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE)))

);

}

}

**package** panel;

**import** View.AbstractView;

**import** View.AuthorizationPane;

**import** View.MainFrame;

**import** View.ViewFactory;

**import** java.util.Arrays;

**import** javax.swing.JTabbedPane;

**import** app.Context;

**import** app.Controller;

/\*\*

\* Клас створений з метою відображення форми входу в систему користувача.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** LoginPanel **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JLabel jLabel2;

**private** javax.swing.JPasswordField passwordField;

**private** javax.swing.JTextField loginField;

**private** javax.swing.JButton loginButton;

**private** javax.swing.JButton recoveryButton;

**private** javax.swing.JButton registrationButton;

**public** AuthorizationPane parent;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **LoginPanel**() {

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

parent.removeAll();

resetData();

parent.add(parent.getLoginJPanel1());

parent.getLoginJPanel1().setVisible(**true**);

parent.repaint();

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**public** **void** **resetData**() {

loginField.setText(**null**);

passwordField.setText(**null**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jLabel1 = **new** javax.swing.JLabel();

jLabel2 = **new** javax.swing.JLabel();

recoveryButton = **new** javax.swing.JButton();

registrationButton = **new** javax.swing.JButton();

loginButton = **new** javax.swing.JButton();

loginField = **new** javax.swing.JTextField();

passwordField = **new** javax.swing.JPasswordField();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Вхід"));

setPreferredSize(**new** java.awt.Dimension(260, 200));

jLabel1.setText("Логін");

jLabel2.setText("Пароль");

jLabel2.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);

recoveryButton.setText("Відновити пароль");

recoveryButton.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

recoveryButtonActionPerformed(evt);

}

});

registrationButton.setText("Реєстрація");

registrationButton.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

registrationButtonActionPerformed(evt);

}

});

loginButton.setText("Вхід");

loginButton.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

loginButtonActionPerformed(evt);

}

});

loginField.setColumns(15);

passwordField.setColumns(15);

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addComponent(registrationButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(loginButton, javax.swing.GroupLayout.PREFERRED\_SIZE, 92, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1)

.addComponent(jLabel2))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(0, 0, Short.MAX\_VALUE)

.addComponent(loginField, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(passwordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE)))))

.addContainerGap())

.addGroup(layout.createSequentialGroup()

.addGap(57, 57, 57)

.addComponent(recoveryButton)

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addComponent(passwordField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(recoveryButton)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(loginButton)

.addComponent(registrationButton))

.addContainerGap())

.addGroup(javax.swing.GroupLayout.Alignment.CENTER, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(loginField, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(18, 18, 18)

.addComponent(jLabel2)

.addGap(88, 88, 88))))

);

}

**private** **void** **recoveryButtonActionPerformed**(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_recoveryButtonActionPerformed

RecoveryPanel panel = parent.getRecPanel();

panel.updateView();

parent.add(panel);

parent.remove(**this**);

panel = **null**;

}

**private** **void** **registrationButtonActionPerformed**(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_registrationButtonActionPerformed

RegistrationPanel regPanel = parent.getRegPanel();

regPanel.updateView();

parent.add(regPanel);

regPanel = **null**;

parent.remove(**this**);

}

**private** **void** **loginButtonActionPerformed**(java.awt.event.ActionEvent evt) {

Context c = **new** Context();

c.addItem("login", loginField.getText());

StringBuilder password = **new** StringBuilder();

**char**[] passwordChars = passwordField.getPassword();

**for** (**int** i = 0; i < passwordChars.length; i++) {

password.append(passwordChars[i]);

}

Arrays.*fill*(passwordChars, '0');

c.addItem("password",password.toString());

password = **null**;

Controller.*getController*().addRequest(c, "loginValidation");

MainFrame frame = (MainFrame) ViewFactory.*getInstance*().getView("MainFrame");

frame.setMenuEnabled(**true**);

JTabbedPane pane = frame.getjTabbedPane2();

pane.removeAll();

//pane.remove(pane.indexOfTab("Авторизація"));

frame.setAuthorizationPane2(**null**);

frame = **null**;

pane = **null**;

}

}

package panel;

import java.util.Arrays;

import app.Context;

import applicationValidators.LoginInfoValidator;

import View.AbstractView;

/\*\*

\* Панель створена, щоб надати можливість користувачеві змінити інформацію облікового запису.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@SuppressWarnings("serial")

public class LoginInformationPanel extends javax.swing.JPanel implements AbstractView{

private LoginInfoValidator validator;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JLabel jLabel4;

private javax.swing.JPasswordField oldPassField;

private javax.swing.JTextField loginField;

private javax.swing.JPasswordField newPassField;

private javax.swing.JPasswordField СorrectPassField;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Встановлює значення поля {@link #validator}.

\*/

public LoginInformationPanel() {

initComponents();

setValidator(new LoginInfoValidator(this));

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void updateView(Object... models) {

resetData();

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

private void resetData() {

loginField.setText(null);

oldPassField.setText(null);

newPassField.setText(null);

СorrectPassField.setText(null);

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* @return створений об'єкт типу {@link Context}

\*/

public Context formContext(){

Context c = new Context();

c.addItem("login", loginField.getText());

StringBuilder newPassword = new StringBuilder();

char[] passwordChars = newPassField.getPassword();

for (int i = 0; i < passwordChars.length; i++) {

newPassword.append(passwordChars[i]);

}

Arrays.fill(passwordChars, '0');

System.out.println(newPassword);

c.addItem("password",newPassword.toString());

newPassword = null;

return c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void initComponents() {

jLabel1 = new javax.swing.JLabel();

loginField = new javax.swing.JTextField();

jLabel3 = new javax.swing.JLabel();

newPassField = new javax.swing.JPasswordField();

СorrectPassField = new javax.swing.JPasswordField();

jLabel4 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

oldPassField = new javax.swing.JPasswordField();

setBorder(javax.swing.BorderFactory.createTitledBorder("Інформація облікового запису"));

setFocusCycleRoot(true);

setFont(new java.awt.Font("SansSerif", 0, 12));

jLabel1.setFont(new java.awt.Font("SansSerif", 0, 12));

jLabel1.setText("Логін:");

jLabel1.setFocusCycleRoot(true);

loginField.setColumns(15);

jLabel3.setFont(new java.awt.Font("SansSerif", 0, 12)); // NOI18N

jLabel3.setText("Новий пароль:");

jLabel3.setFocusCycleRoot(true);

newPassField.setColumns(15);

СorrectPassField.setColumns(15);

jLabel4.setFont(new java.awt.Font("SansSerif", 0, 12)); // NOI18N

jLabel4.setText("Підтвердження паролю:");

jLabel4.setFocusCycleRoot(true);

jLabel2.setFont(new java.awt.Font("SansSerif", 0, 12)); // NOI18N

jLabel2.setText("Пароль:");

oldPassField.setColumns(15);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);

this.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 35, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel2, javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jLabel3, javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jLabel4, javax.swing.GroupLayout.Alignment.TRAILING))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)

.addComponent(newPassField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(СorrectPassField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(oldPassField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(loginField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(22, 22, 22)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(loginField, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel2)

.addComponent(oldPassField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(newPassField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel4)

.addComponent(СorrectPassField, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

}

/\*\*

\* @return the oldPassField

\*/

public javax.swing.JPasswordField getOldPassField() {

return oldPassField;

}

/\*\*

\* @return the loginField

\*/

public javax.swing.JTextField getLoginField() {

return loginField;

}

/\*\*

\* @return the newPassField

\*/

public javax.swing.JPasswordField getNewPassField() {

return newPassField;

}

/\*\*

\* @return the сorrectPassField

\*/

public javax.swing.JPasswordField getСorrectPassField() {

return СorrectPassField;

}

public LoginInfoValidator getValidator() {

return validator;

}

public void setValidator(LoginInfoValidator validator) {

this.validator = validator;

}

}

**package** panel;

**import** model.ApplicationTableModel;

**import** app.Context;

**import** app.Controller;

**import** View.AbstractView;

/\*\*

\* Клас створений з метою відображення інформації про товари, що будуть додані до наступного

\* замовлення.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** GoodsToOrderPanel **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.JButton jButton1;

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JScrollPane jScrollPane1;

**private** javax.swing.JTable jTable1;

**private** javax.swing.JTextField summaryPrice;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **GoodsToOrderPanel**() {

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

**if**(models.length != 0){

ApplicationTableModel model = (ApplicationTableModel)models[0];

jTable1.setModel(model);

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jScrollPane1 = **new** javax.swing.JScrollPane();

jTable1 = **new** javax.swing.JTable();

jButton1 = **new** javax.swing.JButton();

jLabel1 = **new** javax.swing.JLabel();

summaryPrice = **new** javax.swing.JTextField();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Склад замовлення"));

jScrollPane1.setViewportView(jTable1);

jButton1.setText("Видалити");

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jLabel1.setText("Сума замовлення:");

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**)

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(summaryPrice, javax.swing.GroupLayout.PREFERRED\_SIZE, 80, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 92, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 375, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 275, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jLabel1)

.addComponent(summaryPrice, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jButton1))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

jButton1.getAccessibleContext().setAccessibleName("");

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

**int** selectedRow = jTable1.getSelectedRow();

Context c = **new** Context();

ApplicationTableModel tableModel = (ApplicationTableModel) jTable1.getModel();

c.addItem("GoodsName",(String) tableModel.getValueAt(selectedRow, 2));

c.addItem("Price", (Integer)tableModel.getValueAt(selectedRow, 3));

tableModel.removeRow(selectedRow);

tableModel.fireTableRowsDeleted(selectedRow, selectedRow);

Controller.*getController*().addRequest(c, "deleteItemFromBasket");

**this**.repaint();

}

}

**package** panel;

**import** View.AbstractView;

**import** View.MainFrame;

**import** View.ViewFactory;

**import** javax.swing.JTabbedPane;

**import** app.Context;

**import** app.Controller;

/\*\*

\* Клас створений з метою відображення форми замовлення.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** DetailOrderPanel **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.ButtonGroup buttonGroup1;

**private** javax.swing.ButtonGroup buttonGroup2;

**private** javax.swing.ButtonGroup buttonGroup3;

**private** javax.swing.JPanel payMetod;

**private** javax.swing.JRadioButton byCash;

**private** javax.swing.JRadioButton byCard;

**private** javax.swing.JRadioButton byCourier;

**private** javax.swing.JRadioButton selfDelivery;

**private** javax.swing.JPanel deliveryMetod;

**private** javax.swing.JRadioButton fromAccount;

**private** javax.swing.JRadioButton fromOrderForm;

**private** javax.swing.JButton jButton1;

**private** javax.swing.JPanel contactInfoType;

/\*\*

\* Об'єкт типу {@link ContactInformationPanel}

\*/

**private** panel.ContactInformationPanel contactInformationPanel1;

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

resetData();

contactInformationPanel1.updateView(((ContactInformationPanel)ViewFactory.*getInstance*().getView("ContactInformationPanel")).getContactInfo());

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**public** **void** **resetData**() {

contactInformationPanel1.resetData();

}

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Викликає метод {@link ContactInformationPanel#switchEditableAndFocusable(boolean)}

\*/

**public** **DetailOrderPanel**() {

initComponents();

contactInformationPanel1.switchEditableAndFocusable(**false**);

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* **@return** створений об'єкт типу {@link Context}

\*/

**private** Context **formContext**(){

Context c = **new** Context();

**if**(byCash.isSelected())

c.addItem("payment", "by cash");

**else** c.addItem("payment", "by card");

**if**(byCourier.isSelected())

c.addItem("deliveryMethod", "by courier"); //как-то на укр перевести.

**else** c.addItem("deliveryMethod", "self delivery");

**return** c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

java.awt.GridBagConstraints gridBagConstraints;

buttonGroup1 = **new** javax.swing.ButtonGroup();

buttonGroup2 = **new** javax.swing.ButtonGroup();

buttonGroup3 = **new** javax.swing.ButtonGroup();

contactInfoType = **new** javax.swing.JPanel();

fromAccount = **new** javax.swing.JRadioButton();

fromOrderForm = **new** javax.swing.JRadioButton();

deliveryMetod = **new** javax.swing.JPanel();

byCourier = **new** javax.swing.JRadioButton();

selfDelivery = **new** javax.swing.JRadioButton();

payMetod = **new** javax.swing.JPanel();

byCash = **new** javax.swing.JRadioButton();

byCard = **new** javax.swing.JRadioButton();

contactInformationPanel1 = **new** panel.ContactInformationPanel();

jButton1 = **new** javax.swing.JButton();

setBorder(javax.swing.BorderFactory.*createTitledBorder*("Деталі оформлення замовлення"));

setMinimumSize(**new** java.awt.Dimension(368, 684));

setPreferredSize(**new** java.awt.Dimension(378, 684));

setLayout(**new** java.awt.GridBagLayout());

contactInfoType.setBorder(javax.swing.BorderFactory.*createTitledBorder*("Вид контактної інформації"));

buttonGroup1.add(fromAccount);

fromAccount.setSelected(**true**);

fromAccount.setText("з облікового запису");

fromAccount.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

fromAccountActionPerformed(evt);

}

});

buttonGroup1.add(fromOrderForm);

fromOrderForm.setText("введені дані");

fromOrderForm.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

fromOrderFormActionPerformed(evt);

}

});

javax.swing.GroupLayout contactInfoTypeLayout = **new** javax.swing.GroupLayout(contactInfoType);

contactInfoType.setLayout(contactInfoTypeLayout);

contactInfoTypeLayout.setHorizontalGroup(

contactInfoTypeLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(contactInfoTypeLayout.createSequentialGroup()

.addContainerGap()

.addGroup(contactInfoTypeLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(fromAccount)

.addComponent(fromOrderForm))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

contactInfoTypeLayout.setVerticalGroup(

contactInfoTypeLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(contactInfoTypeLayout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(fromAccount)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(fromOrderForm))

);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 0;

gridBagConstraints.gridwidth = 2;

gridBagConstraints.ipadx = 161;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(0, 20, 0, 0);

add(contactInfoType, gridBagConstraints);

deliveryMetod.setBorder(javax.swing.BorderFactory.*createTitledBorder*("Спосіб доставки"));

buttonGroup2.add(byCourier);

byCourier.setText("доставка курьером");

buttonGroup2.add(selfDelivery);

selfDelivery.setText("самовивіз");

javax.swing.GroupLayout deliveryMetodLayout = **new** javax.swing.GroupLayout(deliveryMetod);

deliveryMetod.setLayout(deliveryMetodLayout);

deliveryMetodLayout.setHorizontalGroup(

deliveryMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(deliveryMetodLayout.createSequentialGroup()

.addContainerGap()

.addGroup(deliveryMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(byCourier)

.addComponent(selfDelivery))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

deliveryMetodLayout.setVerticalGroup(

deliveryMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(deliveryMetodLayout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(byCourier)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(selfDelivery))

);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridwidth = 2;

gridBagConstraints.ipadx = 169;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(0, 20, 0, 14);

add(deliveryMetod, gridBagConstraints);

payMetod.setBorder(javax.swing.BorderFactory.*createTitledBorder*("Спосіб оплати"));

buttonGroup3.add(byCash);

byCash.setText("оплата готівкою");

buttonGroup3.add(byCard);

byCard.setText("безготівковий розрахунок");

javax.swing.GroupLayout payMetodLayout = **new** javax.swing.GroupLayout(payMetod);

payMetod.setLayout(payMetodLayout);

payMetodLayout.setHorizontalGroup(

payMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(payMetodLayout.createSequentialGroup()

.addContainerGap()

.addGroup(payMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(byCash)

.addComponent(byCard))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

payMetodLayout.setVerticalGroup(

payMetodLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(payMetodLayout.createSequentialGroup()

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(byCash)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(byCard))

);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 2;

gridBagConstraints.gridwidth = 2;

gridBagConstraints.ipadx = 127;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(0, 20, 0, 14);

add(payMetod, gridBagConstraints);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 3;

gridBagConstraints.gridwidth = 2;

gridBagConstraints.ipadx = 160;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(0, 20, 0, 14);

add(contactInformationPanel1, gridBagConstraints);

jButton1.setText("Замовити");

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 4;

gridBagConstraints.ipadx = 17;

gridBagConstraints.insets = **new** java.awt.Insets(6, 248, 0, 0);

add(jButton1, gridBagConstraints);

}// </editor-fold>//GEN-END:initComponents

**private** **void** **fromAccountActionPerformed**(java.awt.event.ActionEvent evt) {

**this**.updateView();

contactInformationPanel1.switchEditableAndFocusable(**false**);

}

**private** **void** **fromOrderFormActionPerformed**(java.awt.event.ActionEvent evt) {

contactInformationPanel1.switchEditableAndFocusable(**true**);

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

Context c = formContext();

**if**(fromOrderForm.isSelected())

c.getContext().putAll(contactInformationPanel1.formContext().getContext());

Controller.*getController*().addRequest(c, "addOrder");

JTabbedPane pane;

pane = ((MainFrame)ViewFactory.*getInstance*().getView("MainFrame")).getjTabbedPane2();

pane.removeTabAt(pane.indexOfTab("Замовлення"));

pane = **null**;

}

}

package panel;

import javax.swing.BorderFactory;

import javax.swing.GroupLayout;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.LayoutStyle;

import app.Context;

import app.Controller;

import applicationValidators.ContactInfoValidator;

import model.ContactInformation;

import View.AbstractView;

/\*\*

\* Клас створений для відображення та зміни контактної інформації користувача.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@SuppressWarnings("serial")

public class ContactInformationPanel extends JPanel implements AbstractView{

private ContactInfoValidator validator;

private JTextField adress;

private JTextField city;

private JTextField email;

private JLabel jLabel1;

private JLabel jLabel2;

private JLabel jLabel3;

private JLabel jLabel4;

private JLabel jLabel5;

private JLabel jLabel6;

private JTextField lastName;

private JTextField mobilePhone;

private JTextField nameField;

private ContactInformation info;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Встановлює значення поля {@link #validator}.

\*/

public ContactInformationPanel() {

initComponents();

setValidator(new ContactInfoValidator(this));

}

/\*\*

\* Встановлює здатність текстового компонента до зміни та отримання фокусу згідно параметру.

\*

\* @param param булеве значення.

\*/

public void switchEditableAndFocusable(boolean param){

nameField.setEditable(param);

nameField.setFocusable(param);

lastName.setEditable(param);

lastName.setFocusable(param);

email.setEditable(param);

email.setFocusable(param);

mobilePhone.setEditable(param);

mobilePhone.setFocusable(param);

adress.setEditable(param);

adress.setFocusable(param);

city.setEditable(param);

city.setFocusable(param);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void updateView(Object... models) {

info = null;

boolean flag = false;

for (int i = 0; i < models.length; i++) {

if(models[i] instanceof ContactInformation){

info = (ContactInformation) models[i];

flag = true;

break;

}

}

if(flag == true){

nameField.setText(info.getName());

lastName.setText(info.getLastName());

email.setText(info.getEmail());

adress.setText(info.getAdress());

city.setText(info.getCity());

mobilePhone.setText(info.getTelephone());

}

else Controller.getController().addRequest(null, "updateContactInfo");

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

public void resetData() {

nameField.setText(null);

lastName.setText(null);

email.setText(null);

adress.setText(null);

city.setText(null);

mobilePhone.setText(null);

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* @return створений об'єкт типу {@link Context}

\*/

public Context formContext(){

Context c = new Context();

c.addItem("name", nameField.getText());

c.addItem("lastName",lastName.getText());

c.addItem("email",email.getText());

c.addItem("telephone", mobilePhone.getText());

c.addItem("adress",adress.getText());

c.addItem("city", city.getText());

return c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void initComponents() {

jLabel1 = new JLabel();

nameField = new JTextField();

jLabel2 = new JLabel();

lastName = new JTextField();

jLabel3 = new JLabel();

email = new JTextField();

jLabel4 = new JLabel();

mobilePhone = new JTextField();

jLabel5 = new JLabel();

adress = new JTextField();

jLabel6 = new JLabel();

city = new JTextField();

setBorder(BorderFactory.createTitledBorder("Контактна інформація"));

jLabel1.setText("Ім'я:");

nameField.setColumns(15);

jLabel2.setText("Прізвище:");

lastName.setColumns(15);

jLabel3.setText("Електронна адреса:");

email.setColumns(15);

jLabel4.setText("Мобільний телефон:");

mobilePhone.setColumns(15);

jLabel5.setText("Адреса:");

adress.setColumns(15);

jLabel6.setText("Місто:");

city.setColumns(15);

GroupLayout layout = new GroupLayout(this);

this.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)

.addComponent(jLabel1, GroupLayout.Alignment.TRAILING)

.addComponent(jLabel2, GroupLayout.Alignment.TRAILING)

.addComponent(jLabel3, GroupLayout.Alignment.TRAILING)

.addComponent(jLabel4, GroupLayout.Alignment.TRAILING)

.addComponent(jLabel5, GroupLayout.Alignment.TRAILING)

.addComponent(jLabel6, GroupLayout.Alignment.TRAILING))

.addPreferredGap(LayoutStyle.ComponentPlacement.RELATED, GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.CENTER)

.addComponent(nameField, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(lastName, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(email, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(mobilePhone, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(adress, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(city, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(17, 17, 17)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(nameField, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel1))

.addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(jLabel2)

.addComponent(lastName, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(jLabel3)

.addComponent(email, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(jLabel4)

.addComponent(mobilePhone, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(jLabel5)

.addComponent(adress, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)

.addComponent(jLabel6)

.addComponent(city, GroupLayout.DEFAULT\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)))

);

}// </editor-fold>//GEN-END:initComponents

public ContactInformation getContactInfo() {

return info;

}

public void setContactInfo(ContactInformation contactInfo) {

this.info = contactInfo;

}

/\*\*

\* @return the adress

\*/

public JTextField getAdress() {

return adress;

}

/\*\*

\* @return the city

\*/

public JTextField getCity() {

return city;

}

/\*\*

\* @return the email

\*/

public JTextField getEmail() {

return email;

}

/\*\*

\* @return the lastName

\*/

public JTextField getLastName() {

return lastName;

}

/\*\*

\* @return the mobilePhone

\*/

public JTextField getMobilePhone() {

return mobilePhone;

}

/\*\*

\* @return the name

\*/

public JTextField getNameField() {

return nameField;

}

public ContactInfoValidator getValidator() {

return validator;

}

public void setValidator(ContactInfoValidator validator) {

this.validator = validator;

}

}

**package** panel;

**import** global.GlobalContext;

**import** java.awt.BorderLayout;

**import** java.awt.Image;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** javax.imageio.ImageIO;

**import** javax.swing.ImageIcon;

**import** javax.swing.JScrollPane;

**import** javax.swing.SpinnerNumberModel;

**import** app.Context;

**import** app.Controller;

**import** model.Goods;

**import** model.Profile;

**import** View.AbstractView;

**import** View.CatalogPane;

**import** View.ViewFactory;

/\*\*

\* Клас створений, щоб вивести на екран інформацію про конкретний товар та надати можливість

\* додати його до кошика.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** ConcreteGoodsPanel **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.JButton jButton1;

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JLabel jLabel2;

**private** javax.swing.JLabel jLabel3;

**private** javax.swing.JLabel jLabel4;

**private** javax.swing.JLabel jLabel5;

**private** javax.swing.JLabel jLabel6;

**private** javax.swing.JScrollPane jScrollPane1;

**private** javax.swing.JSpinner jSpinner2;

**private** javax.swing.JTextArea jTextArea1;

**private** javax.swing.JTextField goodsName;

**private** javax.swing.JTextField maker;

**private** javax.swing.JTextField count;

**private** javax.swing.JTextField price;

**private** SpinnerNumberModel spinnerModel;

**private** Goods currentGoods;

**private** CatalogPane parent;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **ConcreteGoodsPanel**() {

initComponents();

jTextArea1.setFocusable(**false**);

jTextArea1.setEditable(**false**);

goodsName.setFocusable(**false**);

goodsName.setEditable(**false**);

maker.setFocusable(**false**);

maker.setEditable(**false**);

count.setEditable(**false**);

count.setFocusable(**false**);

price.setFocusable(**false**);

price.setEditable(**false**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(**final** Object... models) {

parent.remove(**this**);

**if**(models.length != 0 && models[0] **instanceof** Goods){

parent.add(**this**, BorderLayout.CENTER);

currentGoods = **null**;

currentGoods = (Goods)models[0];

**try** {

Image resultImage = ImageIO.*read*(**new** File(currentGoods.getImagePath()));

resultImage = resultImage.getScaledInstance(jLabel1.getWidth(), jLabel1.getHeight(), Image.SCALE\_DEFAULT);

jLabel1.setIcon(**new** ImageIcon(resultImage));

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

goodsName.setText(currentGoods.getName());

maker.setText(currentGoods.getMaker());

count.setText(String.*valueOf*(currentGoods.getNumber()));

price.setText(String.*valueOf*(currentGoods.getPrice()));

String description = formatDescription(currentGoods.getDesctription());

**int** max = currentGoods.getNumber();

SpinnerNumberModel spinnerNumberModel = (SpinnerNumberModel) jSpinner2.getModel();

spinnerNumberModel.setMaximum(max);

spinnerNumberModel.setMinimum(1);

spinnerNumberModel.setValue(1);

jTextArea1.setText(description);

description = **null**;

**this**.repaint();

}

**else** {

parent.remove(**this**);

}

}

/\*\*

\* **@return** the parent

\*/

**public** CatalogPane **getParent**() {

**return** parent;

}

/\*\*

\* **@param** parent the parent to set

\*/

**public** **void** **setParent**(CatalogPane parent) {

**this**.parent = parent;

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**private** **void** **resetData**(){

goodsName.setText(**null**);

maker.setText(**null**);

count.setText(**null**);

price.setText(**null**);

jTextArea1.setText(**null**);

}

/\*\*

\* Метод форматує рядок таким чином, щоб кожне речення виводилось з нового рядка.

\*

\* **@param** description рядок, який необхідно відформатувати.

\* **@return** форматований рядок.

\*/

**private** String **formatDescription**(String description){

StringTokenizer tokenizer = **new** StringTokenizer(description,".");

StringBuilder builder = **new** StringBuilder(" ");

**while**(tokenizer.hasMoreElements())

builder.append(tokenizer.nextToken() + ".\n");

**return** builder.toString();

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* **@return** створений об'єкт типу {@link Context}

\*/

**private** Context **formContext**(){

Context c = **new** Context();

c.addItem("goodsID", currentGoods.getID());

c.addItem("profileID", ((Profile)GlobalContext.*globalContext*.get("profile")).getID());

c.addItem("count",(Integer)jSpinner2.getValue());

**return** c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jLabel1 = **new** javax.swing.JLabel();

jTextArea1 = **new** javax.swing.JTextArea();

jLabel2 = **new** javax.swing.JLabel();

goodsName = **new** javax.swing.JTextField();

jLabel3 = **new** javax.swing.JLabel();

maker = **new** javax.swing.JTextField();

jLabel4 = **new** javax.swing.JLabel();

count = **new** javax.swing.JTextField();

jLabel5 = **new** javax.swing.JLabel();

price = **new** javax.swing.JTextField();

jButton1 = **new** javax.swing.JButton();

jLabel6 = **new** javax.swing.JLabel();

jSpinner2 = **new** javax.swing.JSpinner();

jTextArea1.setColumns(20);

jTextArea1.setRows(5);

jTextArea1.setMaximumSize(**new** java.awt.Dimension(12, 28));

jTextArea1.setPreferredSize(**new** java.awt.Dimension(12, 28));

// jScrollPane1.setViewportView(jTextArea1);

jScrollPane1 = **new** javax.swing.JScrollPane(jTextArea1);

jScrollPane1.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

jLabel2.setText("Назва:");

goodsName.setColumns(15);

goodsName.setMaximumSize(**new** java.awt.Dimension(30, 28));

goodsName.setPreferredSize(**new** java.awt.Dimension(30, 28));

jLabel3.setText("Виробник:");

maker.setColumns(15);

jLabel4.setText("Кількість:");

count.setColumns(15);

jLabel5.setText("Ціна:");

price.setColumns(15);

jButton1.setText("Додати до кошика");

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jLabel6.setText("Кількість на замовлення:");

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 250, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addGap(27, 27, 27)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel6)

.addGap(19, 19, 19)

.addComponent(jSpinner2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(46, 46, 46)

.addComponent(jButton1))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel2)

.addComponent(jLabel3)

.addComponent(jLabel4)

.addComponent(jLabel5))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, **false**)

.addComponent(maker, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT\_SIZE, 140, Short.MAX\_VALUE)

.addComponent(count, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE)

.addComponent(goodsName, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE)

.addComponent(price, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE))))))

.addContainerGap(18, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(6, 6, 6)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 339, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 339, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(goodsName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel2))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(maker, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel3))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(count, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel4))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(price, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel5))

.addGap(46, 46, 46))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton1)

.addComponent(jSpinner2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel6))

.addContainerGap())))

);

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed

Controller.*getController*().addRequest(formContext(), "addItemToBasket");

resetData();

CatalogPane pane = ((CatalogPane)ViewFactory.*getInstance*().getView("CatalogPane"));

pane.remove(**this**);

pane.repaint();

}

**public** SpinnerNumberModel **getSpinnerModel**() {

**return** spinnerModel;

}

**public** **void** **setSpinnerModel**(SpinnerNumberModel spinnerModel) {

**this**.spinnerModel = spinnerModel;

}

}

package global;

import java.awt.Component;

import java.util.HashMap;

import javax.swing.JOptionPane;

import View.ViewFactory;

public class GlobalContext {

public static HashMap<String,Object> globalContext = new HashMap<>();

public static void showSystemErrorMessage(String text){

JOptionPane.showMessageDialog((Component) ViewFactory.getInstance().getView("MainFrame"), text, "Системна помилка", JOptionPane.ERROR\_MESSAGE);

}

public static void showWarningmessage(String text){

JOptionPane.showMessageDialog((Component) ViewFactory.getInstance().getView("MainFrame"), text, "Застереження", JOptionPane.WARNING\_MESSAGE);

}

public static void showOKMessage(String text){

JOptionPane.showMessageDialog((Component) ViewFactory.getInstance().getView("MainFrame"), text, "", JOptionPane.INFORMATION\_MESSAGE);

}

}

**package** View;

**import** java.util.HashMap;

**import** java.util.Properties;

**import** Configurators.ConfigFacade;

**import** exception.RequestException;

**import** global.GlobalContext;

/\*\*

\* Основне завдання даного класу є повернення об'єкту типу {@link AbstractView} за ключем,

\* що передається як параметр.

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **class** ViewFactory {

/\*\*

\* Реалізація шаблону 'Singleton'.

\*/

**private** **static** **final** ViewFactory factory = **new** ViewFactory();

/\*\*

\* завантажені об'єкти-відображення

\*/

**private** HashMap<String,AbstractView> instancedViews;

/\*\*

\* готові до завантаження об'єкти-відображення.

\*/

**private** HashMap<String,String> readyToInstanceViews;

/\*\*

\* Повертає значення поля {@link #factory}

\*

\* **@return** значення поля {@link #factory}

\*/

**public** **static** ViewFactory **getInstance**(){

**return** factory;

}

/\*\*

\* Метод конфігурує фабрику конфігураційними даними.

\*/

**private** **void** **configureFactory**(){

Properties props = ConfigFacade.*getInstance*().getProperties();

**boolean** flag = **true**;

String mainPart = **new** String("ApplicationConfigurations.Views.View\_");

String idPart = **new** String("#ID");

String classPart = **new** String("#class");

**int** inc = 1;

**while**(flag == **true**){

String propertyKey = props.getProperty(mainPart + inc + idPart);

**if**(propertyKey != **null**){

String value = props.getProperty(mainPart + inc + classPart);

readyToInstanceViews.put(propertyKey, value);

inc++;

}

**else** flag = **false**;

}

props = **null**;

mainPart = **null**;

idPart = **null**;

classPart = **null**;

}

/\*\*

\* Клас, що ініціалізує значення полів та завантажує конфігураційні дані для нього.

\*/

**public** **ViewFactory**() {

readyToInstanceViews = **new** HashMap<>();

instancedViews = **new** HashMap<>();

configureFactory();

}

/\*\*

\* Метод повертає об'єкт класу {@link AbstractView} згідно з вхідним параметром.

\* Якщо параметр - пусте значення, то програмний додаток аварійно завершує роботу.

\* Спочатку проводиться пошук об'єктів відображень з уже інстанційованих, якщо

\* об'єкт-відображення не створений, то він створюється на додається до інстанційованих.

\*

\* **@param** ID ключ, за яким шукається об'єкт-відображення.

\* **@return** об'єкт-відображення, що відповідає заданому ключеві.

\*/

**public** AbstractView **getView**(String ID){

**if**(ID.isEmpty())

**try** {

**throw** **new** RequestException("Системна помилка. ID класу відображення не може бути порожнім. Програма буде закрита примусово.");

} **catch** (RequestException e1) {

GlobalContext.*showSystemErrorMessage*(e1.getMessage());

e1.printStackTrace();

System.*exit*(0);

}

AbstractView instance = instancedViews.get(ID);

**if**(instance == **null**){

**try** {

instance = (AbstractView) Class.*forName*(readyToInstanceViews.get(ID)).newInstance();

readyToInstanceViews.remove(ID);

} **catch** (InstantiationException | IllegalAccessException | ClassNotFoundException e) {

**try** {

**throw** **new** RequestException("Системна помилка. Помилка при створенні об'єкту відображення.Програма буде закрита примусово.");

} **catch** (RequestException e1) {

GlobalContext.*showSystemErrorMessage*(e1.getMessage());

e1.printStackTrace();

System.*exit*(0);

}

}

instancedViews.put(ID, instance);

}

**return** instance;

}

}

**package** View;

**import** javax.swing.JTabbedPane;

/\*\*

\* Вкладка створена для того, щоб надати користувачеві максимальну значущу інформацію щодо

\* поточного замовлення.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** OrderPane **extends** javax.swing.JPanel **implements** AbstractView {

**private** panel.DetailOrderPanel detailOrderPanel1;

**private** panel.GoodsToOrderPanel goodsToOrderPanel1;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **OrderPane**() {

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

MainFrame frame = (MainFrame) ViewFactory.*getInstance*().getView("MainFrame");

JTabbedPane pane = frame.getjTabbedPane2();

pane.addTab("Замовлення", **this**);

pane.removeTabAt(pane.indexOfTab("Кошик"));

detailOrderPanel1.updateView(models);

goodsToOrderPanel1.updateView(models);

pane.setSelectedComponent(**this**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

java.awt.GridBagConstraints gridBagConstraints;

detailOrderPanel1 = **new** panel.DetailOrderPanel();

goodsToOrderPanel1 = **new** panel.GoodsToOrderPanel();

setMinimumSize(**new** java.awt.Dimension(787, 680));

setLayout(**new** java.awt.GridBagLayout());

detailOrderPanel1.setMinimumSize(**new** java.awt.Dimension(350, 664));

detailOrderPanel1.setPreferredSize(**new** java.awt.Dimension(355, 680));

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.anchor = java.awt.GridBagConstraints.PAGE\_START;

add(detailOrderPanel1, gridBagConstraints);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.anchor = java.awt.GridBagConstraints.PAGE\_START;

add(goodsToOrderPanel1, gridBagConstraints);

}

}

**package** View;

**import** javax.swing.JTabbedPane;

**import** panel.ContactInformationPanel;

**import** panel.OrderHistory;

/\*\*

\*Вкладка створена для того, щоб користувач мав швидкий доступ до інформації замовлень та

\*контактної інформації.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** MainPane **extends** javax.swing.JPanel **implements** AbstractView{

**private** panel.ContactInformationPanel contactInformationPanel1;

**private** panel.OrderHistory orderHistory1;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **MainPane**() {

initComponents();

contactInformationPanel1.switchEditableAndFocusable(**false**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

contactInformationPanel1.updateView(models);

orderHistory1.updateView();

MainFrame frame = (MainFrame) ViewFactory.*getInstance*().getView("MainFrame");

JTabbedPane pane = frame.getjTabbedPane2();

**int** index = pane.indexOfTab("Головна");

**if**(index >= 0)

pane.removeTabAt(index);

pane.addTab("Головна", (MainPane) ViewFactory.*getInstance*().getView("MainPane"));

frame = **null**;

pane = **null**;

}

/\*\*

\* {@inheritDoc}

\*/

**public** **void** **initComponents**() {

java.awt.GridBagConstraints gridBagConstraints;

contactInformationPanel1 = (ContactInformationPanel) ViewFactory.*getInstance*().getView("ContactInformationPanel");

orderHistory1 = (OrderHistory) ViewFactory.*getInstance*().getView("OrderHistory");

setLayout(**new** java.awt.GridBagLayout());

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 0;

gridBagConstraints.ipadx = 165;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(44, 6, 0, 0);

add(contactInformationPanel1, gridBagConstraints);

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 1;

gridBagConstraints.gridy = 0;

gridBagConstraints.gridheight = 2;

gridBagConstraints.ipadx = 9;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = **new** java.awt.Insets(6, 6, 13, 6);

add(orderHistory1, gridBagConstraints);

}

/\*\*

\* **@return** the contactInformationPanel1

\*/

**public** panel.ContactInformationPanel **getContactInformationPanel1**() {

**return** contactInformationPanel1;

}

/\*\*

\* **@param** contactInformationPanel1 the contactInformationPanel1 to set

\*/

**public** **void** **setContactInformationPanel1**(panel.ContactInformationPanel contactInformationPanel1){

**this**.contactInformationPanel1 = contactInformationPanel1;

}

/\*\*

\* **@return** the orderHistory1

\*/

**public** panel.OrderHistory **getOrderHistory1**() {

**return** orderHistory1;

}

/\*\*

\* **@param** orderHistory1 the orderHistory1 to set

\*/

**public** **void** **setOrderHistory1**(panel.OrderHistory orderHistory1) {

**this**.orderHistory1 = orderHistory1;

}

}

**package** View;

**import** javax.swing.JTabbedPane;

**import** exception.ValidationException;

**import** global.GlobalContext;

**import** app.Controller;

**import** panel.ContactInformationPanel;

**import** panel.LoginInformationPanel;

/\*\*

\* Вкладка створена для того, щоб користувач міг внести зміни до контактної інформації

\* та інформації облікового запису.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** EditingPersonalDataPane **extends** javax.swing.JPanel **implements** AbstractView{

/\*\*

\* {@link ContactInformationPanel}

\*/

**private** panel.ContactInformationPanel contactInformationPanel1;

**private** javax.swing.JButton jButton1;

**private** panel.LoginInformationPanel loginInformation2;

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(Object... models) {

contactInformationPanel1.updateView(models);

contactInformationPanel1.switchEditableAndFocusable(**true**);

loginInformation2.updateView(models);

MainFrame frame = (MainFrame) ViewFactory.*getInstance*().getView("MainFrame");

JTabbedPane pane = frame.getjTabbedPane2();

**int** tabIndex = pane.indexOfTab("Редагувати дані");

**if**(tabIndex >= 0)

pane.remove(tabIndex);

pane.addTab("Редагувати дані", (EditingPersonalDataPane) ViewFactory.*getInstance*().getView("EditPersDataPane"));

pane.setSelectedIndex(pane.indexOfTab("Редагувати дані"));

repaint();

}

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **EditingPersonalDataPane**() {

initComponents();

}

**public** panel.ContactInformationPanel **getContactInformationPanel1**() {

**return** contactInformationPanel1;

}

**public** **void** **setContactInformationPanel1**(

panel.ContactInformationPanel contactInformationPanel1) {

**this**.contactInformationPanel1 = contactInformationPanel1;

}

**public** panel.LoginInformationPanel **getLoginInformation2**() {

**return** loginInformation2;

}

**public** **void** **setLoginInformation2**(panel.LoginInformationPanel loginInformation2) {

**this**.loginInformation2 = loginInformation2;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

java.awt.GridBagConstraints gridBagConstraints;

contactInformationPanel1 = (ContactInformationPanel) ViewFactory.*getInstance*().getView("ContactInformationPanel");

jButton1 = **new** javax.swing.JButton();

loginInformation2 = (LoginInformationPanel) ViewFactory.*getInstance*().getView("LoginInformationPanel");

java.awt.GridBagLayout layout = **new** java.awt.GridBagLayout();

layout.columnWidths = **new** **int**[] {0, 5, 0, 5, 0};

layout.rowHeights = **new** **int**[] {0, 5, 0};

setLayout(layout);

add(contactInformationPanel1, **new** java.awt.GridBagConstraints());

jButton1.setText("Редагувати");

jButton1.setMinimumSize(**new** java.awt.Dimension(100, 23));

jButton1.setPreferredSize(**new** java.awt.Dimension(125, 25));

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

gridBagConstraints = **new** java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 2;

gridBagConstraints.gridwidth = 3;

add(jButton1, gridBagConstraints);

add(loginInformation2, **new** java.awt.GridBagConstraints());

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {

**try** {

contactInformationPanel1.getValidator().isSemanticValid();

loginInformation2.getValidator().isSemanticValid();

} **catch** (ValidationException e) {

GlobalContext.*showWarningmessage*(e.getMessage());

**return**;

}

Controller.*getController*().addRequest(loginInformation2.formContext(), "updateLoginInfo");

Controller.*getController*().addRequest(contactInformationPanel1.formContext(), "updateContactInfo");

}

}

package View;

import java.sql.ResultSet;

import javax.swing.JTabbedPane;

import model.ApplicationTableModel;

import app.Context;

import app.Controller;

/\*\*

\* Вкладку створено для того, щоб користувач міг додати товари в кошик, видалити їх та

\* передати товари до замовлення.

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@SuppressWarnings("serial")

public class BasketPane extends javax.swing.JPanel implements AbstractView {

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JTable jTable1;

/\*\*

\* Зберігає значення {@link ApplicationTableModel}

\*/

private ApplicationTableModel tableModel;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

public BasketPane() {

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void updateView(Object... models) {

JTabbedPane pane = ((MainFrame)ViewFactory.getInstance().getView("MainFrame")).getjTabbedPane2();

int tabIndex = pane.indexOfTab("Кошик");

if(tabIndex >= 0)

pane.remove(tabIndex);

pane.setSelectedIndex(pane.indexOfComponent(pane.add("Кошик", (BasketPane)ViewFactory.getInstance().getView("BasketPane"))));

if(models == null || models.length == 0)

Controller.getController().addRequest(null, "updateBasket");

else{

tableModel = new ApplicationTableModel((ResultSet)models[0]);

jTable1.setModel(tableModel);

this.repaint();

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void initComponents() {

java.awt.GridBagConstraints gridBagConstraints;

jScrollPane1 = new javax.swing.JScrollPane();

jTable1 = new javax.swing.JTable();

jButton1 = new javax.swing.JButton();

jButton2 = new javax.swing.JButton();

setMaximumSize(new java.awt.Dimension(501, 433));

setLayout(new java.awt.GridBagLayout());

jScrollPane1.setViewportView(jTable1);

gridBagConstraints = new java.awt.GridBagConstraints();

gridBagConstraints.gridx = 0;

gridBagConstraints.gridy = 0;

gridBagConstraints.gridwidth = 3;

gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;

gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 35);

add(jScrollPane1, gridBagConstraints);

jButton1.setText("Оформити замовлення");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

gridBagConstraints = new java.awt.GridBagConstraints();

gridBagConstraints.gridx = 1;

gridBagConstraints.gridy = 1;

gridBagConstraints.ipady = 1;

gridBagConstraints.anchor = java.awt.GridBagConstraints.BASELINE;

gridBagConstraints.insets = new java.awt.Insets(0, 51, 0, 83);

add(jButton1, gridBagConstraints);

jButton2.setText("Видалити");

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

gridBagConstraints = new java.awt.GridBagConstraints();

gridBagConstraints.gridx = 2;

gridBagConstraints.gridy = 1;

gridBagConstraints.anchor = java.awt.GridBagConstraints.BASELINE\_LEADING;

add(jButton2, gridBagConstraints);

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

ViewFactory.getInstance().getView("OrderPane").updateView(this.tableModel);

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt){

int selectedRow = jTable1.getSelectedRow();

Context c = new Context();

ApplicationTableModel tableModel = (ApplicationTableModel) jTable1.getModel();

c.addItem("GoodsName",(String) tableModel.getValueAt(selectedRow, 2));

c.addItem("Price", (Integer)tableModel.getValueAt(selectedRow, 4));

tableModel.removeRow(selectedRow);

tableModel.fireTableRowsDeleted(selectedRow, selectedRow);

Controller.getController().addRequest(c, "deleteItemFromBasket");

this.repaint();

}

}

package View;

import javax.swing.JTabbedPane;

import panel.LoginPanel;

import panel.RecoveryPanel;

import panel.RegistrationPanel;

/\*\*

\* Вкладка створена, щоб реалізувати прецедент "Авторизація користувача".

\*

\* @author Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* @version 1.0

\*/

@SuppressWarnings("serial")

public class AuthorizationPane extends javax.swing.JPanel implements AbstractView{

/\*\*

\* Зберігає значення {@link LoginPanel}

\*/

private panel.LoginPanel loginJPanel1;

/\*\*

\* Зберігає значення {@link RecoveryPanel}

\*/

private panel.RecoveryPanel recPanel;

/\*\*

\* Зберігає значення {@link RegistrationPanel}

\*/

private panel.RegistrationPanel regPanel;

/\*\*

\* {@inheritDoc}

\*/

@Override

public void updateView(Object... models) {

MainFrame frame = (MainFrame) ViewFactory.getInstance().getView("MainFrame");

frame.setMenuEnabled(false);

JTabbedPane pane = frame.getjTabbedPane2();

pane.removeAll();

resetData();

loginJPanel1.parent = this;

getRegPanel().setAuthorizationPanel(this);

getRecPanel().setAuthorizationPanel(this);

pane.addTab("Авторизація",this);

}

/\*\*

\* Метод видаляэ дані з текстових компонентів.

\*/

public void resetData() {

loginJPanel1.resetData();

recPanel.resetData();

regPanel.resetData();

removeAll();

this.add(loginJPanel1);

}

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\* Встановює значення {@link RecoveryPanel}, {@link RegistrationPanel}

\*/

public AuthorizationPane() {

recPanel = (RecoveryPanel)ViewFactory.getInstance().getView("RecoveryPanel");

regPanel = (RegistrationPanel)ViewFactory.getInstance().getView("RegistrationPanel");

recPanel.setAuthorizationPanel(this);

regPanel.setAuthorizationPanel(this);

initComponents();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void initComponents() {

loginJPanel1 = (LoginPanel) ViewFactory.getInstance().getView("LoginPanel");

loginJPanel1.parent = this;

setLayout(new java.awt.GridBagLayout());

add(loginJPanel1, new java.awt.GridBagConstraints());

}// </editor-fold>//GEN-END:initComponents

/\*\*

\* @return the loginJPanel1

\*/

public panel.LoginPanel getLoginJPanel1() {

return loginJPanel1;

}

/\*\*

\* @param loginJPanel1 the loginJPanel1 to set

\*/

public void setLoginJPanel1(panel.LoginPanel loginJPanel1) {

this.loginJPanel1 = loginJPanel1;

}

/\*\*

\* @return the recPanel

\*/

public panel.RecoveryPanel getRecPanel() {

return recPanel;

}

/\*\*

\* @param recPanel the recPanel to set

\*/

public void setRecPanel(panel.RecoveryPanel recPanel) {

this.recPanel = recPanel;

}

/\*\*

\* @return the regPanel

\*/

public panel.RegistrationPanel getRegPanel() {

return regPanel;

}

/\*\*

\* @param regPanel the regPanel to set

\*/

public void setRegPanel(panel.RegistrationPanel regPanel) {

this.regPanel = regPanel;

}

}

**package** View;

/\*\*

\* Клас є коренем ієрархії об'єктів відображень, задає загальний протокол, який має бути реалізований

\* з метою правильної роботи програмного додатку.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

**public** **interface** AbstractView {

/\*\*

\* Метод має інкапсулювати логіку оновлення об'єкту-відображення згідно

\* з переданими йому параметрами-моделями. Об'єкти-відображення, що безпосередньо

\* відображають дані моделі, мають заповнювати свою форму самостійно, взявши дані

\* з моделі, або очищати форму, якщо даних не існує. Інші об'єкти-відображення

\* мають розмістити себе у головному фреймі або батьківських компонентах та передати

\* об'єкти моделі об'єктам-нащадкам для відображення.

\*

\* **@param** models масив об'єктів-моделей, що потребують відображення.

\*/

**abstract** **void** **updateView**(Object ... models);

/\*\*

\* Метод інкапсулює логіку розміщення елементів графічного інтерфейсу.

\*/

**abstract** **void** **initComponents**();

}

**package** panel;

**import** global.GlobalContext;

**import** java.awt.BorderLayout;

**import** java.awt.Image;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** javax.imageio.ImageIO;

**import** javax.swing.ImageIcon;

**import** javax.swing.JScrollPane;

**import** javax.swing.SpinnerNumberModel;

**import** app.Context;

**import** app.Controller;

**import** model.Goods;

**import** model.Profile;

**import** View.AbstractView;

**import** View.CatalogPane;

**import** View.ViewFactory;

/\*\*

\* Клас створений, щоб вивести на екран інформацію про конкретний товар та надати можливість

\* додати його до кошика.

\*

\* **@author** Олег Кахерський, студент групи ІО-21, ФІОТ, НТУУ "КПІ", olehkakherskiy@mail.ru

\* **@version** 1.0

\*/

@SuppressWarnings("serial")

**public** **class** ConcreteGoodsPanel **extends** javax.swing.JPanel **implements** AbstractView{

**private** javax.swing.JButton jButton1;

**private** javax.swing.JLabel jLabel1;

**private** javax.swing.JLabel jLabel2;

**private** javax.swing.JLabel jLabel3;

**private** javax.swing.JLabel jLabel4;

**private** javax.swing.JLabel jLabel5;

**private** javax.swing.JLabel jLabel6;

**private** javax.swing.JScrollPane jScrollPane1;

**private** javax.swing.JSpinner jSpinner2;

**private** javax.swing.JTextArea jTextArea1;

**private** javax.swing.JTextField goodsName;

**private** javax.swing.JTextField maker;

**private** javax.swing.JTextField count;

**private** javax.swing.JTextField price;

**private** SpinnerNumberModel spinnerModel;

**private** Goods currentGoods;

**private** CatalogPane parent;

/\*\*

\* Конструктор класу без параметрів. Викликає {@link #initComponents()}.

\*/

**public** **ConcreteGoodsPanel**() {

initComponents();

jTextArea1.setFocusable(**false**);

jTextArea1.setEditable(**false**);

goodsName.setFocusable(**false**);

goodsName.setEditable(**false**);

maker.setFocusable(**false**);

maker.setEditable(**false**);

count.setEditable(**false**);

count.setFocusable(**false**);

price.setFocusable(**false**);

price.setEditable(**false**);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **updateView**(**final** Object... models) {

parent.remove(**this**);

**if**(models.length != 0 && models[0] **instanceof** Goods){

parent.add(**this**, BorderLayout.CENTER);

currentGoods = **null**;

currentGoods = (Goods)models[0];

**try** {

Image resultImage = ImageIO.*read*(**new** File(currentGoods.getImagePath()));

resultImage = resultImage.getScaledInstance(jLabel1.getWidth(), jLabel1.getHeight(), Image.SCALE\_DEFAULT);

jLabel1.setIcon(**new** ImageIcon(resultImage));

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

goodsName.setText(currentGoods.getName());

maker.setText(currentGoods.getMaker());

count.setText(String.*valueOf*(currentGoods.getNumber()));

price.setText(String.*valueOf*(currentGoods.getPrice()));

String description = formatDescription(currentGoods.getDesctription());

**int** max = currentGoods.getNumber();

SpinnerNumberModel spinnerNumberModel = (SpinnerNumberModel) jSpinner2.getModel();

spinnerNumberModel.setMaximum(max);

spinnerNumberModel.setMinimum(1);

spinnerNumberModel.setValue(1);

jTextArea1.setText(description);

description = **null**;

**this**.repaint();

}

**else** {

parent.remove(**this**);

}

}

/\*\*

\* **@return** the parent

\*/

**public** CatalogPane **getParent**() {

**return** parent;

}

/\*\*

\* **@param** parent the parent to set

\*/

**public** **void** **setParent**(CatalogPane parent) {

**this**.parent = parent;

}

/\*\*

\* Метод видаляє дані з текстових компонентів.

\*/

**private** **void** **resetData**(){

goodsName.setText(**null**);

maker.setText(**null**);

count.setText(**null**);

price.setText(**null**);

jTextArea1.setText(**null**);

}

/\*\*

\* Метод форматує рядок таким чином, щоб кожне речення виводилось з нового рядка.

\*

\* **@param** description рядок, який необхідно відформатувати.

\* **@return** форматований рядок.

\*/

**private** String **formatDescription**(String description){

StringTokenizer tokenizer = **new** StringTokenizer(description,".");

StringBuilder builder = **new** StringBuilder(" ");

**while**(tokenizer.hasMoreElements())

builder.append(tokenizer.nextToken() + ".\n");

**return** builder.toString();

}

/\*\*

\* Метод формує об'єкт типу {@link Context}

\*

\* **@return** створений об'єкт типу {@link Context}

\*/

**private** Context **formContext**(){

Context c = **new** Context();

c.addItem("goodsID", currentGoods.getID());

c.addItem("profileID", ((Profile)GlobalContext.*globalContext*.get("profile")).getID());

c.addItem("count",(Integer)jSpinner2.getValue());

**return** c;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

**public** **void** **initComponents**() {

jLabel1 = **new** javax.swing.JLabel();

jTextArea1 = **new** javax.swing.JTextArea();

jLabel2 = **new** javax.swing.JLabel();

goodsName = **new** javax.swing.JTextField();

jLabel3 = **new** javax.swing.JLabel();

maker = **new** javax.swing.JTextField();

jLabel4 = **new** javax.swing.JLabel();

count = **new** javax.swing.JTextField();

jLabel5 = **new** javax.swing.JLabel();

price = **new** javax.swing.JTextField();

jButton1 = **new** javax.swing.JButton();

jLabel6 = **new** javax.swing.JLabel();

jSpinner2 = **new** javax.swing.JSpinner();

jTextArea1.setColumns(20);

jTextArea1.setRows(5);

jTextArea1.setMaximumSize(**new** java.awt.Dimension(12, 28));

jTextArea1.setPreferredSize(**new** java.awt.Dimension(12, 28));

// jScrollPane1.setViewportView(jTextArea1);

jScrollPane1 = **new** javax.swing.JScrollPane(jTextArea1);

jScrollPane1.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

jLabel2.setText("Назва:");

goodsName.setColumns(15);

goodsName.setMaximumSize(**new** java.awt.Dimension(30, 28));

goodsName.setPreferredSize(**new** java.awt.Dimension(30, 28));

jLabel3.setText("Виробник:");

maker.setColumns(15);

jLabel4.setText("Кількість:");

count.setColumns(15);

jLabel5.setText("Ціна:");

price.setColumns(15);

jButton1.setText("Додати до кошика");

jButton1.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** **actionPerformed**(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jLabel6.setText("Кількість на замовлення:");

javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(**this**);

**this**.setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 265, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 250, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addGap(27, 27, 27)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(jLabel6)

.addGap(19, 19, 19)

.addComponent(jSpinner2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(46, 46, 46)

.addComponent(jButton1))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel2)

.addComponent(jLabel3)

.addComponent(jLabel4)

.addComponent(jLabel5))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, **false**)

.addComponent(maker, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT\_SIZE, 140, Short.MAX\_VALUE)

.addComponent(count, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE)

.addComponent(goodsName, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE)

.addComponent(price, javax.swing.GroupLayout.PREFERRED\_SIZE, 1, Short.MAX\_VALUE))))))

.addContainerGap(18, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(6, 6, 6)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 339, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 339, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(goodsName, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel2))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(maker, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel3))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(count, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel4))

.addGap(12, 12, 12)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(price, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel5))

.addGap(46, 46, 46))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jButton1)

.addComponent(jSpinner2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jLabel6))

.addContainerGap())))

);

}

**private** **void** **jButton1ActionPerformed**(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_jButton1ActionPerformed

Controller.*getController*().addRequest(formContext(), "addItemToBasket");

resetData();

CatalogPane pane = ((CatalogPane)ViewFactory.*getInstance*().getView("CatalogPane"));

pane.remove(**this**);

pane.repaint();

}

**public** SpinnerNumberModel **getSpinnerModel**() {

**return** spinnerModel;

}

**public** **void** **setSpinnerModel**(SpinnerNumberModel spinnerModel) {

**this**.spinnerModel = spinnerModel;

}

}

CПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* + - 1. Адам Фриман. История MVC: [Електронний ресурс] // Адам Фриман. ASP .NET MVC 4. 2013. URL: <http://smarly.net/pro-asp-net-mvc-4/introducing-asp-net-mvc-4/mvc-pattern/the-history-of-mvc> .(Дата звернення: 2.03.2014).
      2. Адам Фриман. Понимание MVC паттерна: [Электронный ресурс] // Адам Фриман. ASP .NET MVC 4. 2013. URL: <http://smarly.net/pro-asp-net-mvc-4/introducing-asp-net-mvc-4/mvc-pattern/understanding-the-mvc-pattern> .

(Дата звернення: 2.03.2014).

* + - 1. Александр Ерохин. Паттерн MVP [Електронний ресурс]// ProfessorWeb. 2011 - 2014.URL: <http://professorweb.ru/my/WPF/documents_WPF/level36/36_4.php>.

(Дата звернення: 2.03.2014).

* + - 1. Александр Ерохин. Паттерн MVVM [Електронний ресурс]// ProfessorWeb. 2011 - 2014.URL: <http://professorweb.ru/my/WPF/documents_WPF/level36/36_5.php>.

(Дата звернення: 2.03.2014).

* + - 1. Иван Портянкин SWING. Эффектные пользовательские интерфейсы; Издательство ,,Лори” ,2011 – 640 с. – ISBN 978 – 5 – 85582 – 305 – 9 (рус.)
      2. Model-View-Controller: A Design Pattern for Software.2004.[Електронний ресурс]// IST UC Berkeley. California. URL: https://ist.berkeley.edu/as-ag/pub/pdf/mvc-seminar.pdf . (Дата звернення: 2.03.2014) .
      3. Model – View – Controller [Електронний ресурс] // Вікіпедія. URL: http://uk.wikipedia.org/wiki/Модель-вид-контролер. (Дата звернення: 2.03.2014).
      4. Model – View – Presenter [Електронний ресурс] // Вікіпедія. URL: http://uk.wikipedia.org/wiki/Model-View-Presenter. (Дата звернення: 2.03.2014).