НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

КУРСОВА РОБОТА

КУРСОВА РОБОТА

з дисципліни: "Інженерія програмного забезпечення"

(назва дисципліни)

на тему: Система інтернет-магазину «TexnoUа». Функціональність для обслуговуючого персоналу

Студента 2 курсу ІО-21 групи

спеціальності комп’ютерна інженерія\_\_\_\_\_\_\_\_\_\_

Кузьменка Влодимира Зіновійовича

(прізвище та ініціали)

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

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

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

Кількість балів:\_\_\_\_ Оцінка: ECTS\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії

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

(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

Анотація

Система інтернет-магазину «TexnoUa» це програмний додаток, що автоматизує покупку товарів за допомогою Інтернет.

Об’єктом розробки є підсистема системи керування інтернет-магазином – система для взаємодії обслуговуючого персоналу із іншим компонентами системи інтернет-магазину.

Метою дослідження є отримання навичок у розробці та підтримці програмного забезпечення, на прикладі системи взаємодії обслуговуючого персоналу.

Результатом роботи є створення програмного додатку який призначений для співробітників конкретного інтернет-магазину. Програмний додаток взаємодіє з користувачем та базою даних. Новизна роботи полягає у подальшій розробці даного проекту та впровадження його у бізнес.

Значущість роботи полягає у набутті практичних навичок розробки та підтримки програмних додатків.

ЗМІСТ

ВСТУП…….……………………………………………………………….………...4

1. РОЗДІЛ 1. ОПИС ШАБЛОНУ MVC.…………………………..………….…...5

1.1 Опис…………………………………………………………………………..5

1.2 Реалізація шаблону MVC……………...……………………………………7

1.3. Переваги та недоліки шаблону MVC. Споріднені шаблони……….…….9

1. РОЗДІЛ 2. ПРОЕКТУВАННЯ………...…………………………………….....12

2.1 Сценарії та діаграми………….…………………………………………….12

2.2Ескізи інтерфейсу користувача…..………………………………………...14

2.3 Діаграма граничних класів……………………………….…………….….17

2.4 Відповідність граничних класів бібліотекам Java……….……………….18

2.5 Проектування структури програми………………………………….……18

1. РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ………….……………23

3.1 Розробка структури програмного додатку……………………………….19

3.2 Інструкції для користувача………………………………………………..15

3.3 Специфікації класів……………………………………………………...…27

1. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ……………………….………56
2. ВИСНОВКИ……………………………………………………………………61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……………...………………………62

ДОДАТКИ

ДОДАТОК А. Діаграма класів вигляду ……………..................…………….63

ДОДАТОК Б. Діаграма класів контролера…………...………………...…….64

ДОДАТОК В. Програмний код………………………………………………..65

ВСТУП

У даній курсовій роботі буде розроблятись програмний додаток, що представляє собою частину системи управління інтернет-магазином «TexnoUa». Буде реалізовуватись функціональність для обслуговуючого персоналу (працівник складу, кур’єр).

Розробку програмного додатку буде проводитись мовою програмування Java. Для розробки будемо використовувати вільне модульне інтегроване середовище розробки програмного забезпечення – Eclipse.

Програма буде зв’язуватись з базою даних MySQL. Для побудови, та управління цією базою даних буде використовуватись програма MySQL Workbeanch.

Для реалізації графічного інтерфейсу будемо використовувати бібліотеку Swing.

Для зв’язку програми та бази даних будемо використовувати шаблон проектування DAO (Data Access Object).

Основним шаблоном програми буде шаблон MVC

РОЗДІЛ 1. ОПИС ШАБЛОНУ MVC

1.1 Опис.

Розробники об'єктно-орієнтованих систем постійно стикаються з проблемою створення універсальних компонентів. Ця проблема проявляється особливо гостро в тих випадках , коли компоненти повинні бути за своєю природою складними або гнучкими.

Розглянемо , наприклад , таку таблицю . Концепцію таблиці можна застосовувати різними способами , тобто залежно від потреб програми . Можна, наприклад , підійти до реалізації таблиці як до способу зберігання даних у вигляді логічної структури, що складається з комірок , рядків і стовпців. Однак цього недостатньо , оскільки існує багато способів керування збереженими даними і їх поданням.

Наприклад , якщо говорити про зберігання , то таблицями може підтримувати лише певні форми даних (тільки десяткові числа) або ж може дозволяти виконання спеціальних операцій (таких , як підсумовування). Вона може поводитися , як таблиця бази даних , в якій рядки представляють окремі записи (групи елементів даних, що представляють одну сутність), а стовпці представляють. Може бути і зовсім інша ситуація, коли таблиця не пред'являє ніяких обмежень до даних, що в ній зберігаються, а також не вимагає

якось по особливому розглядати її рядки і стовпці.

Коли навіть настільки простий об'єкт, як таблиця, може використовуватись багатьма різними способами, розробник стикається з дилемою. З одного боку, добре мати можливість повторного використання програмного коду, щоб не створювати кожну нову таблицю з самого початку. Але з іншого боку, важко зрозуміти , як розробити подібний універсальний компонент, щоб його можна було використати повторно. Занадто типова реалізація потребуватиме значного доопрацювання при кожному повторному використанні, яка може звести нанівець всі переваги цього повторного використання.

Шаблон MVC пропонує елегантну альтернативу. Згідно з цим шаблоном, складний елемент визначається в термінах трьох логічних складових:

* Модель (model). Сукупність даних, що відображає стан елемента, а також засоби, що забезпечують зміну стану.
* Представлення (view). Подання елементу (як візуальне , так і не візуальне).
* Контролер (controller). Керуюча функціональність елемента , яка забезпечує відповідність між операціями , виконуваними з його представленням, і станом, утвореним моделлю.

Застосування шаблону MVC повсюдно можна зустріти , наприклад , в області управлінням бізнесом. Дійсно, керівники підприємств формують модель, визначаючи цілі і встановлюючи правила, які забезпечують зростання підприємства і виконання його функцій. Відділи продажів і маркетингу здійснюють представлення моделі, оскільки вони представляють компанію і її продукцію світу. Нарешті, виробничі відділи відіграють роль контролера , так як вони на основі інформації, отриманої від представлення, роблять певні кроки, спрямовані на зміну моделі .

Подібне розділення елемента дозволяє розглядати кожну частину незалежно від інших (у всякому разі, практично незалежно ). Але для того щоб елемент поводився, як єдине ціле, необхідно, щоб кожна частина мала відповідний інтерфейс з двома іншими. Так, наприклад, подання має мати можливість відправляти повідомлення контролеру і отримувати інформацію від моделі, щоб виконувати висунуті до поданням вимоги. Однак шаблон MVC дає значний виграш у тому, що дозволяє порівняно легко замінювати окремі частини компонента, забезпечуючи тим самим надзвичайно високий рівень гнучкості системи.

Наприклад, таблиця, реалізована відповідно до шаблону MVC, може змінюватися з сітки в діаграму шляхом простої зміни її подання.

Розглянутий приклад з таблицею демонструє застосування шаблону на рівні окремого компонента, однак, ніщо не перешкоджає використовувати шаблон MVC на рівні архітектури всієї системи в цілому. На рівні компонента модель керує станом цього компонента , представлення забезпечує інтерфейс користувача з компонентом, а контролер виконує обробку подій або прив'язку операцій. На архітектурному рівні ці функції можна транслювати в підсистеми: модель представляє всю бізнес-модель, представлення – відображення моделі, а контролер визначає бізнес-процеси.

Шаблон MVC відноситься до розряду тих шаблонів, які «підштовхують» розробників до інтенсивного використання інкапсуляції . Відповідно до принципів об'єктно-орієнтованого програмування, рекомендується визначати елементи в термінах їх інтерфейсу (тобто як вони взаємодіють із зовнішнім світом, іншими об'єктами, компонентами або системами) та реалізації (як вони забезпечують підтримку певного стану і як функціонують ). Шаблон MVC підтримує такий підхід, оскільки відповідно з ним елемент явно розділяється на

три наступні частини:

* Model. Реалізація (стан: атрибути і внутрішня поведінка);
* View. Зовнішній інтерфейс (поведінка: визначення служб, як можуть використовуватись для подання моделі);
* Controller. Внутрішній інтерфейс (поведінка: обслуговування запитів на оновлення моделі).

1.2 Реалізація шаблону MVC.

Компонентна діаграма шаблону MVC представлена ​​на рис. 1.1. Для опису даного шаблону використовується компонентна діаграма. Кожна з трьох частин шаблону MVC являє собою компонент, який містить, в свою чергу, багато класів та інтерфейсів.

При реалізації шаблону MVC зазвичай використовуються наступні компоненти[7, c.223]:

* Model. Цей компонент містить один або більше класів і інтерфейсів, відповідає за обслуговування моделі даних. Стан моделі зберігається в атрибутах і реалізації методів. Для того щоб мати можливість повідомляти компоненти-представлення про будь-які зміни, модель зберігає посилання на кожне зареєстроване представлення (одночасно може бути декілька представлень). Коли відбувається зміна, кожен зареєстрований компонент-представлення сповіщається про цю зміну.
* View. Класи і інтерфейси уявлення забезпечують презентацію даних, що зберігаються в компоненті-моделі. Подання може (але не зобов'язана) складатися з компонентів візуального графічного інтерфейсу. Для того щоб уявлення могло отримувати повідомлення про зміну даних від моделі, воно має бути зареєстроване в моделі. Отримуючи сповіщення про зміну, компонент-представлення приймає рішення, чи потрібно їх відображати, і якщо потрібно, то як саме це зробити. Компонент-подання також зберігає посилання на модель для отримання даних від моделі, але при цьому подання може лише отримувати дані без можливості їх зміни. Представлення може також використовуватися для приховування контролера, але при цьому всі запити на зміну завжди пересилаються компоненту-контролеру. У цьому зв'язку подання повинно зберігати посилання на один або декілька контролерів.



Рис 1.1. Компонентна діаграма шаблону MVC

* Controller. Цей компонент керує змінами моделі. він зберігає посилання на компонент-модель, який відповідає за здійснення змін, і сам в свою чергу відповідає за виклик одного або декількох методів обновлення. Запит на зміни може надходити від компонента-подання.

Варіанти шаблону MVC найчастіше виникають навколо різних реалізацій представлення:

* Розсилка інформації моделлю або її отримання поданням. Реалізувати шаблон MVC можна одним із двох способів . У першому випадку модель самостійно розсилає сповіщення про оновлення своєму поданню (або поданням), а в другому – представлення, в міру необхідності, запитує інформацію у моделі. Вибір конкретного способу впливає на реалізацію відносин компонентів в системі.
* Кілька представлень. Модель може передавати інформацію кільком представленням. Цей варіант особливо часто використовується в деяких реалізаціях графічного інтерфейсу, оскільки в них одні й ті ж дані інколи повинні представлятися по різному.
* Подання «дивись , але не чіпай». Не всім представленням потрібен контролер. Деякі подання лише відображають дані моделі , але не підтримують ніяких засобів, які забезпечували б зміни в моделі за допомогою представлення.

1.3. Переваги та недоліки шаблону MVC. Споріднені шаблони.

Шаблон MVC надає відмінний спосіб створення гнучких і адаптованих до різних нових ситуацій елементів програми. При цьому гнучкість може використовуватися як статично, так і динамічно. Під статичної гнучкістю розуміємо можливість додавання в додаток нового класу подання або контролера, а під динамічною – можливість заміни об'єкта подання або контролера під час роботи програми[7, c. 225].

Зазвичай найбільша складність при реалізації шаблону MVC полягає в тому, як правильно вибрати базову презентацію елемента. Іншими словами, потрібно правильно розробити інтерфейси між моделлю, представленням і контролером. Елемент, виконаний відповідно до шаблоном MVC, часто, як і більшість інших програмних об'єктів , повинен задовольняти якогось певного набору вимог. Тому для реалізації елемента таким чином, щоб він не ніс на собі відбитку специфічних особливостей програми, необхідно мати певне бачення і виконати ретельний аналіз.

До споріднених можна віднести наступні шаблони.

* Observer [4, с. 280]. Шаблон MVC для управління комунікаціями часто користується шаблоном Observer. Зазвичай, останній реалізується наступними частинами системи.
  + - * Представленням і контролером, щоб зміни в уявленні викликали реакцію контролера.
      * Моделлю та поданням, щоб подання сповіщалося про всі зміни в моделі.
      * Strategy [4, с. 300]. Контролер часто реалізують на основі шаблону Strategy, що дозволяє спростити процес його заміни іншим контролером.

РОЗДІЛ 2. ПРОЕКТУВАННЯ

2.1. Прецеденти.

Робота з програмою починається з головного вікна. Користувачу пропонується авторизуватись у системі, або зареєструватись.

Прецедент реєстрації нового користувача (рис 2.1).

Дійові особи: Програма, Незареєстрований користувач.

1. Програма пропонує користувачу ввести адресу своєї електронної пошти та пароль, або розпочати процес реєстрації.
2. Користувач вводить свої особисті дані (Ім’я, Прізвище, телефон, електронну адресу).
3. Користувач підтверджує правильність введених даних.
4. Програма зберігає введені дані.



Рис. 2.1 Реєстрація нового користувача

Прецедент авторизації користувача (рис 2.2).

Дійові особи: Користувач, Програма.

1. Система пропонує користувачу ввести адресу своєї електронної пошти та пароль.
2. Користувач вводить адресу своєї електронної пошти та пароль.
3. Система виконує авторизацію користувача в системі.

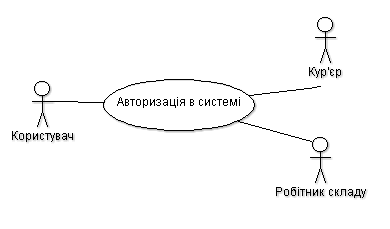


Рис. 2.2 Авторизація користувача

Далі розглянемо окремо роботу з програмою працівника складу та кур’єра. При зменшенні кількості товару на складі, або його відсутності працівник складу повинен виконати запит на поповнення товару. Система прийме цей запит, та виконає відповідні дії того щоб новий товар прислали на склад.

Прецедент запиту на поповнення товару на складі(рис 2.3).

Дійові особи: Робітник складу, Програма.

1. Система пропонує робітнику складу переглянути наявний на складі товар.
2. Робітник складу вказує який товар необхідно поповнити на складі.
3. Робітник складу підтверджує інформацію про поповнення товару.
4. Програма виконує запит на поповнення товару.



Рис 2.3 Запит на поповнення товару

Коли клієнт у своєму замовленні вказує що самостійно бажає забрати свій товар зі складу, то працівник складу повинен видати товар клієнту.

Прецедент видачі товару клієнту із складу (рис 2.4).

Дійові особи: Клієнт, Робітник складу, Програма.

1. Програма пропонує робітнику складу ввести номер замовлення.
2. Програма показує робітнику складу інформацію про замовлення.
3. Робітник складу видає товар клієнту та приймає оплату за нього.
4. Програма пропонує підтвердити оплату замовлення.
5. Програма змінює статус замовлення.



Рис 2.4 Видача товару клієнту зі складу

Кур’єру для доставки замовлення на склад необхідно спочатку отримати товар на складі. Після авторизації кур’єр вказує свій ідентифікаційний номер, та отримує товар.

Прецедент видачі товару кур’єру для доставки (рис 2.5).

Дійові особи: Працівник складу, Кур’єр, Програма.

1. Програма пропонує Працівнику складу ввести ідентифікаційний номер Кур’єра.
2. Програма показує інформацію про товар, який потрібно видати кур’єру.
3. Працівник складу видає товар кур’єру.
4. Програма пропонує підтвердити видачу товару кур’єрові.
5. Програма змінює статус замовлення.



Рис 2.5 Видача товару кур’єру для доставки

Коли відбувається доставка товару клієнту, виконується наступний прецедент.

Прецедент доставки замовлення кур’єром (рис 2.6).

Дійові особи: Кур’єр. Клієнт,

1. Програма пропонує Кур’єру переглянути список замовлень
2. Кур’єр вибирає конкретне замовлення.
3. Програма показує інформацію про замовлення.
4. Кур’єр доставляє замовлення Клієнту та приймає оплату.
5. Програма пропонує підтвердити доставку.

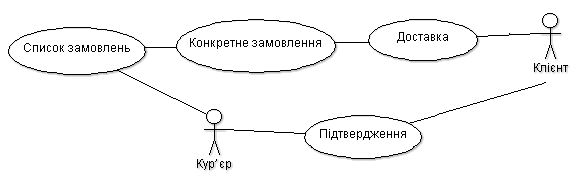


Рис 2.6 Доставка кур’єром

Кур’єр та Працівник складу мають можливість переглянути особисту інформацію, та редагувати її.

Прецедент редагування та перегляду особистої інформації (рис 2.7).

Дійові особи: Авторизований користувач(Працівник складу, Кур’єр), Програма.

1. Програма пропонує авторизованому користувачу переглянути свою особисту інформацію.
2. Авторизований користувач вводить нову інформацію.
3. Програма зберігає зміни.

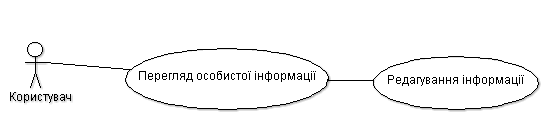


Рис 2.7

Користувачі також мають змогу змінювати стиль інтерфейсу програми.

Прецедент зміни стилю графічного інтерфейсу(рис. 2.8).

Дійові особи: користувач, програма.

1. Програма пропонує користувачу вибрати зі встановлених у його операційній системі, стиль інтерфейсу.
2. Користувач вибирає один варіант.
3. Програма відображає вибраний стиль.



Рис 2.8. Зміна стилю вікон

Користувач також має змогу авторизуватись під іншим логіном та паролем, не завершуючи при цьому роботу програми.

Прецедент виходу з системи (2.9).

Дійові особи: зареєстрований користувач, програма.

1. Програма пропонує зареєстрованому користувачу вийти з системи.
2. Користувач виходить з системи та стає незареєстрованим користувачем.



Рис 2.9. Вихід з системи

Прецедент зміна мови. (рис 2.10)

Дійові особи: авторизований користувач, програма.

1. Програма пропонує вибрати мову графічного інтерфейсу.
2. Користувач робить вибір, та підтверджує його.



Рис 2.10. Зміна мови графічного інтерфейсу

2.2. Ескізи інтерфейсу користувача.

Після запуску програми користувачу необхідно авторизуватись(рис 2.6)., або зареєструватись(рис. 2.7).



Рис 2.6. Вікно авторизації користувача



Рис. 2.7. Вікно реєстрації користувача.

Далі, залежно від свого профілю, користувач працює з програмою як працівник складу, або як кур’єр.

Розглянемо ескізи інтерфейсів для працівника складу. Працівник складу має наступні можливості:

1. Видавати товар для клієнта (Рис. 2.8.)



Рис. 2.8. Вкладка «Товар для клієнта»

1. Видавати товар кур’єру (Рис 2.9).



Рис2.9. Вкладка «Товар для кур’єра»

1. Замовляти товар на склад (Рис 2.10)



Рис2.10 Вкладка «Товар на складі»

1. Переглядати та редагувати особисту інформацію(Рис 2.11)



Рис. 2.11. Вкладка «Особиста інформація»

Розглянемо ескізи інтерфейсів для кур’єра. Кур’єр має наступні можливості:

1. Доставляти товар на клієнту (Рис 2.12).



Рис 2.12. Вкладка «Товар для доставки»

1. Переглядати та редагувати особисту інформацію. (Рис 2.11).

2.3. Діаграма граничних класів.

На рисунку 2.13. зображена діаграма граничних класів функціоналу для обслуговуючого персоналу інтернет-магазину «TexnoUА». Ці класи будуть безпосередньо взаємодіяти з користувачем.



Рис 2.13. Діаграма граничних класів.

2.4. Відповідність граничних класів бібліотекам Java.

|  |  |
| --- | --- |
| Авторизація | javax.swing.JDialog; |
| Головне вікно | javax.swing.JFrame; |
| Реєстрація | javax.swing.JDialog; |
| Меню | javax.swing.JMenuBar; |
| Головне меню | javax.swing.JMenu; |
| Меню працівника складу | javax.swing.JMenu; |
| Меню кур’єра | javax.swing.JMenu; |
| Меню налаштувань | javax.swing.JMenu; |
| Панель вкладок | javax.swing.JTabbedPane; |
| Вкладка доставки товару | javax.swing.JPanel; |
| Вкладка видачі товару зі складу | javax.swing.JPanel; |
| Вкладка перегляду особистої інформації | javax.swing.JPanel; |
| Вікно списку товарів | javax.swing.JDialog; |
| Вікно зміни стилю | javax.swing.JDialog; |
| Вкладка дозаказу товару | javax.swing.JPanel; |

Таблиця 2.1

2.5 Проектування структури програми.

На етапі проектування були створені наступні класи (Рис 2.13, 214). Ці класи(рис 2.13) будуть безпосередньо взаємодіяти з користувачем.

Роль моделі у програмі відіграє база даних MySQL. У програмі база даних відображається об’єктом dao. Через цей об’єкт здійснюється обмін інформацією між базою даних, та командами, що відправляються із класів інтерфейсу у контролер, де проходять валідацію, та ініціалізуються. Інтерфейс передає лише контекст команди, та посилання на себе. При виконанні команда викликає метод update(Data) що оновлює інформацію яку подає інтерфейс.

Пакет dao містить інтерфейс для реалізації різних способів зберігання моделі (наприклад різних баз даних). У програмі будемо використовувати MySQL реалізацію, за яку відповідатиме клас MySqlCRUID.

Пакет config містить у собі засоби для конфігурування програми. Для зберігання значень використовуватимемо XML-файли. Клас AppConfig – клас що зберігає усі конфігурації, представляє собою композитну структуру.

Пакет model містить у собі класи, що відповідають таблицям у реляційній базі даних, з якою працює програма. Ці класи використовуватиме реалізація доступу до бази даних. Також ці класи використовуватимуть класи з пакету view для представлення інформації.

Пакет view містить у собі класи, що реалізують графічний інтерфейс користувача. Вони взаємодіють з ним, та відправляють контексти команд у контролер.

Пакет controller місить у собі класи що реалізують команди графічного інтерфейсу. У класі Controller відбувається виконання команди, після того, як вона пройшла валідацію, за яку відповідає клас Validator.



Рис 2.14. Структура програмного додатку «Функціонал для обслуговуючого персоналу інтернет-магазину "TexnoUa"»

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Розробка структури програмного додатку.

Діаграма класів програмного додатку «Система інтернет-магазину «TexnoUа». Функціональність для обслуговуючого персоналу» зображена на рис. 2.14. Розглянемо детальніше кожен з її компонентів.

На рис. 3.1 зображена діаграма класів конфігуратора. При необхідності збереження налаштувань програми використовується клас AppConfig. Він являє собою композитну структуру. Клас Configuration відповідає за збереження параметрів, клас ConfigFactory – фабрика що створює класи для зчитування та запису конфігурацій у файли. Клас що зчитують параметри повинні реалізовувати інтерфейс Reader, класи що записують параметри повинні реалізовувати інтерфейс Writer.

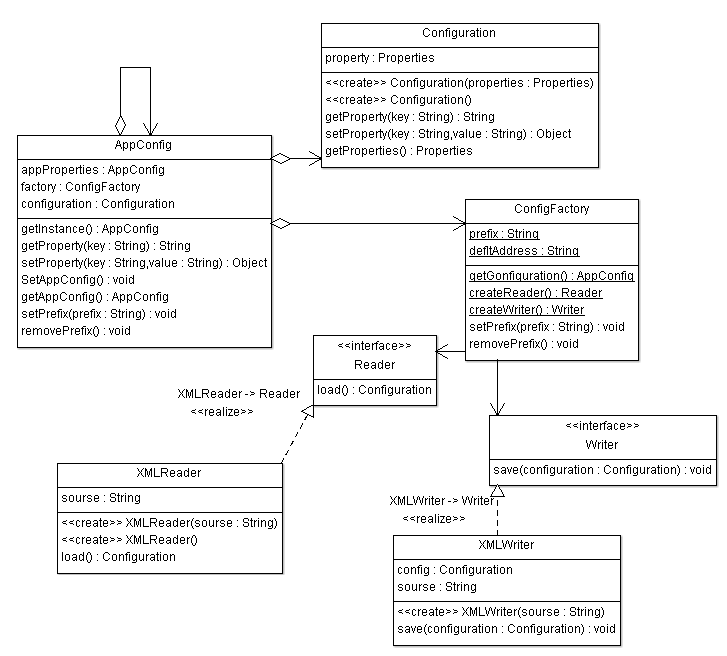


Рис 3.1. Діаграма класів конфігуратора

На рис. 3.2. зображена діаграма класів шаблону DAO(Data Access Object). Об’єкт шаблону DAO виконує операції з базою даних(клас MySqlCRUID) і надає доступ до цих операцій через інтерфейс (інтерфейс CRUIDInteface). Для реалізації операцій з базою даних клас MySqlCRUID використовує методи класу DAOAnnotationUtils. Для того, щоб не реалізовувати для кожного класу моделі операції що визначені у інтерфейсі CRUIDInteface використовуємо механізм рефлексії, та анотацій. Позначаємо поля класів моделі анотацією @Stored. Анотація @Stored містить у собі мета дані про номер поля як колонки таблиці та який клас-конвертер необхідно використовувати для роботи з цим полем. Класи-конвертери – це класи що перетворюють значення полів для запису у базу даних, та для зчитування і роботи з даними полів у програмі. Конвертери повинні реалізовувати інтерфейс ValueConverter.

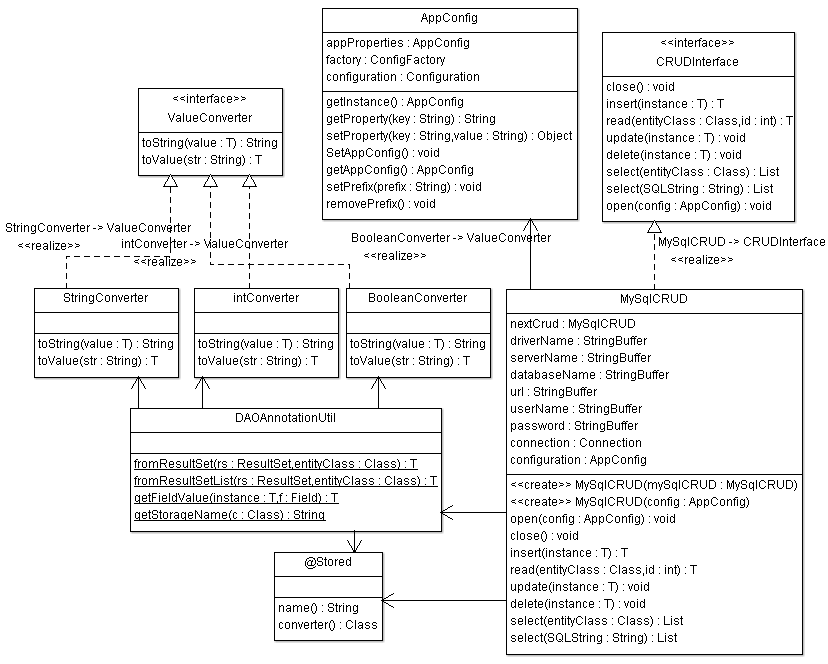


Рис 3.2 Діаграма класів DAO

На рис 3.3. зображено діаграми класів моделі. Ці класи представляють таблиці бази даних. У класах, поля відображають колонки таблиць у базі даних.

Ці класи призначені для зберігання та обміну інформацією у програмі. Детальніше призначення кожного класу описані у документації для них.

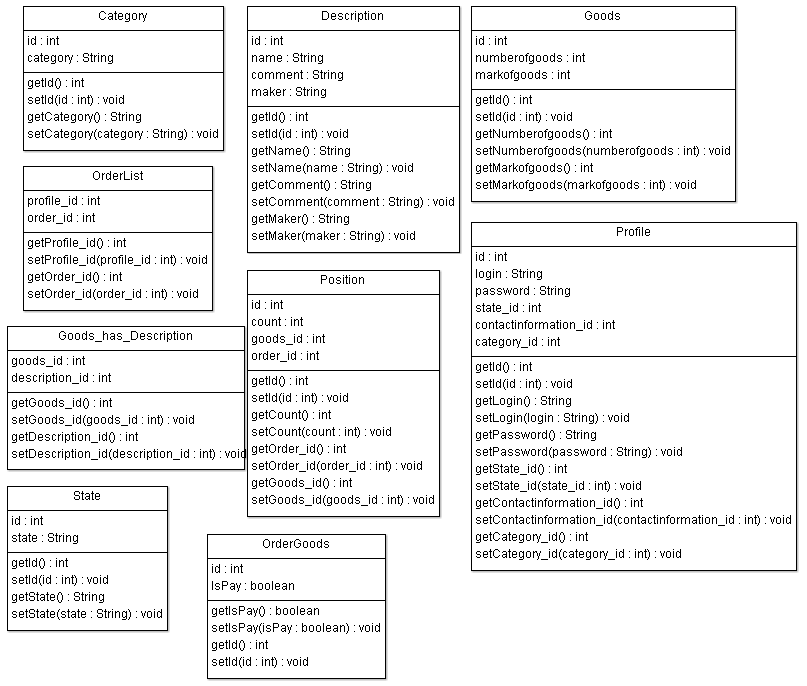


Рис 3.3 Діаграма класів моделі.

У додатку 1 зображена діаграма класів вигляду. Це класи, що реалізують графічний інтерфейс користувача. Для отримання інформації з моделі у кожному з цих класів створюється контекст, що відповідає певній команді. Цей контекст передається контролеру, також з контекстом передається посилання на цей же клас вигляду. У кожному класі вигляду є методи, які викликаються у команді, цим методам передаються дані, для оновлення моделі.

У додатку 2 зображена діаграма класів контролера. Метод контролера toContoreler() приймає посилання, контекст а команду, виконує валідацію, створює команду, та ставить її у чергу на виконання. Кожна команда реалізовує інтерфейс Command, що в свою чергу унаслідується від інтерфейсу Runnable.

Усі операції команди виконуються у методі run().

Головний клас програми MainClass містить у собі головний метод main(String [] args), що є вихідною точкою роботи програми. У цьому методі створюється об’єкт класу Controller, що виступає як Singleton у програмі. Також як Singleton виступає dao об’єкт.

3.2 Інструкції для користувача.

Робота з програмою розпочинається з авторизації. Якщо користувач не авторизований, то йому необхідно виконати реєстрацію. Після успішної реєстрації авторизуватись.

Для перегляду особистої інформації користувачу необхідно вибрати в головному меню пункт «Мої дані». Для редагування цих даних необхідно у цьому ж вікні нажати кнопку «Внести зміни». Далі ввести нові дані у відповідних полях, та нажати кнопку «ОК».

Кур’єру для доставки товару клієнту необхідно у меню «Кур’єр» вибрати пункт «Доставка товарів». Далі необхідно вибрати зі списку замовлення. Після вибору у вікні відобразиться адреса клієнта, його ідентифікаційний номер, телефон, та сума замовлення. Після оплати замовлення кур’єр підтверджує оплату шляхом встановлення прапорця «оплачено» та на жаттям кнопки «Підтвердження». При видачі товару кур’єр переглядає список товарів, який можна побачити нажавши кнопку «Список товарів».

Працівнику складу для видачі товару клієнту зі складу необхідно виконати такі ж дії, як і кур’єру, лише у своєму меню вибрати пункт «Товар для клієнта».

Працівнику складу щоб видати кур’єру товар необхідно ввести id кур’єра, та нажати кнопку «Enter». Після цього відобразиться ID кур’єра, та стане доступною кнопка «Показати товари». Після видачі товарів необхідно підтвердити видачу нажавши кнопку «Видано». Працівнику складу для того, щоб поповнити товар на складі необхідно вибрати в пункт «Дозаказ товарів» у меню «Робітник складу».

Далі вибрати зі списку товар, нажати кнопку «Замовити ще» та ввести число, на скільки поповнити товар.

Користувачам для зміни стилю програми необхідно вибрати пункт «Стиль» у меню «Налаштування». Із представленого списку вибрати один із стилів та нажати кнопку «ОК». Для того, щоб вийти з системи та змінити користувача програми необхідно у головному меню вибрати пункт «Вихід з системи». Для завершення роботи з програмою необхідно у головному меню вибрати пункт «Вийти з програми».

3.3 Специфікації класів.

public class AppConfig

extends java.util.Properties

клас що що містить у собі усі конфігурації. Представляє собою композитну структуру

Методи класу

getInstance

public AppConfig getInstance()

конструктор класу

Повертає :this повертає обєкт цього класу

getProperty

public java.lang.String getProperty(java.lang.String key)

метод повертає значення по ключу

Overrides:getProperty in class java.util.PropertiesПараметри :key - ключПовертає :значення по ключу

setProperty

public java.lang.Object setProperty(java.lang.String key,

java.lang.String value)

Метод для встановлення значення

Overrides:setProperty in class java.util.PropertiesПараметри :key - ключзначення -

SetAppConfig

public void SetAppConfig()

метод додає до композитної струкрури новий елемент

getAppConfig

public AppConfig getAppConfig()

повертає композитний елемент

Повертає :appProperties елемент

setPrefix

public void setPrefix(java.lang.String prefix)

метод встановлює префікс. Префікс призначений для того щоб створити новий файл конфігурації.

Параметри :prefix - - префікс для нового файла конфігурації

removePrefix

public void removePrefix()

метод видаляє префікс

Конструктор

Authorisation

public Authorisation()

конструктор вікна. Додає до панелі усі елементи.

Методи класу

update

public void update(java.lang.Object... objects)

метод що оновлює модель

Параметри :objects - дані для оновлення

getState

public State getState()

метод повертає стан

Повертає :стан

setState

public void setState(State state)

метод встановлює стан

Параметри :state - стан

getCategory

public static Category getCategory()

метод повертає категорі

Повертає :категорія

setCategory

public void setCategory(Category category)

меьод встановлює категорію

Параметри :category – категорія

setProfile

public void setProfile(Profile profile)

метод встановлює інформацію про користувача - профайл

Параметри :profile - профайл

getProfile

public static Profile getProfile()

метод повертає інформацію про користувача

Повертає :профайл

Конструктор

AutorisationCommand

public AutorisationCommand()

конструктор rkfce

Методи класу

setDataView

public void setDataView(java.lang.Object dataView)

метод встановлює посилання вигляду

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - вигляд

run

public void run()

виконавчий метод команди

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

getLogin

public java.lang.String getLogin()

повертає текущий

Повертає :логін

setLogin

public void setLogin(java.lang.String login)

встановлює новий логін замість текущого

Параметри :login - новий логін

getPassword

public java.lang.String getPassword()

повертає текущий пароль

Повертає :пароль

setPassword

public void setPassword(java.lang.String password)

встановлює новий текущий пароль

Параметри :password - новий пароль

public class BooleanConverter

Наслідує java.lang.Object

Реалізує ValueConverter

конвертер для булевського типу

BooleanConverter

public BooleanConverter()

Методи класу

toString

public <T> java.lang.String toString(T value)

метод що перетворює значення value у строку

Зазначений у:toString у інтерфейсіValueConverterType Параметри :T - тип значенняПараметри :value - значення що перетворюєтьсяПовертає :конвертований об'єкт

toValue

public <T> T toValue(java.lang.String str)

метод, що перетворює строку у значення типу T

Зазначений у:toValue у інтерфейсіValueConverterType Параметри :T - тип у який теобхідно перетворити строкуПараметри :str - строка для перетворенняПовертає :значення

public class CarryGoods

Наслідує javax.swing.JPanel

Клас що реалізує інтерфейс доставки доставки товарів клієнту

Конструктор

CarryGoods

public CarryGoods(Profile p)

Створюєься панель. Додаються елементи інтерфейсу користувача

Методи класу

update

public void update(java.util.List<OrderList> listWorker,

java.util.List<OrderList> clientList)

метод що оновляє дані для представлення

Параметри :listWorker - список працівнківclientList - список клієнтів

update

public void update(java.lang.String addres,

java.lang.String Telephone,

double sum,

java.util.List<Goods> goods,

java.util.List<Position> positions,

java.util.List<Description> descriptions)

метод що оновлює дані вигляду

Параметри :addres - адреса клієнтаTelephone - телефон клієнтаsum - сума замовленняgoods - список товарівpositions - список позиційdescriptions - список описів товарів

getProfileWorker

public Profile getProfileWorker()

меотод що повертає профайл користувача

Повертає :профайл

setProfileWorker

public void setProfileWorker(Profile profileWorker)

метод що встановлює список користувача

Параметри :profileWorker -

getOrderListClient

public OrderList getOrderListClient()

метод що поврертає замовлення

Повертає :замовлення

setOrderListClient

public void setOrderListClient(OrderList orderListClient)

метод що встановлює замовлення

Параметри :orderListClient - замовлення клієнта

setStyle

public static void setStyle(java.lang.String className)

метод встановлює стиль панелі

Параметри :className - назві класу стиля

public class Category

Наслідує java.lang.Object

Клас моделі. Представляє собою категорію користувача

Методи класу

getId

public int getId()

повертає ідентифікаційний номер

Повертає :ідентифікаційний номер

setId

public void setId(int id)

встановлює ідентифікаційний номер

Параметри :id - ідентифікаційний номер

getCategory

public java.lang.String getCategory()

повертає категорію

Повертає :категорія

setCategory

public void setCategory(java.lang.String category)

встановлює категорію

Параметри :category – категорія  
@Retention(value=RUNTIME)

@Target(value=TYPE)

public @interface COMMAND

Анотація для команди

Елементи

key

public abstract java.lang.String key

поле зберігає ключ команди

Повертає :ключ команди

public interface Command

Наслідує java.lang.Runnable

інтерфейс команди

Методи класу

run

void run()

виконавчий метод команди

Зазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

void setDataView(java.lang.Object dataView)

метод для встановлення зв'язку з виглядом

Параметри :dataView - посилання на клас вигляду  
public class ConfigFactory

Наслідує java.lang.Object

Фабрика рідерерів та райтерів. також повертає композитну структуру конфігурації методом getConfiguration()

Конструктор

ConfigFactory

public ConfigFactory()

Методи класу

getGonfiguration

public static AppConfig getGonfiguration()

Повертає :композитний обєкт

createReader

public static Reader createReader()

створення нового рідера

Повертає :рідер

createWriter

public static Writer createWriter()

створення новго райтера

Повертає :райтер

setPrefix

public void setPrefix(java.lang.String prefix)

встановлення нового префікса

Параметри :prefix - новий префікс

removePrefix

public void removePrefix()

видалення текущого префікса

public class Configuration

Наслідує java.util.Properties

клас конфігурації. використовується для роботи з обєктами конфігурації у програмі

Конструктор

Configuration

public Configuration(java.util.Properties properties)

конструктор. встановлює нові параметри для роботи з нею

Параметри :properties - нові параметри

Configuration

public Configuration()

пустий конструктор

Методи класу

getProperty

public java.lang.String getProperty(java.lang.String key)

повертає текущі параметри за заданим ключом

Overrides:getProperty in class java.util.PropertiesПараметри :key - ключ

setProperty

public java.lang.Object setProperty(java.lang.String key,

java.lang.String value)

встановлює нові параметри

Overrides:setProperty in class java.util.PropertiesПараметри :key - ключvalue - значення

getProperties

public java.util.Properties getProperties()

повертає всі текущі параметри

Повертає :gffhvtnhb

public class ConfirmedGoodsCourierCommand

Наслідує java.lang.Object

Реалізує Command

команда, що виконує підтвердження видачі товару кур'єру

Method Summary

Methods

Modifier and Type

Method and Description

void run()

виконавчий метод

void setDataView(java.lang.Object dataView)

метод що встановлює посилання на вигляд

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Конструктор

ConfirmedGoodsCourierCommand

public ConfirmedGoodsCourierCommand()

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class ContactInformation

Наслідує java.lang.Object

Клас моделі що відповібає за збереження контактної інформації

Методи класу

getId

public int getId()

метод повертає ідентифікаційний номер

Повертає :ідентифікаційний номер

setId

public void setId(int id)

метод встановлює ідентифікаційний номер

Параметри :id - ідентифікаційний номер

getTelephone

public java.lang.String getTelephone()

метод повертає номер телефону

Повертає :номер телефну

setTelephone

public void setTelephone(java.lang.String telephone)

метод встановлює номер телефону

Параметри :telephone - номер телефону

getEmail

public java.lang.String getEmail()

метод поервтає адресу електронної пошти

Повертає :адреса електронної пошти

setEmail

public void setEmail(java.lang.String email)

метод встановлює адресу електронної пошти

Параметри :email - адреса електронної пошти

getAdress

public java.lang.String getAdress()

метод повретає домашнью адресу

Повертає :домашня адреса

setAdress

public void setAdress(java.lang.String adress)

метод встановлює домашню адресу

Параметри :adress – адреса

@Retention(value=RUNTIME)

@Target(value=TYPE)

public @interface CONTEXT

Контекст команди, складається з параметрів

Елементи

list

public abstract PARAMETER[] list

поле містить у собі список параметрів

Повертає :список параметрів

public class Controller

Наслідує java.lang.Object

Клас контролер

Конструктор

Controller

public Controller()

конструктор, що встановлює команди в хеш-карту та ініціалізує dao об’єк

Методи класу

toController

public static void toController(java.lang.Object dataView,

CONTEXT context,

COMMAND command)

метод який валідує та виконує команду

Параметри :dataView - силка виглядуcontext - контекст командиcommand - команда

getDao

public static CRUDInterface getDao()

метод що повертає об'єкт ДАО. якщо об'єкт ДАО створений, повертає його, якщо ні, створює новий

Повертає :ДАО-об'єкт

Методи класу

getCommandKey

public static java.lang.String getCommandKey(java.lang.Class c)

мето повертає ключ команди

Параметри :c - клас командиПовертає :ключ команди

isValiedKey

public static boolean isValiedKey(PARAMETER[] paramOne,

PARAMETER[] paramTwo)

метод перевіряє валідність команди

Параметри :paramOne - параметри для порівняння з paramTwoparamTwo - параметри для порявняння з paramOneПовертає :результат варіації

public interface CRUDInterface

інтерфейс для класів, що будуть на пряму працювати з базою даних

Методи класу

close

void close()

Виключні ситуаціїjava.lang.Exception

метод що відкриває з\*єдання

Throws:java.lang.Exception

insert

<T> T insert(T instance)

Виключні ситуаціїjava.lang.Exception

метод що записує інформацію в базу даних

Параметри :instance - значення які треба записати

Повертає :NullThrows:java.lang.Exception

read

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

int id)

Виключні ситуаціїjava.lang.Exception

метод що зчитує інформацію з бази даних

Параметри :entityClass - клас що представляє інформацію у моделі програми яку ми будемо зчитувати з бази данихid - ключ по якому буде робитись пошукПовертає :значення типу моделі з якою буде працювати програмаThrows:java.lang.Exception

update

<T> void update(T instance)

Виключні ситуаціїjava.lang.Exception

метод що виконує команду оновлення даних

Параметри :instance - дані для оновленняThrows:java.lang.Exception

delete

<T> void delete(T instance)

Виключні ситуаціїjava.lang.Exception

метод що видаляє дані з бази даних

Параметри :instance - дані що треба видалитиThrows:java.lang.Exception

select

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

Виключні ситуаціїjava.lang.Exception

метод що вибирає всі дані із заданого типу моделі

Параметри :entityClass - клас - тип моделіПовертає :список значень моделіThrows:java.lang.Exception

select

<T> java.util.List<T> select(java.lang.String SQLString)

Виключні ситуаціїjava.lang.Exception

Throws:java.lang.Exception

open

void open(AppConfig config)

Виключні ситуаціїjava.lang.Exception

метод що встановлює з'вязок з базою даних

Параметри :config - конфігурація для встановлення звязку з базою даних

Throws:java.lang.Exception

public class DAOAnnotationUtil

Наслідує java.lang.Object

Клас для методів що використовуються при роботі з базою данних та класоми представленням моделі

Методи класу

fromResultSet

public static <T> T fromResultSet(java.sql.ResultSet rs,

java.lang.Class entityClass)

трансформування значень

Параметри :rs - значення з бази данихentityClass - клас значень що відповідають моделі програмиПовертає :T значення T що повертаються

fromResultSetList

public static <T> T fromResultSetList(java.sql.ResultSet rs,

java.lang.Class entityClass)

пперетворення списку значень у дані для моделі

Параметри :тип - що повертаємоrs - значення що передаються з бази данихentityClass - клас що відповідає типу даних що повертаються з моделіПовертає :список значень

getFieldValue

public static <T> T getFieldValue(T instance,

java.lang.reflect.Field f)

повертає значення поля

Параметри :instance - змінна, значення поля якої треба повернутиf - поле, значення якого треба повернутиПовертає :значення що повертається

getStorageName

public static java.lang.String getStorageName(java.lang.Class c)

метод що повератє назву класу за анотацією Stored

Параметри :c - клас назву якого треба повернути

Повертає :строка з назвою

public class Description

Наслідує java.lang.Object

клас моделі що відповідає за збереження опису товару

Методи класу

getId

public int getId()

метод повертає ідентифікаційний номер

Повертає :ідентифікаційний номер

setId

public void setId(int id)

метод встановлює ідентифікаційний номер

Параметри :id - ідентифікаційний номер

getName

public java.lang.String getName()

метод повертає ім’я

Повертає :ім’я

setName

public void setName(java.lang.String name)

метод встановлює ім’я

Параметри :name - ім’я

getComment

public java.lang.String getComment()

метод повертає характеристики

Повертає :характеристики

setComment

public void setComment(java.lang.String comment)

метод встановлює характеристики

Параметри :comment - характеристика

getMaker

public java.lang.String getMaker()

метод повертає виробника

Повертає :виробник

setMaker

public void setMaker(java.lang.String maker)

метод встановлює виробника

Параметри :maker – виробник

public class EditInfo

Наслідує javax.swing.JDialog

клас що реалізує інтерфейс редагування інформації користувача

Конструктор

EditInfo

public EditInfo()

створення панелі, виконання команд

Методи класу

getLogin

public java.lang.String getLogin()

метод повертає логін

Повертає :логін

getEmail

public java.lang.String getEmail()

метод повертає email

Повертає :email

getAddres

public java.lang.String getAddres()

метод повертає адресу

Повертає :адреса

getTelephone

public java.lang.String getTelephone()

метод повретає телефон

Повертає : номер телефону

getPassword

public java.lang.String getPassword()

метод повертає пароль

Повертає : пароль

setStyle

public static void setStyle(java.lang.String className)

становлення стилю вікна

Параметри : className - назва стилю

Конструктор

EditInfo

public EditInfo()

створення панелі, виконання команд

Методи класу

getLogin

public java.lang.String getLogin()

метод повертає логін

Повертає :логін

getEmail

public java.lang.String getEmail()

метод повертає email

Повертає :email

getAddres

public java.lang.String getAddres()

метод повертає адресу

Повертає :адреса

getTelephone

public java.lang.String getTelephone()

метод повретає телефон

Повертає :номер телефону

getPassword

public java.lang.String getPassword()

метод повертає пароль

Повертає :пароль

setStyle

public static void setStyle(java.lang.String className)

становлення стилю вікна

Параметри :className - назва стилю

public class EditInfoCommand

Наслідує java.lang.Object

Реалізує Command

команда що виконує редагування інформації

Методи класу

run

public void run()

виконавчий метод команди

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

метод для встановлення силки на обєкт представлення

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на представлення

public class GetAllGoodsCommand

Наслідує java.lang.Object

Реалізує Command

команда, що повертає усі товари

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView

метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class GetAllGoodsCommand

Наслідує java.lang.Object

Реалізує Command

команда, що повертає усі товари

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class GetGoodsForCourierCommand

Наслідує java.lang.Object

Реалізує Command

команда що повертає усі товари, призначені для текущого кур'єра

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class GetGoodsListCommand

Наслідує java.lang.Object

Реалізує Command

команда для отримання списку списку товарів

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class GetGourierCommand

Наслідує java.lang.Object

Реалізує Command

команда що повертає дані про кур'єра

Методи класу

run

public void run()

виконавчий метод команди

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class GetOrderListCommand

Наслідує java.lang.Object

Реалізує Command

команда що шукає список замовлень

Методи класу

run

public void run()

виконавчий метод команди

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class Goods

Наслідує java.lang.Object

клас моделі що відповідає за товар

Методи класу

getId

public int getId()

метод повертає ідентифікаційний номер

Повертає :ідентифікаційний номер

setId

public void setId(int id)

метод встановлює ідентифікаційний номер

Параметри :id - ідентифікаційний номе

getNumberofgoods

public int getNumberofgoods()

метод повертає кількість товарів

Повертає :кількість товарів

setNumberofgoods

public void setNumberofgoods(int numberofgoods)

метод встановлює кількість товарів

Параметри :numberofgoods - кілкьість товарів

getMarkofgoods

public int getMarkofgoods()

метод повертає ціну товару

Повертає :ціна

setMarkofgoods

public void setMarkofgoods(int markofgoods)

метод встановлює ціну товару

Параметри :markofgoods - ціна товару

public class Goods\_has\_Description

Наслідує java.lang.Object

клас моделі що зв’язує опис і товар

Методи класу

getGoods\_id

public int getGoods\_id()

повертає ідентифікаційний номер товару

Повертає :ідентифікаційний номер

setGoods\_id

public void setGoods\_id(int goods\_id)

метод встановлює ідентифікаційний номер товару

Параметри :goods\_id - ідентифікаційний номер

getDescription\_id

public int getDescription\_id()

метод повертає ідентифікаційний номер опису

Повертає :ідентифікаційний номер

setDescription\_id

public void setDescription\_id(int description\_id)

метод встановлює ідентифікаційний номер опису

Параметри :description\_id – ідентифікаційний номер опису

public class GoodsForCourier

Наслідує javax.swing.JPanel

Клас що реалізує графічний інтерфейс користувача для відображення товарів для кур’єра

Методи класу

update

public void update(Profile profile)

метод що оновлює дані для інтерфейсу

Параметри :profile - профайл користувача

update

public void update(java.util.List<Goods> goods,

java.util.List<Description> descriptions,

java.util.List<Position> positions)

метод що оновлює інформацію для представлення

Параметри :goods - список товарів

descriptions - список описів

positionsсписок - позицій

getCourierId

public java.lang.Integer getCourierId()

метод повертає ідентифікаційний номер кур'єра

Повертає :ідентифікаційний норме

setCourierId

public void setCourierId(java.lang.Integer courierId)

метод встановлює ідентифікацйний номер кур'єра

Параметри :courierId - ідентифікаійний номер

getProfile

public Profile getProfile()

метод що повертає профайл

Повертає :профайл

setProfile

public void setProfile(Profile profile)

метод що встановлює профайл

Параметри :profile - профайл

setStyle

public static void setStyle(java.lang.String className)

метод що встановлює стиль вікна

Параметри :className - назва стилю

public class intConverter

Наслідує java.lang.Object

Реалізує ValueConverter

конвертер для типів з якими оперує модель даних у програмі

Методи класу

toString

public <T> java.lang.String toString(T value)

метод перетворює ціле значення у об'єкт Класу String

Зазначений у:toString у інтерфейсіValueConverterПараметри :value - що перетворюємо у StringПовертає :строкове представлення value

toValue

public <T> T toValue(java.lang.String str)

метод повертає значення типу Т що задани об'єктом типу String

Зазначений у:toValue у інтерфейсіValueConverterПараметри :str - строка з якої беремо значненняПовертає :Integer представлення строки str

public class IsPayCommand

Наслідує java.lang.Object

Реалізує Command

команда, що визначає чи оплачене замовлення

Методи класу

run

public void run()

виконавчий метож

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class MainClass

Наслідує java.lang.Object

головний клас. тут відбувається запуск програми. Стоврення контролера

Методи класу

main

public static void main(java.lang.String[] args)

головний метод

public class MainFrame

Наслідує javax.swing.JFrame

Клас що реалізує головне вікно програми

Конструктор

MainFrame

public MainFrame()

конструктор головного вікна.

Методи класу

setStyle

public void setStyle(java.lang.String className)

метод встановлює стиль головного вікна та меню

Параметри :className - назва стилю

public class MySqlCRUD

Наслідує java.lang.Object

Реалізує CRUDInterface

клас що працює з базою даних

Конструктор

MySqlCRUD

public MySqlCRUD(MySqlCRUD mySqlCRUD)

конструктор класу.

Параметри :mySqlCRUD - наступний обєкт для роботи з базою.

MySqlCRUD

public MySqlCRUD(AppConfig config)

конструктор класу

Параметри :config - конфігурація для під'єднання до бази даних

Методи класу

open

public void open(AppConfig config)

Виключні ситуаціїjava.lang.Exception

Description copied from interface: CRUDInterface

метод що встановлює з'вязок з базою даних

Зазначений у:open у інтерфейсіCRUDInterfaceПараметри :config - конфігурація для відкриття з'єднання з базою данихThrows:java.lang.Exception

close

public void close()

Виключні ситуаціїjava.lang.Exception

Description copied from interface: CRUDInterface

метод що відкриває з\*єдання

Зазначений у:close у інтерфейсіCRUDInterfaceThrows:java.lang.Exception

insert

public <T> T insert(T instance)

Виключні ситуаціїjava.lang.Exception

Description copied from interface: CRUDInterface

метод що записує інформацію в базу даних

Зазначений у:insert у інтерфейсіCRUDInterfaceПараметри :instance - значення які треба записатиПовертає :NullThrows:java.lang.Exception

read

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

int id)

Виключні ситуаціїjava.lang.Exception

метод що зчитує інформацію із бази даних

Зазначений у:read у інтерфейсіCRUDInterfaceType Параметри :T - тим з яким працює методПараметри :entityClass - клас що представляє таблицю з якої треба зчитати даніid - ключ для пошукуПовертає :значення типу моделі з якою буде працювати програмаThrows:java.lang.Exception

update

public <T> void update(T instance)

Виключні ситуаціїjava.lang.Exception

метод що поновлює дані з у базі.

Зазначений у:update у інтерфейсіCRUDInterfaceType Параметри :T - тип цього значенняПараметри :instance - значення що треба оновитиThrows:java.lang.Exception

delete

public <T> void delete(T instance)

Виключні ситуаціїjava.lang.Exception

метод що виконує запит видалення інформації з бази даних

Зазначений у:delete у інтерфейсіCRUDInterfaceType Параметри :T - тип данихПараметри :instance - дані що треба видалитиThrows:java.lang.Exception

select

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

Виключні ситуаціїjava.lang.Exception

метод що виконує запит на вибір даних, але повертає список значень

Зазначений у:select у інтерфейсіCRUDInterfaceType Параметри :T - тип даних що повертаються у спискуПараметри :entityClass - клас що выдповыдаэ таблиці з якої вибираємо даніПовертає :список значень моделіThrows:java.lang.Exception

select

public <T> java.util.List<T> select(java.lang.String SQLString)

Виключні ситуаціїjava.lang.Exception

Зазначений у:select у інтерфейсіCRUDInterfaceThrows:java.lang.Exception

public class OrderGoods

Наслідує java.lang.Object

Клас що представляє собою замовлення

Методи класу

getIsPay

public boolean getIsPay()

повертає статус замевлення

Повертає :статус

setIsPay

public void setIsPay(boolean isPay)

встановлює статус замовлення

Параметри :isPay - статус

getId

public int getId()

повертає ідентифікаційний номер замовлення

Повертає :ідентифікаційний номмер

setId

public void setId(int id)

метод встановлює ідентифікаційний норме замовлення

Параметри :id - ідентифікаційний номер замовлення

getIsOnCourier

public boolean getIsOnCourier()

метод повертає статус замовлення

Повертає :статус

setIsOnCourier

public void setIsOnCourier(boolean isOnCourier)

метод встановлює статуст замолення

Параметри :isOnCourier – статус

public class OrderGoodsList

Наслідує javax.swing.JDialog

клас вигляду що реалізує виведення користувачу списку товарів

Конструктор

OrderGoodsList

public OrderGoodsList()

Конструктор вікна. Створює та відображає елементи графічного інтерфейсу користувача

Методи класу

setStyle

public static void setStyle(java.lang.String className)

Метод що встановлює стиль вікна

Параметри :className - назва стилю

public class OrderList

Наслідує java.lang.Object

клас моделі що відповідає за роботу з списком замовлень

Методи класу

getProfile\_id

public int getProfile\_id()

повертає ідентифікаційний норме користувача

Повертає :ідентифікаційний номер

setProfile\_id

public void setProfile\_id(int profile\_id)

метод встановлює ідентифікацййний номер користувача

Параметри :profile\_id - ідентифікаційний номер користувача

getOrder\_id

public int getOrder\_id()

метод повертає номер замовлення

Повертає :номер замовленняя

setOrder\_id

public void setOrder\_id(int order\_id)

метод встановлює номер замовлення

Параметри :order\_id - номер замолення

@Retention(value=RUNTIME)

@Target(value=TYPE)

public @interface PARAMETER

анотація для кожного параметру команди

Елементи

key

public abstract java.lang.String key

ключ параметру

Повертає :значення ключа

type

public abstract java.lang.Class<?> type

тип ключа параметру. По замовчуванню Object.class

Повертає :тип ключаDefault:java.lang.Object.class

optional

public abstract boolean optional

реалізований опціонал. По замовчуванню false

Повертає :існування реалізаціїDefault:false

public class Position

Наслідує java.lang.Object

Клас моделі, що зв’язує товари, та конкретне замовлення

Методи класу

getId

public int getId()

метод що повретає ідентифікаційний норер замолення

Повертає :номер замовлення

setId

public void setId(int id)

метод що встановлює ідентифікаційний номер замовлення

Параметри :id - ідентифікаційний номер замовлення

getCount

public int getCount()

метод що повертає кількість товарів в позиції

Повертає :кількість товарів

setCount

public void setCount(int count)

метод що встановлює кількість товарів

Параметри :count - кількість товарів

getOrder\_id

public int getOrder\_id()

метод що повертає номер замовлення

Повертає :номер замовлення

setOrder\_id

public void setOrder\_id(int order\_id)

метод що встановлює номер замовлення

Параметри :order\_id - номер замовлення

getGoods\_id

public int getGoods\_id()

метод що повертає номер товару

Повертає :номер товару

setGoods\_id

public void setGoods\_id(int goods\_id)

метод що встановлює номер товару

Параметри :goods\_id - номер товару

public class Profile

Наслідує java.lang.Object

клас моделі що відображає інформацію користувача

Методи класу

getId

public int getId()

метод поретає ідентифікаційний номер користувача

Повертає :ідентифікаційний нормер

setId

public void setId(int id)

метод встановлює ідентифікаційний номер

Параметри :id - ідентиіфкаційний номер

getLogin

public java.lang.String getLogin()

метод повертає логін

Повертає :логін

setLogin

public void setLogin(java.lang.String login)

метод встановляє логін

Параметри :login - логін

getPassword

public java.lang.String getPassword()

метод повертає пароль

Повертає :пароль

setPassword

public void setPassword(java.lang.String password)

метод встановляє пароль

Параметри :password - пароль

getState\_id

public int getState\_id()

метод повертає номер стану корисутвача

Повертає :номер стану

setState\_id

public void setState\_id(int state\_id)

метод встановляє номер стану користувача

Параметри :state\_id - номер стану

getContactinformation\_id

public int getContactinformation\_id()

метод повертає номер контактної інормації користувача

Повертає :номер контантної інформації

setContactinformation\_id

public void setContactinformation\_id(int contactinformation\_id)

метод встановлює номер контактної інормації користувача

Параметри :contactinformation\_id - номер контактної інормації

getCategory\_id

public int getCategory\_id()

метод повертає номер категорії користувача

Повертає :номер категорії користувача

setCategory\_id

public void setCategory\_id(int category\_id)

метод встановлює номер категорії користувача

Параметри :category\_id - номер категорії

public interface Reader

Інтерфейс для зчитувачів конфігурацій

Методи класу

load

Configuration load()

Клас-зчитувачповинен реалізовувати метод load() для зчитування конфігурації

Повертає :обєкт конфігурації

public class RegistrationCommand

Наслідує java.lang.Object

Реалізує Command

команда що виконує реєстрацію нового користувача

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class ResupplyCommand

Наслідує java.lang.Object

Реалізує Command

команда, що виконує запит на поповнення товару

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public class ResupplyGoods

Наслідує javax.swing.JPanel

Клас, що реалізує інтерфейс користувача для попвнення товару

Конструктор

ResupplyGoods

public ResupplyGoods()

створення нової панелі.

Методи класу

update

public void update(java.util.List<Goods> goods,

java.util.List<Description> descriptions)

метод що оновлює дані для інтерфейсу

Параметри :goods - список товарівdescriptions - список описів

getNumberOfGoods

public java.lang.Integer getNumberOfGoods()

метод що повретає кілкькість товару

Повертає :кількість товарів

setNumberOfGoods

public void setNumberOfGoods(java.lang.Integer numberOfGoods)

метод, що встановлює кількість товарів

Параметри :numberOfGoods - кількість товарів

getGoodsID

public java.lang.Integer getGoodsID()

метод що повертає ідентифікаційний номер товару

Повертає :ідентифікаційний номер

setGoodsID

public void setGoodsID(java.lang.Integer goodsID)

метод що встановлює ідентифікаційний номер товару

Параметри :goodsID - ідентифікаційний номер

setStyle

public static void setStyle(java.lang.String className)

метод, що встановлює стиль вікна

Параметри :className - назва стилю

public class SelectStyle

Наслідує javax.swing.JDialog

Клас, що реалізує інтерфейс користувача для вибору стилю вікон

Конструктор

SelectStyle

public SelectStyle(MainFrame mainFrame)

Створення вікна для вибору стилю.

Параметри :mainFrame - головне вікно програми

Методи класу

getStyle

public java.lang.String getStyle()

метод, що повертає стиль вікна

Повертає :назва стилю

setStyle

public void setStyle(java.lang.String style)

метод що встановлює стиль

Параметри :style - назва стилю

public class SetGoodsList

Наслідує javax.swing.JDialog

клас вигляду що реалізує відображення списку товарів

Конструктор

SetGoodsList

public SetGoodsList(java.util.List<Goods> goods,

java.util.List<Description> descriptions,

java.util.List<Position> positions)

ствоерння нового вікна

Параметри :goods - список товарівdescriptions - список описівpositions - список позицій

Методи класу

setStyle

public static void setStyle(java.lang.String className)

метод, що встановлює стиль вікна

Параметри :className - назва стилю

public class State

Наслідує java.lang.Object

клас моделі, що визначає стан користувача

Методи класу

getId

public int getId()

метод що повертає ідентифікаційний номер стану

Повертає :ідентифікаційний номер

setId

public void setId(int id)

метод що встановлює ідентифікаційний номер стану

Параметри :id - ідентифікаційний номер стану

getState

public java.lang.String getState()

метод що повертає стан

Повертає :стан

setState

public void setState(java.lang.String state)

метод що встановляє стан користувача

Параметри :state – стан

@Retention(value=RUNTIME)

public @interface Stored

анотація для зчитування мета-даних з моделі

Елементи

name

public abstract java.lang.String name

ім\*я

Повертає :значення

converter

public abstract java.lang.Class converter

конвертер для даного елемента, по замовчуванню StringConverter.class

Повертає :клас конвертораDefault:dao.annotation.utils.converter.StringConverter.class

public class StringConverter

Наслідує java.lang.Object

Реалізує ValueConverter

Строковий конвертор

Методи класу

toString

public <T> java.lang.String toString(T value)

перетворення значення value у строку з типу T

Зазначений у:toString у інтерфейсіValueConverterПараметри :value - знгачення для перетворенняПовертає :строкове значення з типу T

toValue

public <T> T toValue(java.lang.String str)

перетворення із строкового значення в значення T

Зазначений у:toValue у інтерфейсіValueConverterПараметри :str - строка що перетворюємо у значення типу TПовертає :значення типу Т

public class TabbedPane

Наслідує javax.swing.JPanel

клас вигляду що реалізує відображення панелей

Field Detail

tabbedPane

public static javax.swing.JTabbedPane tabbedPane

панель вкладок

Конструктор TabbedPane

public TabbedPane()

створення нової панелі

Методи класу

addTabPanel

public static void addTabPanel(javax.swing.JPanel panel,

java.lang.String name)

метод, що додає до панелі нову вкладу

Параметри :panel - нова вкладкаname - ім’я вкладки

setStyle

public static void setStyle(java.lang.String className)

метод, що встановлює новий стиль

Параметри :className - назва стилю

public class Validator

Наслідує java.lang.Object

Валідатор для команд

Методи класу

isValide

public static boolean isValide(java.lang.Class<Command> command,

CONTEXT context)

метод що виконує валідацію

Параметри :command - сомандаcontext - контекст командиПовертає :результат валідації

public interface ValueConverter

інтерфейс для конвертерів типів

toString

<T> java.lang.String toString(T value)

переведення зі значення типу Т у строку

Параметри :value - значення що конвертуэтьсяПовертає :конвертований об'єкт

toValue

<T> T toValue(java.lang.String str)

конвертер із строки в значення типу Т

Параметри :str - строка з якої конвертуватиПовертає :значення

public class WorkerInfo

Наслідує javax.swing.JPanel

клас вигляду що реалізує відображення інформації про користувача

public WorkerInfo()

конструктор класу. Створює вікно

Методи класу

update

public void update(java.util.HashMap<java.lang.String,java.lang.String> mapinfo)

метод що оновлює дані графічного інтерфейсу

Параметри :mapinfo - дані

getProfile\_id

public int getProfile\_id()

метод що повертає профайл

Повертає :профайл

setProfile\_id

public void setProfile\_id(int profile\_id)

метод, що встановлює ідентифікаційний номер профайла

Параметри :profile\_id - ідентифікаційний номер профайла

setStyle

public static void setStyle(java.lang.String className)

метод що встановлює стиль вікна

Параметри :className - назва стилю

public class WorkerInfoCommand

Наслідує java.lang.Object

Реалізує Command

команда що шукає інформацію про користувача

Методи класу

run

public void run()

виконавчий метод

Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object dataView)

Метод що встановлює посилання на вигляд

Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд

public interface Writer

інтерфейс для райтерів

Методи класу

save

void save(Configuration configuration)

метод записує конфігурацію

Параметри :configuration - сонфігурація для запису

public class XMLReader

Наслідує java.lang.Object

Реалізує Reader

Клас для зчитування конфігурацій з XML файла

Конструктор

XMLReader

public XMLReader(java.lang.String sourse)

конструктор класу. встановлює адресу XML-файла

Параметри :sourse - адреса XML файла

XMLReader

public XMLReader()

пустий конструктор. Буде використовуватись адреса за замовчуванням.

Методи класу

load

public Configuration load()

метод завантажує файл конфігурації

Зазначений у:load у інтерфейсіReaderПовертає :об'єкт конфігурації

public class XMLWriter

Наслідує java.lang.Object

Реалізує Writer

Клас-записувач. Записує конфігурацію в XML файл

Конструктор

XMLWriter

public XMLWriter(java.lang.String sourse)

конструктор класу.

Параметри :sourse - встановлює адресу файлу запису замість тієї, що викоритсовується по замовчуванню.

Методи класу

save

public void save(Configuration configuration)

метод записує коннфігурацію

Зазначений у:save у інтерфейсі WriterПараметри :onfiguration - обєкт конфігурації для запису у XML файл

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

Програмний додаток «Система інтернет-магазину «TexnoUа». Функціональність для обслуговуючого персоналу» реалізований згідно всіх зазначених вимог.

Реалізація(рис. 4.1) прецеденту реєстрації нового користувача (рис.2.1)

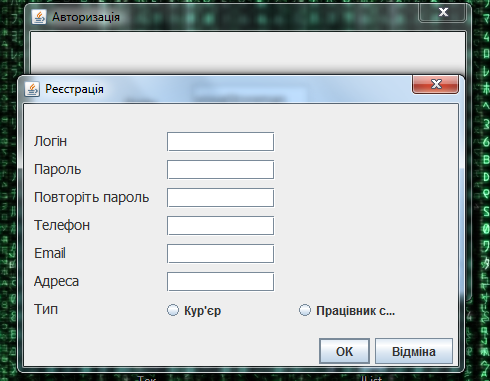


Рис 4.1 Вікно реєстрації

Реалізація(рис4.2) прецеденту авторизації (рис 2.2), перший запуск програми,.

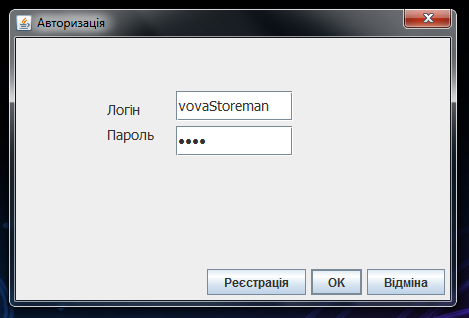


Рис4.2 Вікно авторизації

Реалізація прецеденту (рис 2.7) перегляду особистої інформації (рис 4.3),

та редагування особистої інформації (рис 4.4).

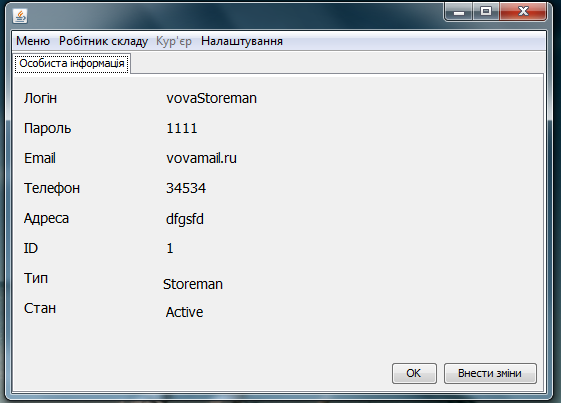


Рис 4.3 Вкладка «Особиста інформація»

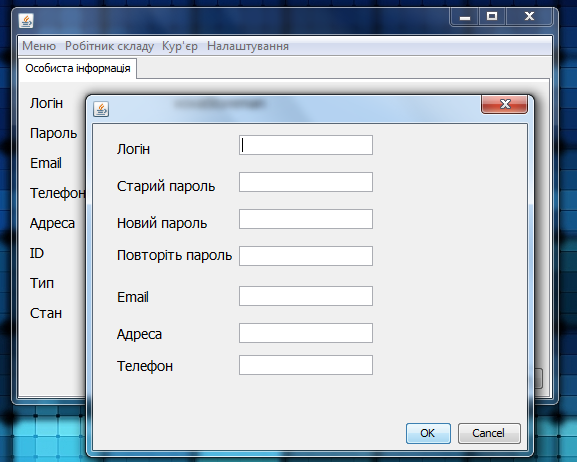


Рис4.4 Діалог редагування особистої інформації

Реалізація(рис 4.5) прецеденту доставки товару кур’єром(рис 2.6).

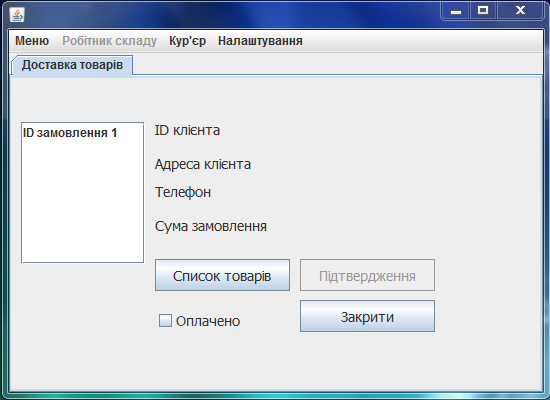


Рис 4.5 вкладка «Доставка товарів»

Перегляд списку товарів (рис 4.6).



Рис 4.6. Список товарів

Реалізація прецеденту(рис 2.5) видачі товару кур’єру (рис4.7) та товари, що треба видати (рис 4.8).

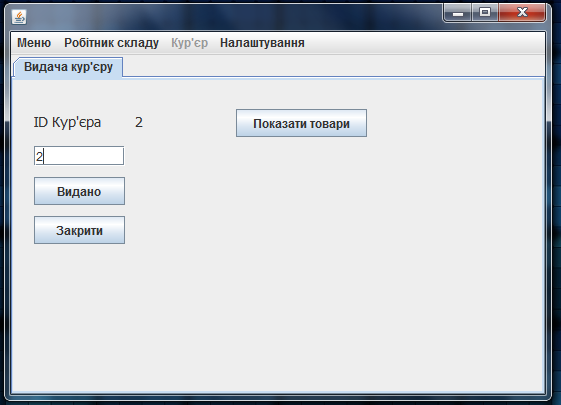


Рис 4.7. вкладка «Видача кур’єру»

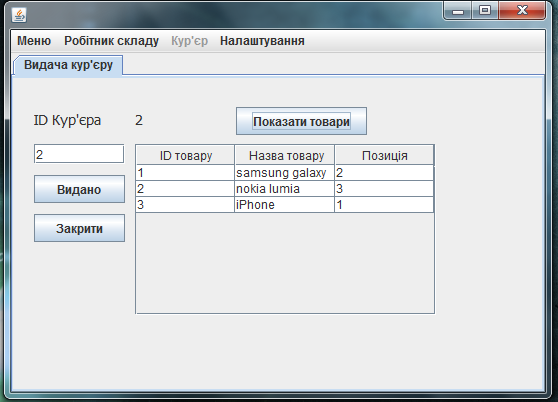


Рис 4.8. відображення товарів для видачі.

Реалізація(рис 4.9) прецеденту поповнення товару на складі (рис 2.3), та діалог поповнення (4.10).

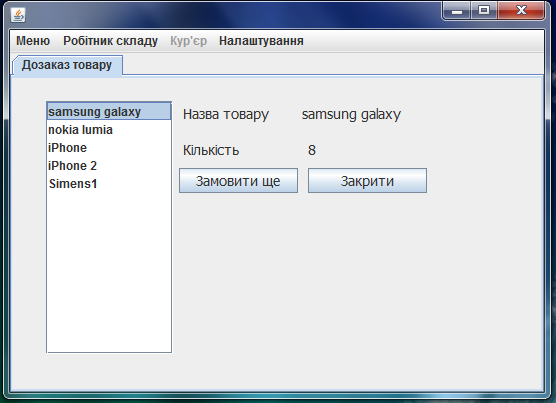


Рис 4.9 Вкладка «Дозаказ товару»

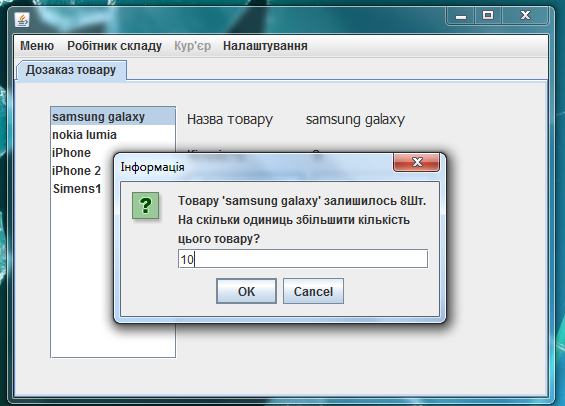


Рис 4.10. Діалог поповнення товару

Реалізація(рис 4.11) прецеденту(рис 2.8) зміни стилю графічного інтерфейсу .

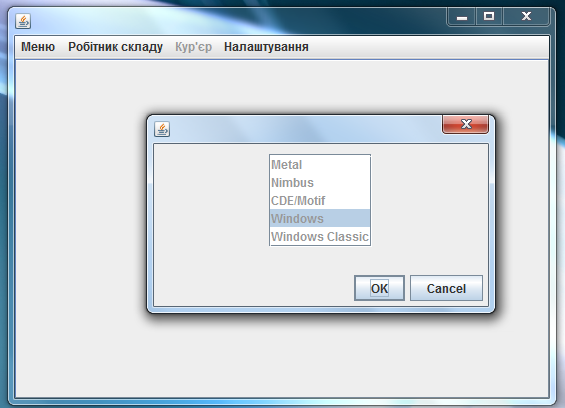


Рис 4.11. Вікно зміни стилю.

Реалізація (рис. 4.12) прецеденту (рис 2.10) зміни мови графічного інтерфейсу.

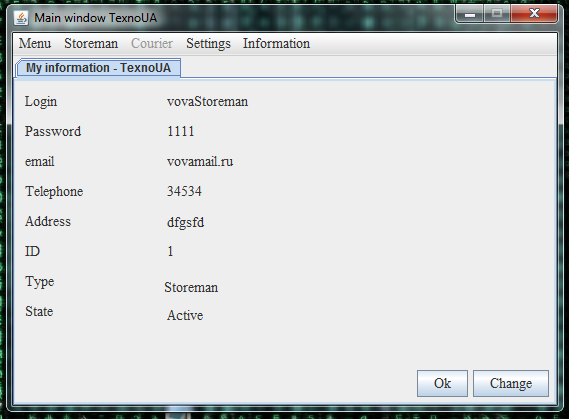


Рис. 4.12 Інтерфейс англійською мовою

Під час тестування програми не виявлено помилок чи зависань. На всіх етапах роботи вона працює коректно. Усі прецеденти були реалізовані.

ВИСНОВКИ

Програмний додаток реалізовано мовою програмування Java. Для розробки програмного додатку використовувалось вільне модульне інтегроване середовище розробки програмного забезпечення – Eclipse. Для розробки графічного інтерфейсу використовувався пагін WindowsBuilder.

Реалізований шаблон MVC що представляє структуру програми.

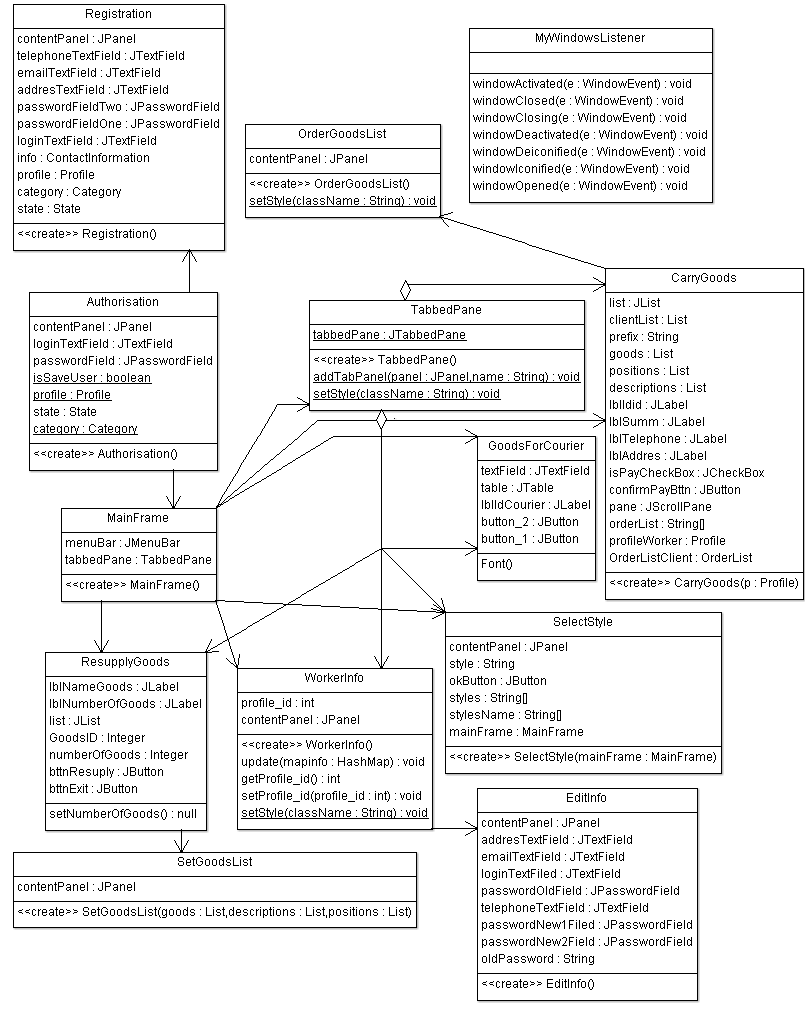
Використана база даних MySQL. Реалізовано операції для роботи з цією базою даних. Для роботи з базою даних реалізовано шаблон проектування DAO.

Графічний інтерфейс користувача реалізований засобами бібліотеки SWING

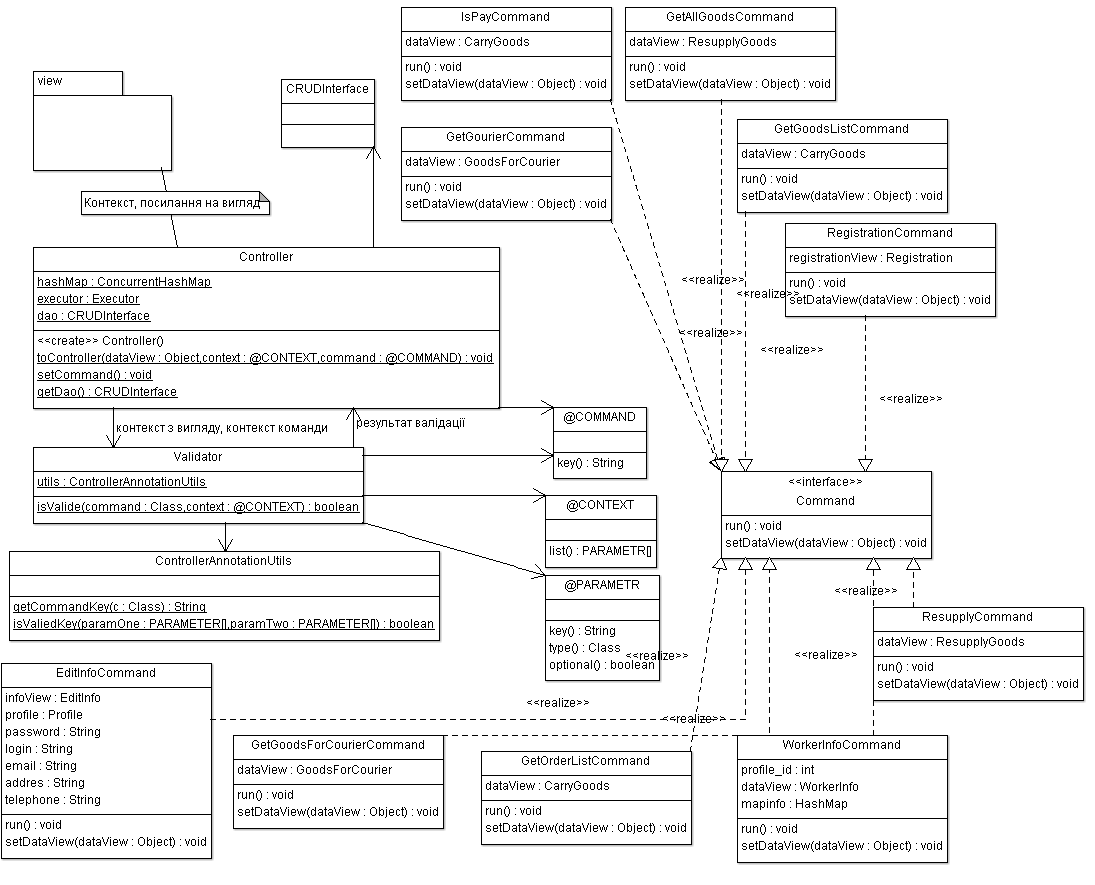
СПИСОК ВИКОРИСТАНИ ДЖЕРЕЛ

1. Герберт Шилдт SWING: руководство для начинающих– М.:“Вильямс”, 2007. – С. 720. – ISBN 0-07-226314-8.
2. Герберт Шилдт Java. Полное руководство, 8-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 20012. – 1104 с. – ISBN 978-5-8459-1759-1 (рус.)
3. Эккель Б. Философия Java / Эккель Брюс; Пер.с англ. Е.Матвеев.– 4-е изд.–СПб.: Питер, 2010. – 640с.: ил. – (Библиотека программиста). – Алф.указ.:с.631. – ISBN 978-5-388-00003-3.
4. Приемы объектно-ориентированого проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2011 – 368 с.: ил. –ISBN 978-5-469-01136-1.
5. Хорстманн Кей С. Java 2. Том 1. Основы / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 816 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1378-4 (рус.).
6. Хорстманн Кей С. Java 2. Том 2. Тонкости программирования / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 992 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1482-8 (рус.).
7. Стелтинг Стивен Применение шаблонов Java /Стелтинг Стивен, Маасен Олав; Пер. с англ. –М.: Издательский дом “Вильямс”, 2002. – 576 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). – ISBN 5-8459-0339-4 (рус.).

ДОДАТОК 1. ДІАГРАМА КЛАСІВ ВИГЛЯДУ



ДОДАТОК 2. ДІАГРАМА КЛАСІВ КОНТРОЛЛЕРА



ДОДАТОК 3. ПРОГРАМНИЙ КОД.

package app.controller.command;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.ContactInformation;

import app.model.Profile;

import app.view.Authorisation;

import app.view.EditInfo;

/\*\*

\* команда що виконує редагування інформації

\*

\*

\*/

@COMMAND(key = "editInfo")

@CONTEXT(list = {

@PARAMETER(key = "profile", type = Profile.class, optional = true),

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

@PARAMETER(key = "email", type = String.class, optional = true),

@PARAMETER(key = "addres", type = String.class, optional = true),

@PARAMETER(key = "telephone", type = String.class, optional = true) })

public class EditInfoCommand implements Command {

/\*\*

\* поле для зберігання силки на представлення

\*/

private EditInfo infoView;

/\*\*

\* інформація про текущого користувача

\*/

private Profile profile = Authorisation.getProfile();

/\*\*

\* текущий пароль

\*/

private String password;

/\*\*

\* текущий логі

\*/

private String login;

/\*\*

\* текуща адреса електронної пошти

\*/

private String email;

/\*\*

\* текуща домашня адреса

\*/

private String addres;

/\*\*

\* текущий номер телефону

\*/

private String telephone;

/\*\*

\* виконавчий метод команди

\*/

@Override

public void run() {

CRUDInterface crud = Controller.getDao();

try {

ContactInformation contInform = crud.read(ContactInformation.class,

profile.getContactinformation\_id());

Profile prfl = crud.read(Profile.class, profile.getId());

contInform.setId(profile.getContactinformation\_id());

contInform.setAdress(addres);

contInform.setTelephone(telephone);

contInform.setEmail(email);

prfl.setCategory\_id(profile.getCategory\_id());

prfl.setContactinformation\_id(profile.getContactinformation\_id());

prfl.setState\_id(profile.getState\_id());

prfl.setId(profile.getId());

prfl.setLogin(login);

prfl.setPassword(password);

crud.update(contInform);

crud.update(prfl);

crud.close();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

} finally {

try {

crud.close();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

/\*\*

\* метод для встановлення силки на обєкт представлення

\* @param dataView посилання на представлення

\*/

@Override

public void setDataView(Object dataView) {

EditInfoCommand.this.infoView = (EditInfo) dataView;

EditInfoCommand.this.addres = EditInfoCommand.this.infoView.getAddres();

EditInfoCommand.this.email = EditInfoCommand.this.infoView.getEmail();

EditInfoCommand.this.telephone = EditInfoCommand.this.infoView

.getTelephone();

EditInfoCommand.this.login = EditInfoCommand.this.infoView.getLogin();

EditInfoCommand.this.password = EditInfoCommand.this.infoView

.getPassword();

}

}

package app.controller.command;

import java.util.List;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.Description;

import app.model.Goods;

import app.view.ResupplyGoods;

/\*\*

\* команда, що повертає усі товари

\*/

@COMMAND(key = "getAllGoods")

@CONTEXT(list={

@PARAMETER(key = "goods", type = Goods.class, optional = true),

@PARAMETER(key = "listGoods", type = List.class, optional = true)

})

public class GetAllGoodsCommand implements Command {

/\*\*

\* посилання на представлення

\*/

private ResupplyGoods dataView;

/\*\*

\* виконавчий метод

\*/

@Override

public void run() {

CRUDInterface crud = Controller.getDao();

try {

List<Goods> goods = crud.select(Goods.class);

List<Description> descriptions = crud.select(Description.class);

dataView.update(goods,descriptions);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* метод що встановлює посилання на вигляд

\* @param dataView посилання на вигляд

\*/

@Override

public void setDataView(Object dataView) {

// TODO Auto-generated method stub

this.dataView = (ResupplyGoods) dataView;

}

}

package app.controller.command;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.Description;

import app.model.Goods;

import app.model.Goods\_has\_Description;

import app.model.OrderGoods;

import app.model.OrderList;

import app.model.Position;

import app.view.GoodsForCourier;

/\*\*

\* команда що повертає усі товари, призначені для текущого кур'єра

\*/

@COMMAND(key = "getGoodsForCourier")

@CONTEXT(list = {

@PARAMETER(key = "goodsList", type = Goods.class, optional = true),

@PARAMETER(key = "positionList", type = Position.class, optional = true),

@PARAMETER(key = "descriptionList", type = Description.class, optional = true) })

public class GetGoodsForCourierCommand implements Command {

/\*\*

\* посилання на вигляд

\*/

private GoodsForCourier dataView;

/\*\*

\* виконавчий метод

\*/

@Override

public void run() {

int profile\_id = dataView.getCourierId();

CRUDInterface crud = Controller.getDao();

try {

List<OrderList> orderLists = crud.select(OrderList.class);

Iterator<OrderList> itr = orderLists.iterator();

while (itr.hasNext()) {

OrderList orderList = (OrderList) itr.next();

OrderGoods goods = crud.read(OrderGoods.class,

orderList.getOrder\_id());

if (orderList.getProfile\_id() != profile\_id

|| goods.getIsOnCourier() == true) {

itr.remove();

}

}

List<Position> positions = new ArrayList<Position>();

List<Position> pList = crud.select(Position.class);

for (OrderList orderList : orderLists) {

for (Position position : pList) {

if (orderList.getOrder\_id() == position.getOrder\_id()) {

positions.add(position);

}

}

}

List<Goods> goods = new ArrayList<Goods>();

for (Position position : positions) {

int id = position.getGoods\_id();

goods.add((Goods) crud.read(Goods.class, id));

}

List<Description> descriptions = new ArrayList<Description>();

for (Goods goods2 : goods) {

int goods\_id = goods2.getId();

Goods\_has\_Description item = crud.read(

Goods\_has\_Description.class, goods\_id);

int id = item.getDescription\_id();

descriptions

.add((Description) crud.read(Description.class, id));

}

dataView.update(goods, descriptions, positions);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* метод що встановлює посилання на вигляд

\*

\* @param dataView

\* посилання на вигляд

\*/

@Override

public void setDataView(Object dataView) {

this.dataView = (GoodsForCourier) dataView;

}

}

package app.controller.command;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.ContactInformation;

import app.model.Description;

import app.model.Goods;

import app.model.Goods\_has\_Description;

import app.model.Position;

import app.view.CarryGoods;

/\*\*

\* команда для отримання списку списку товарів

\*

\*/

@COMMAND(key = "getGoodsListCommand")

@CONTEXT(list = {

@PARAMETER(key = "orderGoodsid", type = Integer.class, optional = true),

@PARAMETER(key = "addres", type = String.class, optional = true),

@PARAMETER(key = "telephone", type = Integer.class, optional = true),

@PARAMETER(key = "summ", type = Integer.class, optional = true),

@PARAMETER(key = "goods", type = Goods.class, optional = true),

@PARAMETER(key = "position", type = Position.class, optional = true),

@PARAMETER(key = "description", type = Description.class, optional = true)

})

public class GetGoodsListCommand implements Command {

/\*\*

\* посилання на вигляд

\*/

private CarryGoods dataView;

/\*\*

\* виконавчий метод

\*/

@Override

public void run() {

int order\_id = dataView.getOrderListClient().getOrder\_id();

CRUDInterface crud = Controller.getDao();

try {

List<Position> listPosition = crud.select(Position.class);

Iterator<Position> itr = listPosition.iterator();

Position item;

while (itr.hasNext()) {

item = itr.next();

if (item.getOrder\_id() != order\_id)

itr.remove();

}

List<Goods> goodsList = new ArrayList<>();

itr = listPosition.iterator();

while (itr.hasNext()) {

Position position = (Position) itr.next();

goodsList.add((Goods) crud.read(Goods.class,

position.getGoods\_id()));

}

List<Description> listDescriptions = new ArrayList<>();

for (Goods goods : goodsList) {

listDescriptions.add((Description) crud.read(Description.class,

((Goods\_has\_Description) (crud.read(

Goods\_has\_Description.class, goods.getId())))

.getDescription\_id()));

}

double sum=0;

Iterator<Position> pIterator = listPosition.iterator();

Iterator<Goods> gIteratot = goodsList.iterator();

while (pIterator.hasNext()&&gIteratot.hasNext()) {

sum += pIterator.next().getCount()\*gIteratot.next().getMarkofgoods();

}

ContactInformation info = crud.read(ContactInformation.class, dataView.getOrderListClient().getProfile\_id());

dataView.update(info.getAdress(), info.getTelephone(), sum, goodsList, listPosition, listDescriptions);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* метод що встановлює посилання на вигляд

\* @param dataView посилання на вигляд

\*/

@Override

public void setDataView(Object dataView) {

GetGoodsListCommand.this.dataView = (CarryGoods) dataView;

}

}

package app.controller.command;

import javax.swing.JOptionPane;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.Category;

import app.model.Profile;

import app.view.GoodsForCourier;

/\*\*

\*

\* команда що повертає дані про кур'єра

\* @author Vova

\*/

@COMMAND(key = "getGourier")

@CONTEXT(list = {

@PARAMETER(key = "profile", type = Profile.class, optional = true),

@PARAMETER(key = "GoodsForCourier", type = GoodsForCourier.class, optional = true) })

public class GetGourierCommand implements Command {

/\*\*

\* посилання на вигляд

\*/

private GoodsForCourier dataView;

/\*\*

\* виконавчий метод команди

\*/

@Override

public void run() {

CRUDInterface crud = Controller.getDao();

try {

Profile item = crud.read(Profile.class, dataView.getCourierId());

Category category = crud

.read(Category.class, item.getCategory\_id());

if (category.getCategory().equalsIgnoreCase("Courier")&&item.getId()!=0){

System.out.println(item.getId());

dataView.update(item);

}

else {

item.setId(-1);

dataView.update(item);

}

} catch (NullPointerException e) {

JOptionPane.showMessageDialog(dataView, "Помилка, неправильний ввід", "Інформація", JOptionPane.ERROR\_MESSAGE);

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Метод що встановлює посилання на вигляд

\* @param dataView посилання на вигляд

\*/

@Override

public void setDataView(Object dataView) {

this.dataView = (GoodsForCourier) dataView;

}

}

package app.controller.command;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import dao.CRUDInterface;

import app.controller.Controller;

import app.controller.command.annotation.COMMAND;

import app.controller.command.annotation.CONTEXT;

import app.controller.command.annotation.PARAMETER;

import app.model.OrderGoods;

import app.model.OrderList;

import app.model.Profile;

import app.view.CarryGoods;

/\*\*

\* команда що шукає список замовлень

\* @author Vova

\*

\*/

@COMMAND(key = "GetOrderListCommand")

@CONTEXT(list = {

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

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

public class GetOrderListCommand implements Command {

/\*\*

\* посилання на вигляд

\*/

private CarryGoods dataView;

/\*\*

\* виконавчий метод команди

\*/

@Override

public void run() {

CRUDInterface crud = Controller.getDao();

try {

List<OrderList> list = crud.select(OrderList.class);

// список замовлень конкретного працівника

List<OrderList> listOrder = new ArrayList<>();

// список клієнтів для відповідного працівника

List<OrderList> listClient = new ArrayList<>();

Iterator<OrderList> iteratog = list.listIterator();

while (iteratog.hasNext()) {

OrderList orderList = iteratog.next();

OrderGoods ordrGoods = crud.read(OrderGoods.class,

orderList.getOrder\_id());

if (ordrGoods.getIsPay() == true) {

iteratog.remove();

}

}

iteratog = list.listIterator();

while (iteratog.hasNext()) {

OrderList item = iteratog.next();

if (item.getProfile\_id() == dataView.getProfileWorker().getId()) {

listOrder.add(item);

} else

listClient.add(item);

}

dataView.update(listOrder, listClient);

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Метод що встановлює посилання на вигляд

\*

\* @param dataView

\* посилання на вигляд

\*/

@Override

public void setDataView(Object dataView) {

GetOrderListCommand.this.dataView = (CarryGoods) dataView;

}

}