

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 «TechnoUa». Функціональність для
обслуговуючого персоналу

Студента 2 курсу ІО-21 групи

спеціальності комп'ютерна інженерія

Кузьменка Володимира 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 «TechnoUa» 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
2. РОЗДІЛ 2. ПРОЕКТУВАННЯ.....	11
2.1 Сценарії та діаграми.....	10
2.2 Ескізи інтерфейсу користувача.....	14
2.3 Діаграма граничних класів.....	17
2.4 Відповідність граничних класів бібліотекам Java.....	18
2.5 Проектування структури програми.....	18
3. РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ.....	23
3.1 Розробка структури програмного додатку.....	19
3.2 Інструкції для користувача.....	15
3.3 Специфікації класів.....	27
4. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ.....	56
5. 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 розроблятися програмний додаток, що представляє собою частину системи управління інтернет-магазином «ТехноUa». 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 Workbench.

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 елемента (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 управління 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 поведився, 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 вииграш 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, с.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 оновлення. 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, с. 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 шаблону 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.

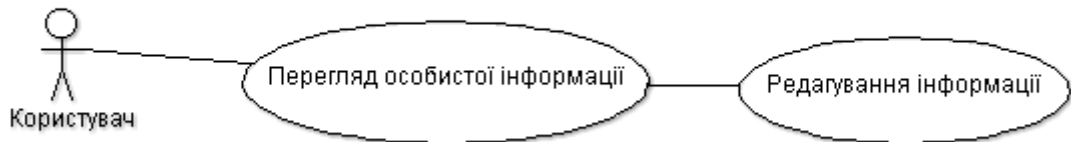


Рис 2.7

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.

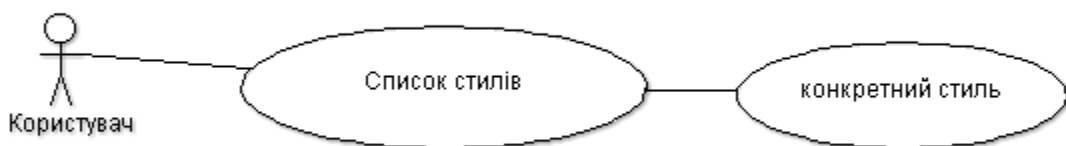


Рис 2.8. Зміна стилю вікон

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).

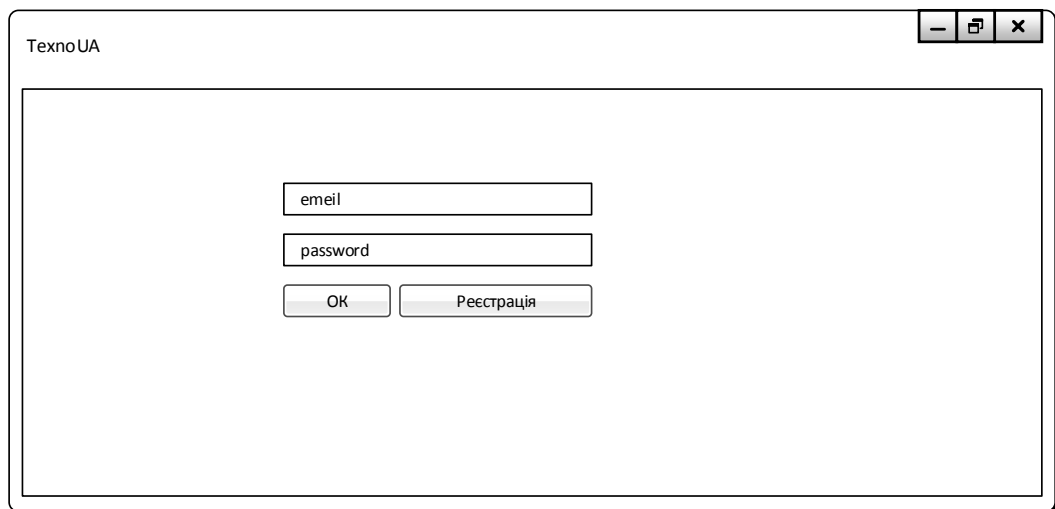


Рис 2.6. Вікно авторизації користувача

ТехноUA

Реєстрація

Електронна адреса:

Контактний телефон:

Пароль:

Підтвердіть пароль:

☐ Ви майбутній працівник складу:
 ☐ Ви майбутній кур'єр:

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.)

ТехноUA. Працівник складу

Особиста інформація Товар на складі Товар для кур'єра **Товар для клієнта** Вийти з системи Вийти з програми

Ідентифікаційний номер замовлення

ID замовлення Загальна ціна:

Список замовлень	
ID замовлення1	<input checked="" type="checkbox"/>
ID замовлення2	<input checked="" type="checkbox"/>

ID замовлення1
Назва товару1
Назва товару2
Назва товару3

Рис. 2.8. Вкладка «Товар для клієнта»

2. Видавати товар кур'єру (Рис 2.9).

ТехноUA. Працівник складу

Особиста інформація Товар на складі **Товар для кур'єра** Товар для клієнта Вийти з системи Вийти з програми

Ідентифікаційний номер кур'єра

Введіть номер

ID Кур'єра

Список замовлень

ID замовлення1	<input checked="" type="checkbox"/>
ID замовлення2	<input checked="" type="checkbox"/>

ID замовлення1

Назва товару1
Назва товару2
Назва товару3

Рис2.9. Вкладка «Товар для кур'єра»

3. Замовляти товар на склад (Рис 2.10)

ТехноUA. Працівник складу

Особиста інформація **Товар на складі** Товар для кур'єра Товар для клієнта Вийти з системи Вийти з програми

ID товару	Назва товару	Кількість на складі	Кількість дозаказати

Рис2.10 Вкладка «Товар на складі»

4. Переглядати та редагувати особисту інформацію(Рис 2.11)

ТехноUA. Працівник складу

Особиста інформація Товар на складі Товар для кур'єра Товар для клієнта Вийти з системи Вийти з програми

Прізвище

Ім'я

По Батькові

Адреса електронної пошти:

Номер телефону:

Пароль:

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).

The screenshot shows a window titled "TechnoUA. Кур'єр" with standard Windows window controls. It features four tabs: "Особиста інформація", "Товар для доставки" (which is active and has a mouse cursor over it), "Вийти з програми", and "Вийти з системи".

Under the "Товар для доставки" tab, there are two main sections:

- Список замовлень (Order List):** A table with three rows, each containing an order number and a checked checkbox.

Список замовлень	
Замовлення1	<input checked="" type="checkbox"/>
Замовлення2	<input checked="" type="checkbox"/>
Замовлення3	<input checked="" type="checkbox"/>
- Client Information Form:** Four text input fields for "Адреса клієнта", "Контактний телефон клієнта", "Прізвище та ім'я клієнта", and "Загальна сума".

Below the order list are two buttons: "Оплачено" and "Доставлено".

At the bottom right, there is a table with four columns: "ID товару", "Назва товару", "Кількість одиниць товару", and "Ціна". The table has three empty rows for data entry.

Рис 2.12. Вкладка «Товар для доставки»

2. 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 «TechnoUA». Ці класи будуть безпосередньо взаємодіяти з користувачем.

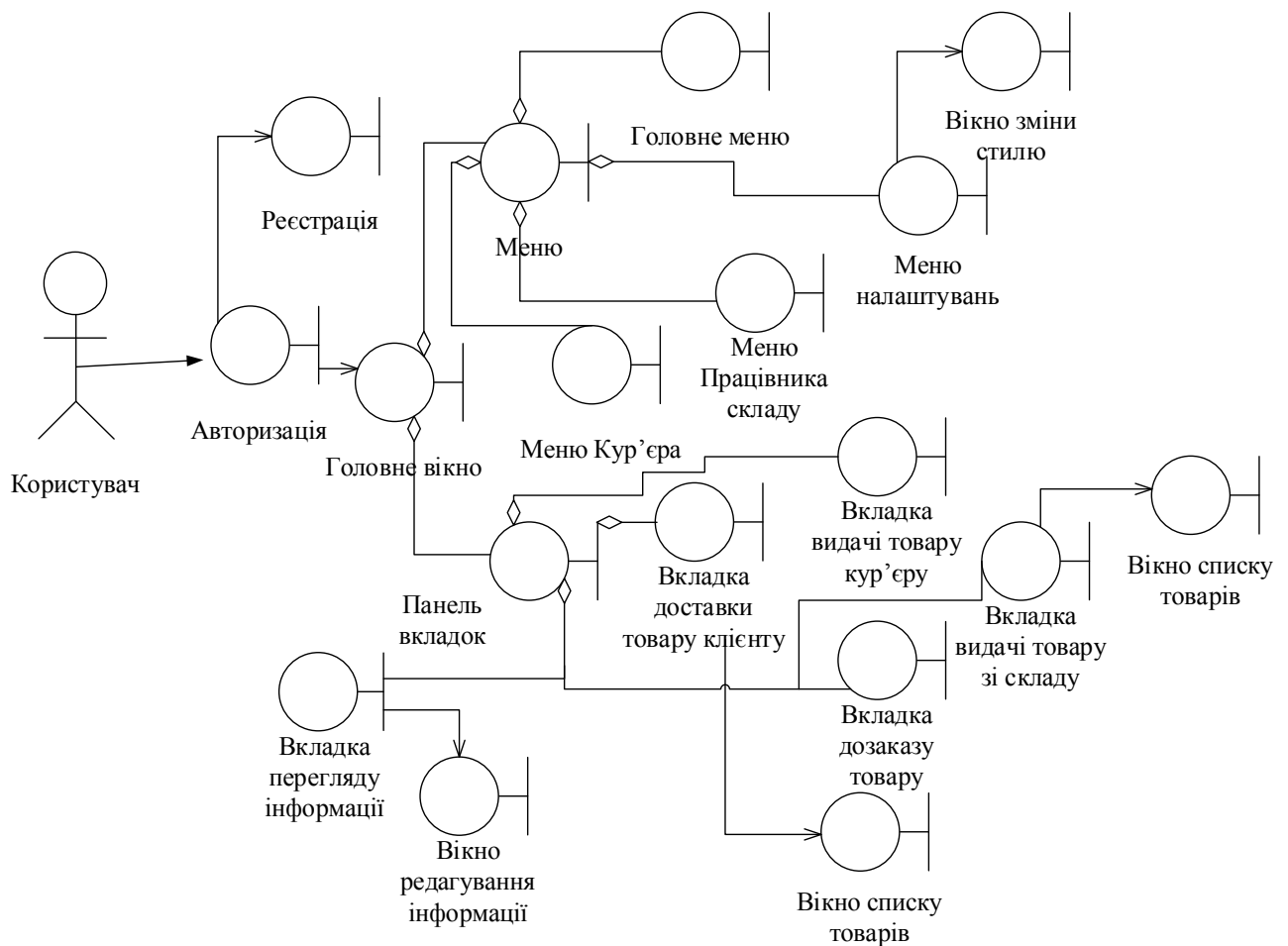


Рис 2.13. Діаграма граничних класів.

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 містить 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.

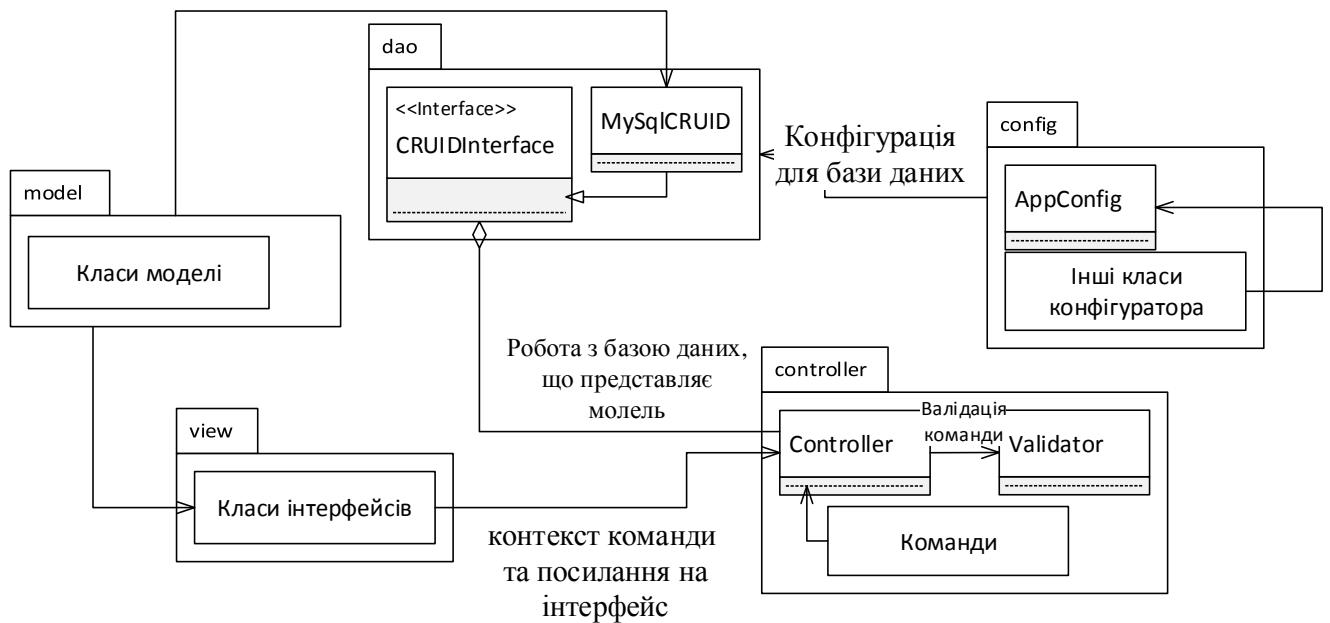


Рис 2.14. Структура програмного додатку «Функціонал для обслуговуючого персоналу інтернет-магазину "TechnoUa"»

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Розробка структури програмного додатку.

Діаграма класів програмного додатку «Система інтернет-магазину «ТехноУа». 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.

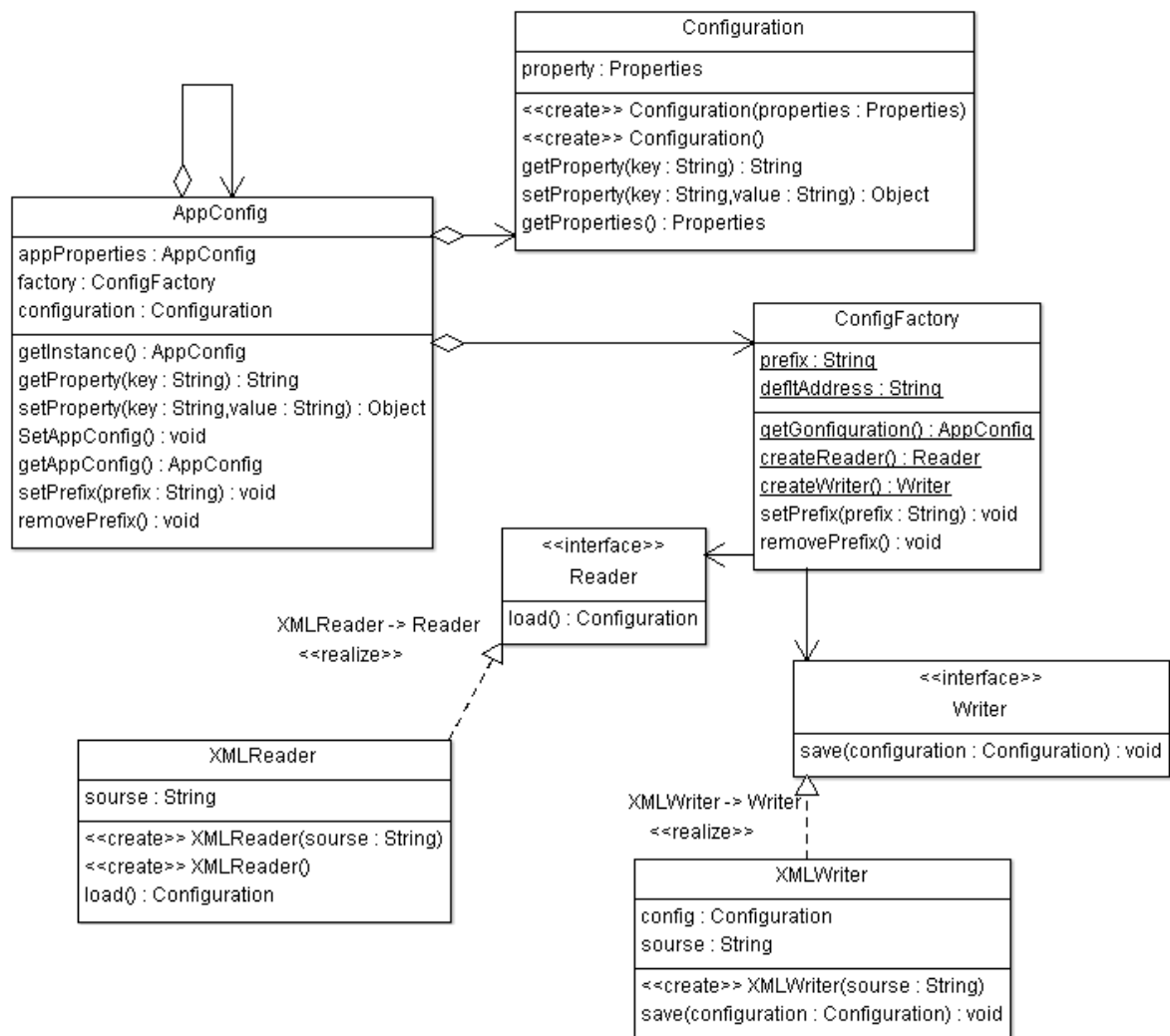


Рис 3.1. Діаграма класів конфігуратора

На рис. 3.2. зображена діаграма класів шаблону DAO(Data Access Object). Об'єкт шаблону DAO виконує операції з базою даних(клас MySQLCRUD) і надає доступ до цих операцій через інтерфейс (інтерфейс CRUDInterface). Для реалізації операцій з базою даних клас MySQLCRUD використовує методи класу DAOAnnotationUtils. Для того, щоб не реалізовувати для кожного класу моделі операції що визначені у інтерфейсі CRUDInterface 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.

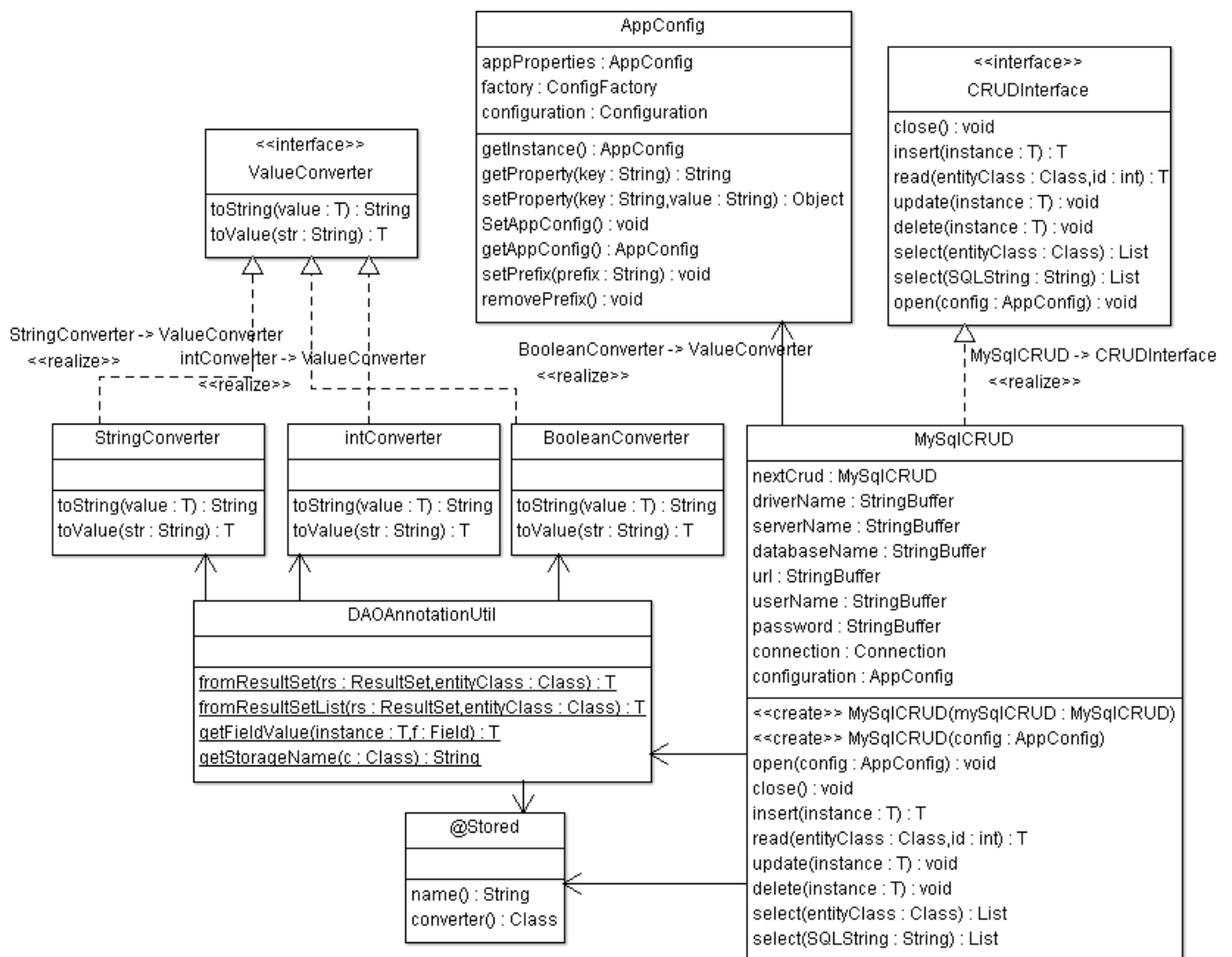


Рис 3.2 Діаграма класів DAO

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.

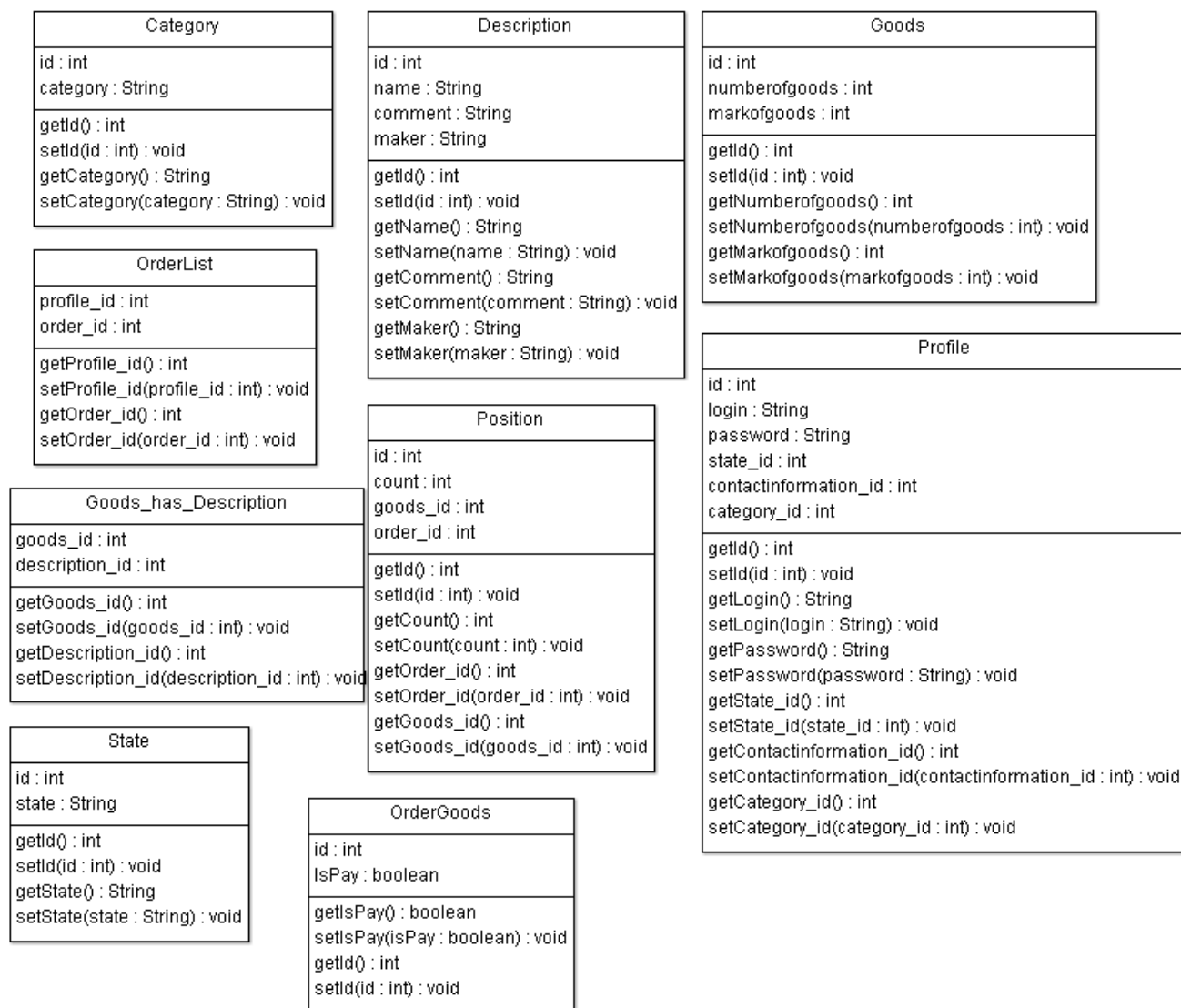


Рис 3.3 Діаграма класів моделі.

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 натиснути кнопку «Внести зміни». Далі ввести нові дані у відповідних полях, та натиснути 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 натиснути 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, натиснути 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 натиснути 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.

<code>public class AppConfig</code>	<code>setProperty</code>
<code>extends java.util.Properties</code>	<code>public</code> <code>java.lang.Object</code>
клас що містить у собі усі конфігурації. Представляє собою композитну структуру	<code>setProperty(java.lang.String key, java.lang.String value)</code>
Методи класу	Метод для встановлення значення
<code>getInstance</code>	<code>Overrides: setProperty</code> in class <code>java.util.Properties</code>
<code>public AppConfig getInstance()</code>	Параметри :key - ключзначення -
конструктор класу	<code>SetAppConfig</code>
Повертає :this повертає об'єкт цього класу	<code>public void SetAppConfig()</code>
<code>getProperty</code>	метод додає до композитної структури новий елемент
<code>public</code> <code>java.lang.String</code>	<code>getAppConfig</code>
<code>getProperty(java.lang.String key)</code>	<code>public AppConfig getAppConfig()</code>
метод повертає значення по ключу	повертає композитний елемент
<code>Overrides: getProperty</code> in class <code>java.util.Properties</code>	Повертає :appProperties елемент
Параметри :key - ключ	<code>setPrefix</code>
Повертає :значення по ключу	

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 категорію

Повертає :категорія

setCategory

public void setCategory(Category category)

метод 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

AuthorisationCommand

public AuthorisationCommand()

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 BooleanConverter()
public void run()	Методи класу
виконавчий метод команди	toString
Зазначений у:run у	public <T> java.lang.String toString(T
інтерфейсіCommandЗазначений у:run у	value)
інтерфейсіjava.lang.Runnable	метод що перетворює значення value у
getLogin	строку
public java.lang.String getLogin()	Зазначений у:toString у
повертає текущий	інтерфейсіValueConverterType
Повертає :логін	Параметри :T - тип значенняПараметри
setLogin	:value - значення що
public void setLogin(java.lang.String	перетворюєтьсяПовертає
login)	:конвертований об'єкт
встановлює новий логін замість	toValue
текущого	public <T> T toValue(java.lang.String str)
Параметри :login - новий логін	метод, що перетворює строку у
getPassword	значення типу T
public java.lang.String getPassword()	Зазначений у:toValue у
повертає текущий пароль	інтерфейсіValueConverterType
Повертає :пароль	Параметри :T - тип у який теобхідно
setPassword	перетворити строкуПараметри :str -
public void setPassword(java.lang.String	строка для перетворенняПовертає
password)	:значення
встановлює новий текущий пароль	public class CarryGoods
Параметри :password - новий пароль	Наслідує javax.swing.JPanel
public class BooleanConverter	Клас що реалізує інтерфейс доставки
Наслідує java.lang.Object	доставки товарів клієнту
Реалізує ValueConverter	Конструктор
конвертер для булевського типу	CarryGoods
BooleanConverter	public CarryGoods(Profile p)

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

інтерфейсjava.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 рідерів та райтерів. також

повертає композитну структуру

конфігурації методом getConfiguration()

Конструктор

ConfigFactory

public ConfigFactory()

Методи класу

getGonfiguration

public static AppConfig

getGonfiguration()

Повертає :композитний об'єкт

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 текущего префікса
 public class Configuration
 Наслідує java.util.Properties
 клас конфігурації. використовується
 для роботи з об'єктами 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	Наслідує java.lang.Object
void setDataView(java.lang.Object dataView)	Клас моделі що відповідає за збереження контактної інформації
метод що встановлює посилання на вигляд	Методи класу
Methods inherited from class java.lang.Object	getId
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	public int getId()
Конструктор	метод повертає ідентифікаційний номер
ConfirmedGoodsCourierCommand	Повертає :ідентифікаційний номер
public	setId
ConfirmedGoodsCourierCommand()	public void setId(int id)
Методи класу	метод встановлює ідентифікаційний номер
run	Параметри :id - ідентифікаційний номер
public void run()	getTelephone
виконавчий метод	public java.lang.String getTelephone()
Зазначений у:run у	метод повертає номер телефону
інтерфейсіCommandЗазначений у:run у	Повертає :номер телефону
інтерфейсіjava.lang.Runnable	setTelephone
setDataView	public void setTelephone(java.lang.String telephone)
public void setDataView(java.lang.Object dataView)	метод встановлює номер телефону
метод що встановлює посилання на вигляд	Параметри :telephone - номер телефону
Зазначений у:setDataView у	getEmail
інтерфейсіCommandПараметри :dataView - посилання на вигляд	public java.lang.String getEmail()
	метод поєрвтає адресу електронної пошти
	Повертає :адреса електронної пошти
	setEmail

public class ContactInformation

```
public void setEmail(java.lang.String email)
```

метод встановлює адресу електронної пошти

Параметри :email - адреса електронної пошти

```
getAdress
```

```
public java.lang.String getAdress()
```

метод повертає домашню 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
```

34

```
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

командисcommand - 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)	Виключні
метод перевіряє валідність команди	ситуаціїjava.lang.Exception
Параметри :paramOne - параметри для порівняння з paramTwo	метод що зчитує інформацію з бази даних
paramTwo - параметри для порівняння з paramOne	Параметри :entityClass - клас що представляє інформацію у моделі програми яку ми будемо зчитувати з бази даних
Повертає :результат варіації	id - ключ по якому буде робитись пошук
public interface CRUDInterface	Повертає :значення типу моделі з якою буде працювати програма
інтерфейс для класів, що будуть на пряму працювати з базою даних	Throws:java.lang.Exception
Методи класу	
close	
void close()	
Виключні	update
ситуаціїjava.lang.Exception	<T> void update(T instance)
метод що відкриває з'єднання	Виключні
Throws:java.lang.Exception	ситуаціїjava.lang.Exception
insert	
<T> T insert(T instance)	метод що виконує команду оновлення даних
Виключні	Параметри :instance - дані для оновлення
ситуаціїjava.lang.Exception	Throws:java.lang.Exception
метод що записує інформацію в базу даних	delete
Параметри :instance - значення які треба записати	<T> void delete(T instance)
Повертає	Виключні
:NullThrows:java.lang.Exception	ситуаціїjava.lang.Exception
read	метод що видаляє дані з бази даних
<T> T read(java.lang.Class entityClass, int id)	Параметри :instance - дані що треба видалити
	Throws:java.lang.Exception
	select

<T>	java.util.List<T>	
select(java.lang.Class entityClass)	fromResultSet	
Виключні	public static <T> T	
ситуаціїjava.lang.Exception	fromResultSet(java.sql.ResultSet rs,	
метод що вибирає всі дані із заданого	java.lang.Class entityClass)	
типу моделі	трансформування значень	
Параметри :entityClass - клас - тип	Параметри :rs - значення з бази	
моделіПовертає :список значень	данихentityClass - клас значень що	
моделіThrows:java.lang.Exception	відповідають моделі програмиПовертає	
select	:T значення T що повертаються	
<T>	fromResultSetList	
select(java.lang.String SQLString)	public static <T> T	
Виключні	fromResultSetList(java.sql.ResultSet rs,	
ситуаціїjava.lang.Exception	java.lang.Class entityClass)	
Throws:java.lang.Exception	пперетворення списку значень у дані	
open	для моделі	
void open(AppConfig config)	Параметри :тип - що повертаєmors -	
Виключні	значення що передаються з бази	
ситуаціїjava.lang.Exception	данихentityClass - клас що відповідає	
метод що встановлює з'вязок з базою	типу даних що повертаються з	
даних	моделіПовертає :список значень	
Параметри :config - конфігурація для	getFieldValue	
встановлення звязку з базою даних	public static <T> T getFieldValue(T	
Throws:java.lang.Exception	instance,	
public class DAOAnnotationUtil	java.lang.reflect.Field f)	
Наслідує java.lang.Object	повертає значення поля	
Клас для методів що використовуються	Параметри :instance - змінна, значення	
при роботі з базою даних та класами	поля якої треба повернутиf - поле,	
представленням моделі	значення якого треба	
Методи класу		

повернутиПовертає значення що повертається	public void setName(java.lang.String name)
getStorageName	метод встановлює ім'я
public static java.lang.String	Параметри :name - ім'я
getStorageName(java.lang.Class c)	getComment
метод що повертає назву класу за анотацією Stored	public java.lang.String getComment()
Параметри :c - клас назву якого треба повернути	метод повертає характеристики
Повертає :строка з назвою	Повертає :характеристики
public class Description	setComment
Наслідує java.lang.Object	public void setComment(java.lang.String comment)
клас моделі що відповідає за збереження опису товару	метод встановлює характеристики
Методи класу	Параметри :comment - характеристика
getId	getMaker
public int getId()	public java.lang.String getMaker()
метод повертає ідентифікаційний номер	метод повертає виробника
Повертає :ідентифікаційний номер	Повертає :виробник
setId	setMaker
public void setId(int id)	public void setMaker(java.lang.String maker)
метод встановлює ідентифікаційний номер	метод встановлює виробника
Параметри :id - ідентифікаційний номер	Параметри :maker – виробник
getName	public class EditInfo
public java.lang.String getName()	Наслідує javax.swing.JDialog
метод повертає ім'я	клас що реалізує інтерфейс редагування інформації користувача
Повертає :ім'я	Конструктор
setName	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

getAddress

public java.lang.String getAddress()

метод повертає адресу

Повертає :адреса

getTelephone

public java.lang.String getTelephone()

метод повертає 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

getAddress

public java.lang.String getAddress()

метод повертає адресу

Повертає :адреса

getTelephone

public java.lang.String getTelephone()

метод повертає 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	<code>public void setDataView(java.lang.Object dataView)</code>
Методи класу	метод що встановлює посилання на вигляд
<code>run</code>	Зазначений <code>y.setDataView</code> у
<code>public void run()</code>	інтерфейсі <code>Command</code> Параметри
виконавчий метод команди	<code>:dataView</code> - посилання на вигляд
Зазначений <code>y.run</code> у	<code>public class GetAllGoodsCommand</code>
інтерфейсі <code>Command</code> Зазначений <code>y.run</code> у	Наслідує <code>java.lang.Object</code>
інтерфейсі <code>java.lang.Runnable</code>	Реалізує <code>Command</code>
<code>setDataView</code>	команда, що повертає усі товари
<code>public void setDataView(java.lang.Object dataView)</code>	Методи класу
метод для встановлення силки на об'єкт представлення	<code>run</code>
Зазначений <code>y.setDataView</code> у	<code>public void run()</code>
інтерфейсі <code>Command</code> Параметри	виконавчий метод
<code>:dataView</code> - посилання на представлення	Зазначений <code>y.run</code> у
<code>public class GetAllGoodsCommand</code>	інтерфейсі <code>Command</code> Зазначений <code>y.run</code> у
Наслідує <code>java.lang.Object</code>	інтерфейсі <code>java.lang.Runnable</code>
Реалізує <code>Command</code>	<code>setDataView</code>
команда, що повертає усі товари	<code>public void setDataView(java.lang.Object dataView)</code>
Методи класу	метод що встановлює посилання на вигляд
<code>run</code>	Зазначений <code>y.setDataView</code> у
<code>public void run()</code>	інтерфейсі <code>Command</code> Параметри
виконавчий метод	<code>:dataView</code> - посилання на вигляд
Зазначений <code>y.run</code> у	<code>public</code> <code>class</code>
інтерфейсі <code>Command</code> Зазначений <code>y.run</code> у	<code>GetGoodsForCourierCommand</code>
інтерфейсі <code>java.lang.Runnable</code>	Наслідує <code>java.lang.Object</code>
<code>setDataView</code>	

Реалізує Command

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

Методи класу

run

public void run()

виконавчий метод

Зазначений y:run у

інтерфейсіCommandЗазначений y:run у

інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object
dataView)

метод що встановлює посилання на
вигляд

Зазначений y: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 y:run у

інтерфейсіCommandЗазначений y:run у

інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object
dataView)

метод що встановлює посилання на
вигляд

Зазначений y: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 y:run у

інтерфейсіCommandЗазначений y:run у

інтерфейсіjava.lang.Runnable

setDataView

public void setDataView(java.lang.Object
dataView)

Метод що встановлює посилання на
вигляд

Зазначений y:setDataView у

інтерфейсіCommandПараметри

:dataView - посилання на вигляд

public class GetOrderListCommand

Наслідує java.lang.Object

Реалізує Command

команда що шукає список замовлень	<code>public int getNumberofgoods()</code>
Методи класу	метод повертає кількість товарів
<code>run</code>	Повертає :кількість товарів
<code>public void run()</code>	<code>setNumberofgoods</code>
виконавчий метод команди	<code>public void setNumberofgoods(int</code>
Зазначений <code>y:run</code> у	<code>numberofgoods)</code>
інтерфейсі <code>Command</code> Зазначений <code>y:run</code> у	метод встановлює кількість товарів
інтерфейсі <code>java.lang.Runnable</code>	Параметри : <code>numberofgoods</code> - кількість
<code>setDataView</code>	товарів
<code>public void setDataView(java.lang.Object</code>	<code>getMarkofgoods</code>
<code>dataView)</code>	<code>public int getMarkofgoods()</code>
Метод що встановлює посилання на	метод повертає ціну товару
вигляд	Повертає :ціна
Зазначений <code>y:setDataView</code> у	<code>setMarkofgoods</code>
інтерфейсі <code>Command</code> Параметри	<code>public void setMarkofgoods(int</code>
: <code>dataView</code> - посилання на вигляд	<code>markofgoods)</code>
<code>public class Goods</code>	метод встановлює ціну товару
Наслідує <code>java.lang.Object</code>	Параметри : <code>markofgoods</code> - ціна товару
клас моделі що відповідає за товар	<code>public class Goods_has_Description</code>
Методи класу	Наслідує <code>java.lang.Object</code>
<code>getId</code>	клас моделі що зв'язує опис і товар
<code>public int getId()</code>	Методи класу
метод повертає ідентифікаційний номер	<code>getGoods_id</code>
Повертає :ідентифікаційний номер	<code>public int getGoods_id()</code>
<code>setId</code>	повертає ідентифікаційний номер
<code>public void setId(int id)</code>	товару
метод встановлює ідентифікаційний	Повертає :ідентифікаційний номер
номер	<code>setGoods_id</code>
Параметри : <code>id</code> - ідентифікаційний номе	<code>public void setGoods_id(int goods_id)</code>
<code>getNumberofgoods</code>	

метод встановлює ідентифікаційний номер товару	java.util.List<Description> descriptions,
Параметри :goods_id - ідентифікаційний номер	java.util.List<Position> positions)
getDescription_id	метод що оновлює інформацію для представлення
public int getDescription_id()	Параметри :goods - список товарів
метод повертає ідентифікаційний номер опису	descriptions - список описів
Повертає :ідентифікаційний номер	positions - список - позицій
setDescription_id	getCourierId
public void setDescription_id(int description_id)	public java.lang.Integer getCourierId()
метод встановлює ідентифікаційний номер опису	метод повертає ідентифікаційний номер кур'єра
Параметри :description_id – ідентифікаційний номер опису	Повертає :ідентифікаційний номер
public class GoodsForCourier	setCourierId
Наслідує javax.swing.JPanel	public void setCourierId(java.lang.Integer courierId)
Клас що реалізує графічний інтерфейс користувача для відображення товарів для кур'єра	метод встановлює ідентифікаційний номер кур'єра
Методи класу	Параметри :courierId - ідентифікаційний номер
update	getProfile
public void update(Profile profile)	public Profile getProfile()
метод що оновлює дані для інтерфейсу	метод що повертає профайл
Параметри :profile - профайл користувача	Повертає :профайл
update	setProfile
public void update(java.util.List<Goods> goods,	public void setProfile(Profile profile)
	метод що встановлює профайл
	Параметри :profile - профайл
	setStyle

<code>public static void setStyle(java.lang.String className)</code>	Наслідує <code>java.lang.Object</code> Реалізує <code>Command</code>
метод що встановлює стиль вікна	команда, що визначає чи оплачене замовлення
Параметри :className - назва стилю	
<code>public class intConverter</code>	Методи класу
Наслідує <code>java.lang.Object</code>	<code>run</code>
Реалізує <code>ValueConverter</code>	<code>public void run()</code>
конвертер для типів з якими оперує модель даних у програмі	виконавчий метод
Методи класу	Зазначений <code>y:run</code> у інтерфейсі <code>Command</code> Зазначений <code>y:run</code> у інтерфейсі <code>java.lang.Runnable</code>
<code>toString</code>	
<code>public <T> java.lang.String toString(T value)</code>	<code>setDataView</code> <code>public void setDataView(java.lang.Object dataView)</code>
метод перетворює ціле значення у об'єкт Класу <code>String</code>	Метод що встановлює посилання на вигляд
Зазначений <code>y:toString</code> у інтерфейсі <code>ValueConverter</code> Параметри :value - що перетворюємо у <code>String</code> Повертає :строкове представлення value	Зазначений <code>y:setDataView</code> у інтерфейсі <code>Command</code> Параметри :dataView - посилання на вигляд
<code>toValue</code>	<code>public class MainClass</code> Наслідує <code>java.lang.Object</code>
<code>public <T> T toValue(java.lang.String str)</code>	головний клас. тут відбувається запуск програми. Створення контролера
метод повертає значення типу T що задані об'єктом типу <code>String</code>	Методи класу
Зазначений <code>y:toValue</code> у інтерфейсі <code>ValueConverter</code> Параметри :str - строка з якої беремо значенняПовертає :Integer представлення строки str	<code>main</code> <code>public static void main(java.lang.String[] args)</code>
<code>public class IsPayCommand</code>	головний метод <code>public class MainFrame</code> Наслідує <code>javax.swing.JFrame</code>

Клас що реалізує головне вікно програми	Методи класу
Конструктор	open
MainFrame	public void open(AppConfig config)
public MainFrame()	Виключні
конструктор головного вікна.	ситуаціїjava.lang.Exception
Методи класу	Description copied from interface: CRUDInterface
setStyle	метод що встановлює зв'язок з базою даних
public void setStyle(java.lang.String className)	Зазначений у:open у інтерфейсіCRUDInterface
метод встановлює стиль головного вікна та меню	Параметри :config - конфігурація для відкриття з'єднання з базою даних
Параметри :className - назва стилю	Throws:java.lang.Exception
public class MySqlCRUD	close
Наслідує java.lang.Object	public void close()
Реалізує CRUDInterface	Виключні
клас що працює з базою даних	ситуаціїjava.lang.Exception
Конструктор	Description copied from interface: CRUDInterface
MySqlCRUD	метод що відкриває з*єднання
public MySqlCRUD(MySqlCRUD mySqlCRUD)	Зазначений у:close у інтерфейсіCRUDInterface
конструктор класу.	Throws:java.lang.Exception
Параметри :mySqlCRUD - наступний об'єкт для роботи з базою.	insert
MySqlCRUD	public <T> T insert(T instance)
public MySqlCRUD(AppConfig config)	Виключні
конструктор класу	ситуаціїjava.lang.Exception
Параметри :config - конфігурація для під'єднання до бази даних	Description copied from interface: CRUDInterface

метод що записує інформацію в базу даних

Зазначений `y:insert` у інтерфейсі `CRUDInterface` Параметри `:instance` - значення які треба записати Повертає

`:Null` `Throws: java.lang.Exception`

`read`

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

`int id)`

Виключні

ситуації `java.lang.Exception`

метод що зчитує інформацію із бази даних

Зазначений `y:read` у інтерфейсі `CRUDInterface` Тип

Параметри `:T` - тим з яким працює метод Параметри `:entityClass` - клас що представляє таблицю з якої треба зчитати дані `id` - ключ для

пошуку Повертає значення типу моделі з якою буде працювати програма `Throws: java.lang.Exception`

`update`

`public <T> void update(T instance)`

Виключні

ситуації `java.lang.Exception`

метод що поновлює дані з у базі.

Зазначений `y:update` у інтерфейсі `CRUDInterface` Тип

Параметри `:T` - тип цього значення Параметри `:instance` - значення що треба

оновити `Throws: java.lang.Exception`

`delete`

`public <T> void delete(T instance)`

Виключні

ситуації `java.lang.Exception`

метод що виконує запит видалення інформації з бази даних

Зазначений `y:delete` у інтерфейсі `CRUDInterface` Тип

Параметри `:T` - тип даних Параметри `:instance` - дані що треба видалити `Throws: java.lang.Exception`

`select`

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

Виключні

ситуації `java.lang.Exception`

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

Зазначений `y:select` у інтерфейсі `CRUDInterface` Тип

Параметри `:T` - тип даних що повертаються у списку Параметри `:entityClass` - клас що відповідає таблиці з якої вибираємо дані Повертає

:список	значень	метод	встановлює	ідентифікаційний
моделі	Throws:java.lang.Exception		норме	замовлення
select			Параметри	:id - ідентифікаційний
public	<T>	java.util.List<T>	номер	замовлення
select(java.lang.String SQLString)			getIsOnCourier	
	Виключні		public boolean getIsOnCourier()	
ситуації	java.lang.Exception		метод	повертає статус замовлення
Зазначений	y:select	y	Повертає	:статус
інтерфейсі	CRUDInterfaceThrows:java.la		setIsOnCourier	
ng.Exception			public void	setIsOnCourier(boolean
public class OrderGoods			isOnCourier)	
Наслідуює	java.lang.Object		метод	встановлює статуст замолення
Клас що представляє собою замовлення			Параметри	:isOnCourier – статус
Методи класу			public class OrderGoodsList	
getIsPay			Наслідуює	javax.swing.JDialog
public boolean getIsPay()			клас	вигляду що реалізує виведення
повертає статус замевлення			користувачу	списку товарів
Повертає	:статус		Конструктор	
setIsPay			OrderGoodsList	
public void setIsPay(boolean isPay)			public OrderGoodsList()	
встановлює статус замовлення			Конструктор	вікна. Створює та
Параметри :isPay - статус			відображає	елементи графічного
getId			інтерфейсу	користувача
public int getId()			Методи класу	
повертає	ідентифікаційний	номер	setStyle	
замовлення			public static void setStyle(java.lang.String	
Повертає	:ідентифікаційний	номмер	className)	
setId			Метод що встановлює стиль вікна	
public void setId(int id)			Параметри :className - назва стилю	
			public class OrderList	

Наслідує java.lang.Object	public abstract java.lang.String key
клас моделі що відповідає за роботу з	ключ параметру
списком замовлень	Повертає :значення ключа
Методи класу	type
getProfile_id	public abstract java.lang.Class<?> type
public int getProfile_id()	тип ключа параметру. По
повертає ідентифікаційний номе	замовчуванню Object.class
користувача	Повертає :тип
Повертає :ідентифікаційний номер	ключаDefault:java.lang.Object.class
setProfile_id	optional
public void setProfile_id(int profile_id)	public abstract boolean optional
метод встановлює ідентифікаційний	реалізований опціонал. По
номер користувача	замовчуванню false
Параметри :profile_id -	Повертає :існування
ідентифікаційний номер користувача	реалізаціїDefault:false
getOrder_id	public class Position
public int getOrder_id()	Наслідує java.lang.Object
метод повертає номер замовлення	Клас моделі, що зв'язує товари, та
Повертає :номер замовлення	конкретне замовлення
setOrder_id	Методи класу
public void setOrder_id(int order_id)	getId
метод встановлює номер замовлення	public int getId()
Параметри :order_id - номер замолення	метод що повретає ідентифікаційний
@Retention(value=RUNTIME)	норер замолення
@Target(value=TYPE)	Повертає :номер замовлення
public @interface PARAMETER	setId
анотація для кожного параметру	public void setId(int id)
команди	
Елементи	метод що встановлює ідентифікаційний
key	номер замовлення

Параметри :id - ідентифікаційний номер замовлення	клас моделі що відображає інформацію користувача
getCount	Методи класу
public int getCount()	getId
метод що повертає кількість товарів в позиції	public int getId()
Повертає :кількість товарів	метод поретає ідентифікаційний номер користувача
setCount	Повертає :ідентифікаційний номер
public void setCount(int count)	setId
метод що встановлює кількість товарів	public void setId(int id)
Параметри :count - кількість товарів	метод встановлює ідентифікаційний номер
getOrder_id	Параметри :id - ідентифікаційний номер
public int getOrder_id()	getLogin
метод що повертає номер замовлення	public java.lang.String getLogin()
Повертає :номер замовлення	метод повертає логін
setOrder_id	Повертає :логін
public void setOrder_id(int order_id)	setLogin
метод що встановлює номер замовлення	public void setLogin(java.lang.String login)
Параметри :order_id - номер замовлення	метод встановлює логін
getGoods_id	Параметри :login - логін
public int getGoods_id()	getPassword
метод що повертає номер товару	public java.lang.String getPassword()
Повертає :номер товару	метод повертає пароль
setGoods_id	Повертає :пароль
public void setGoods_id(int goods_id)	setPassword
метод що встановлює номер товару	public void setPassword(java.lang.String password)
Параметри :goods_id - номер товару	
public class Profile	
Наслідує java.lang.Object	

метод встановлює пароль	<code>public void setCategory_id(int</code>
Параметри :password - пароль	<code>category_id)</code>
<code>getState_id</code>	метод встановлює номер категорії
<code>public int getState_id()</code>	користувача
метод повертає номер стану	Параметри :category_id - номер
користувача	категорії
Повертає :номер стану	<code>public interface Reader</code>
<code>setState_id</code>	Інтерфейс для зчитувачів конфігурацій
<code>public void setState_id(int state_id)</code>	Методи класу
метод встановлює номер стану	<code>load</code>
користувача	<code>Configuration load()</code>
Параметри :state_id - номер стану	Клас-зчитувач повинен реалізовувати
<code>getContactinformation_id</code>	метод <code>load()</code> для зчитування
<code>public int getContactinformation_id()</code>	конфігурації
метод повертає номер контактної	Повертає :об'єкт конфігурації
інформації користувача	<code>public class RegistrationCommand</code>
Повертає :номер контактної інформації	Наслідує <code>java.lang.Object</code>
<code>setContactinformation_id</code>	Реалізує <code>Command</code>
<code>public void setContactinformation_id(int</code>	команда що виконує реєстрацію нового
<code>contactinformation_id)</code>	користувача
метод встановлює номер контактної	Методи класу
інформації користувача	<code>run</code>
Параметри :contactinformation_id -	<code>public void run()</code>
номер контактної інформації	виконавчий метод
<code>getCategory_id</code>	Зазначений у:run у
<code>public int getCategory_id()</code>	інтерфейсі <code>Command</code> Зазначений у:run у
метод повертає номер категорії	інтерфейсі <code>java.lang.Runnable</code>
користувача	<code>setDataView</code>
Повертає :номер категорії користувача	<code>public void setDataView(java.lang.Object</code>
<code>setCategory_id</code>	<code>dataView)</code>

Метод що встановлює посилання на вигляд

Зазначений `y:setDataView`

інтерфейсі `Command` Параметри

`: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 `y:run`

інтерфейсі `Command` Зазначений `y:run`

інтерфейсі `java.lang.Runnable`

`setDataView`

`public void setDataView(java.lang.Object
dataView)`

Метод що встановлює посилання на вигляд

Зазначений `y:setDataView`

інтерфейсі `Command` Параметри

`:dataView` - посилання на вигляд

`public class ResupplyGoods`

Наслідує `javax.swing.JPanel`

Клас, що реалізує інтерфейс користувача для поповнення товару

Конструктор

50

`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

товарів `descriptions` - список описів

`getNumberOfGoods`

`public java.lang.Integer`

`getNumberOfGoods()`

метод що повертає кількість 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 setStyle(java.lang.String style)
public void setGoodsID(java.lang.Integer goodsID)	метод що встановлює стиль
метод що встановлює ідентифікаційний номер товару	Параметри :style - назва стилю
Параметри :goodsID - ідентифікаційний номер	public class SetGoodsList
setStyle	Наслідує javax.swing.JDialog
public static void setStyle(java.lang.String className)	клас вигляду що реалізує відображення списку товарів
метод, що встановлює стиль вікна	Конструктор
Параметри :className - назва стилю	SetGoodsList
	public
	SetGoodsList(java.util.List<Goods> goods,
	java.util.List<Description>
	descriptions,
	java.util.List<Position> positions)
public class SelectStyle	створення нового вікна
Наслідує javax.swing.JDialog	Параметри :goods - список товарів
Клас, що реалізує інтерфейс користувача для вибору стилю вікон	descriptions - список описів
Конструктор	positions - список позицій
SelectStyle	Методи класу
public SelectStyle(MainFrame mainFrame)	
Створення вікна для вибору стилю.	setStyle
Параметри :mainFrame - головне вікно програми	public static void setStyle(java.lang.String className)
Методи класу	метод, що встановлює стиль вікна
getStyle	Параметри :className - назва стилю
public java.lang.String getStyle()	public class State
метод, що повертає стиль вікна	Наслідує java.lang.Object
Повертає :назва стилю	клас моделі, що визначає стан користувача
setStyle	

Методи класу	<code>public abstract java.lang.Class converter</code>
<code>getId</code>	конвертер для даного елемента, по
<code>public int getId()</code>	замовчуванню <code>StringConverter.class</code>
метод що повертає ідентифікаційний	Повертає :клас
номер стану	конвертора <code>Default:dao.annotation.utils.co</code>
Повертає :ідентифікаційний номер	<code>nverter.StringConverter.class</code>
<code>setId</code>	<code>public class StringConverter</code>
<code>public void setId(int id)</code>	Наслідує <code>java.lang.Object</code>
метод що встановлює ідентифікаційний	Реалізує <code>ValueConverter</code>
номер стану	Строковий конвертор
Параметри :id - ідентифікаційний	Методи класу
номер стану	<code>toString</code>
<code>getState</code>	<code>public <T> java.lang.String toString(T</code>
<code>public java.lang.String getState()</code>	<code>value)</code>
метод що повертає стан	перетворення значення <code>value</code> у строку з
Повертає :стан	типу <code>T</code>
<code>setState</code>	Зазначений <code>y:toString</code> у
<code>public void setState(java.lang.String state)</code>	інтерфейсі <code>ValueConverter</code> Параметри
метод що встановлює стан користувача	: <code>value</code> - значення для
Параметри :state – стан	перетворення Повертає :строкове
<code>@Retention(value=RUNTIME)</code>	значення з типу <code>T</code>
<code>public @interface Stored</code>	<code>toValue</code>
анотація для зчитування мета-даних з	<code>public <T> T toValue(java.lang.String str)</code>
моделі	перетворення із строкового значення в
Елементи	значення <code>T</code>
<code>name</code>	Зазначений <code>y:toValue</code> у
<code>public abstract java.lang.String name</code>	інтерфейсі <code>ValueConverter</code> Параметри : <code>str</code>
ім'я	- строка що перетворюємо у значення
Повертає :значення	типу <code>T</code> Повертає :значення типу <code>T</code>
<code>converter</code>	<code>public class TabbedPane</code>

Наслідує javax.swing.JPanel	isValidate
клас вигляду що реалізує відображення панелей	public static boolean isValidate(java.lang.Class<Command> command,
Field Detail	CONTEXT context)
tabbedPane	метод що виконує валідацію
public static javax.swing.JTabbedPane tabbedPane	Параметри :command - командаcontext - контекст командиПовертає :результат валідації
панель вкладок	public interface ValueConverter
	інтерфейс для конвертерів типів
Конструктор TabbedPane	toString
public TabbedPane()	<T> java.lang.String toString(T value)
створення нової панелі	переведення зі значення типу T у строку
Методи класу	Параметри :value - значення що конвертуєтьсяПовертає :конвертований об'єкт
addTabPanel	toValue
public static void addTabPanel(javax.swing.JPanel panel, java.lang.String name)	<T> T toValue(java.lang.String str)
метод, що додає до панелі нову вкладку	конвертер із строки в значення типу T
Параметри :panel - нова вкладкаname - ім'я вкладки	Параметри :str - строка з якої конвертуватиПовертає :значення
setStyle	public class WorkerInfo
public static void setStyle(java.lang.String className)	Наслідує javax.swing.JPanel
метод, що встановлює новий стиль	клас вигляду що реалізує відображення інформації про користувача
Параметри :className - назва стилю	public WorkerInfo()
public class Validator	конструктор класу. Створює вікно
Наслідує java.lang.Object	Методи класу
Валідатор для команд	
Методи класу	

update	виконавчий метод
public void update(java.util.HashMap<java.lang.String,java.lang.String> mapinfo)	Зазначений у:run у інтерфейсіCommandЗазначений у:run у інтерфейсіjava.lang.Runnable
метод що оновлює дані графічного інтерфейсу	setDataView
Параметри :mapinfo - дані	public void setDataView(java.lang.Object dataView)
getProfile_id	Метод що встановлює посилання на вигляд
public int getProfile_id()	
метод що повертає профайл	Зазначений у:setDataView у інтерфейсіCommandПараметри :dataView - посилання на вигляд
Повертає :профайл	
setProfile_id	public interface Writer
public void setProfile_id(int profile_id)	інтерфейс для райтерів
метод, що встановлює ідентифікаційний номер профайла	Методи класу
Параметри :profile_id - ідентифікаційний номер профайла	save
setStyle	void save(Configuration configuration)
public static void setStyle(java.lang.String className)	метод записує конфігурацію
метод що встановлює стиль вікна	Параметри :configuration - конфігурація для запису
Параметри :className - назва стилю	public class XMLReader
public class WorkerInfoCommand	Наслідує java.lang.Object
Наслідує java.lang.Object	Реалізує Reader
Реалізує Command	Клас для зчитування конфігурацій з XML файла
команда що шукає інформацію про користувача	Конструктор XMLReader
Методи класу	public XMLReader(java.lang.String source)
run	
public void run()	

конструктор класу. встановлює адресу XML-файла	Клас-записувач. Записує конфігурацію в XML файл
Параметри :source - адреса XML файла	Конструктор
XMLReader	XMLWriter
public XMLReader()	public XMLWriter(java.lang.String
пустий конструктор. Буде	source)
використовуватись адреса за	конструктор класу.
замовчуванням.	Параметри :source - встановлює адресу
Методи класу	файлу запису замість тієї, що
load	викоритсовується по замовчуванню.
public Configuration load()	Методи класу
метод завантажує файл конфігурації	save
Зазначений у:load у	public void save(Configuration
інтерфейсіReaderПовертає :об'єкт	configuration)
конфігурації	метод записує коннфігурацію
public class XMLWriter	Зазначений у:save у інтерфейсі
Наслідує java.lang.Object	WriterПараметри :onfiguration - об'єкт
Реалізує Writer	конфігурації для запису у XML файл

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

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

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

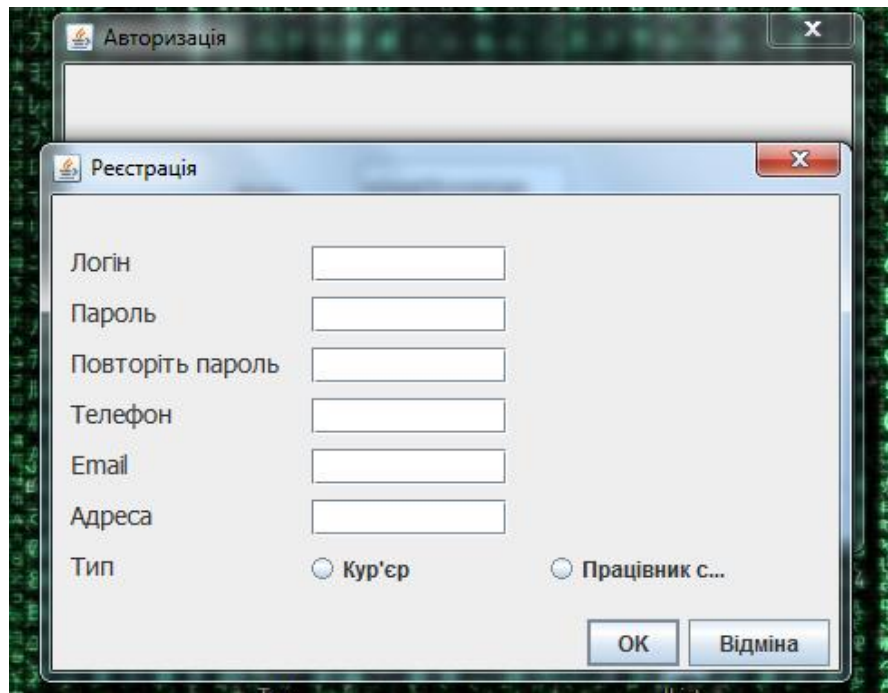


Рис 4.1 Вікно реєстрації

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

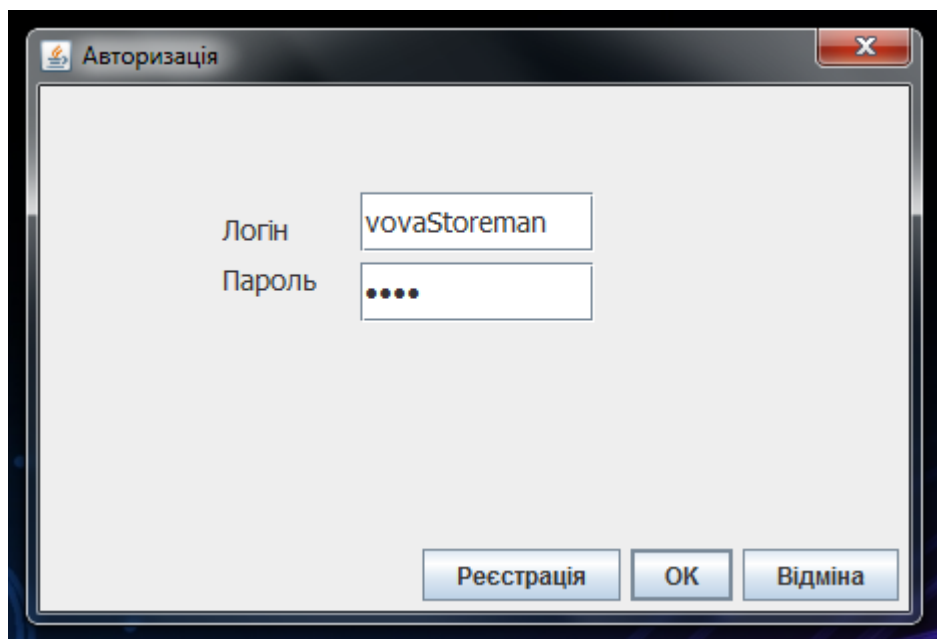


Рис4.2 Вікно авторизації

Реалізація прецеденту (рис 2.7) перегляду особистої інформації (рис 4.3), та редагування особистої інформації (рис 4.4).

The screenshot shows a window titled 'Меню Робітник складу Кур'єр Налаштування'. The 'Особиста інформація' tab is selected. It displays a list of personal information fields with their corresponding values:

Логін	vovaStoreman
Пароль	1111
Email	vovamail.ru
Телефон	34534
Адреса	dfgsfd
ID	1
Тип	Storeman
Стан	Active

At the bottom right, there are two buttons: 'OK' and 'Внести зміни'.

Рис 4.3 Вкладка «Особиста інформація»

The screenshot shows the same 'Меню Робітник складу Кур'єр Налаштування' window, but with a modal dialog box open for editing personal information. The dialog box contains the following fields:

- Логін
- Старий пароль
- Новий пароль
- Повторіть пароль
- Email
- Адреса
- Телефон

At the bottom right of the dialog box, there are two buttons: 'OK' and 'Cancel'.

Рис4.4 Діалог редагування особистої інформації

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

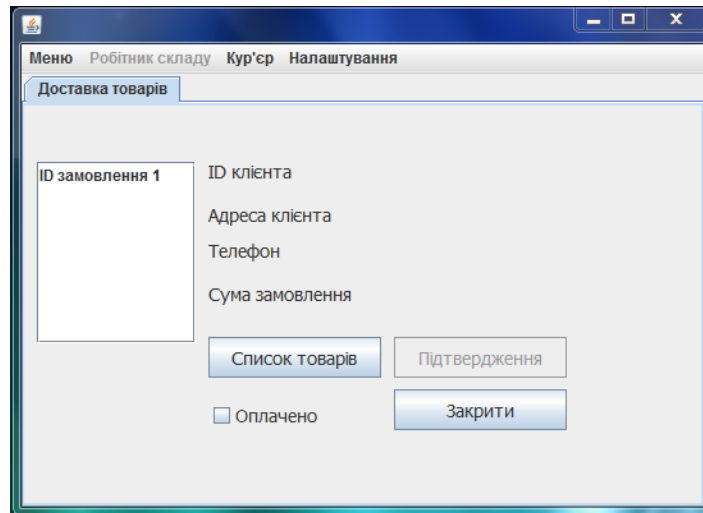


Рис 4.5 вкладка «Доставка товарів»

Перегляд списку товарів (рис 4.6).

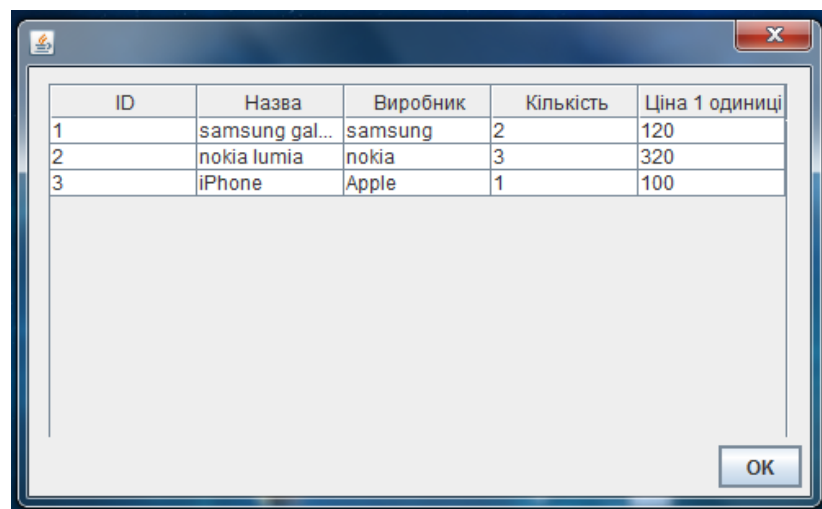


Рис 4.6. Список товарів

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

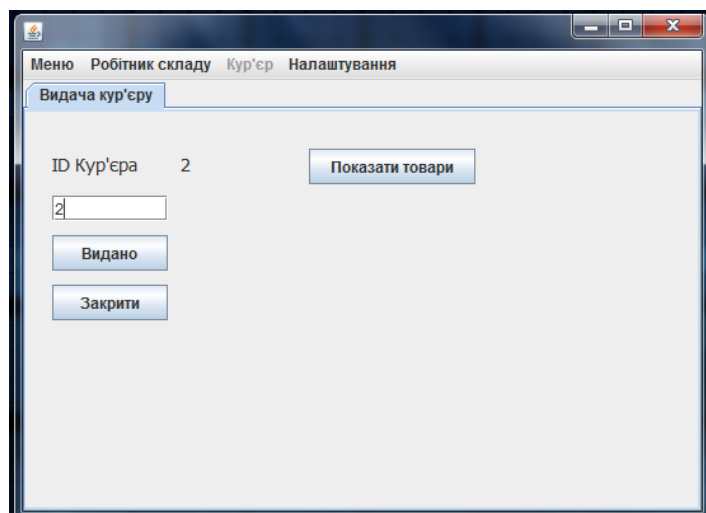


Рис 4.7. вкладка «Видача кур'єру»

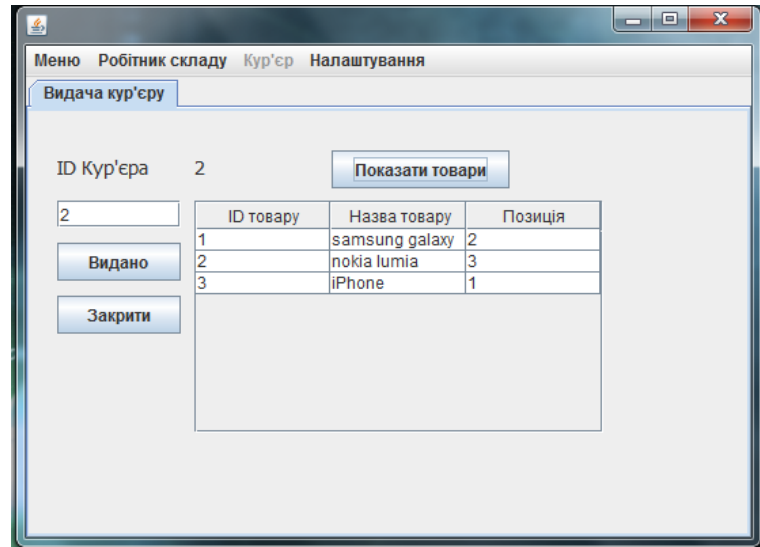


Рис 4.8. відображення товарів для видачі.

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

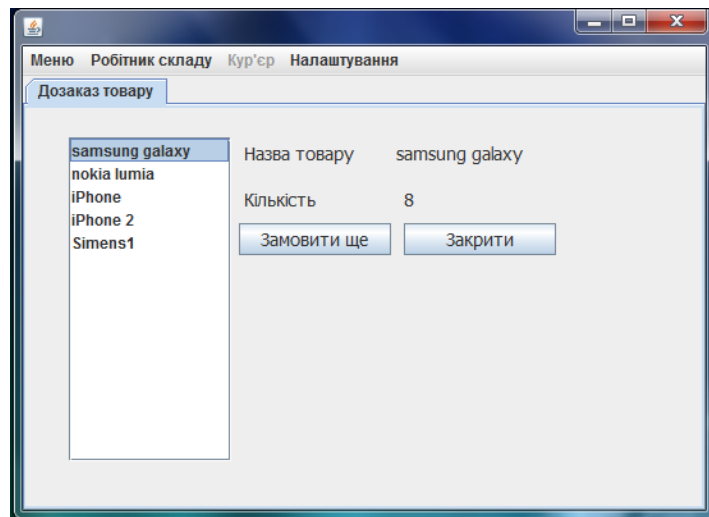


Рис 4.9 Вкладка «Дозаказ товару»

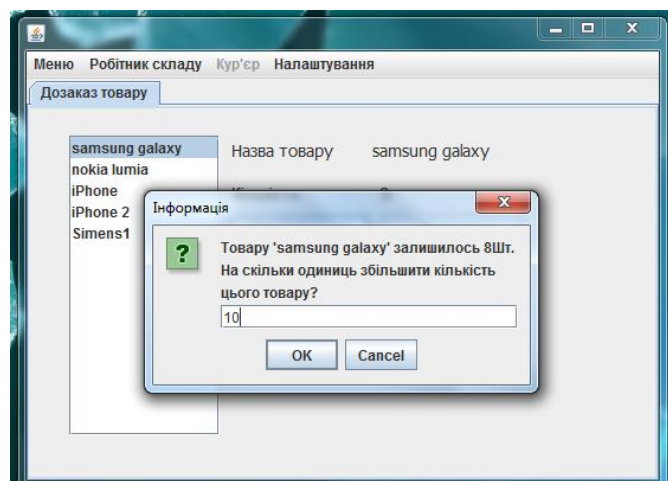


Рис 4.10. Діалог поповнення товару

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

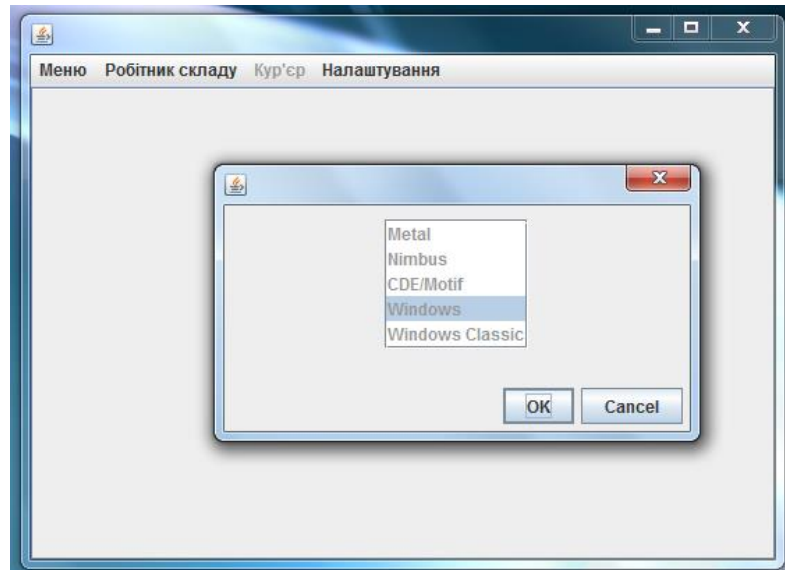


Рис 4.11. Вікно зміни стилю.

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

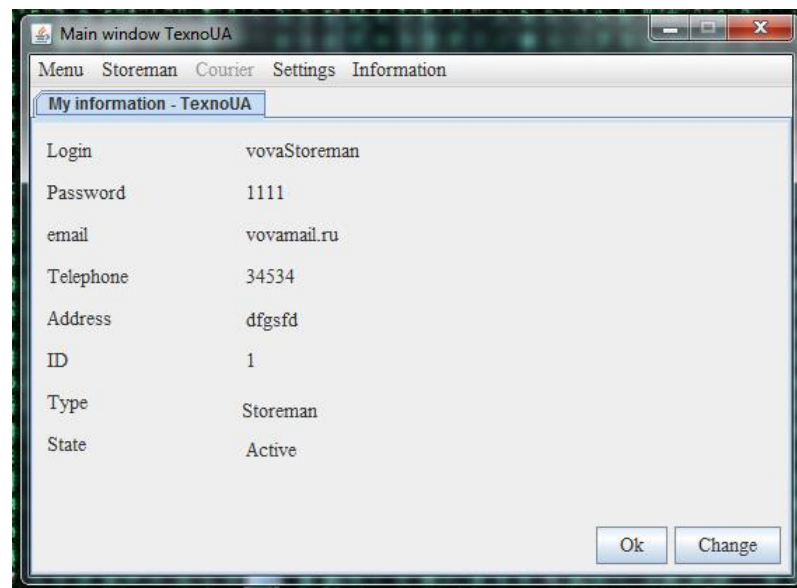


Рис. 4.12 Інтерфейс англійською мовою

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.

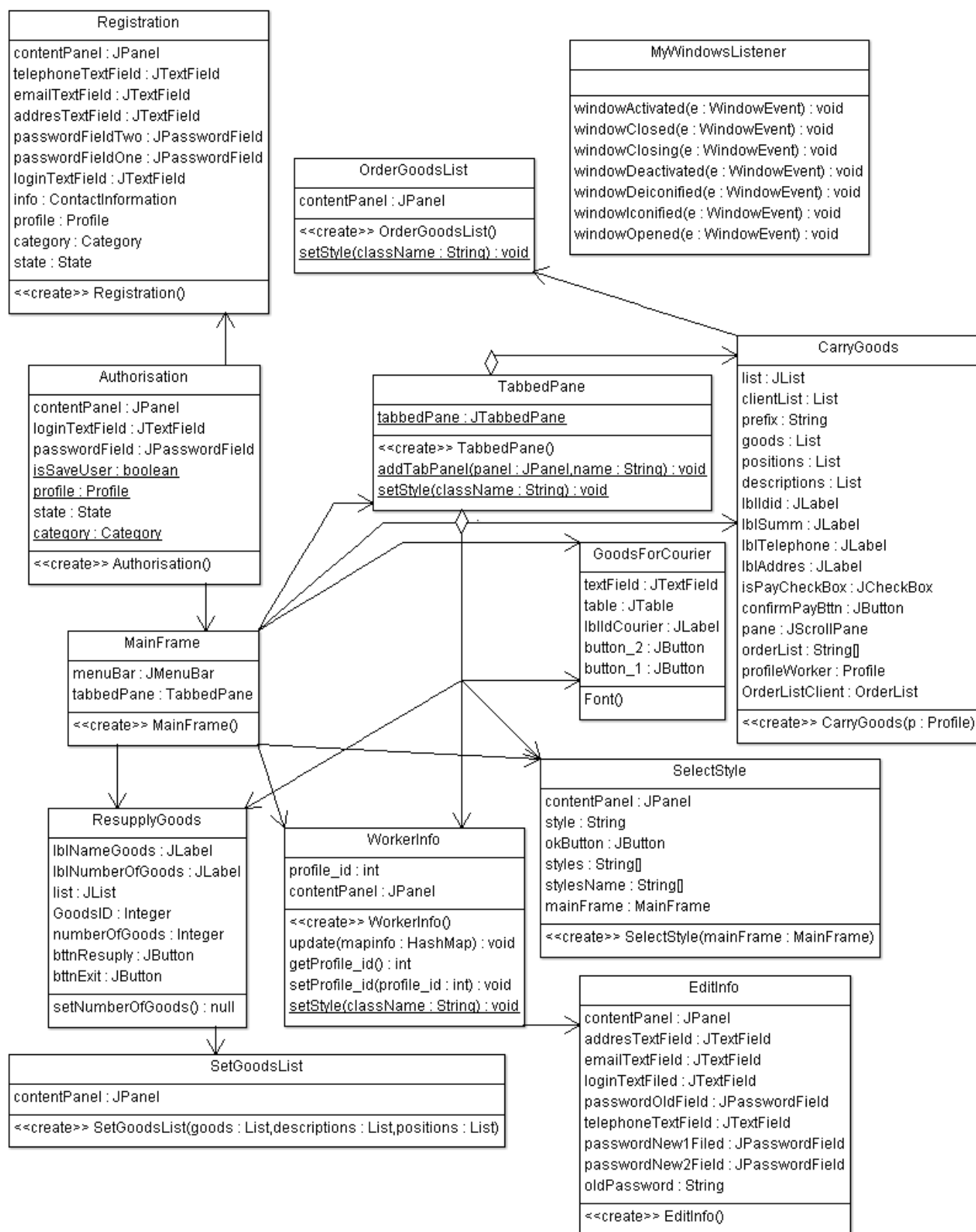
Використана база даних MySQL. Реалізовано операції для роботи з цією базою даних. Для роботи з базою даних реалізовано шаблон проектування DAO.

Графічний інтерфейс користувача реалізований засобами бібліотеки SWING

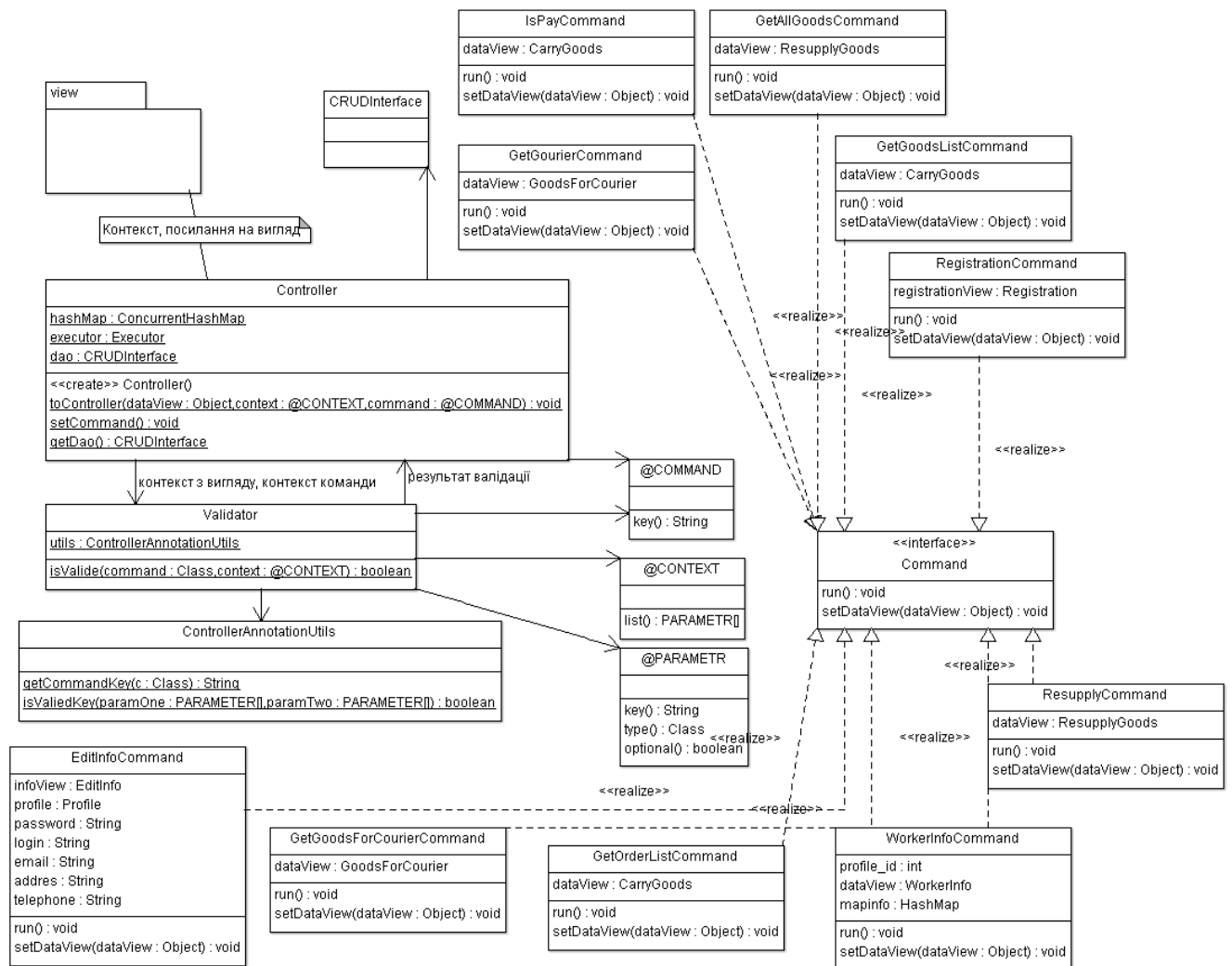
СПИСОК ВИКОРИСТАНИ ДЖЕРЕЛ

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 с.: 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 с.: 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 с.: 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 текущего 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(addrres);
            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 об'єкт 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 {
    /**
     * посилення 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 текущего 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 {
    /**
     * посилення 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
 *      посилає 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 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 {
/**
 * посилення 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 {
/**
 * посилення 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 {
    /**
     * посилення 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;

```

}

}