НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**КУРСОВА РОБОТА**

з дисципліни «Інженерія програмного забезпечення»

на тему: «Інтернет-магазин. Функціональність для адміністратора»

Студента 2 курсу групи ІО-21

напряму підготовки 6.050102 Комп’ютерна інженерія

спеціальності 7.8.05010201 Комп’ютерні системи та мережі

Коноз Андрій Олександрович

Керівник доцент, кандидат технічних наук

Абу Усбах Олексій Нідалійович

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**

В даній курсовій роботі розглянуті етапи створення програмного продукту. Даний програмний продукт є частиною набору програм для інтернет-магазину. Програма реалізує функціональність адміністратора даного магазину.

В першому розділі представлений огляд схеми використання декількох шаблонів проектування Model-view-controller («Модель-вигляд-контролер»).

В другому розділі відбувається опис програмного продукту з використанням UML діаграм класів і діаграм прецедентів. Також в цьому розділі приводиться детальний опис інтерфейсу користувача.

В третьому розділі надається перелік засобів, які були використані при створенні програми, а також фото інтерфейсу і інструкція для користувача даної програми.

В четвертому розділі приводяться результати тестування програми.

**Зміст**

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

1. ОГЛЯД MVС………………………………………………………………………...5
   1. Загальні відомості про MVC……………………………………………..…….5
   2. Model-View-Presenter………………………………………………………..….7
   3. Model-View-View-Model…………………………………………………...…...9
   4. Model–view–adapter……………………………………………………..……..10

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ…………………….….12

2.1 Прецеденти……………………………………………………………………..12

2.2 Діаграма граничних класів……………………………………………………24

2.3 Проектування графічного інтерфейсу………………………………………..31

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ………………………..……….33

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……………………………52

ВИСНОВОК……………………………………………………………………...…....73

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………………….76

ДОДАТКИ……………………………………………………………………...…….77

1.ВСТУП.

Метою курсової роботи є отримання навичок у розробці програмного забезпечення з графічним інтерфейсом користувача, ознайомлення і засвоєння основних навиків роботи з графічними пакетами мови програмування Java Swing та AWT а також освоєння принципів роботи з базами даних. З цією метою розробляється програмний додаток, який реалізує функціональність для адміністратора інтернет-магазину. Програмний додаток повинен мати простий і зручний інтерфейс, а також набір операцій, які дозволять керувати магазином. Розробка даного програмного забезпечення проводитиметься на мові програмування Java в середовищі Eclipse. Робота містить повну документацію до коду програми та інструкцію для користувача.

1. ОГЛЯД MVС

На сьогоднішній день окрім шаблону MVC існує безліч його модифікацій,

які мають свої переваги і недоліки. Метою даного розділу є ознайомлення з найбільш поширеними модифікаціями цього патерну.

2.1. Загальний огляд MVC.

Model - view - controller ( MVC або «модель - відображення - контролер» ) – являє собою схему використання декількох шаблонів проектування, за допомогою яких модель даних програми, інтерфейс користувача і взаємодія з користувачем розділені на три окремих компоненти таким чином, щоб редагування одного з цих компонентів не впливало на два інші. Дана схема проектування часто використовується для побудови архітектурного каркаса, коли переходять від теорії до реалізації в конкретній предметній області[1].

Концепція MVC була описана в 1979 році Трюгве Реенскаугом, який на той час займався розробкою Smalltalk в Xerox PARC. Оригінальна реалізація описана в статті «Applications Programming in Smalltalk-80: How to use Model-View-Controller». Потім Джим Алтофф з командою розробників реалізували версію MVC для бібліотеки класів Smalltalk-80.

В початковій концепції була описана роль кожного з елементів: моделі, відображення і контролера. Але зв'язки між ними не були описані конкретно . Існує два різновиди даного шаблону:

1. Пасивна модель. Модель жодним чином не впливає на контролер чи відображення, і використовується ними як джерело даних для відображення. Всі зміни стану моделі відслідковує контролер. Контролер також відповідає за оновлення відображення, якщо це необхідно. Така модель частіше використовується в структурному програмуванні , так як в цьому випадку модель являє просто структуру даних , без методів їх обробки.
2. Активна модель. Ця модель, на відміну від пасивної, оповіщає відображення про те, що в ній відбулися зміни, а відображення, які зацікавлені в оповіщенні , підписуються на ці повідомлення. Це дозволяє зберегти незалежність моделі як від контролера , так і від відображення.

Класичною реалізацією концепції MVC прийнято вважати версію саме з активною моделлю[2] .

В основі класичної моделі лежать такі принципи:

1. Слабке зв'язування.
2. Модель нічого не знає ні про кого. Модель розсилає оповіщення, які може слухати, наприклад, відображення, якщо хоче.
3. Відображення знає про модель але не може її змінювати, відображення може маніпулювати контролером.
4. Контролер знає про Моделі і може її змінювати, а також знає про відображення і може його (їх) змінювати.

Концепція MVC дозволяє розділити дані , подання та обробку дій користувача на три окремих компонента :

1. Модель. Модель надає знання: дані і методи роботи з цими даними, реагує на запити, змінюючи свій стан. Не містить інформації, як ці знання можна візуалізувати.
2. Відображення, подання. Відповідає за відображення інформації (візуалізацію) . Часто як відображення виступає форма (вікно) з графічними елементами .
3. Контролер. Забезпечує зв'язок між користувачем і системою: контролює введення даних користувачем і використовує модель та відображення для реалізації необхідної реакції.

Важливо відзначити , що як відображення , так і контролер залежать від моделі. Однак модель не залежить ні від відображення, ні від контролера[3]. Тим самим досягається призначення такого поділу: вон дозволяє будувати модель незалежно від візуального представлення, а також створювати кілька різних відображень для однієї моделі .

Для реалізації схеми Model -View - Controller використовується досить велика кількість шаблонів проектування (залежно від складності архітектурного рішення) , основні з яких « спостерігач », «стратегія», « компонувальник » .

Найбільш типова реалізація відокремлює відображення від моделі шляхом встановлення між ними протоколу взаємодії , використовуючи апарат подій (підписка або сповіщення) . При кожній зміні внутрішніх даних в моделі вона оповіщає всі залежні від неї відображення, і відображення оновлюється. Для цього використовується шаблон «спостерігач». При обробці реакції користувача відображення обирає, залежно від потрібної реакції, потрібний контролер , який забезпечить той чи інший зв'язок з моделлю . Для цього використовується шаблон «стратегія» , або замість цього може бути модифікація з використанням шаблону «команда». А для можливості однотипного поводження з підоб'єктами складно- складеного ієрархічного виду може використовуватися шаблон «компонувальник». Крім того, можуть використовуватися й інші шаблони проектування, наприклад, «фабричний метод», який дозволить задати за замовчуванням тип контролера для відповідного виду[4] .

З розвитком об'єктно-орієнтованого програмування та поняття про шаблони проектування був створений ряд модифікацій концепції MVC , які при реалізації у різних авторів можуть відрізнятися від оригінальної .

2.2. Model-View-Presenter.

Model-View-Presenter (MVP) - шаблон проектування, похідний від MVC, який використовується в основному для побудови інтерфейсу користувача.

У MVP Presenter бере на себе функціональність посередника (граючи роль, аналогічну контролеру в MVC). Крім того, Presenter відповідає за управління подіями для користувача інтерфейсу (наприклад, mouseDown, keyDown і т.д.), яке зазвичай було турботою відображення. В результаті, модель стає чітко моделлю предметної області.

MVP - шаблон проектування інтерфейсу користувача, який був розроблений для полегшення автоматичного модульного тестування і поліпшення розподілу відповідальності у презентаційній логіці (відділення логіки від відображення) :

1. Модель являє собою інтерфейс , що визначає дані, які відображаються або беруть участь в інтерфейсі іншим чином .
2. Відображення - це інтерфейс, який відображає дані (Модель) і маршрутизує команди користувача (або події) Presenter-у, щоб той виконував дії над цими даними.
3. Presenter діє над Моделлю і Відображенням . Він витягає дані зі сховища (Моделі) і форматує їх для подання у Відображення.

Зазвичай, реалізація Відображення створює екземпляр конкретного об'єкта Presenter-а і надає посилання на себе. (Формально параметром конструктора Presenter'a є інтерфейс Відображення, в той час як в конструктор передається конкретна реалізація Відображення) . Коли викликається подія Відображення, воно викликає конкретний метод Presenter'а , що не має параметрів і значення, що повертається . Далі Presenter через інтерфейс отримує дані з Відображення, після чого викликає методи Моделі і встановлює через інтерфейс дані з Моделі в Відображення[5].

Кількість логіки, допустимої в Відображенні, різниться для різних реалізацій.

З точки зору багаторівневої моделі програм в ООП, Presenter може розглядатися як самостійний рівень між рівнем програми і рівнем інтерфейсом користувача.

2.3. Model-View-View-Model.

Шаблон Model-View-View-Model (MVVM) - застосовується при проектуванні архітектури програми. Спочатку був представлений співтовариству Джоном Госсманом в 2005 році як модифікація шаблону Presentation Model. Більшою своєю частиною він базується на MVC. MVVM орієнтований на сучасні платформи розробки, такі як Windows Presentation Foundation, Silverlight від компанії Microsoft, ZK framework[6].

MVVM використовується для розділення моделі та її відображення, що необхідно для їхніх змін окремо один від одного.

MVVM зручно використовувати замість класичного MVC і йому подібних в тих випадках, коли в платформі, на якій ведеться розробка, присутнє «зв'язування даних».

У шаблонах проектування MVC/MVP зміни в інтерфейсі не впливають безпосередньо на Mодель , а попередньо йдуть через Контролер або Presenter . У таких технологіях як WPF і Silverlight є концепція «зв'язування даних», що дозволяє пов'язувати дані з візуальними елементами в обидві сторони. Отже , при використанні цього прийому застосування моделі MVC стає вкрай незручним через те, що прив'язка даних до подання безпосередньо не вкладається в концепцію MVC/MVP [7].

Шаблон MVVM ділиться на три частини:

1. Модель, так само, як у класичному MVC, являє собою фундаментальні дані, необхідні для роботи програми .
2. Представлення - це графічний інтерфейс, тобто вікно, кнопки і т.п. Відображення є підписником на подію зміни значень властивостей або команд, що надаються Моделлю подання. У разі, якщо в Моделі відображення змінилась якась властивість, то вона сповіщає всіх підписників про це, і Відображення, в свою чергу, запрошує оновлене значення властивості з Моделі представлення. У випадку, якщо користувач впливає на якийсь елемент інтерфейсу, Відображення викликає відповідну команду, надану Моделлю подання.
3. Модель відображення (англ. View Model ) є , з одного боку , абстракцією Відображення, а з іншого, надає обгортку даних з Моделі, які підлягають скріпленню. Тобто, вона містить Модель, яка перетворена до Відображення, а також містить у собі команди, якими може користуватися Відображення, щоб впливати на Модель .

2.4. Model–view–adapter.

Model–view–adapter (MVA) або посередник-контролер MVC є архітектурним шаблоном в багаторівневій архітектурі, який використовується в розробці програмного забезпечення. У складних комп'ютерних системах, які представляють великі обсяги даних для користувачів, розробники часто хочуть, щоб модель та інтерфейс користувача вид відносились , так що зміни в інтерфейсі не впливали на обробку даних і щоб дані могли бути реорганізовані без зміни інтерфейсу користувача. MVA і традиційний MVC і намагалися вирішити цю проблему , але двома різними стилями рішення. Традиційний MVC влаштовує модель, відображення і контролерв трикутнику , з моделі , уявлення і контролера , як вершин , так що деякі інформаційні потоки між моделлю і відображенням знаходяться за межами прямого контролю контролера. Model–view–adapter вирішує цю проблему трішки інакше, влаштовуючи модель, адаптер або посередник-контролер , а також відображення лінійно без яких-небудь з'єднань безпосередньо між моделлю і представленням.

Відображення повністю відокремлене від моделі. Це дозволяє відображенню і моделі взаємодіяти тільки через контролер-посередник або адаптер. За допомогою цього устрою, тільки адаптер або посередник-контролер має знання про модель і вигляд, тому що це є обов'язком виключно адаптера або посередника-контролера. Модель і відображення зберігаються навмисно не знаючи один одного. У традиційній MVC , модель і відображення знали про існування один одного[8].

Це рішення дозволяє широкому спектру різних відображень отримати доступ до тієї ж моделі або через точно такий же адаптер, або через той же клас адаптерів. Наприклад, одна базова модель даних для зберігання і схеми і технології може бути доступна через широкий спектр дещо різних відображень -наприклад, Qt GUI, Microsoft MFC GUI, GTK + GUI, Microsoft. Адаптер або клас адаптерів тримає модель абсолютно не звертаючи, що вона підтримує велику кількість інтерфейсів користувача і, можливо, навіть підтримуючи це розмаїття одночасно.

Для виконання даної курсової роботи буде використовуватися Model–view–adapter, так як за мету поставлено створити програму з якнайбільш самостійними модулями.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Прецеденти.

Після запуску програми неавторизований користувач повинен авторизуватися заповнивши форми «Логін», «Пароль» і натиснувши кнопку «Ввійти»(Рис. 2.1).



Рис. 2.1 Діаграма прецедентів операції авторизації.

Авторизований користувач (далі адміністратор) може переглядати товари і каталоги в головному вікні програми обираючи потрібний каталог в таблиці (Рис. 2.2).



Рис. 2.2 Діаграма прецедентів операції перегляду товарів і каталогів.

Адміністратор може перейменувати каталог, зайшовши в потрібний і натиснувши кнопку «Перейменувати каталог» в головному вікні програми, ввівши нову назву і натиснувши кнопку «Перейменувати» в діалоговому вікні.(Рис. 2.3)

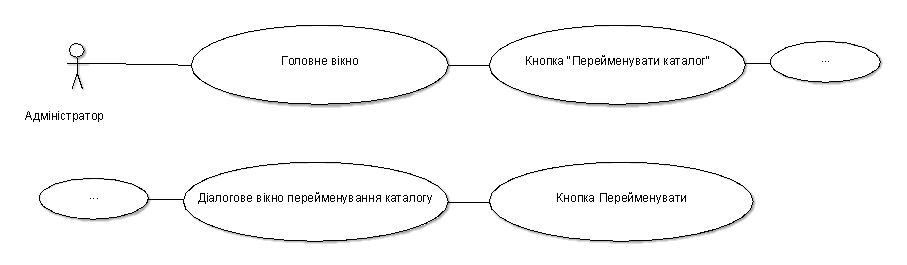


Рис. 2.3 Діаграми прецедентів операцій управління каталогами.

Адміністратор може видаляти потрібні каталоги зайшовши в потрібний каталог і натиснувши кнопку «Видалити каталог». Після натискання кнопки каталог разом з усіма товарами, які належать йому, буде видалений. Операція можлива за умови відсутності у каталозі призначеному на видалення інших каталогів (Рис. 2.4).

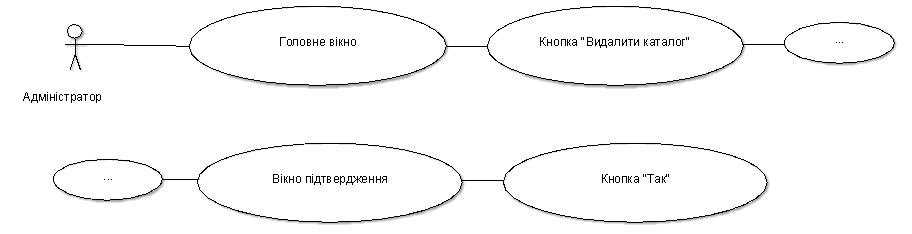


Рис. 2.4 Діаграми прецедентів операції видалення каталогу.

Адміністратор може створювати нові каталоги зайшовши в потрібний каталог і натиснувши кнопку «Створити каталог». Після цього адміністратор повинен ввести наву каталогу і натиснути кнопку «Створити» (Рис. 2.5).

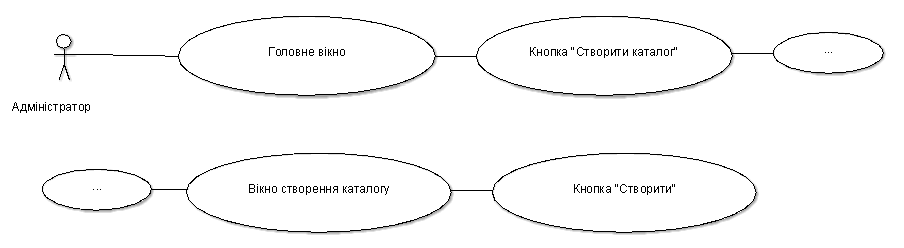


Рис. 2.5 Діаграма прецедентів для операції створення каталогу.

Адміністратор може детально переглядати товари. Для цього він повинен обрати потрібний товар і натиснути кнопку «Відкрити товар».

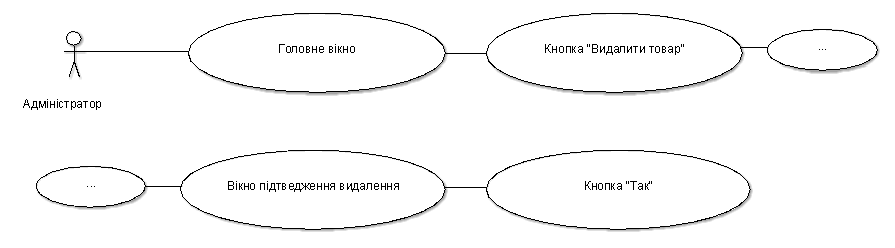


Рис. 2.6 Діаграма прецедентів для операції детального перегляду товару.

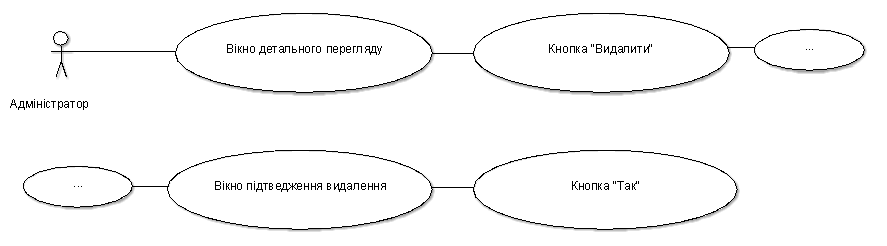
Адміністратор може видаляти конкретний товар двома способами:

1) В головному вікні програми обравши потрібний товар і натиснувши кнопку «Видалити товар». Після натиску кнопки, потрібно підтвердити свою дію у вікні підтвердження видалення (Рис. 2.7 а).

2) В вікні детального перегляду товару натиснувши кнопку «Видалити» і підтвердивши свою дію у вікні підтвердження видалення (Рис. 2.7 б).



а) Перший спосіб.



б) Другис спосіб.

Рис. 2.7 Варіанти видалення товару.

В обох випадках після підтвердження видалення товар видаляється з бази даних разом з всіма записами, що стосуються даного товару.

Адміністратор може редагувати товари. Для цього йому потрібно у вікні детального перегляду товару натиснути кнопку «Редагувати», ввести всі необхідні корективи у діалоговому вікні редагування товару і натиснути в цьому ж вікні кнопку «Редагувати» (Рис. 2.8).

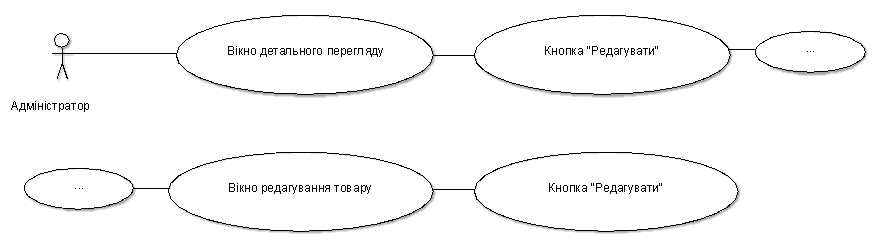


Рис. 2.8 Діаграма прецедентів для операції редагування товару.

Після натискання кнопки «Редагувати» в вікні редагування товару, інформація про товар в базі даних про даний товар буде оновлена.

Адміністратор може створювати товари. Для цього йому необхідно зайти в потрібний каталог і натиснути в головному вікні програми кнопку «Створити товар». Після натискання кнопки з’явиться діалогове вікно створення товару. Після завершення введення даних, адміністратор повинен натиснути кнопку «Створити» (Рис. 2.9).

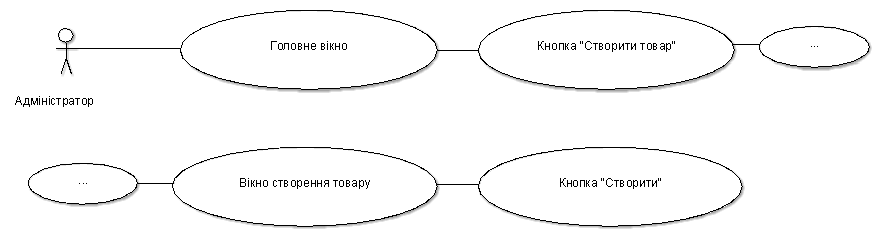


Рис. 2.9 Діаграма прецедентів для операції створення товару.

Після натиску на кнопку, в базу даних буде додано товар з заданими характеристиками, а головне вікно програми буде оновлене.

Адміністратор може переглядати списки всих користувачів системи. Для цього йому необхідно в головному вікні програми натиснути кнопку «Користувачі»(Рис. 2.10).

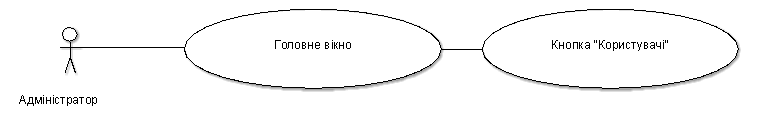


Рис. 2.10 Діаграма прецедентів для операції перегляду списку користувачів.

Після натиску на кнопку в головному вікні програми буде відображена таблиця з короткою інформацією про користувачів.

Адміністратор може переглядати детальну інформацію про користувачів. Для цього йому нуобхідно натиснут в головному вікні програми кнопку «Користувачі», обрати потрібного користувача і натиснути кнопку «Детальна інформація» (Рис. 2.11).

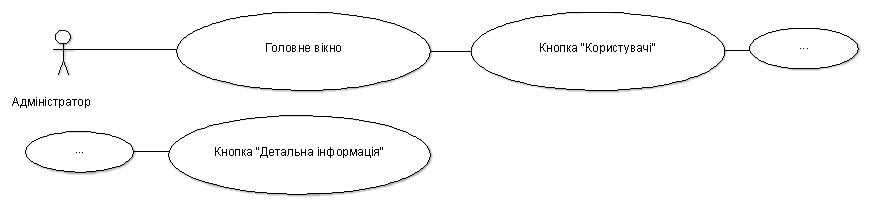


Рис. 2.11 Діаграма прецедентів для операції перегляду детальної інформації про користувача.

Адміністратор може заблокувати за необхідноті користувача-споживача. Для цього необхідно в головному вікні програми натиснути кнопку «Користувачі», обрати в таблиці потрібного користувача, натиснути кнопку «Детальна інформація» і натиснути у вікні перегляду детальної інформації кнопку «Заблокувати» (Рис. 2.12).

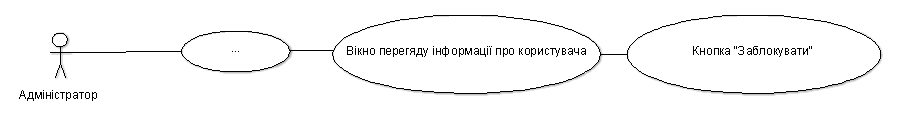


Рис. 2.12 Діаграма прецедентів для операції блокування користувача.

Адміністратор може розблокувати за необхідноті користувача-споживача. Для цього необхідно в головному вікні програми натиснути кнопку «Користувачі», обрати в таблиці потрібного користувача, натиснути кнопку «Детальна інформація» і натиснути у вікні перегляду детальної інформації кнопку «Розблокувати» (Рис. 2.13).

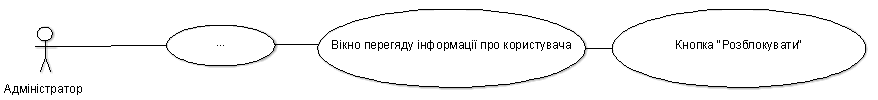


Рис. 2.13 Діаграма прецедентів для операції розблокування користувача.

Адміністратор може змінювати візуальне оформлення програми (стиль). Для цього йому потрібно зайти в меню «Налаштування» і в підменю «Стиль» обрати потрібний йому стиль (Рис. 2.14).

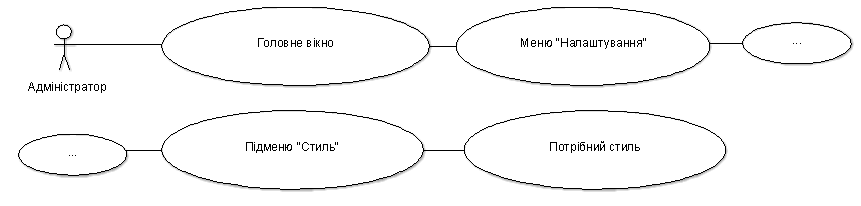


Рис. 2.14 Діагарма прецедентів для операції зміни стилю.

Після вибору необхідного пункту, головне вікно програми динамічно змінить стиль на обраний.

Адміністратор може змінити мову інтерфейсу. Для цього йому необхідно зайти в меню «Налаштування» і обрати в підменю «Мова» потрібну мову (Рис.2.15).

Після обрання потрібного пункту, програма динамічно змінить мову інтерфейсу на обрану.

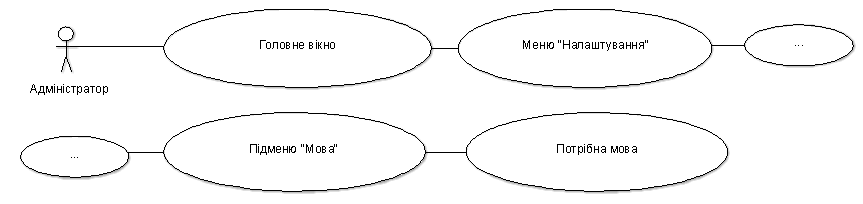


Рис. 2.15 Діаграма прецедентів для операції зміни мови.

Адміністратор може переглянути інформацію про програму. Для цього йому необхідно в меню «Довідка» обрати пункт «Про програму» (Рис. 2.16).

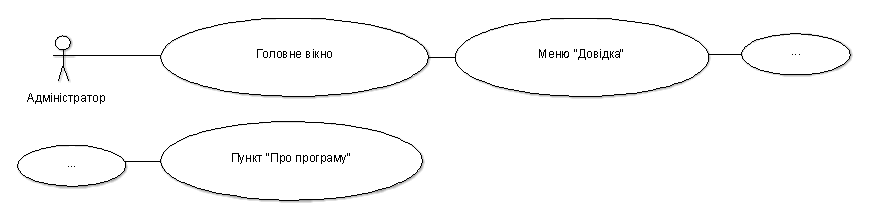


Рис. 2.16 Діаграма прецедентів для операції перегляду інформації про програму.

Після вибору даного пункту, інформація про програму з’явиться в окремому вікні.

Адміністратор може отримати допомогу щодо усунення проблем в роботі програми. Для цього він повинен зайти в меню «Довідка» і обрати пункт «Допомога» (Рис. 2.17).

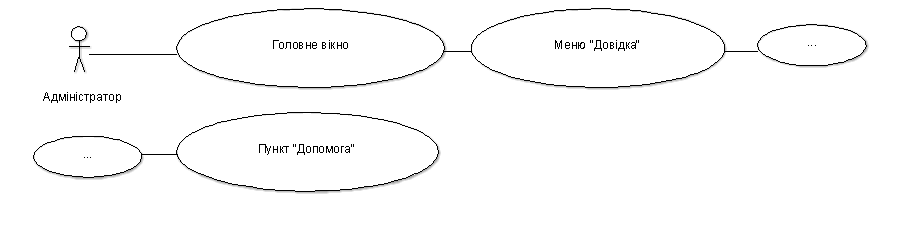


Рис. 2.17 Діаграма прецедентів для операції отримання допомоги.

Початковий опис потоку подій роботи з програмою.

1) Прецедент розпочинається коли користувач запускає програму.

2) Після авторизації користувач може виконати одну з наступних дій:

а) Переглянути каталоги та їх вміст.

b) Перейменувати каталог.

с) Створити каталог.

d) Видалити каталог.

e) Переглянути детальну інформацію про товар.

f) Створити новий товар.

g) Видалити товар.

h) Редагувати існуючий товар.

i) Переглянути список користувачів.

j) Переглянути детальну інформацію про користувача.

k) Заблокувати користувача.

l) Розблокувати користувача.

m) Змінити тему програми.

n) Змінити мову програми.

o) Переглянути інформацію про програму.

p) Переглянути допомогу по програмі.

3) Прецедент закінчується коли користувач закінчує роботу з програмою.

Розробка орієнтирів.

1) Прецедент розпочинається коли користувач запускає програму.

2) Після авторизації користувач може виконати одну з наступних дій:

а) Переглянути каталоги та їх вміст..[Адміністратор може переглянути

каталоги і товари]

b) Перейменувати каталог. [Адміністратор може змінити назву каталогу]

с) Створити каталог. [Адміністратор може створити новий каталог,

вказавши назву]

d) Видалити каталог. [Адміністратор може видалити каталог]

e) Переглянути детальну інформацію про товар. [Адміністратор може

переглядати детальну інформацію про конкретний товар]

f) Створити новий товар. [Адміністратор може додати новий товар вказавши

назву і інші параметри]

g) Видалити товар. [Адміністратор може видалити товар, обравши

конкретний товар]

h) Редагувати існуючий товар. [Адміністратор може редагувати інформацію

вже існуючого товару]

i) Переглянути список користувачів. [Адміністратор може переглядати

список всих користувачів системи]

j) Переглянути детальну інформацію про користувача. [Адміністратор може

переглядати детальну інформацію про користувача]

k) Заблокувати користувача. [Адміністратор може заблокувати за

необхідності користувача]

l) Розблокувати користувача. [Адміністратор може розблокувати за

необхідності користувача]

m) Змінити тему програми. [Адміністратор може змінювати візуальне

оформлення програми]

n) Змінити мову програми. [Адміністратор може змінити мову програмного

інтерфейсу]

o) Переглянути інформацію про програму. [Адміністратор може

переглянути інформацію про програму]

p) Переглянути допомогу по програмі. [Адміністратор може переглянути

допомогу по програмі]

3) Прецедент закінчується коли користувач закінчує роботу з програмою.

Розробка атрибутів.

1) Прецедент розпочинається коли користувач запускає програму.

2) Після авторизації користувач може виконати одну з наступних дій:

а) Переглянути каталоги та їх вміст.{Головне вікно буде оновлювати вміст

таблиці}

b) Перейменувати каталог. {Відобрразиться діалогове вікно редагування

каталогу}

с) Створити каталог. {Відобразиться діалогове вікно створення каталогу}

d) Видалити каталог. {Програма запросить підтвердження}

e) Переглянути детальну інформацію про товар. {Відкриється нове вікно з

інформацією про товар}

f) Створити новий товар. {Відкриється діалогове вікно створення товару}

g) Видалити товар. {З'євиться форма підтвердження}

h) Редагувати існуючий товар. {З'явиться вікно редагування товару}

i) Переглянути список користувачів. {В головному вікні відобразиться

таблиця користувачів}

j) Переглянути детальну інформацію про користувача. {З'явиться нове вікно

з детальною інформацією про користувача}

k) Заблокувати користувача. {Стан користувача зміниться на

«Заблокований»}

l) Розблокувати користувача. {Стан користувача зміниться на

«Розблокований»}

m) Змінити тему програми. {Зміниться віуальне оформлення програми}

n) Змінити мову програми. {Зміниться мова інтерфейсу програми}

o) Переглянути інформацію про програму. {З'явиться вікно з інформацією

про програму}

p) Переглянути допомогу по програмі. {З'явиться вікно допомого по

програмі}

3) Прецедент закінчується коли користувач закінчує роботу з програмою.

Розробка інтенсивності використання.

1) Прецедент розпочинається коли користувач запускає програму.

2) Після авторизації користувач може виконати одну з наступних дій:

а) Переглянути каталоги та їх вміст. (дуже часто)

b) Перейменувати каталог. (рідко)

с) Створити каталог. (рідко)

d) Видалити каталог. (рідко)

e) Переглянути детальну інформацію про товар. (середньо)

f) Створити новий товар. (рідко)

g) Видалити товар. (рідко)

h) Редагувати існуючий товар. (часто)

i) Переглянути список користувачів. (середньо)

j) Переглянути детальну інформацію про користувача. (рідко)

k) Заблокувати користувача. (рідко)

l) Розблокувати користувача. (рідко)

m) Змінити тему програми. (рідко)

n) Змінити мову програми. (рідко)

o) Переглянути інформацію про програму. (рідко)

p) Переглянути допомогу по програмі. (рідко)

3) Прецедент закінчується коли користувач закінчує роботу з програмою.

Результуючий опис подій.

1) Прецедент розпочинається коли користувач запускає програму.

2) Після авторизації користувач може виконати одну з наступних дій:

а) Переглянути каталоги та їх вміст..[Адміністратор може переглянути

каталоги і товари] {Головне вікно буде оновлювати вміст таблиці} (дуже

часто)

b) Перейменувати каталог. [Адміністратор може змінити назву каталогу]

{Відобрразиться діалогове вікно редагування каталогу} (рідко)

с) Створити каталог. [Адміністратор може створити новий каталог,

вказавши назву] {Відобразиться діалогове вікно створення каталогу}

(рідко)

d) Видалити каталог. [Адміністратор може видалити каталог] {Програма

запросить підтвердження} (рідко)

e) Переглянути детальну інформацію про товар. [Адміністратор може

переглядати детальну інформацію про конкретний товар] {Відкриється

нове вікно з інформацією про товар} (середньо)

f) Створити новий товар. [Адміністратор може додати новий товар вказавши

назву і інші параметри] {Відкриється діалогове вікно створення товару}

(рідко)

g) Видалити товар. [Адміністратор може видалити товар, обравши

конкретний товар] {З'євиться форма підтвердження} (рідко)

h) Редагувати існуючий товар. [Адміністратор може редагувати інформацію

вже існуючого товару] {З'явиться вікно редагування товару} (часто)

i) Переглянути список користувачів. [Адміністратор може переглядати

список всих користувачів системи] {В головному вікні відобразиться

таблиця користувачів} (середньо)

j) Переглянути детальну інформацію про користувача. [Адміністратор може

переглядати детальну інформацію про користувача] {З'явиться нове вікно

з детальною інформацією про користувача} (рідко)

k) Заблокувати користувача. [Адміністратор може заблокувати за

необхідності користувача] {Стан користувача зміниться на

«Заблокований»} (рідко)

l) Розблокувати користувача. [Адміністратор може розблокувати за

необхідності користувача] {Стан користувача зміниться на

«Розблокований»} (рідко)

m) Змінити тему програми. [Адміністратор може змінювати візуальне

оформлення програми] {Зміниться віуальне оформлення програми}

(рідко)

n) Змінити мову програми. [Адміністратор може змінити мову програмного

інтерфейсу] {Зміниться мова інтерфейсу програми} (рідко)

o) Переглянути інформацію про програму. [Адміністратор може

переглянути інформацію про програму] {З'явиться вікно з інформацією

про програму} (рідко)

p) Переглянути допомогу по програмі. [Адміністратор може переглянути

допомогу по програмі] {З'явиться вікно допомого по програмі} (рідко)

3) Прецедент закінчується коли користувач закінчує роботу з програмою.

2.2 Діаграма граничних класів.

Графічний інтерфейс програми є багатовіконним. Тому для кращого розуміння буде складено до кожного вікна програми своя діаграма класів і таблиця відповідностей. Таблиці відповідностей винесені в додаток Б.

Вікно авторизації.

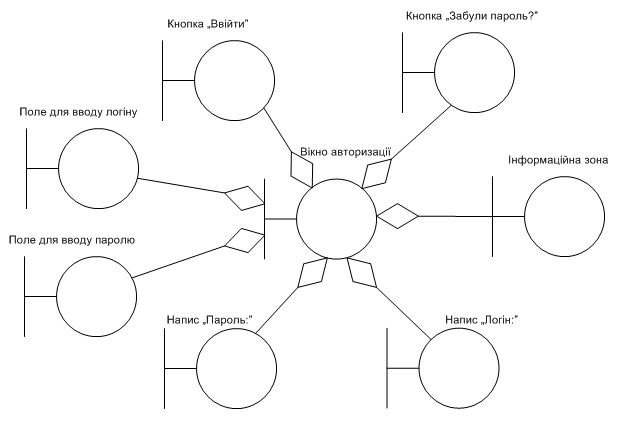


Рис. 2.18 Гранична діаграма класів для вікна авторизації.

Вікно авторизації є формою для введення логіну і паролю користувачем. Користувач в даному вікні може ввести дані для авторизації або отримати інформацію про відновлення доступу, якщо ці дані було втрачено. Гранична діаграма класів для нього зображена на Рис. 2.18.

Головне вікно програми.

Після авторизації з’явиться головне вікно програми, з яким користувач буде взаємодіяти до завершення роботи програми. Гранична діаграма класів приведена нижче (Рис. 2.19).

Структура графічного інтерфейсу головного вікна набагато складніша. Це можна пояснити тим, що головне вікно програми має велику функціональність. Саме з воно створює нові діалогові вікна для специфічних операцій. Вікно містить компоненти багатьох типів. Серед них найпоширеніші JButton, JMenu і JMenuItem. Також наявні такі елементи як JSeparator, JScrollPane, JTable, JMenuBar, JLabel. Саме завдяки складній комбінації цих всих елементів буде реалізовано дуже зручний інтерфейс для користувача.

Оскільки діаграма граничних класів має значний розмір, вона буде роздрукована в форматі А3 і наведена в додатку В (Рис. В.1).

Вікно створення каталогу.

Вікно створення каталогу буде з’являтися після того, як користувач натисне кнопку «Створити каталог» в головному вікні програми. Діаграма граничних класів для даного вікна приведена нижче (Рис. 2.20).

Дане вікно матиме вигляд форми з одним полем – назвою каталогу. Окрім поля в формі будуть наявні кнопки підтвердження створення і сказування операції.

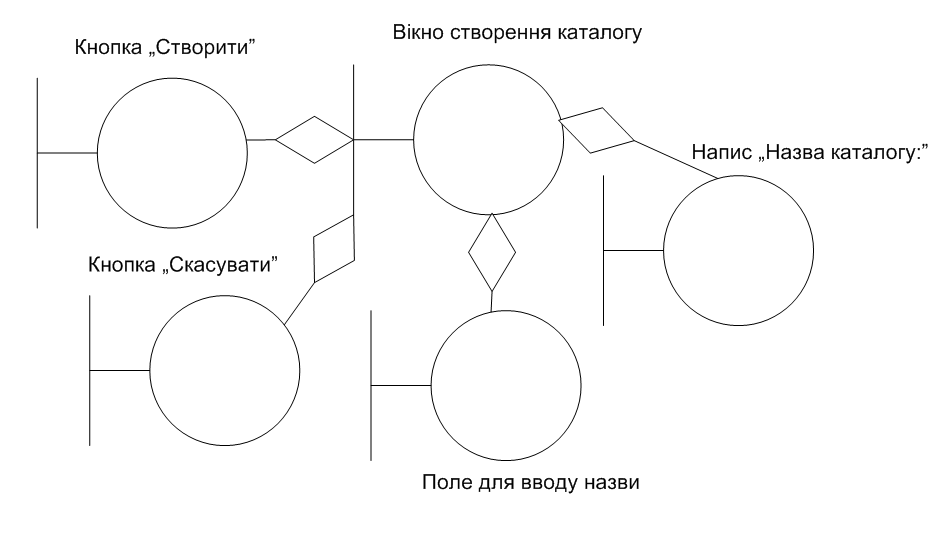


Рис. 2.20 Діаграма граничних класів для вікна створення каталогу.

Вікно створення товару.

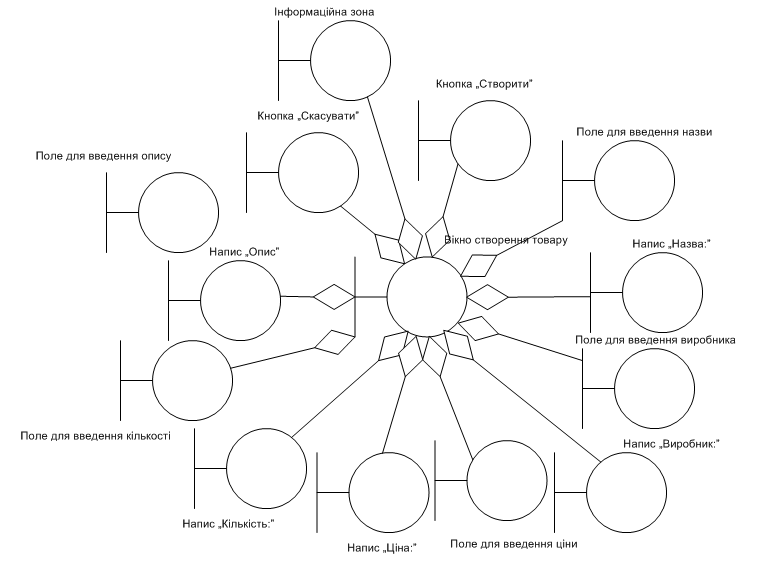


Рис. 2.21 Діаграма граничних класів для вікна створення товару.

Вікно створення товару матиме вигляд форми з полями для заповнення і кнопками підтвердження створення і скасування. Для створення вікна будуть масово використані компоненти лише трьох типів: JButton, JLabel, JTextField. Гранична діаграма класів наведена вище(Рис. 2.22 ).

Вікно перейменовування каталогу.

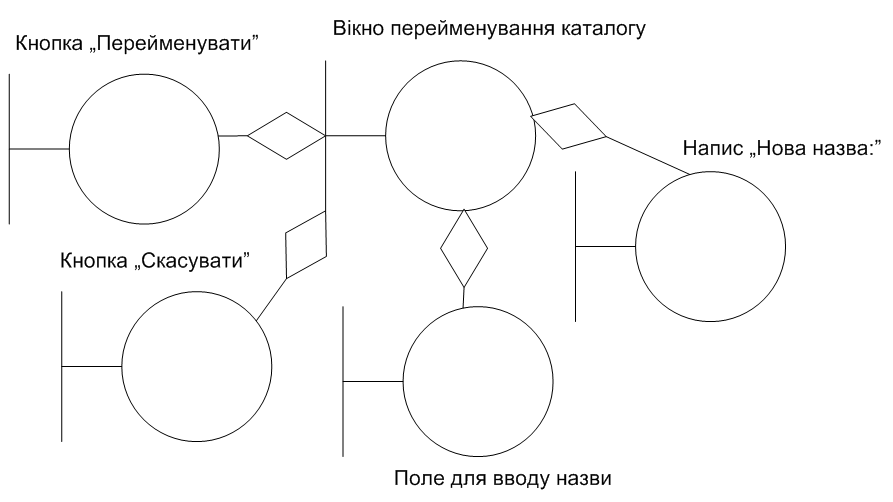


Рис. 2.22 Діаграма граничних класів для вікна перейменування каталогу.

Вікно представляє собою форму для введення нової назви каталогу. В ньому наявне поле для введення назви і кнопки підтвердження і скасування операції. Діаграма граничних класів для даного вікна наведена вище (Рис. 2.22).

Вікно редагування товару.

В даному вікні використовуватимуться лише мітки і поля для введення тексту. Окрім цих елементів у вікні також присутні кнопки для підтвердження і скасування операції редагування. Діаграма граничних класів для даного вікна приведена на Рис. 2.23.

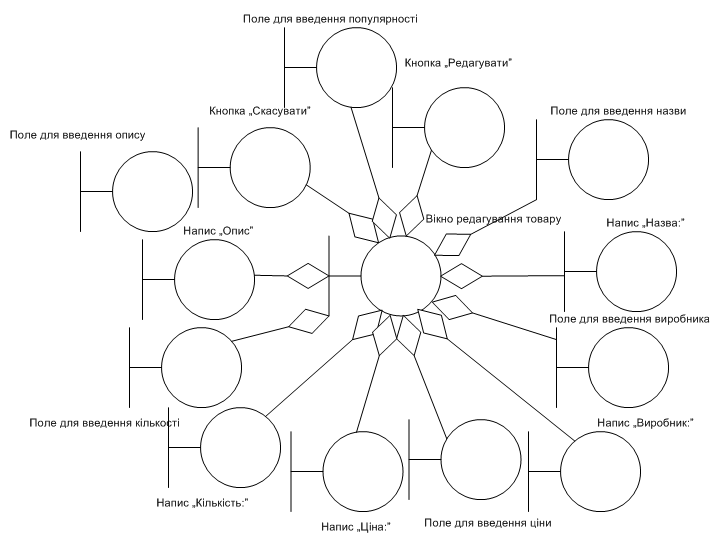


Рис. 2.23 Діаграма граничних класів для вікна редагування товару.

Вікно детального перегляду товару.

В даному вікні використовуватимуться лише мітки для відображення тексту. Окрім цих елементів у вікні також присутні кнопки для операцій редагування і видалення товару. Діаграма граничних класів для даного вікна приведена на Рис. 2.24.

Таблиця відповідностей елементів графічного інтерфейсу вікна детального перегляду інформації про товар класам бібліотеки javax.siwng i java.awt наведена в додатку Б.

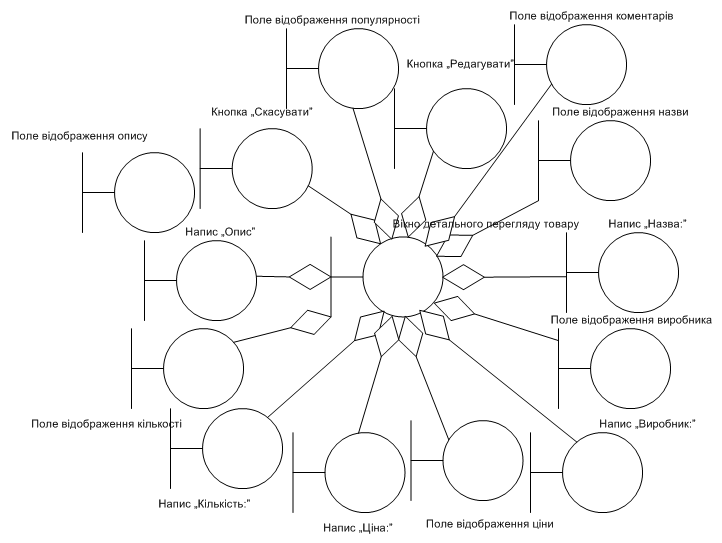


Рис. 2.24 Діаграма граничних класів для вікна перегляду товару.

Вікно перегляду користувача.

Дане вікно дозволятиме користувачу переглядати детальну інформацію про інших користувачів системи, а також блокувати і розблоковувати користувачів, якщо це дозволено.

Для реалізації даного вікна використовувалися компоненти двох типів не рахуючи самого вікна: JButton, i JLabel. Гранична діаграма класів для даного вікна наведена нижче (Рис. 2.25).

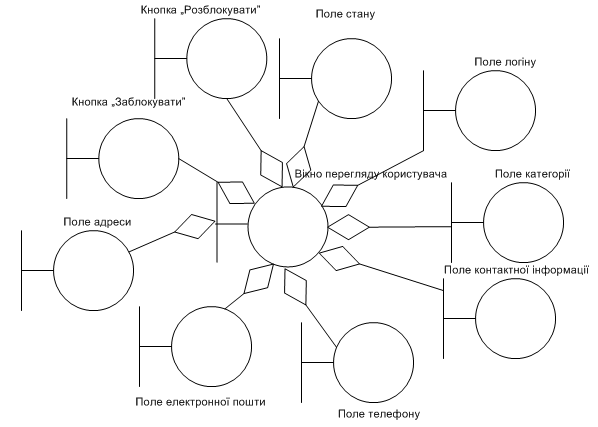


Рис. 2.25 Гранична діаграма класів для вікна перегляду користувача.

2.3 Проектування графічного інтерфейсу.

Для реалізації графічного інтерфейсу користувача був підібраний багато віконний підхід через те, що він дозволяє розділити складний і різновекторний інтерфейс на безліч окремих вікон. Це дозволяє в певній мірі спростити інтерфейс. Адже тепер користувач взаємодіє з головним вікном, а допоміжні вікна з’являються лише в разі необхідності. Завдяки цьому користувачу стає простіше орієнтуватися під час роботи з програмою. На Рис. 2.26 зображено модель головного вікна програми.

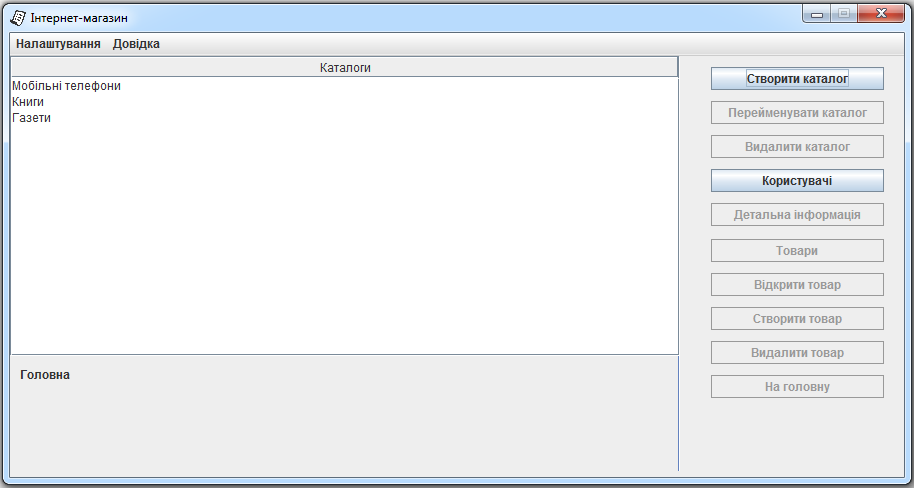


Рис. 2.26 Модель графічного інтерфейсу головного вікна програми.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

application

## Клас Autorization

Клас реалізує діалогове вікно авторизації.

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

Основна панель вікна.

#### textField

private javax.swing.JTextField textField

Поле для введення логіну.

#### passwordField

private javax.swing.JPasswordField passwordField

Поле для введення паролю.

#### link

private [Autorization](file:///D:\Education\4\SourseCode\KursWork\doc\application\Autorization.html) link

Посилання на самого себе.

#### lblEnterLoginAnd

private javax.swing.JLabel lblEnterLoginAnd

Мітка з тестом "логін"

#### btnForgotPassword

private javax.swing.JButton btnForgotPassword

Кнопка для відображення дій при втраті паролю.

#### lblPassword

private javax.swing.JLabel lblPassword

Мітка з текстом "пороль".

#### btnLogin

private javax.swing.JButton btnLogin

Кнопка для входження в систему.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця, в якій зберігаються написи компонентів

### Конструктори

#### Autorization

public Autorization()

Створення і налаштування діалогового вікна.

### Методи

#### main

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

Запуск програми

#### initComponents

private void initComponents()

Ініціалізація компонентів.

#### allocateComponents

private void allocateComponents()

Розміщення компонентів в діалоговому вікні.

#### getSolution

public void getSolution(int solution)

Метод для зворотнього зв'язку. Використовується для отримання результату виконання команди авторизування. Приймає параметром клас користувача.

****Параметри:****

solution - клас користувача.

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно

application

## Клас ContextUtils

Клас надає методи для роботи з різними мовами.

### Методи

#### loadContext

public static java.util.Hashtable<java.lang.String,java.lang.String> loadContext(java.lang.String component)

Метод завантажує назви елементів конкретного вікна.

****Параметри:****

component - назва вікна, для якого потрібно завантажити список

****Повертає:****

таблицю з назвами компонентів

application

## Клас CreateCatalog

Клас реалізовує функціонал діалогового вікна створення каталогу

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

Основна панель вікна.

#### textField

private javax.swing.JTextField textField

Поле для введення імені каталогу.

#### okButton

private javax.swing.JButton okButton

Кнопка для підтвердження.

#### link

private [MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link

Посилання на головне вікно.

#### cancelButton

private javax.swing.JButton cancelButton

Кнопка для скасування.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця з написами на компонентах.

### Конструктори

#### CreateCatalog

public CreateCatalog([MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link)

Створення і налаштування діалогового вікна.

****Параметри:****

link - посилання на головне вікно.

### Методи

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно

application

## Клас CreateGood

Клас реалізовує діалогове вікно для створення товару.

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

Основна панель вікна.

#### nameField

private javax.swing.JTextField nameField

Поле для введення назви товару.

#### manufacturedField

private javax.swing.JTextField manufacturedField

Поле для введення виробника.

#### priceField

private javax.swing.JTextField priceField

Поле для введення ціни.

#### countField

private javax.swing.JTextField countField

Поле для введення кількості товару.

#### descriptionArea

private javax.swing.JTextArea descriptionArea

Поле для введення опису товару.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця з написами на компонентах.

#### lblError

private javax.swing.JLabel lblError

Мітка для відображення попереджень.

#### link

private [MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link

Посилання на власника вікна.

#### cancelButton

private javax.swing.JButton cancelButton

Кнопка для скасування створення.

### Конструктори

#### CreateGood

public CreateGood([MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link)

Створення і налаштування діалогового вікна.

****Параметри:****

link - посилання на головне вікно.

### Методи

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно

application

## Клас EditCatalog

Клас реалізовує діалогове вікно для редагування товару.

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

Панель, на якій розташовуються всі компоненти вікна.

#### textField

private javax.swing.JTextField textField

Поле для введення нової назви.

#### link

private [MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link

Посилання на головне вікно.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця з написами на компонентах.

### Конструктори

#### EditCatalog

public EditCatalog([MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link)

Створення і налаштування діалогового вікна

****Параметри:****

link - посилання на головне вікно.

### Методи

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно

application

## Клас MainFrame

Клас реалізує функціонал головного вікна програми.

### Поля

#### contentPane

private javax.swing.JPanel contentPane

Основна панель, в якій будуть міститися всі інші об'єкти.

#### goodsTable

private javax.swing.JTable goodsTable

Таблиця, в якій буде відображатися інформація директорій з товарами.

#### clientsTable

private javax.swing.JTable clientsTable

Таблиця, в якій буде відображатися інформація про клієнтів.

#### btnGoods

private javax.swing.JButton btnGoods

Кнопка, яка відображає таблицю товарів.

#### btnEditCatalog

private javax.swing.JButton btnEditCatalog

Кнопка, яка викликає вікно для редагування коталогу.

#### lblNewLabel

private javax.swing.JLabel lblNewLabel

Мітка, якій свідображається підказки.

#### btnDeleteCatalog

private javax.swing.JButton btnDeleteCatalog

Кнопка, яка видаляє каталог.

#### btnClients

private javax.swing.JButton btnClients

Кнопка, яка відображає таблицю клієнтів.

#### btnOpenGoods

private javax.swing.JButton btnOpenGoods

Кнопка, яка відкриває детальний перегляд товару.

#### btnCreateCatalog

private javax.swing.JButton btnCreateCatalog

Кнопка, яка відкриває вікно для створення каталогу.

#### btnCreateGood

private javax.swing.JButton btnCreateGood

Кнопка, яка відкриває вікно для створення товару.

#### separator

private javax.swing.JSeparator separator

Розділяє таблицю і кнопки лінією для кращого візуального ефекту.

#### btnNewButton

private javax.swing.JButton btnNewButton

Повертає таблицю в корінь каталогу.

#### scrollPane

private javax.swing.JScrollPane scrollPane

Панель в яку поміщають таблицю для кращоговізуального ефекту.

#### link

private [MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link

Посилання на самого себе. Використовується при створенні команд

#### lblTitle

private javax.swing.JLabel lblTitle

Мітка, яка відображає назву вузла, вміст якого відображається

#### btnDeleteGoods

private javax.swing.JButton btnDeleteGoods

Видаляє товар

#### btnOpenClient

private javax.swing.JButton btnOpenClient

Відкриває список клієнтів.

#### rdbtnmntmEnglish

private javax.swing.JRadioButtonMenuItem rdbtnmntmEnglish

Пункт меню, який відповідає англійській мові

#### rdbtnmntmUkrainian

private javax.swing.JRadioButtonMenuItem rdbtnmntmUkrainian

Пункт меню, який відповідає українській мові

#### rdbtnmntmSystem

private javax.swing.JRadioButtonMenuItem rdbtnmntmSystem

Пункт меню, який відповідає системному стилю

#### rdbtnmntmStandart

private javax.swing.JRadioButtonMenuItem rdbtnmntmStandart

Пункт меню, який відповідає стандартному стилю

#### rdbtnmntmNimbus

private javax.swing.JRadioButtonMenuItem rdbtnmntmNimbus

Пункт меню, який відповідає за стиль німб

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця, в якій зберігаються написи компонентів

#### mnHelp

private javax.swing.JMenu mnHelp

Меню допомоги

#### menuBar

private javax.swing.JMenuBar menuBar

Компонент, на якому розміщюються всі пункти меню

#### mnSettings

private javax.swing.JMenu mnSettings

Меню налаштувань

#### mnLanguage

private javax.swing.JMenu mnLanguage

Підменю мов

#### mnStyle

private javax.swing.JMenu mnStyle

Підменю стилів

#### mntmAbout

private javax.swing.JMenuItem mntmAbout

Пункт меню, який відповідає за інформацію про програму

#### mntmHelp

private javax.swing.JMenuItem mntmHelp

Пункт меню, який відповідає за інформацію допомоги

### Конструктори

#### MainFrame

public MainFrame()

Конструктор, в якому створюється і налаштовується головне вікно прогами.

### Методи

#### initComponents

private void initComponents()

Відбувається ініціалізація компонентів.

#### allocateComponents

private void allocateComponents()

Відбувається додавання до основного вікна.

#### runGetTable

public void runGetTable(java.lang.String node)

Метод генерує оновлення таблиці для конкретного коріня.

****Параметри:****

pane - панель, в якій знаходиться таблиця.

node - корінь, відносно якого відбувається оновлення.

#### getTitleLabel

public javax.swing.JLabel getTitleLabel()

****Повертає:****

мітку, яка відображає назву вузла, зміст якого відображається

#### getScrolPane

public javax.swing.JScrollPane getScrolPane()

****Повертає:****

панель, в середині якої знаходиться таблиця

#### setButtonsEnabled

* + - public void setButtonsEnabled(boolean createCatalog,
    - boolean editCatalog,
    - boolean deleteCatalog,
    - boolean createGoods,
    - boolean openGoods,
    - boolean goods,
    - boolean clients,
    - boolean getRoot,
    - boolean deleteGoods,

boolean openClient)

Встановлює стан кнопок головного вікна

****Параметри:****

createCatalog - стан кнопки створення каталогу

editCatalog - стан кнопки редагування каталогу

deleteCatalog - стан кнопки видалення каталогу

createGoods - стан кнопки створення товару

openGoods - стан кнопки відкриття товару

goods - стан кнопки, яка відкриває таблицю товарів

clients - стан кнопки, яка відкриває таблицю клієнтів

getRoot - стан кнопки повернення на головну

deleteGoods - стан кнопки видалення товару

openClient - стан кнопки детального перегляду клієнта

#### refresh

public void refresh()

Метод оновлює таблицю товарів відносно головного вузла.

#### refresh

public void refresh(java.lang.String point)

Метод оновлює таблицю товарів відносно конкретного вузла

****Параметри:****

point - назва вузла

#### setTipForComponent

public void setTipForComponent(java.lang.String title)

Метод встановлює повідомлення в головному вікні

****Параметри:****

title - повідомлення, яке потрібно відобразити

#### setSettings

private void setSettings()

Метод налаштовує пункти меню згідно завантажених налаштувань

#### changeLanguage

private void changeLanguage()

Метод змінює написи на компонентах головного вікна

#### setIcon

private void setIcon()

Встановлює піктограму на дане вікно

application

## Клас OpenGoods

Клас реалізовує функціонал для відображення детальної інформації про товар.

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

Панель, яка містить всі компоненти вікна.

#### lblNamefield

private javax.swing.JLabel lblNamefield

Мітка, для відображення назви.

#### lblManufacturedfield

private javax.swing.JLabel lblManufacturedfield

Мітка для відображення виробника.

#### lblPrisefield

private javax.swing.JLabel lblPrisefield

Мітка для відображення ціни.

#### lblCountfield

private javax.swing.JLabel lblCountfield

Мітка для відображення кількості.

#### txtrDescriptionArea

private javax.swing.JTextPane txtrDescriptionArea

Поле для відображення опису.

#### txtrComments

private javax.swing.JTextPane txtrComments

Поле для відображення коментарів.

#### lblPopularityfield

private javax.swing.JLabel lblPopularityfield

Поле для відображення популярності.

#### link

protected [MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link

Посилання на головне вікно.

#### goods

private [Goods](file:///D:\Education\4\SourseCode\KursWork\doc\application\Model\Goods.html) goods

Товар для відображення.

#### descr

private [Description](file:///D:\Education\4\SourseCode\KursWork\doc\application\Model\Description.html) descr

Опис товару для відображення.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця з написами компонентів.

### Конструктори

#### OpenGoods

public OpenGoods([MainFrame](file:///D:\Education\4\SourseCode\KursWork\doc\application\MainFrame.html) link)

Конструктор налаштовує діалогове вікно.

****Параметри:****

link - посилання наголовне вікно.

### Методи

#### setInformation

* + - public void setInformation([Goods](file:///D:\Education\4\SourseCode\KursWork\doc\application\Model\Goods.html) goods,
    - [Description](file:///D:\Education\4\SourseCode\KursWork\doc\application\Model\Description.html) descr,

java.lang.String comments)

Метод встановлює інформацію для відображення.

****Параметри:****

goods - товар для відображення.

descr - опис для відображення.

comments - коментарі.

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно

application

## Клас User

Клас реалізовує діалогове вікно, в якому буде відображатися інформація про користувача.

### Поля

#### contentPanel

private final javax.swing.JPanel contentPanel

основна панель діалогового вікна, в якій будуть розміщуватися всі інші елементи.

#### lblLoginlabel

private javax.swing.JLabel lblLoginlabel

Поле для відображення логіну.

#### lblCategorylabel

private javax.swing.JLabel lblCategorylabel

Поле для відображення категорії.

#### lblTelephonelabel

private javax.swing.JLabel lblTelephonelabel

Поле для відображення телефону.

#### lblEmaillable

private javax.swing.JLabel lblEmaillable

Поле для відображення електронної пошти.

#### lblAddresslable

private javax.swing.JLabel lblAddresslable

Поле для відображення адреси.

#### lblStatelabel

private javax.swing.JLabel lblStatelabel

Поле для відображення стану.

#### btnUnblockUser

private javax.swing.JButton btnUnblockUser

Кнопка для розблокування користувача.

#### btnBlockUser

private javax.swing.JButton btnBlockUser

Кнопка для блокування користувача.

#### context

private java.util.Hashtable<java.lang.String,java.lang.String> context

Таблиця з написани на компонентах.

### Методи

#### setInformation

* + - public void setInformation(java.lang.String login,
    - java.lang.String category,
    - int state,

[ContactInformation](file:///D:\Education\4\SourseCode\KursWork\doc\application\Model\ContactInformation.html) contInfo)

Метод для встановлення інформації для відображення.

****Параметри:****

login - логін

category - категорія

state - стан

contInfo - контактна інформація

#### getState

private java.lang.String getState(int id)

Метод для отримання назви стану за переданою цифрою.

****Параметри:****

id - числовий індентифікатор стану

****Повертає:****

назву стану

#### setIcon

private void setIcon()

Метод встановлює піктограму на вікно.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Авторизація (Прецедент Рис. 2.1).

Користувач після запуску програми повинен авторизуватися, ввівши свій логін і пароль у вікно авторизації програми (Рис. 4.1) і натиснувши кнопку «Ввійти».

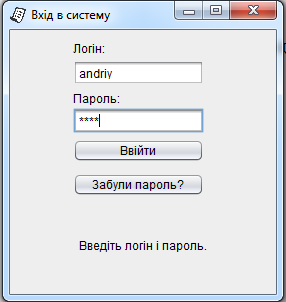
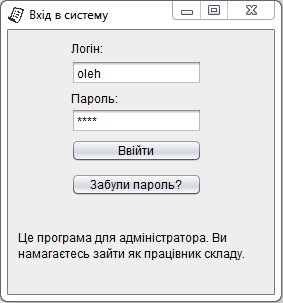
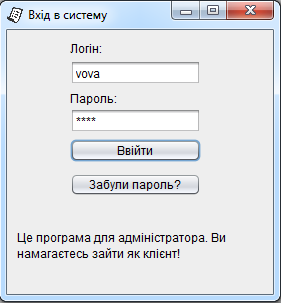


Рис. 4.1 Вікно авторизації.

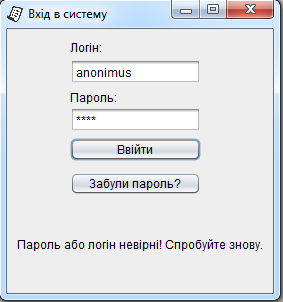
Якщо введені логін і пароль не належать адміністратору, то можливі три варіанти реакції програми(Рис. 4.2 а,б,в).



а)Були введені логін і пароль працівника складу.



б) Були введені логін і пароль клієнта.



в) Були введені некоректні логін і пароль.

Рис. 4.2 Варіанти реакції програми на некоректні авторизаційні дані.

У всих трьох випадках (Рис. 4.2 а,б,в) програма видає попереджувальні повідомлення різного змісту і надає можливість користувачу можливість повторно ввести авторизаційні дані.

В разі успішної авторизації користувач потрапить на головне вікно програми (Рис.4.3).

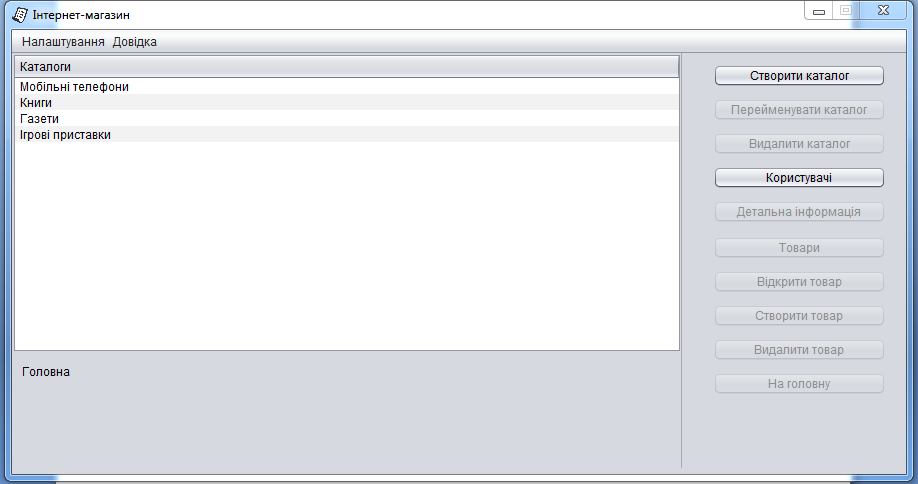


Рис. 4.3 Головне вікно програми.

В головному вікні розташований таблиця каталогів і товарів, з яким користувач буде взаємодіяти, а також набір кнопок для редагування вмісту таблиці.

Всі наступні сценарії доступні користувачу лише після авторизації.

Перегляд товарів і каталогів(Прецедент Рис. 2.2).

Користувач може переглядати список товарів і каталогів натискаючи на потрібний рядок в таблиці (Рис. 4.4).

Після обрання потрібного елементу, буде відкрито його вміст, якщо це каталог, або будуть розблоковані кнопки взаємодії з товаром, ящо обраний компонент був товаром.

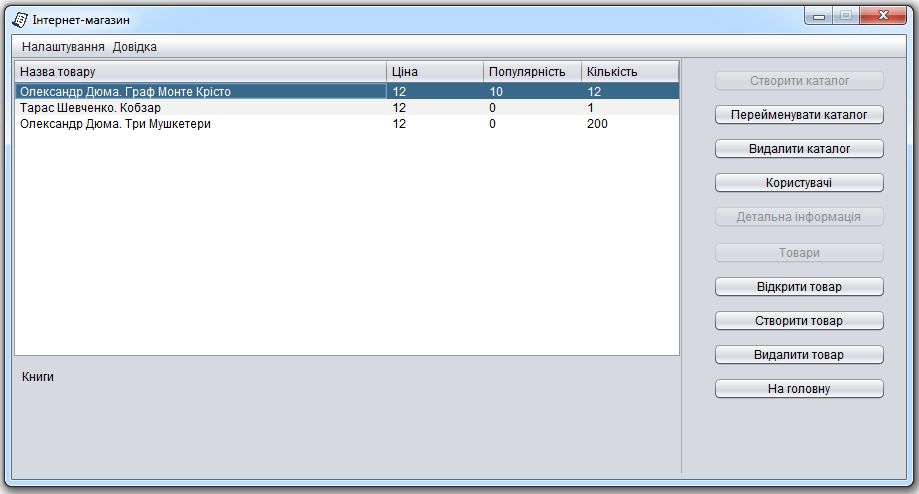


Рис. 4.4 Вміст підкаталогу «Книги».

Перейменування товару(Прецедент Рис. 2.3).

Користувач може перейменовувати каталоги, зайшовши у потрібний каталог і натиснувши кнопку «Перейменувати каталог». Після натиску кнопки, програма видасть користувачу діалогове вікно (Рис. 4.5).

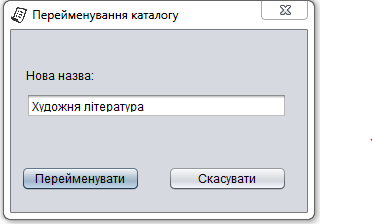
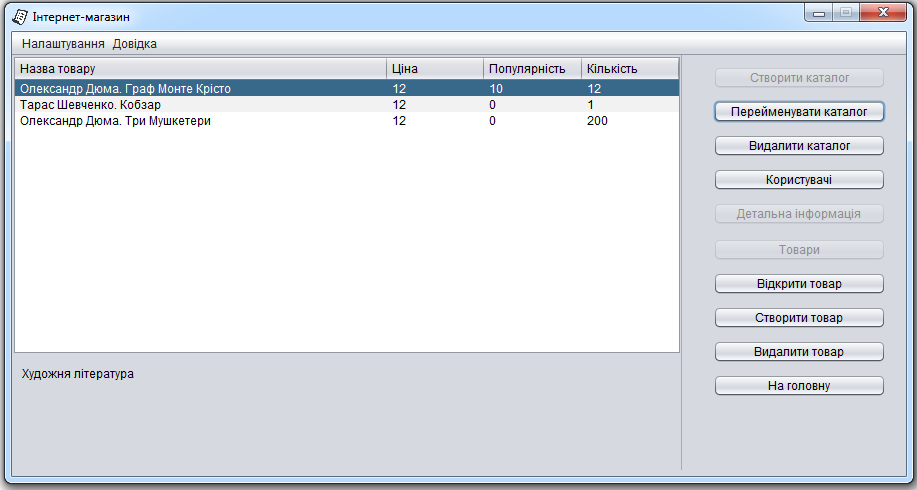


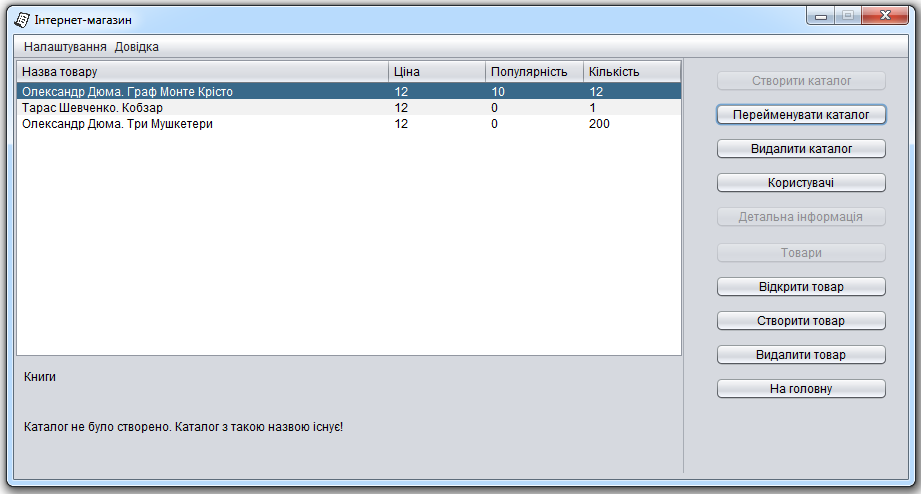
Рис. 4.5 Діалогове вікно перейменування каталогу.

Користувач може скасувати перейменування натиснувши на кнопку «Скасувати». Після натискання на кнопку діалогове вікно зникне і користувач зможе продовжити роботу з програмою.

Для перейменування каталогу користувач повинен ввести в даному вікні нову назву каталогу і натиснути кнопку «Перейменувати». Після натиску кнопки можливі два варіанти розвитку програми (Рис. 4.6 а,б).



а)Перейменовування пройшло вдало.



б) Перейменовування не вдалося.

Рис. 4.6 Варіанти реакції програми на команду перейменування.

При першому варіанті(Рис. 4.6 а) розвитку подій, перейменування каталогу проходить вдало і користувач бачить під таблицею нову назву каталогу.

При другому варіанті (Рис. 4.6 б) розвитку подій, нова назва конфліктує зі старою назвою товару або каталогу . В цьому випадку перейменування не відбувається і користувачу видається повідомлення про невдачу.

Видалення каталогу(Прецедент Рис. 2.4).

Користувач може видалити каталог за умови, що в ньому відсутні інші каталоги.

Для цього він повинен зайти в потрібний каталог і натиснути кнопку «Видалити каталог». Після натиску кнопки програма запросить підтвердження видалення(Рис. 4.7).

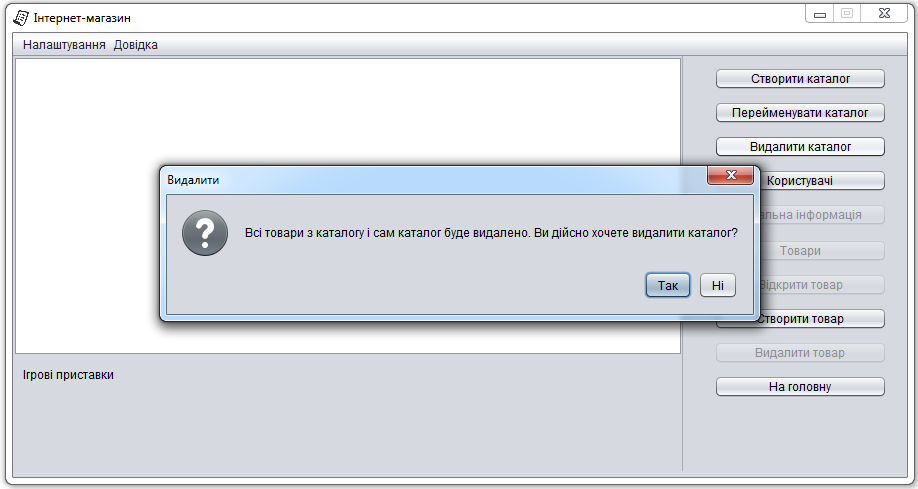


Рис. 4.7 Підтверження видалення.

Користувач може скасувати видалення, натиснувши кнопку «Ні». Вцьому випадку видалення не відбудеться і діалогове вікно зникне.

Якщо користувач підтвердить видалення, натиснувши на кнопку «Так», каталог буде видалено разом зі всіма товарами (Рис. 4.8).

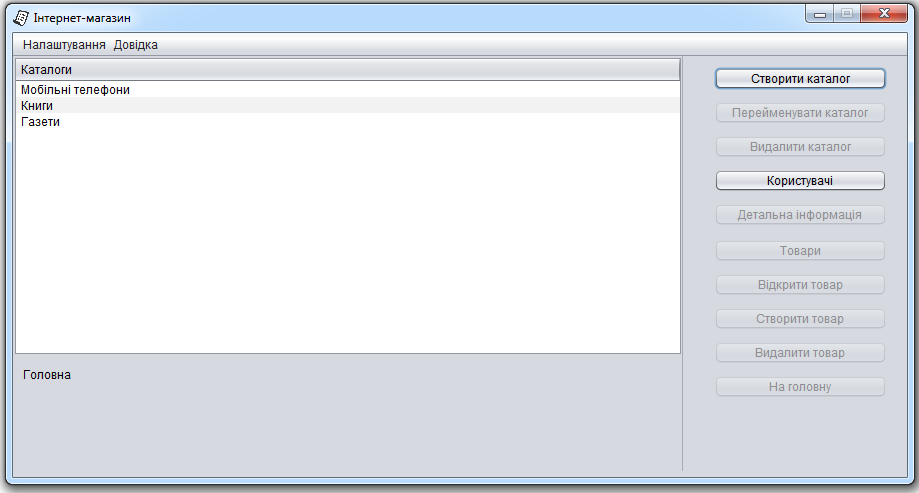


Рис. 4.8 Вікно після видалення каталогу.

Створення каталогу(Прецедент Рис. 2.5).

Користувач може створювати каталоги на головній, а також в середині інших каталогів за умови, що в них відсутні товари.

Для стоврення каталогу користувач повинен зайти в потрібний каталог і натиснути кнопку «Створити каталог». Після натиску кнопки, програма видасть діалогове вікно для створення каталогу (Рис 4.9).

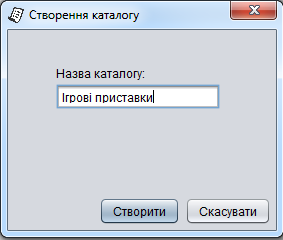


Рис. 4.9 Діалогове вікно створення каталогу.

Користувач може скасувати створення натиснувши на кнопку «Скасувати». Після натискання на кнопку діалогове вікно зникне і користувач зможе продовжити роботу з програмою.

Для створення каталогу користувач повинен ввести в даному вікні нову назву каталогу і натиснути кнопку «Створити». Після натиску кнопки можливі два варіанти розвитку програми (Рис. 4.10 , Рис 4.11).

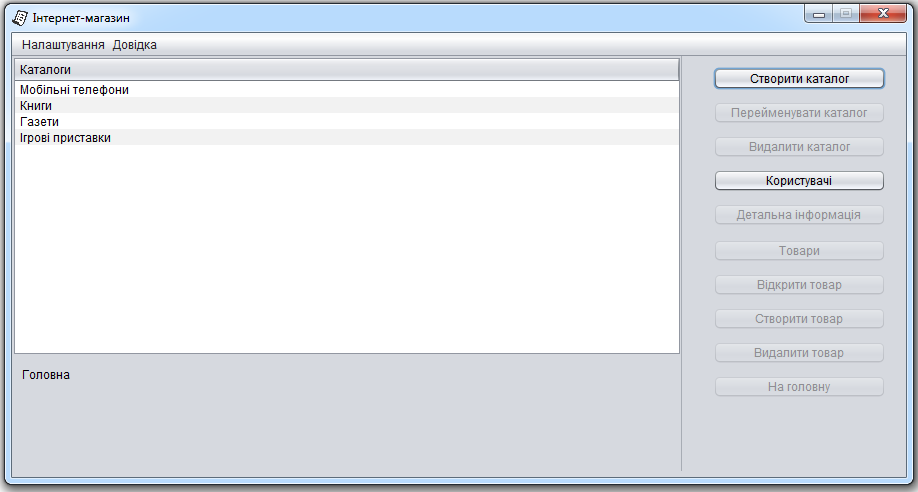


Рис. 4.10 Створення пройшло успішно.

Якщо каталогу або товару з таким іменем не існує в системі, то каталог буде створено. Таблиця буде оновлена і користувач відразу побачить створений ним каталог(Рис. 4.10).

В іншому випадку, якщо в системі існує товар або каталог з таким іменем, то каталог не буде створений, а користувач буде повідомлений повідомленням в головному вікні програми (Рис. 4.11).

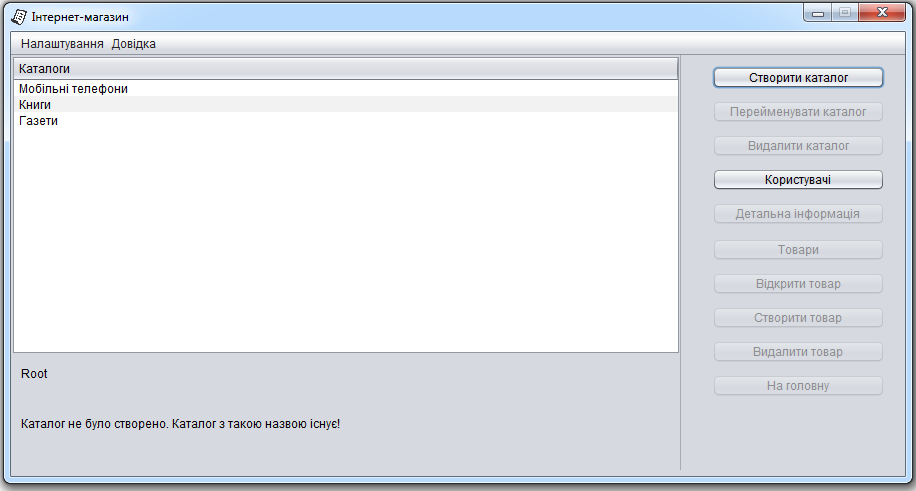


Рис. 4.11 Каталог не було створено.

Детальний перегляд товару(Прецедент Рис. 2.6).

Користувач повинен обрати потрібний товар і натиснути кнопку «Відкрити товар». Після натиску на кнопку програма відкриє діалогове вікно (Рис 4.12).

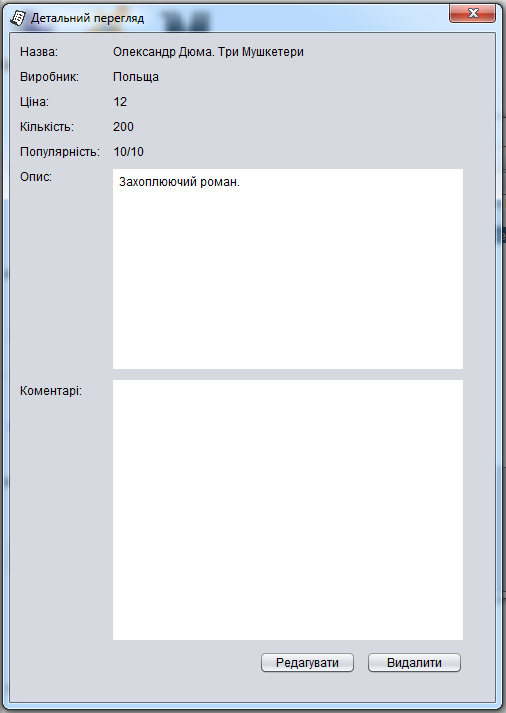


Рис. 4.12 Діалогове вікно з детальною інформацією.

В діалоговому вікні детального перегляду відображаються назва товару, виробник, популярність, ціна, кількість, опис і коментарі.

Видалення товару(Прецедент Рис. 2.7).

Користувач може видалити товар двома способами.

1) В головному вікні програми обрати потрібний товар і натиснути кнопку «Видалити».

2) В головному вікні програми обрати потрібний товар і натиснути кнопку «Відкрити товар». У діалоговому вікні натиснути кнопку «Видалити».

В обох способах після натискання кнопки «Видалити», програма виводить діалогове вікно підтвердження (Рис. 4.13)

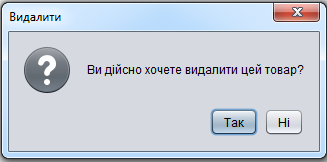


Рис. 4.13 Діалогове вікно підтвердження видалення.

Якщо користувач підтвердить видалення, натиснувши на кнопку «Так», то товар буде видалений з бази.

Якщо користувач захоче скасувати видалення, натиснувши кнопку «Ні», діалогове вікно зникне.

Редагування товару(Прецедент Рис. 2.8).

Користувач може редагувати товар, відкривши його на перегляд і натиснувши на кнопку «Редагувати». Після натиску на кнопку, вікно детального перегляду зникне і програма відобразить діалогове вікно редагування товару (Рис 4.14).

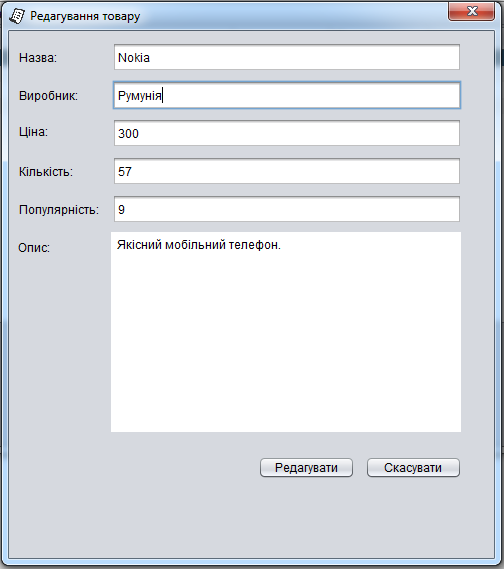
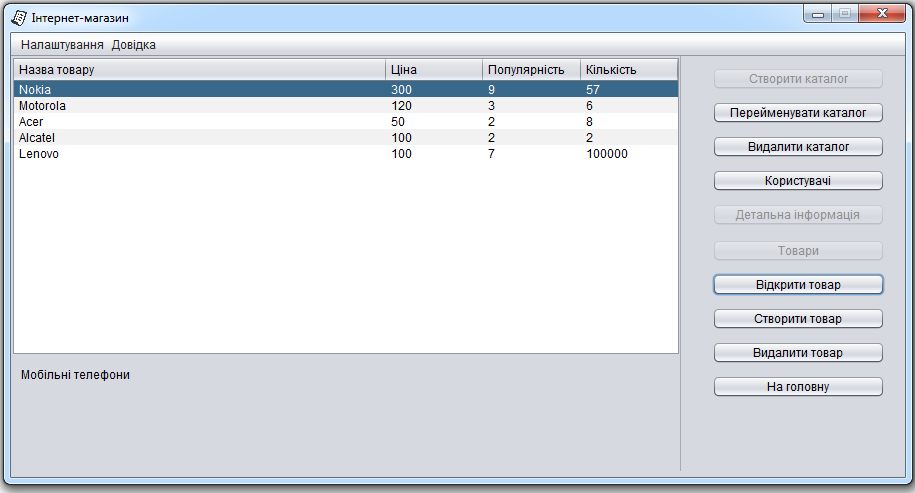


Рис. 4.14 Діалогове вікно редагування товару.

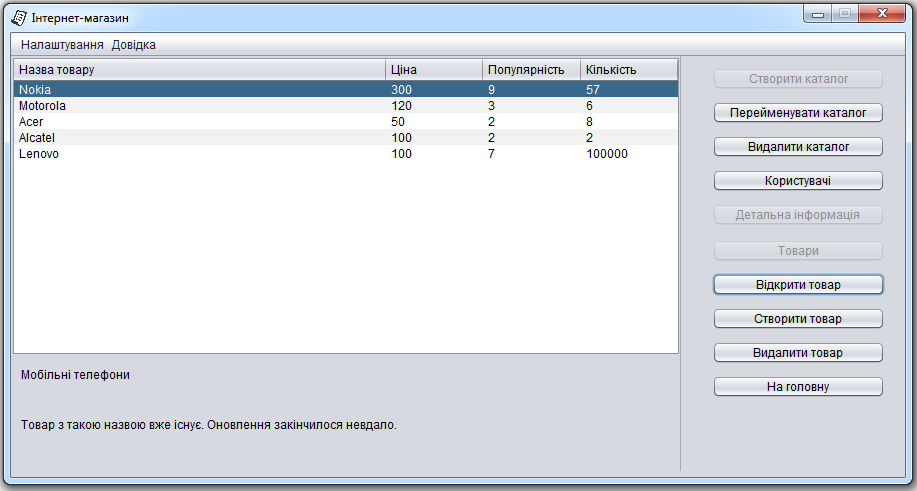
Якщо користувач передумав редагувати товар, то він натискає кнопку «Скасувати». Після натиску кнопки діалогове вікно зникає і користувач може продовжувати взаємодіяти з головним вікном програми.

В іншому випадку користувач редагує потрібні поля і натискає кнопку «Редагувати». Після натискання кнопки, можливі два варіанти реакції програми(Рис. 4.15 а,б).

Якщо після редагування назва товару не дублюється в списку товарів, то інформація редагованого товару обновлюється в базі і головне вікно обновлюється, надаючи користувачу оновлену інформацію про товари.



а) Редагування пройшло успішно.



б) Редагування закінчилось невдало.

Рис. 4.15 Варіанти реагування програми на команду редагування.

Якщо після редагування товару виникає конфлікт між новою назвою і назвами товарів або каталогів зі списку, оновлення інформації про редагований товар не відбувається, а користувачу видається повідомлення про те, що редагування відбулося невдало.

Створення товару(Прецедент Рис. 2.9).

Користувач може створити товар в каталозі, в якому відсутні інші каталоги.Для створення товару користувач повинен зайти в потрібний каталог і натиснути кнопку «Створити товар». Після натиску на кнопку програма відображає діалогове вікно створення товару (Рис. 4.16).

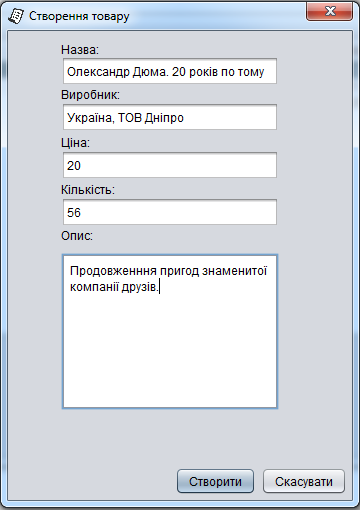
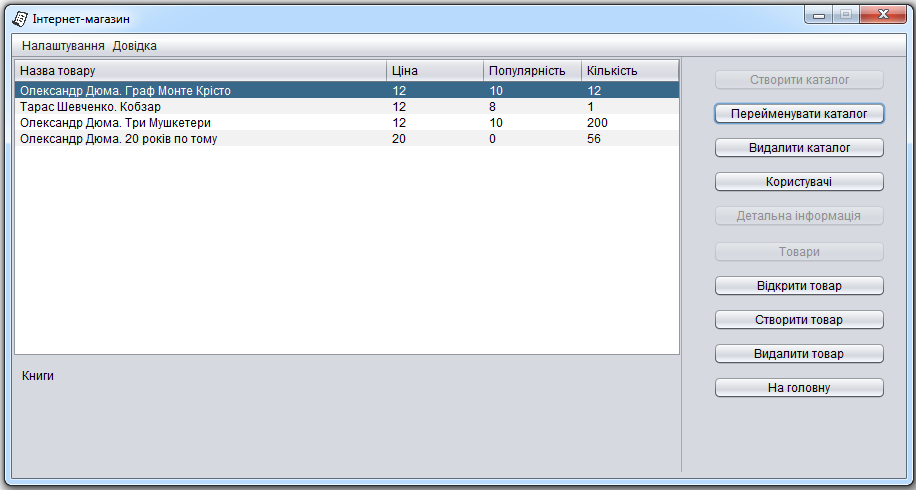


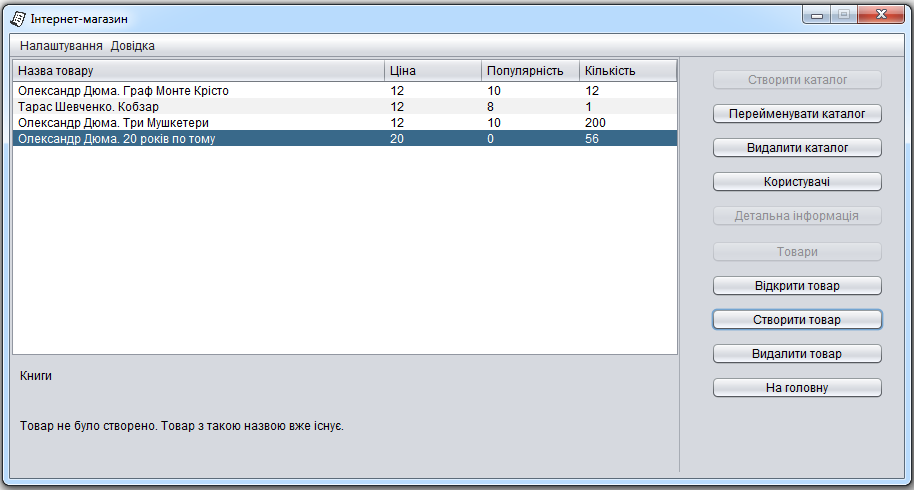
Рис. 4.16 Діалогове вікно створення товару.

Якщо користувач хоче відмінити створення товару, то він натискає кнопку «Скасувати». Після натиску цієї кнопки діалогове вікно зникає і користувач зможе взаємодіяти з головним вікном програми.

Якщо користувач ззаповнить коректно всі поля і натисне кнопку «Створити» можливі два варіанти реакції програми(Рис 4.17 а,б).



а) Створення пройшло вдало.



б) Створення товару закінчилось невдало.

Рис. 4.17 Реакція програми на команду створення товару.

Якщо назва товару не конфліктує з назвами товарів і каталогів, то товар успішно створюється(Рис. 4.17, а). В іншому випадку товар не створюється і

користувач повідомляється про це повідомленням в головному вікні.

Були перевірені всі описані в розділі 2 прецеденти були протестовані. Результати тестування приведенні в цьому розділі вище. Всі прецеденти реалізовані і нормально функціонують.

Перегляд списку користувачів(Прецедент Рис. 2.10).

Користувач може переглядати список всих користувачів системи. Для цього йому необхідно в головному вікні програми натиснути кнопку «Користувачі» (Рис. 4.18).

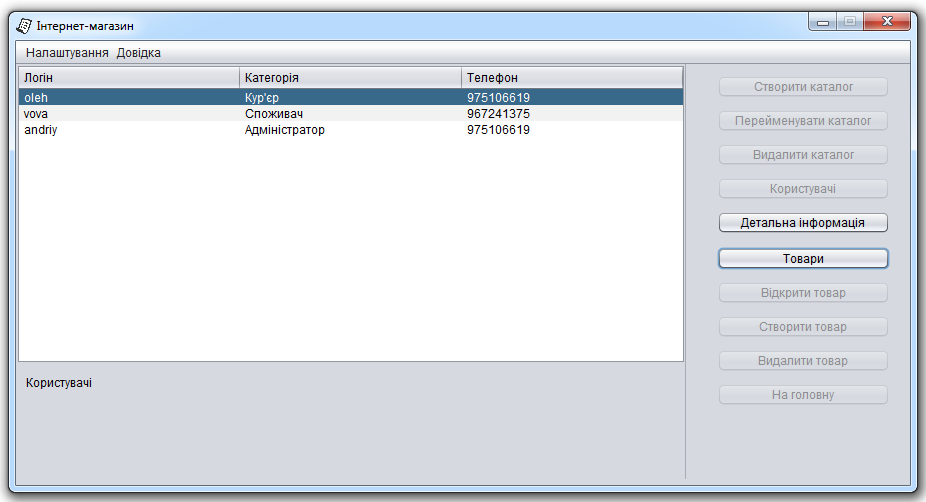


Рис. 4.18 Список всих користувачів системи.

Після натиску кнопки, в головному вікні програми буде відображена таблиця з короткою інформацією про користувачів системи.

Детальний перегляд інформації про користувача (Прецедент Рис. 2.11).

Для детального перегляду інформації про користувача, адміністратору потрібно обрати користувача в таблиці і натиснути кнопку «Детальна інформація».

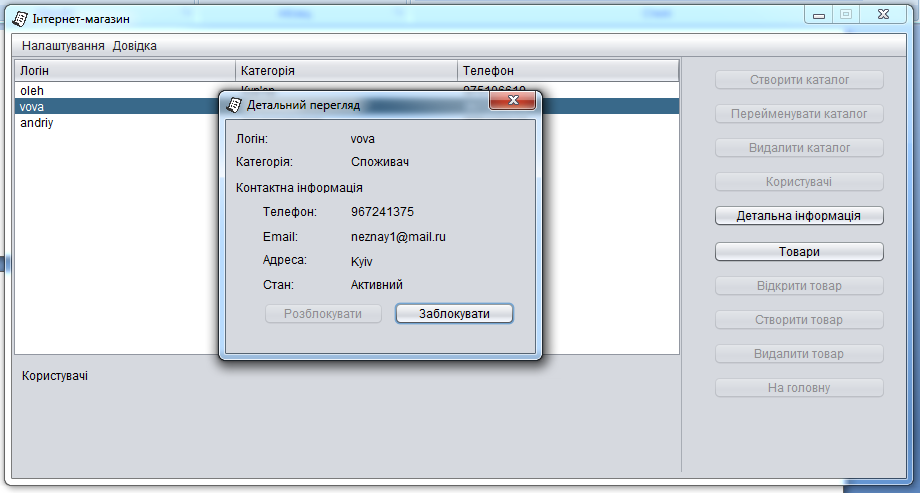


Рис. 4.19 Вікно перегляду детальної інформації про користувача.

Після натиску на кнопку, з’явиться нове діалогове вікно, в якому буде відображена вся інформація про користувача (Рис. 4.19).

Блокування споживача (Прецедент Рис. 4.12).

Адміністратор може заблокувати споживача натиснувши в вікні детального перегляду інформації користувача кнопку «Заблокувати» (Рис. 4.20).

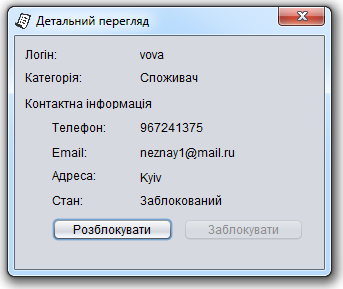


Рис. 4.20 Реакція програми на команду блокування споживача.

Після натиску на кнопку стан користувача змінюється на заблокований, що можна спостерігати в вікні детального перегляду інформації про користувача. Потрібно звернути увагу на те, що в кур’єрів і адміністраторів неможливо змінити стан профілю.

Розблокування споживача (Прецедент Рис. 2.13).

Адміністратор може розблокувати споживача натиснувши на кнопку «Розблокувати» у вікні детального перегляду інформації про користувача (Рис.4.21).

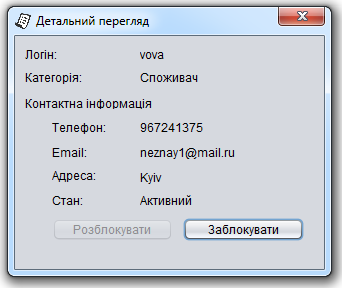


Рис. 4.21 Реакція програми на команду блокування споживача.

Після натиску на кнопку стан споживача змінюється на «Активний» і вікно детальної інформації про користувача оновлюється.

Зміна візуального оформлення (Прецедент Рис. 2.14).

Адміністратор може змінювати візуальний вигляд програми, зайшовши в меню «Налаштування» і в підменю «Стиль» обравши потрібну йому тему програми (Рис 4.22).

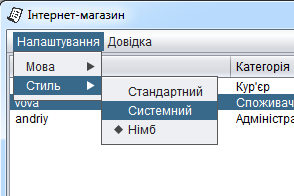


Рис. 4.22 Зміна теми програми.

Після обрання потрібної теми програма відразу змінить своє оформлення на те, що було обране (Рис 4.23).

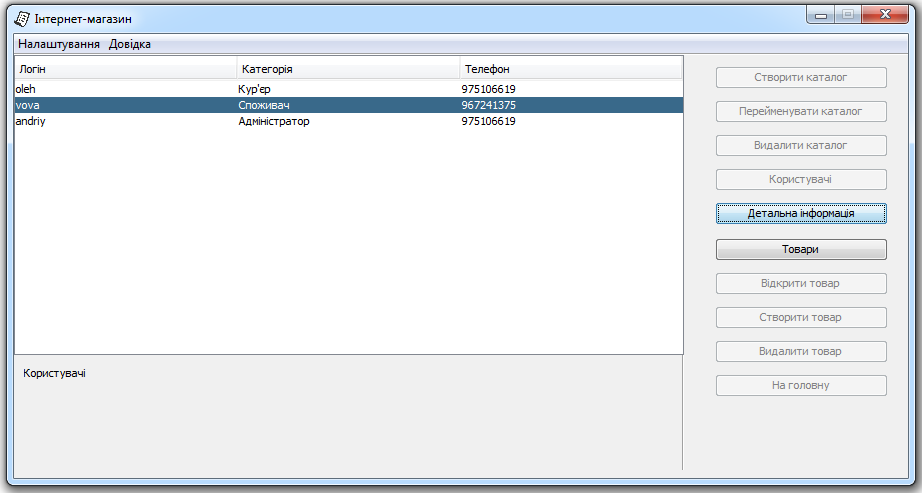


Рис. 4.23 Головне вікно програми після зміни стилю з «Німб» на «Системний».

Тобто користувач може динамічно змінювати оформлення програми обираючи одиз з стандартних стилів.

Зміна мови інтерфейсу (Прецедент Рис. 2.15).

Користувач може змінювати мову інтерфейсу. Для цього йому необхідно зайти в меню «Налаштування» і в підменю «Мови» обрати один з запропонованих пунктів (Рис 4.24).

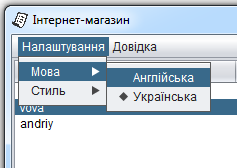


Рис. 4.24 Зміна мови програми.

Після того, як користувач обере одну з мов, мова інтерфейсу зміниться (Рис.4.25).

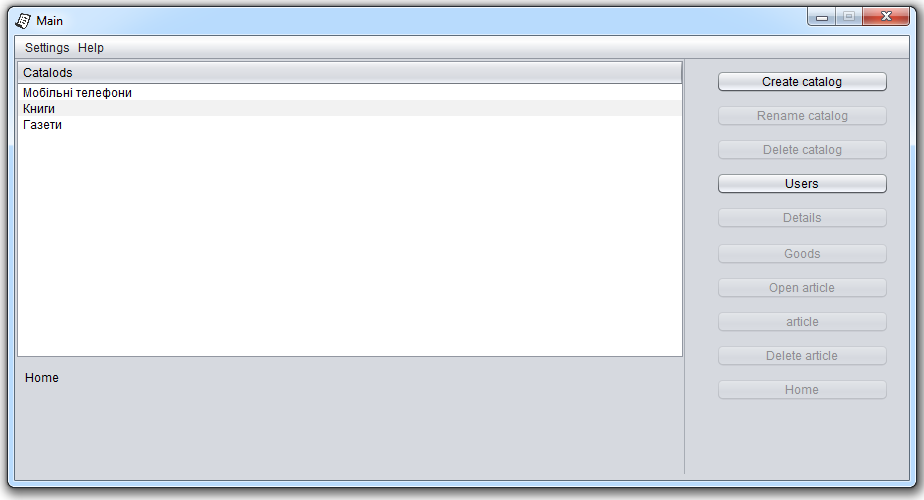


Рис. 4.25 Програма після зміни мови інтерфейсу на англійську.

Мова змінюється динамічно. Після зміни мови програма повернеться на головну.

Перегляд інформації про програму (Прецедент Рис. 2.16).

Користувач може переглянути інформацію про програму , обравши в меню «Довідка» пункт «Про програму». Після вибору пункту з’явиться вікно з інформацією про програму (Рис. 4.26).

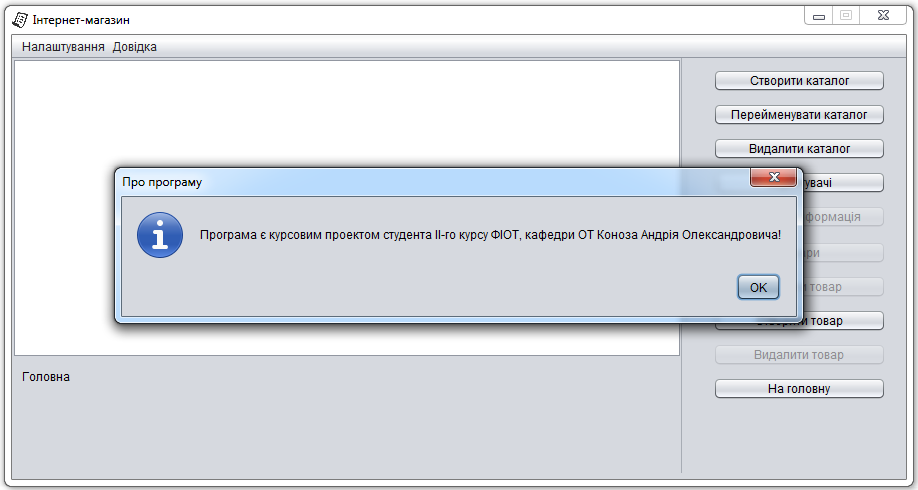


Рис. 4.26 Вікно з інформацією про програму.

Перегляд допомоги по програмі (Прецедент Рис. 2.17).

Користувач може за необхідності переглянути допоміжну інформацію відносно програми. Для цього йому необхідно обрати в меню «Довідка» пункт «Допомога».

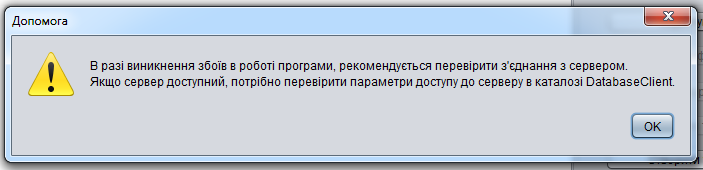


Рис. 4.27 Вікно допомоги.

Після вибору даного пункту програма згенерує текст допомоги в окремому вікні.

ВИСНОВКИ.

В результаті виконання курсової роботи була створена повноцінний програмний продукт, при якого були освоєні і використані комплексні знання як стандартів програмування, так і самої мови програмування Java.

Під час розробки графічного інтерфейсу були освоєні навички роботи з графічними бібліотеками мови програмування Java. Використовуючи графічні компоненти даних бібліотек був побудовиний складний графічний інтерфейс користувача, який базувався на взаємодії різних компонентів бібліотек Swing i AWT.

Розробляючи структуру інтерфейсу було вирішено сконцентрувати основну логіку програми в одному вікні, а деталі операцій винести в окремі діалогові вікна. Для реалізації цього рішення були використані JFrame для головного вікна програми і форми авторизації, JDialog для допоміжних вікон. При цьому була використана ще одна особливість класу JDialog – модальність. Даний клас дозволяє створювати об’єкти модального типу (при відображенні таких об’єктів активним є лише вікно даного об’єкта, всі інші вікна є недоступними для взаємодії). Це було зроблено для того, щоб користувач не заплутувався при виклиці допоміжних вікон.

Згідно обраного паттерну (MVA) графічний інтерфейс займався лише відображенням інформації, яку йому давав контролер. Тому для отримання інформації з моделі, вигляд (графічний інтерфейс) звертається до контролера. Це відбувається наступним чином.

Якщо користувач викликає операцію інтерфейсу, яка потребує інформації з моделі, то інтерфейс формує контекст для команди і передає контролеру цей контекст разом з індентифікаторои команди. Після отримання інформації від контролера інтерфейс відображає отриману інформацію.

Розробляючи частину програми, яка називається контролером, були освоєні і використані навички роботи з рефлексією, синхронізованого виконання команд, роботи з базою чаних і XML-файлами, а також навички роботи з анотаціями і написання власних анотацій.

Рефлексія використовувалась для написання гнучкого DAO, який міг би без перекомпіляції працювати з класами, кількість яких могла зростати. Для цього класи які повинні були записуватися до бази даних розмічалися спеціально створеними анотаціями. Ці анотації містили в собі інформацію про назву таблиці, полів і типу цих полів. За допомогою рефлексії об’єкт DAO отримував доступ до цікавлячих його полів і їх значень. Завдяки анотаціям об’єкт DAO дізнавався як правильно сформувати запит до бази даних, а також що куди і під яким доменом повинне зберігатися.

За схожим принципом рефлексія і анотації використовувалися у валідаторі. Валідатор перевіряв відповідність контексту переданого інтерфейсом до параметрів, які необхідні команді. Тут анотації несли в собі назви параметрів і їх тип.

Взагалі роль контролера полягає в тому щоб за даними отриманими від інтерфейсу він відправив кпотрібну команду на виконання. Для цього він спочатку за ідентифікатором у фабрики команд отримує потрібну команду. Потім він цю команду і контекст до команди передає валідатору. Валідатор перевіряє коректність переданого контексту. Якщо контекст підходить команді, контролер передає команді контекст, а потім команду вже зі встановленим контекстом передає в чергу класу, який запускає команди на виконання.

Для того, щоб контролер завжди був лише в одному екземплярі, він був зроблений Одинаком.

Клас, який займається запуском команд на виконання (далі виконавець) реалізований за допомогою стандартної бібліотеки java.util.concurrent. Клас має чергу, яка містить всі команди передані на виконання. Виконавець запускає всі команди в окремому потоці . Він запускає наступну команду лише після того, як виконається попередня. Цей підхід дозволяє уникну и одночасного доступу декількох команд до моделі, що веде до усунення різних прикрих помилок при роботі з моделлю.

Для того, щоб команди завжди запускались в одному потоці, метод, що приймає команду був зробленим статичним.

Фабрика команд генерує на кожен запит нову команду, оскільки в черзі може бути декілька однакових команд, що призведе до запирання контексту більш ранніх команд. Єдиною причиною створення фабрики була ідея централізації створення.

Моделлю взагалі є сама база данних. Як частину моделі можна вважати класи, які призначені для зберігання рядків таблиці, хоч їх можна віднести також і до контролера.

Контролер не зберігає жодних частин моделі. Тому кожна команда під час виконання звертається до бази данних. Це рішення було прийняте через те, що дані в базі можуть швидко оновлюватись, а адміністратор повинен завжди працювати лише зі свіжими даними.

Для правильного налаштування роботи програми було створено композитну структуру, яка зберігає налаштування. Тут були освоєні і використані навички роботи з класом Properties бібліотеки java.util. Для збереження використовувались XML-файли і стандартні засоби роботи з ними.

Для правильного функціонування програми було необхідно, щоб композитна структура налаштувань перебувала в одному екземплярі. Для цього було використано паттерн Одинак. Він дозволяв з будь-якої точки програми отримати доступ до єдиного композиту налаштувань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Приемы объектно-ориентированого проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2011 – 368 с.: ил. –ISBN 978-5-469-01136-1.
2. Герберт Шилдт SWING: руководство для начинающих– М.:“Вильямс”, 2007. – С. 720. – ISBN 0-07-226314-8.
3. 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.)
4. 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.
5. Хорстманн Кей С. Java 2. Том 1. Основы / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 816 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1378-4 (рус.).
6. Хорстманн Кей С. Java 2. Том 2. Тонкости программирования / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 992 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1482-8 (рус.).
7. Стелтинг Стивен Применение шаблонов Java /Стелтинг Стивен, Маасен Олав; Пер. с англ. –М.: Издательский дом “Вильямс”, 2002. – 576 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). – ISBN 5-8459-0339-4 (рус.).
8. [А. К. Гультяев](http://www.ozon.ru/context/detail/id/5192567/#tab_person)  Проектирование и дизайн пользовательского интерфейса / В. А. Машин; Пер. с англ. : Издательский дом “ [Корона-Принт](http://www.ozon.ru/context/detail/id/856652/) ”, 2010. – 350 c. – ISBN 978-5-7931-0814-0 (рус.).

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ.

**package** application;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.util.Hashtable;

**import** javax.swing.\*;

**import** javax.swing.border.EmptyBorder;

**import** configuration.CompositeStructure.\*;

**import** controller.\*;

**import** controller.Factory.CommandFactory;

**import** dao.CRUIDRealisation;

/\*\*

\* Клас реалізує діалогове вікно авторизації.

\* **@author** Коноз Андрій

\*/

**public** **class** Autorization **extends** JFrame {

/\*\*

\* Основна панель вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для введення логіну.

\*/

**private** JTextField textField;

/\*\*

\* Поле для введення паролю.

\*/

**private** JPasswordField passwordField;

/\*\*

\* Посилання на самого себе.

\*/

**private** Autorization link = **this**;

/\*\*

\* Мітка з тестом "логін"

\*/

**private** JLabel lblEnterLoginAnd;

/\*\*

\* Кнопка для відображення дій при втраті паролю.

\*/

**private** JButton btnForgotPassword;

/\*\*

\* Мітка з текстом "пороль".

\*/

**private** JLabel lblPassword;

/\*\*

\* Кнопка для входження в систему.

\*/

**private** JButton btnLogin;

/\*\*

\* Таблиця, в якій зберігаються написи компонентів

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Запуск програми

\*/

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

**try** {

Autorization dialog = **new** Autorization();

dialog.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

dialog.setVisible(**true**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Створення і налаштування діалогового вікна.

\*/

**public** Autorization() {

Controller.*getController*().dao = **new** CRUIDRealisation();

CompositeComponent comp = **new** CompositePropery();

comp.load("D:\\DatabaseClient\\mainConfig.xml");

AppConfiguration appConfig = AppConfiguration.*getAppConfiguration*(comp);

**try** {

Controller.*getController*().dao.open(appConfig);

}**catch**(Exception e){

System.*out*.println("kurwa!");

e.printStackTrace();

}

**this**.context = ContextUtils.*loadContext*("Autorization");

setIcon();

**try** {

CompositeComponent ob = AppConfiguration.*getAppConfiguration*()

.getProperties("ProgramTheme");

UIManager.*setLookAndFeel*(ob.getProperty("theme"));

} **catch** (Exception e) {

e.printStackTrace();

}

setTitle(context.get("title"));

setBounds(100, 100, 282, 302);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblLogin = **new** JLabel(context.get("lblLogin"));

lblLogin.setBounds(63, 11, 131, 14);

contentPanel.add(lblLogin);

initComponents();

allocateComponents();

btnLogin.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", link);

context.put("login", textField.getText());

String pass = **new** String();

**for** (**char** ch : passwordField.getPassword()) {

pass += ch;

}

context.put("password", pass);

Controller.*getController*().execute(context,

CommandFactory.*AUTORIZATION*);

}

});

}

});

btnForgotPassword.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

lblEnterLoginAnd.setText(context.get("lblEnterLoginAnd"));

}

});

}

/\*\*

\* Ініціалізація компонентів.

\*/

**private** **void** initComponents() {

textField = **new** JTextField();

textField.setBounds(63, 30, 131, 25);

textField.setColumns(10);

lblPassword = **new** JLabel(context.get("lblPassword"));

lblPassword.setBounds(63, 61, 131, 14);

passwordField = **new** JPasswordField();

passwordField.setBounds(63, 78, 131, 25);

btnLogin = **new** JButton(context.get("btnLogin"));

btnLogin.setBounds(63, 109, 131, 23);

btnForgotPassword = **new** JButton(context.get("btnForgotPassword"));

btnForgotPassword.setBounds(63, 143, 131, 23);

lblEnterLoginAnd = **new** JLabel(context.get("lblEnterLoginAndPassword"));

lblEnterLoginAnd.setHorizontalAlignment(SwingConstants.*CENTER*);

lblEnterLoginAnd.setBounds(10, 177, 246, 76);

}

/\*\*

\* Розміщення компонентів в діалоговому вікні.

\*/

**private** **void** allocateComponents() {

contentPanel.add(textField);

contentPanel.add(lblPassword);

contentPanel.add(passwordField);

contentPanel.add(btnLogin);

contentPanel.add(btnForgotPassword);

contentPanel.add(lblEnterLoginAnd);

}

/\*\*

\* Метод для зворотнього зв'язку. Використовується для отримання результату

\* виконання команди авторизування. Приймає параметром клас користувача.

\*

\* **@param** solution

\* клас користувача.

\*/

**public** **void** getSolution(**int** solution) {

**if** (solution == 3) {

**try** {

setVisible(**false**);

**new** MainFrame();

} **catch** (Exception e) {

e.printStackTrace();

}

} **else** **if** (solution == 2) {

lblEnterLoginAnd

.setText(context.get("StoremanError"));

} **else** **if** (solution == 1) {

lblEnterLoginAnd

.setText(context.get("CustomerError"));

} **else** {

lblEnterLoginAnd.setText(context.get("AnotherError"));

}

}

/\*\*

\* Метод встановлює піктограму на вікно

\*/

**private** **void** setIcon() {

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.FlowLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

/\*\*

\* Клас реалізовує функціонал діалогового вікна створення

\* каталогу

\* **@author** Коноз Андрій

\*/

**public** **class** CreateCatalog **extends** JDialog {

/\*\*

\* Основна панель вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для введення імені каталогу.

\*/

**private** JTextField textField;

/\*\*

\* Кнопка для підтвердження.

\*/

**private** JButton okButton;

/\*\*

\* Посилання на головне вікно.

\*/

**private** MainFrame link;

/\*\*

\* Кнопка для скасування.

\*/

**private** JButton cancelButton;

/\*\*

\* Таблиця з написами на компонентах.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Створення і налаштування діалогового вікна.

\* **@param** link посилання на головне вікно.

\*/

**public** CreateCatalog(MainFrame link) {

setIcon();

setResizable(**false**);

**this**.context = ContextUtils.*loadContext*("CreateCatalog");

**this**.link = link;

**this**.setModal(**true**);

setTitle(context.get("title"));

setBounds(100, 100, 274, 228);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

textField = **new** JTextField();

textField.setBounds(48, 54, 164, 25);

contentPanel.add(textField);

textField.setColumns(10);

JLabel lblCatalogName = **new** JLabel(context.get("lblCatalogName"));

lblCatalogName.setBounds(48, 36, 164, 14);

contentPanel.add(lblCatalogName);

JPanel buttonPane = **new** JPanel();

buttonPane.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

getContentPane().add(buttonPane, BorderLayout.*SOUTH*);

okButton = **new** JButton(context.get("okButton"));

okButton.setActionCommand("OK");

buttonPane.add(okButton);

getRootPane().setDefaultButton(okButton);

cancelButton = **new** JButton(context.get("cancelButton"));

cancelButton.setActionCommand("Cancel");

buttonPane.add(cancelButton);

okButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", CreateCatalog.**this**.link);

context.put("name", textField.getText());

Controller.*getController*().execute(context,

CommandFactory.*CREATE\_CATALOG*);

CreateCatalog.**this**.dispose();

}

});

cancelButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CreateCatalog.**this**.dispose();

}

});

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

}

/\*\*

\* Метод встановлює піктограму на вікно

\*/

**private** **void** setIcon() {

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.beans.XMLDecoder;

**import** java.io.FileInputStream;

**import** java.util.Hashtable;

**import** configuration.CompositeStructure.AppConfiguration;

**import** configuration.CompositeStructure.CompositeComponent;

/\*\*

\* Клас надає методи для роботи з різними мовами.

\* **@author** Коноз Андрій

\*/

**public** **class** ContextUtils {

/\*\*

\* Метод завантажує назви елементів конкретного вікна.

\* **@param** component назва вікна, для якого потрібно завантажити список

\* **@return** таблицю з назвами компонентів

\*/

**public** **static** Hashtable<String, String> loadContext(String component){

Hashtable<String, String> result = **null**;

CompositeComponent compprop = AppConfiguration.*getAppConfiguration*().getProperties("SelectedLanguage");

String language =(String) compprop.get("language");

String path = (String) AppConfiguration.*getAppConfiguration*().getProperties(language).get(component);

**try**{

FileInputStream fis = **new** FileInputStream(path);

XMLDecoder d = **new** XMLDecoder(fis);

result =(Hashtable<String, String>) d.readObject();

d.close();

}**catch**(Exception e){

e.printStackTrace();

}

**return** result;

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.FlowLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.ImageIcon;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**import** javax.swing.JTextArea;

**import** javax.swing.JScrollPane;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

/\*\*

\* Клас реалізовує діалогове вікно для створення товару.

\* **@author** Коноз Андрій

\*/

**public** **class** CreateGood **extends** JDialog {

/\*\*

\* Основна панель вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для введення назви товару.

\*/

**private** JTextField nameField;

/\*\*

\* Поле для введення виробника.

\*/

**private** JTextField manufacturedField;

/\*\*

\* Поле для введення ціни.

\*/

**private** JTextField priceField;

/\*\*

\* Поле для введення кількості товару.

\*/

**private** JTextField countField;

/\*\*

\* Поле для введення опису товару.

\*/

**private** JTextArea descriptionArea;

/\*\*

\* Таблиця з написами на компонентах.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Мітка для відображення попереджень.

\*/

**private** JLabel lblError;

/\*\*

\* Посилання на власника вікна.

\*/

**private** MainFrame link;

/\*\*

\* Кнопка для скасування створення.

\*/

**private** JButton cancelButton;

/\*\*

\* Створення і налаштування діалогового вікна.

\* **@param** link посилання на головне вікно.

\*/

**public** CreateGood(MainFrame link) {

context = ContextUtils.*loadContext*("CreateGoods");

setIcon();

**this**.link = link;

setTitle(context.get("title"));

setResizable(**false**);

setBounds(100, 100, 350, 497);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblName = **new** JLabel(context.get("lblName"));

lblName.setBounds(53, 11, 218, 14);

contentPanel.add(lblName);

nameField = **new** JTextField();

nameField.setBounds(53, 25, 218, 30);

contentPanel.add(nameField);

nameField.setColumns(10);

JLabel lblManufactured = **new** JLabel(context.get("lblManufactured"));

lblManufactured.setBounds(53, 56, 218, 14);

contentPanel.add(lblManufactured);

manufacturedField = **new** JTextField();

manufacturedField.setBounds(53, 71, 218, 30);

contentPanel.add(manufacturedField);

manufacturedField.setColumns(10);

JLabel lblPrise = **new** JLabel(context.get("lblPrise"));

lblPrise.setBounds(53, 104, 218, 14);

contentPanel.add(lblPrise);

priceField = **new** JTextField();

priceField.setBounds(53, 119, 218, 30);

contentPanel.add(priceField);

priceField.setColumns(10);

JLabel lblCount = **new** JLabel(context.get("lblCount"));

lblCount.setBounds(53, 151, 218, 14);

contentPanel.add(lblCount);

countField = **new** JTextField();

countField.setBounds(53, 166, 218, 30);

contentPanel.add(countField);

countField.setColumns(10);

JLabel lblDescription = **new** JLabel(context.get("lblDescription"));

lblDescription.setBounds(53, 197, 218, 14);

contentPanel.add(lblDescription);

JScrollPane scrollPane = **new** JScrollPane();

scrollPane.setBounds(53, 222, 218, 157);

contentPanel.add(scrollPane);

descriptionArea = **new** JTextArea();

scrollPane.setViewportView(descriptionArea);

lblError = **new** JLabel("");

lblError.setBounds(53, 390, 218, 35);

contentPanel.add(lblError);

JPanel buttonPane = **new** JPanel();

buttonPane.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

getContentPane().add(buttonPane, BorderLayout.*SOUTH*);

JButton okButton = **new** JButton(context.get("okButton"));

okButton.setActionCommand("OK");

buttonPane.add(okButton);

getRootPane().setDefaultButton(okButton);

cancelButton = **new** JButton(context.get("cancelButton"));

cancelButton.setActionCommand("Cancel");

buttonPane.add(cancelButton);

cancelButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CreateGood.**this**.dispose();

}

});

okButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", CreateGood.**this**.link);

context.put("catalog", CreateGood.**this**.link.getTitleLabel()

.getText());

context.put("name", nameField.getText());

context.put("manufactured", manufacturedField.getText());

**try** {

context.put("count", Integer.*parseInt*(countField.getText()));

} **catch** (Exception e) {

lblError.setText(CreateGood.**this**.context

.get("CountMustBeNumber"));

**return**;

}

**try** {

context.put("price", Integer.*parseInt*(priceField.getText()));

} **catch** (Exception e) {

lblError.setText(CreateGood.**this**.context

.get("PriceMustBeNumber"));

**return**;

}

context.put("description", descriptionArea.getText());

Controller.*getController*().execute(context,

CommandFactory.*CREATE\_GOODS*);

setVisible(**false**);

}

});

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

}

/\*\*

\* Метод встановлює піктограму на вікно

\*/

**private** **void** setIcon(){

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

/\*\*

\* Клас реалізовує діалогове вікно для редагування товару.

\* **@author** Коноз Андрій

\*/

**public** **class** EditCatalog **extends** JDialog {

/\*\*

\* Панель, на якій розташовуються всі компоненти вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для введення нової назви.

\*/

**private** JTextField textField;

/\*\*

\* Посилання на головне вікно.

\*/

**private** MainFrame link;

/\*\*

\* Таблиця з написами на компонентах.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Створення і налаштування діалогового вікна

\* **@param** link посилання на головне вікно.

\*/

**public** EditCatalog(MainFrame link) {

**this**.link = link;

setIcon();

setResizable(**false**);

getContentPane().setLayout(**null**);

context = ContextUtils.*loadContext*("EditCatalog");

setTitle(context.get("title"));

setBounds(100, 100, 302, 210);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblNewName = **new** JLabel(context.get("lblNewName"));

lblNewName.setBounds(15, 38, 261, 14);

contentPanel.add(lblNewName);

textField = **new** JTextField();

textField.setBounds(15, 63, 261, 25);

contentPanel.add(textField);

textField.setColumns(10);

JButton okButton = **new** JButton(context.get("okButton"));

okButton.setBounds(10, 136, 119, 25);

contentPanel.add(okButton);

okButton.setActionCommand("OK");

getRootPane().setDefaultButton(okButton);

JButton cancelButton = **new** JButton(context.get("cancelButton"));

cancelButton.setBounds(157, 136, 119, 25);

contentPanel.add(cancelButton);

cancelButton.setActionCommand("Cancel");

cancelButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

dispose();

}

});

okButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", EditCatalog.**this**.link);

context.put("name", textField.getText());

Controller.*getController*().execute(context,

CommandFactory.*EDIT\_CATALOG*);

dispose();

}

});

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

}

/\*\*

\* Метод встановлює піктограму на вікно

\*/

**private** **void** setIcon() {

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.FlowLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JEditorPane;

**import** javax.swing.JPanel;

**import** javax.swing.JTextField;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.JScrollPane;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

**import** application.Model.Description;

**import** application.Model.Goods;

/\*\*

\* Клас реалізовує функціонал діалогового вікна для редагування товару.

\* **@author** Коноз Андрій

\*/

**public** **class** EditGoods **extends** JDialog {

/\*\*

\* Панель, яка містить в собі всі компоненти вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для введення назви товару.

\*/

**private** JTextField namefield;

/\*\*

\* Поле для введення виробника.

\*/

**private** JTextField manufacturedfield;

/\*\*

\* Поле для введення ціни.

\*/

**private** JTextField prisefield;

/\*\*

\* Поле для введення кількості.

\*/

**private** JTextField countfield;

/\*\*

\* Поле для введення опису.

\*/

**private** JEditorPane descriptionArea;

/\*\*

\* Поле для введення популярності.

\*/

**private** JTextField popularityfield;

/\*\*

\* Посилання на головне вікно.

\*/

**protected** MainFrame link;

/\*\*

\* Мітка для відображення попереджень.

\*/

**private** JLabel lblTips;

/\*\*

\* Товар, який редагується.

\*/

**private** Goods goods;

/\*\*

\* Опис товару, який редагується.

\*/

**private** Description descr;

/\*\*

\* Таблиця з написами компонентів.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Конструктор налаштовує діалогове вікно згідно отриманих параметрів.

\* **@param** link посилання на головне вікно.

\* **@param** goods товар, для редагування.

\* **@param** descr опис товару, який редагується.

\*/

**public** EditGoods(MainFrame link, Goods goods, Description descr) {

setResizable(**false**);

context = ContextUtils.*loadContext*("EditGoods");

setIcon();

**this**.link = link;

**this**.goods = goods;

**this**.descr = descr;

setTitle(context.get("title"));

setBounds(10, 10, 491, 555);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblName = **new** JLabel(context.get("lblName"));

lblName.setBounds(10, 11, 89, 30);

contentPanel.add(lblName);

namefield = **new** JTextField(goods.getName());

namefield.setBounds(103, 11, 350, 30);

contentPanel.add(namefield);

JLabel lblManufactured = **new** JLabel(context.get("lblManufactured"));

lblManufactured.setBounds(10, 49, 89, 30);

contentPanel.add(lblManufactured);

manufacturedfield = **new** JTextField(descr.getMaker());

manufacturedfield.setBounds(103, 49, 350, 30);

contentPanel.add(manufacturedfield);

JLabel lblPrice = **new** JLabel(context.get("lblPrice"));

lblPrice.setBounds(10, 87, 89, 27);

contentPanel.add(lblPrice);

prisefield = **new** JTextField(Integer.*toString*(goods.getPrice()));

prisefield.setBounds(103, 87, 350, 30);

contentPanel.add(prisefield);

JLabel lblCount = **new** JLabel(context.get("lblCount"));

lblCount.setBounds(10, 125, 89, 30);

contentPanel.add(lblCount);

countfield = **new** JTextField(Integer.*toString*(goods.getNumberOfgoods()));

countfield.setBounds(103, 125, 350, 30);

contentPanel.add(countfield);

JLabel lblDescription = **new** JLabel(context.get("lblDescription"));

lblDescription.setBounds(9, 201, 90, 30);

contentPanel.add(lblDescription);

JScrollPane scrollPane = **new** JScrollPane();

scrollPane.setBorder(**null**);

scrollPane.setBounds(102, 201, 350, 200);

contentPanel.add(scrollPane);

descriptionArea = **new** JEditorPane();

descriptionArea.setText(descr.getText());

descriptionArea.setEditable(**true**);

scrollPane.setViewportView(descriptionArea);

JButton btnCancel = **new** JButton(context.get("btnCancel"));

btnCancel.setBounds(356, 425, 97, 23);

contentPanel.add(btnCancel);

JButton btnEdit = **new** JButton(context.get("btnEdit"));

btnEdit.setBounds(249, 425, 97, 23);

contentPanel.add(btnEdit);

JLabel lblPopularity = **new** JLabel(context.get("lblPopularity"));

lblPopularity.setBounds(10, 163, 89, 30);

contentPanel.add(lblPopularity);

popularityfield = **new** JTextField(Integer.*toString*(goods.getMark()));

popularityfield.setBounds(103, 163, 350, 30);

contentPanel.add(popularityfield);

lblTips = **new** JLabel("");

lblTips.setBounds(103, 459, 350, 39);

contentPanel.add(lblTips);

JPanel buttonPane = **new** JPanel();

buttonPane.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

getContentPane().add(buttonPane, BorderLayout.*SOUTH*);

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

btnCancel.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

EditGoods.**this**.dispose();

}

});

btnEdit.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

lblTips.setText("");

ContextForCommand context = **new** ContextForCommand();

**boolean** isChange = **false**;

**try** {

**if**(EditGoods.**this**.goods.getMark() != Integer.*parseInt*(popularityfield.getText())){

EditGoods.**this**.goods.setMark(Integer.*parseInt*(popularityfield.getText()));

isChange = **true**;

}

} **catch** (Exception e) {

lblTips.setText(EditGoods.**this**.context.get("MarkMustBeNumber"));

**return**;

}

**try** {

**if**(EditGoods.**this**.goods.getNumberOfgoods() != Integer.*parseInt*(countfield.getText())){

EditGoods.**this**.goods.setNumberOfgoods(Integer.*parseInt*(countfield.getText()));

isChange = **true**;

}

} **catch** (Exception e) {

lblTips.setText(EditGoods.**this**.context.get("CountMustBeNumber"));

**return**;

}

**try** {

**if**(EditGoods.**this**.goods.getPrice() != Integer.*parseInt*(prisefield.getText())){

EditGoods.**this**.goods.setPrice(Integer.*parseInt*(prisefield.getText()));

isChange = **true**;

}

} **catch** (Exception e) {

lblTips.setText(EditGoods.**this**.context.get("PriceMustBeNumber"));

**return**;

}

**if**(!EditGoods.**this**.goods.getName().equals(namefield.getText())){

EditGoods.**this**.goods.setName(namefield.getText());

isChange = **true**;

}

**if**(!EditGoods.**this**.descr.getMaker().equals(manufacturedfield.getText())){

EditGoods.**this**.descr.setMaker(manufacturedfield.getText());

isChange = **true**;

}

**if**(!EditGoods.**this**.descr.getText().equals(descriptionArea.getText())){

EditGoods.**this**.descr.setText(descriptionArea.getText());

isChange = **true**;

}

**if**(isChange){

context.put("goods", EditGoods.**this**.goods);

context.put("description", EditGoods.**this**.descr);

context.put("proprietor", EditGoods.**this**.link);

Controller.*getController*().execute(context, CommandFactory.*EDIT\_GOODS*);

}

EditGoods.**this**.dispose();

}

});

}

/\*\*

\* Метод встановлює піктограму на вікно.

\*/

**private** **void** setIcon(){

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowEvent;

**import** java.awt.event.WindowListener;

**import** java.util.Hashtable;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.ImageIcon;

**import** javax.swing.JOptionPane;

**import** javax.swing.JScrollPane;

**import** javax.swing.JMenuBar;

**import** javax.swing.JTable;

**import** javax.swing.JSeparator;

**import** javax.swing.JViewport;

**import** javax.swing.SwingConstants;

**import** javax.swing.JMenu;

**import** javax.swing.JButton;

**import** javax.swing.JMenuItem;

**import** javax.swing.JRadioButtonMenuItem;

**import** javax.swing.JLabel;

**import** javax.swing.SwingUtilities;

**import** javax.swing.UIManager;

**import** configuration.CompositeStructure.AppConfiguration;

**import** configuration.CompositeStructure.CompositeComponent;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

/\*\*

\* Клас реалізує функціонал головного вікна програми.

\* **@author** Коноз Андрій

\*/

**public** **class** MainFrame **extends** JFrame {

/\*\*

\* Основна панель, в якій будуть міститися всі інші об'єкти.

\*/

**private** JPanel contentPane;

/\*\*

\* Таблиця, в якій буде відображатися інформація директорій з товарами.

\*/

**private** JTable goodsTable;

/\*\*

\* Таблиця, в якій буде відображатися інформація про клієнтів.

\*/

**private** JTable clientsTable;

/\*\*

\* Кнопка, яка відображає таблицю товарів.

\*/

**private** JButton btnGoods;

/\*\*

\* Кнопка, яка викликає вікно для редагування коталогу.

\*/

**private** JButton btnEditCatalog;

/\*\*

\* Мітка, якій свідображається підказки.

\*/

**private** JLabel lblNewLabel;

/\*\*

\* Кнопка, яка видаляє каталог.

\*/

**private** JButton btnDeleteCatalog;

/\*\*

\* Кнопка, яка відображає таблицю клієнтів.

\*/

**private** JButton btnClients;

/\*\*

\* Кнопка, яка відкриває детальний перегляд товару.

\*/

**private** JButton btnOpenGoods;

/\*\*

\* Кнопка, яка відкриває вікно для створення каталогу.

\*/

**private** JButton btnCreateCatalog;

/\*\*

\* Кнопка, яка відкриває вікно для створення товару.

\*/

**private** JButton btnCreateGood;

/\*\*

\* Розділяє таблицю і кнопки лінією для кращого візуального ефекту.

\*/

**private** JSeparator separator;

/\*\*

\* Повертає таблицю в корінь каталогу.

\*/

**private** JButton btnNewButton;

/\*\*

\* Панель в яку поміщають таблицю для кращоговізуального ефекту.

\*/

**private** JScrollPane scrollPane;

/\*\*

\* Посилання на самого себе. Використовується при створенні команд

\*/

**private** MainFrame link = **this**;

/\*\*

\* Мітка, яка відображає назву вузла, вміст якого відображається

\*/

**private** JLabel lblTitle;

/\*\*

\* Видаляє товар

\*/

**private** JButton btnDeleteGoods;

/\*\*

\* Відкриває список клієнтів.

\*/

**private** JButton btnOpenClient;

/\*\*

\* Пункт меню, який відповідає англійській мові

\*/

**private** JRadioButtonMenuItem rdbtnmntmEnglish;

/\*\*

\* Пункт меню, який відповідає українській мові

\*/

**private** JRadioButtonMenuItem rdbtnmntmUkrainian;

/\*\*

\* Пункт меню, який відповідає системному стилю

\*/

**private** JRadioButtonMenuItem rdbtnmntmSystem;

/\*\*

\* Пункт меню, який відповідає стандартному стилю

\*/

**private** JRadioButtonMenuItem rdbtnmntmStandart;

/\*\*

\* Пункт меню, який відповідає за стиль німб

\*/

**private** JRadioButtonMenuItem rdbtnmntmNimbus;

/\*\*

\* Таблиця, в якій зберігаються написи компонентів

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Меню допомоги

\*/

**private** JMenu mnHelp;

/\*\*

\* Компонент, на якому розміщюються всі пункти меню

\*/

**private** JMenuBar menuBar;

/\*\*

\* Меню налаштувань

\*/

**private** JMenu mnSettings;

/\*\*

\* Підменю мов

\*/

**private** JMenu mnLanguage;

/\*\*

\* Підменю стилів

\*/

**private** JMenu mnStyle;

/\*\*

\* Пункт меню, який відповідає за інформацію про програму

\*/

**private** JMenuItem mntmAbout;

/\*\*

\* Пункт меню, який відповідає за інформацію допомоги

\*/

**private** JMenuItem mntmHelp;

/\*\*

\* Конструктор, в якому створюється і налаштовується головне вікно прогами.

\*/

**public** MainFrame() {

setIcon();

setResizable(**false**);

**this**.context = ContextUtils.*loadContext*("MainFrame");

setTitle(context.get("title"));

setDefaultCloseOperation(JFrame.*DO\_NOTHING\_ON\_CLOSE*);

setBounds(100, 100, 900, 472);

initComponents();

allocateComponents();

mntmAbout.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

JOptionPane.*showMessageDialog*(link, context.get("AboutInform"),

context.get("mntmAbout"),

JOptionPane.*INFORMATION\_MESSAGE*);

}

});

mntmHelp.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

JOptionPane.*showMessageDialog*(link, context.get("HelpInform"),

context.get("mntmHelp"), JOptionPane.*WARNING\_MESSAGE*);

}

});

rdbtnmntmEnglish.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CompositeComponent compprop = AppConfiguration

.*getAppConfiguration*()

.getProperties("SelectedLanguage");

compprop.put("language", "EngLanguage");

rdbtnmntmUkrainian.setSelected(**false**);

context = ContextUtils.*loadContext*("MainFrame");

changeLanguage();

}

});

rdbtnmntmUkrainian.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CompositeComponent compprop = AppConfiguration

.*getAppConfiguration*()

.getProperties("SelectedLanguage");

compprop.put("language", "UkrLanguage");

rdbtnmntmEnglish.setSelected(**false**);

context = ContextUtils.*loadContext*("MainFrame");

changeLanguage();

}

});

rdbtnmntmSystem.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CompositeComponent ob = AppConfiguration.*getAppConfiguration*()

.getProperties("ProgramTheme");

ob.put("theme",

"com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

**try** {

rdbtnmntmNimbus.setSelected(**false**);

rdbtnmntmStandart.setSelected(**false**);

UIManager

.*setLookAndFeel*("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

SwingUtilities.*updateComponentTreeUI*(MainFrame.**this**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

rdbtnmntmNimbus.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CompositeComponent ob = AppConfiguration.*getAppConfiguration*()

.getProperties("ProgramTheme");

ob.put("theme",

"com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");

**try** {

rdbtnmntmSystem.setSelected(**false**);

rdbtnmntmStandart.setSelected(**false**);

UIManager

.*setLookAndFeel*("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");

SwingUtilities.*updateComponentTreeUI*(MainFrame.**this**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

rdbtnmntmStandart.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

CompositeComponent ob = AppConfiguration.*getAppConfiguration*()

.getProperties("ProgramTheme");

ob.put("theme", "javax.swing.plaf.metal.MetalLookAndFeel");

**try** {

rdbtnmntmNimbus.setSelected(**false**);

rdbtnmntmSystem.setSelected(**false**);

UIManager

.*setLookAndFeel*("javax.swing.plaf.metal.MetalLookAndFeel");

SwingUtilities.*updateComponentTreeUI*(MainFrame.**this**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

btnNewButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

**if** (!lblTitle.getText().equals(context.get("Root")))

runGetTable(context.get("Root"));

}

});

btnClients.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", link);

Controller.*getController*().execute(context,

CommandFactory.*GET\_LIST\_OF\_USERS*);

setButtonsEnabled(**false**, **false**, **false**, **false**, **false**, **true**,

**false**, **false**, **false**, **true**);

}

});

btnOpenGoods.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

OpenGoods ob = **new** OpenGoods(link);

**int** row = ((JTable) ((JViewport) scrollPane.getComponent(0))

.getComponent(0)).getSelectedRow();

String goodsName = (String) ((JTable) ((JViewport) scrollPane

.getComponent(0)).getComponent(0)).getValueAt(row, 0);

ContextForCommand context = **new** ContextForCommand();

context.put("name", goodsName);

context.put("proprietor", ob);

Controller.*getController*().execute(context,

CommandFactory.*OPEN\_GOODS*);

}

});

btnDeleteGoods.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

Object[] options = { context.get("Yes"), context.get("No") };

**int** result = JOptionPane.*showOptionDialog*(link,

context.get("DeleteQuestionGoods"),

context.get("Delete"), JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*, **null**, options, options[1]);

**if** (result == JOptionPane.*YES\_OPTION*) {

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", link);

**int** row = ((JTable) ((JViewport) scrollPane.getComponent(0))

.getComponent(0)).getSelectedRow();

context.put("name",

(String) ((JTable) ((JViewport) scrollPane

.getComponent(0)).getComponent(0))

.getValueAt(row, 0));

Controller.*getController*().execute(context,

CommandFactory.*DELETE\_GOODS*);

}

}

});

btnGoods.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

runGetTable(context.get("Root"));

setButtonsEnabled(**true**, **false**, **false**, **false**, **false**, **false**,

**true**, **false**, **false**, **false**);

}

});

btnCreateCatalog.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

**new** CreateCatalog(link);

}

});

btnCreateGood.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

**new** CreateGood(link);

}

});

btnEditCatalog.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

**new** EditCatalog(link);

}

});

btnDeleteCatalog.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

Object[] options = { context.get("Yes"), context.get("No") };

**int** confirm = JOptionPane.*showOptionDialog*(link,

context.get("DeleteQuestionCatalog"),

context.get("Delete"), JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*, **null**, options, options[1]);

**if** (confirm == JOptionPane.*YES\_OPTION*) {

ContextForCommand context = **new** ContextForCommand();

context.put("name", lblTitle.getText());

context.put("proprietor", link);

Controller.*getController*().execute(context,

CommandFactory.*DELETE\_CATALOG*);

}

}

});

btnOpenClient.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

User ob = **new** User();

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", ob);

**int** row = ((JTable) ((JViewport) scrollPane.getComponent(0))

.getComponent(0)).getSelectedRow();

String login = (String) ((JTable) ((JViewport) scrollPane

.getComponent(0)).getComponent(0)).getValueAt(row, 0);

context.put("login", login);

Controller.*getController*().execute(context,

CommandFactory.*GET\_USER\_INFO*);

}

});

**this**.addWindowListener(**new** WindowListener() {

@Override

**public** **void** windowOpened(WindowEvent arg0) {

}

@Override

**public** **void** windowIconified(WindowEvent arg0) {

}

@Override

**public** **void** windowDeiconified(WindowEvent arg0) {

}

@Override

**public** **void** windowDeactivated(WindowEvent arg0) {

}

@Override

**public** **void** windowClosing(WindowEvent event) {

Object[] options = { context.get("Yes"), context.get("No") };

**int** n = JOptionPane.*showOptionDialog*(event.getWindow(),

context.get("CloseQuestion"), context.get("Confirm"),

JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*, **null**, options, options[1]);

**if** (n == 0) {

AppConfiguration.*getAppConfiguration*().save();

**try** {

Controller.*getController*().dao.close();

} **catch** (Exception e) {

e.printStackTrace();

}

event.getWindow().setVisible(**false**);

System.*exit*(0);

}

}

@Override

**public** **void** windowClosed(WindowEvent arg0) {

}

@Override

**public** **void** windowActivated(WindowEvent arg0) {

}

});

setVisible(**true**);

}

/\*\*

\* Відбувається ініціалізація компонентів.

\*/

**private** **void** initComponents() {

menuBar = **new** JMenuBar();

mnSettings = **new** JMenu(context.get("mnSettings"));

mnLanguage = **new** JMenu(context.get("mnLanguage"));

rdbtnmntmEnglish = **new** JRadioButtonMenuItem(

context.get("rdbtnmntmEnglish"));

rdbtnmntmUkrainian = **new** JRadioButtonMenuItem(

context.get("rdbtnmntmUkrainian"));

mnStyle = **new** JMenu(context.get("mnStyle"));

rdbtnmntmStandart = **new** JRadioButtonMenuItem(

context.get("rdbtnmntmStandart"));

rdbtnmntmSystem = **new** JRadioButtonMenuItem(

context.get("rdbtnmntmSystem"));

rdbtnmntmNimbus = **new** JRadioButtonMenuItem(

context.get("rdbtnmntmNimbus"));

mnHelp = **new** JMenu(context.get("mnHelp"));

mntmAbout = **new** JMenuItem(context.get("mntmAbout"));

mntmHelp = **new** JMenuItem(context.get("mntmHelp"));

contentPane = **new** JPanel();

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

contentPane.setLayout(**null**);

scrollPane = **new** JScrollPane();

scrollPane.setBounds(0, 0, 670, 300);

separator = **new** JSeparator();

separator.setOrientation(SwingConstants.*VERTICAL*);

separator.setBounds(668, 0, 11, 415);

btnCreateCatalog = **new** JButton(context.get("btnCreateCatalog"));

btnCreateCatalog.setBounds(701, 11, 173, 23);

btnCreateGood = **new** JButton(context.get("btnCreateGood"));

btnCreateGood.setBounds(701, 251, 173, 23);

btnOpenGoods = **new** JButton(context.get("btnOpenGoods"));

btnOpenGoods.setBounds(701, 217, 173, 23);

btnClients = **new** JButton(context.get("btnClients"));

btnClients.setBounds(701, 113, 173, 23);

btnDeleteCatalog = **new** JButton(context.get("btnDeleteCatalog"));

btnDeleteCatalog.setBounds(701, 79, 173, 23);

btnEditCatalog = **new** JButton(context.get("btnEditCatalog"));

btnEditCatalog.setBounds(701, 45, 173, 23);

lblNewLabel = **new** JLabel("");

lblNewLabel.setBounds(10, 332, 479, 72);

btnGoods = **new** JButton(context.get("btnGoods"));

btnGoods.setEnabled(**false**);

btnGoods.setBounds(701, 183, 173, 23);

lblTitle = **new** JLabel(context.get("Root"));

lblTitle.setBounds(10, 307, 479, 23);

btnNewButton = **new** JButton(context.get("btnNewButton"));

btnNewButton.setBounds(701, 319, 173, 23);

btnDeleteGoods = **new** JButton(context.get("btnDeleteGoods"));

btnDeleteGoods.setBounds(701, 285, 173, 23);

btnOpenClient = **new** JButton(context.get("btnOpenClient"));

btnOpenClient.setBounds(701, 147, 173, 23);

}

/\*\*

\* Відбувається додавання до основного вікна.

\*/

**private** **void** allocateComponents() {

setJMenuBar(menuBar);

menuBar.add(mnSettings);

mnSettings.add(mnLanguage);

mnLanguage.add(rdbtnmntmEnglish);

mnLanguage.add(rdbtnmntmUkrainian);

mnSettings.add(mnStyle);

mnStyle.add(rdbtnmntmStandart);

mnStyle.add(rdbtnmntmSystem);

mnStyle.add(rdbtnmntmNimbus);

setSettings();

menuBar.add(mnHelp);

mnHelp.add(mntmAbout);

mnHelp.add(mntmHelp);

setContentPane(contentPane);

contentPane.add(scrollPane);

runGetTable(context.get("Root"));

contentPane.add(separator);

contentPane.add(btnCreateCatalog);

contentPane.add(btnCreateGood);

contentPane.add(btnOpenGoods);

contentPane.add(btnClients);

contentPane.add(btnDeleteCatalog);

contentPane.add(btnEditCatalog);

contentPane.add(lblNewLabel);

contentPane.add(btnGoods);

contentPane.add(lblTitle);

contentPane.add(btnNewButton);

contentPane.add(btnDeleteGoods);

contentPane.add(btnOpenClient);

}

/\*\*

\* Метод генерує оновлення таблиці для конкретного коріня.

\*

\* **@param** pane

\* панель, в якій знаходиться таблиця.

\* **@param** node

\* корінь, відносно якого відбувається оновлення.

\*/

**public** **void** runGetTable(String node) {

ContextForCommand context = **new** ContextForCommand();

context.put("node", node);

context.put("proprietor", link);

Controller.*getController*().execute(context,

CommandFactory.*GET\_LIST\_OF\_ITEM*);

}

/\*\*

\* **@return** мітку, яка відображає назву вузла, зміст якого відображається

\*/

**public** JLabel getTitleLabel() {

**return** lblTitle;

}

/\*\*

\* **@return** панель, в середині якої знаходиться таблиця

\*/

**public** JScrollPane getScrolPane() {

**return** scrollPane;

}

/\*\*

\* Встановлює стан кнопок головного вікна

\* **@param** createCatalog стан кнопки створення каталогу

\* **@param** editCatalog стан кнопки редагування каталогу

\* **@param** deleteCatalog стан кнопки видалення каталогу

\* **@param** createGoods стан кнопки створення товару

\* **@param** openGoods стан кнопки відкриття товару

\* **@param** goods стан кнопки, яка відкриває таблицю товарів

\* **@param** clients стан кнопки, яка відкриває таблицю клієнтів

\* **@param** getRoot стан кнопки повернення на головну

\* **@param** deleteGoods стан кнопки видалення товару

\* **@param** openClient стан кнопки детального перегляду клієнта

\*/

**public** **void** setButtonsEnabled(**boolean** createCatalog, **boolean** editCatalog,

**boolean** deleteCatalog, **boolean** createGoods, **boolean** openGoods,

**boolean** goods, **boolean** clients, **boolean** getRoot,

**boolean** deleteGoods, **boolean** openClient) {

btnCreateCatalog.setEnabled(createCatalog);

btnEditCatalog.setEnabled(editCatalog);

btnDeleteCatalog.setEnabled(deleteCatalog);

btnCreateGood.setEnabled(createGoods);

btnOpenGoods.setEnabled(openGoods);

btnGoods.setEnabled(goods);

btnClients.setEnabled(clients);

btnNewButton.setEnabled(getRoot);

btnDeleteGoods.setEnabled(deleteGoods);

btnOpenClient.setEnabled(openClient);

}

/\*\*

\* Метод оновлює таблицю товарів відносно головного вузла.

\*/

**public** **void** refresh() {

runGetTable(lblTitle.getText());

}

/\*\*

\* Метод оновлює таблицю товарів відносно конкретного вузла

\* **@param** point назва вузла

\*/

**public** **void** refresh(String point) {

runGetTable(point);

}

/\*\*

\* Метод встановлює повідомлення в головному вікні

\* **@param** title повідомлення, яке потрібно відобразити

\*/

**public** **void** setTipForComponent(String title) {

lblNewLabel.setText(title);

}

/\*\*

\* Метод налаштовує пункти меню згідно завантажених налаштувань

\*/

**private** **void** setSettings() {

CompositeComponent ob = AppConfiguration.*getAppConfiguration*()

.getProperties("ProgramTheme");

String language = AppConfiguration.*getAppConfiguration*()

.getProperties("SelectedLanguage").getProperty("language");

String theme = ob.getProperty("theme");

**switch** (language) {

**case** "UkrLanguage":

rdbtnmntmEnglish.setSelected(**false**);

rdbtnmntmUkrainian.setSelected(**true**);

**break**;

**case** "EngLanguage":

rdbtnmntmEnglish.setSelected(**true**);

rdbtnmntmUkrainian.setSelected(**false**);

**break**;

**default**:

**break**;

}

**switch** (theme) {

**case** "com.sun.java.swing.plaf.windows.WindowsLookAndFeel":

rdbtnmntmSystem.setSelected(**true**);

**break**;

**case** "com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel":

rdbtnmntmNimbus.setSelected(**true**);

**break**;

**case** "javax.swing.plaf.metal.MetalLookAndFeel":

rdbtnmntmStandart.setSelected(**true**);

**break**;

**default**:

**break**;

}

}

/\*\*

\* Метод змінює написи на компонентах головного вікна

\*/

**private** **void** changeLanguage() {

setTitle(context.get("title"));

btnCreateCatalog.setText(context.get("btnCreateCatalog"));

btnNewButton.setText(context.get("btnNewButton"));

lblTitle.setText(context.get("Root"));

runGetTable(context.get("Root"));

btnDeleteGoods.setText(context.get("btnDeleteGoods"));

btnOpenClient.setText(context.get("btnOpenClient"));

rdbtnmntmEnglish.setText(context.get("rdbtnmntmEnglish"));

rdbtnmntmUkrainian.setText(context.get("rdbtnmntmUkrainian"));

rdbtnmntmSystem.setText(context.get("rdbtnmntmSystem"));

rdbtnmntmStandart.setText(context.get("rdbtnmntmStandart"));

rdbtnmntmNimbus.setText(context.get("rdbtnmntmNimbus"));

mnHelp.setText(context.get("mnHelp"));

mnSettings.setText(context.get("mnSettings"));

mnLanguage.setText(context.get("mnLanguage"));

mnStyle.setText(context.get("mnStyle"));

mntmAbout.setText(context.get("mntmAbout"));

mntmHelp.setText(context.get("mntmHelp"));

btnGoods.setText(context.get("btnGoods"));

btnEditCatalog.setText(context.get("btnEditCatalog"));

lblNewLabel.setText(context.get("lblNewLabel"));

btnDeleteCatalog.setText(context.get("btnDeleteCatalog"));

btnClients.setText(context.get("btnClients"));

btnOpenGoods.setText(context.get("btnOpenGoods"));

btnCreateGood.setText(context.get("btnCreateGood"));

}

/\*\*

\* Встановлює піктограму на дане вікно

\*/

**private** **void** setIcon() {

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.FlowLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JTextPane;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.JScrollPane;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

**import** application.Model.Description;

**import** application.Model.Goods;

/\*\*

\* Клас реалізовує функціонал для відображення детальної інформації про товар.

\* **@author** Коноз Андрій

\*/

**public** **class** OpenGoods **extends** JDialog {

/\*\*

\* Панель, яка містить всі компоненти вікна.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Мітка, для відображення назви.

\*/

**private** JLabel lblNamefield;

/\*\*

\* Мітка для відображення виробника.

\*/

**private** JLabel lblManufacturedfield;

/\*\*

\* Мітка для відображення ціни.

\*/

**private** JLabel lblPrisefield;

/\*\*

\* Мітка для відображення кількості.

\*/

**private** JLabel lblCountfield;

/\*\*

\* Поле для відображення опису.

\*/

**private** JTextPane txtrDescriptionArea;

/\*\*

\* Поле для відображення коментарів.

\*/

**private** JTextPane txtrComments;

/\*\*

\* Поле для відображення популярності.

\*/

**private** JLabel lblPopularityfield;

/\*\*

\* Посилання на головне вікно.

\*/

**protected** MainFrame link;

/\*\*

\* Товар для відображення.

\*/

**private** Goods goods;

/\*\*

\* Опис товару для відображення.

\*/

**private** Description descr;

/\*\*

\* Тадлиця з написами компонентів.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* Конструктор налаштовує діалогове вікно.

\* **@param** link посилання наголовне вікно.

\*/

**public** OpenGoods(MainFrame link) {

setResizable(**false**);

context = ContextUtils.*loadContext*("OpenGoods");

setIcon();

**this**.link = link;

setTitle(context.get("title"));

setBounds(10, 10, 491, 697);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblName = **new** JLabel(context.get("lblName"));

lblName.setBounds(10, 11, 89, 14);

contentPanel.add(lblName);

lblNamefield = **new** JLabel("NameField");

lblNamefield.setBounds(103, 11, 350, 14);

contentPanel.add(lblNamefield);

JLabel lblManufactured = **new** JLabel(context.get("lblManufactured"));

lblManufactured.setBounds(10, 36, 89, 14);

contentPanel.add(lblManufactured);

lblManufacturedfield = **new** JLabel("ManufacturedField");

lblManufacturedfield.setBounds(103, 36, 350, 14);

contentPanel.add(lblManufacturedfield);

JLabel lblPrice = **new** JLabel(context.get("lblPrice"));

lblPrice.setBounds(10, 61, 89, 14);

contentPanel.add(lblPrice);

lblPrisefield = **new** JLabel("PriseField");

lblPrisefield.setBounds(103, 61, 350, 14);

contentPanel.add(lblPrisefield);

JLabel lblCount = **new** JLabel(context.get("lblCount"));

lblCount.setBounds(10, 86, 89, 14);

contentPanel.add(lblCount);

lblCountfield = **new** JLabel("CountField");

lblCountfield.setBounds(103, 86, 350, 14);

contentPanel.add(lblCountfield);

JLabel lblDescription = **new** JLabel(context.get("lblDescription"));

lblDescription.setBounds(10, 136, 89, 14);

contentPanel.add(lblDescription);

JScrollPane scrollPane = **new** JScrollPane();

scrollPane.setBorder(**null**);

scrollPane.setBounds(103, 136, 350, 200);

contentPanel.add(scrollPane);

txtrDescriptionArea = **new** JTextPane();

txtrDescriptionArea.setText("Description area");

txtrDescriptionArea.setEditable(**false**);

scrollPane.setViewportView(txtrDescriptionArea);

JLabel lblComments = **new** JLabel(context.get("lblComments"));

lblComments.setBounds(10, 350, 83, 14);

contentPanel.add(lblComments);

JScrollPane scrollPane\_1 = **new** JScrollPane();

scrollPane\_1.setBorder(**null**);

scrollPane\_1.setBounds(103, 347, 350, 260);

contentPanel.add(scrollPane\_1);

txtrComments = **new** JTextPane();

txtrComments.setText("Comments");

txtrComments.setContentType("text/html");

txtrComments.setEditable(**false**);

scrollPane\_1.setViewportView(txtrComments);

JButton btnDelete = **new** JButton(context.get("btnDelete"));

btnDelete.setBounds(356, 618, 97, 23);

contentPanel.add(btnDelete);

JButton btnEdit = **new** JButton(context.get("btnEdit"));

btnEdit.setBounds(249, 618, 97, 23);

contentPanel.add(btnEdit);

JLabel lblPopularity = **new** JLabel(context.get("lblPopularity"));

lblPopularity.setBounds(10, 111, 89, 14);

contentPanel.add(lblPopularity);

lblPopularityfield = **new** JLabel("PopularityField");

lblPopularityfield.setBounds(103, 111, 350, 14);

contentPanel.add(lblPopularityfield);

JPanel buttonPane = **new** JPanel();

buttonPane.setLayout(**new** FlowLayout(FlowLayout.*RIGHT*));

getContentPane().add(buttonPane, BorderLayout.*SOUTH*);

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

btnDelete.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

Object[] options = { context.get("Yes"), context.get("No")};

**int** result = JOptionPane.*showOptionDialog*(OpenGoods.**this**, context.get("DeleteQuestionGoods"), context.get("Delete"), JOptionPane.*YES\_NO\_OPTION*,

JOptionPane.*QUESTION\_MESSAGE*, **null**, options,

options[0]);

**if**(result == JOptionPane.*YES\_OPTION*){

ContextForCommand context = **new** ContextForCommand();

context.put("proprietor", OpenGoods.**this**.link);

context.put("name", lblNamefield.getText());

Controller.*getController*().execute(context, CommandFactory.*DELETE\_GOODS*);

OpenGoods.**this**.dispose();

}

}

});

btnEdit.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

**new** EditGoods(OpenGoods.**this**.link, OpenGoods.**this**.goods, OpenGoods.**this**.descr);

OpenGoods.**this**.dispose();

}

});

}

/\*\*

\* Метод встановлює інформацію для відображення.

\* **@param** goods товар для відображення.

\* **@param** descr опис для відображення.

\* **@param** comments коментарі.

\*/

**public** **void** setInformation(Goods goods, Description descr, String comments){

**this**.goods = goods;

**this**.descr = descr;

lblNamefield.setText(goods.getName());

lblManufacturedfield.setText(descr.getMaker());

lblCountfield.setText(Integer.*toString*(goods.getNumberOfgoods()));

lblPrisefield.setText(Integer.*toString*(goods.getPrice()));

lblPopularityfield.setText(Integer.*toString*(goods.getMark()) +"/10");

txtrDescriptionArea.setText(descr.getText());

txtrComments.setText(comments);

**if**(comments.length() == 0)

txtrComments.setBackground(**this**.getBackground());

}

/\*\*

\* Метод встановлює піктограму на вікно

\*/

**private** **void** setIcon(){

ImageIcon icon = **new** ImageIcon("D:\\DatabaseClient\\icon.png");

setIconImage(icon.getImage());

}

}

**package** application;

**import** java.awt.BorderLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Hashtable;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JDialog;

**import** javax.swing.JPanel;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.SwingConstants;

**import** controller.ContextForCommand;

**import** controller.Controller;

**import** controller.Factory.CommandFactory;

**import** application.Model.ContactInformation;

/\*\*

\* Клас реалізовує діалогове вікно, в якому буде відображатися

\* інформація про користувача.

\* **@author** Коноз Андрій

\*/

**public** **class** User **extends** JDialog {

/\*\*

\* основна панель діалогового вікна, в якій будуть розміщуватися всі інші

\* елементи.

\*/

**private** **final** JPanel contentPanel = **new** JPanel();

/\*\*

\* Поле для відображення логіну.

\*/

**private** JLabel lblLoginlabel;

/\*\*

\* Поле для відображення категорії.

\*/

**private** JLabel lblCategorylabel;

/\*\*

\* Поле для відображення телефону.

\*/

**private** JLabel lblTelephonelabel;

/\*\*

\* Поле для відображення електронної пошти.

\*/

**private** JLabel lblEmaillable;

/\*\*

\* Поле для відображення адреси.

\*/

**private** JLabel lblAddresslable;

/\*\*

\* Поле для відображення стану.

\*/

**private** JLabel lblStatelabel;

/\*\*

\* Кнопка для розблокування користувача.

\*/

**private** JButton btnUnblockUser;

/\*\*

\* Кнопка для блокування користувача.

\*/

**private** JButton btnBlockUser;

/\*\*

\* Таблиця з написани на компонентах.

\*/

**private** Hashtable<String, String> context;

/\*\*

\* В конструкторі відбувається створення діалогового вікна і налаштування

\* його параметрів.

\*/

**public** User() {

context = ContextUtils.*loadContext*("User");

setIcon();

setTitle(context.get("title"));

setBounds(100, 100, 450, 263);

getContentPane().setLayout(**new** BorderLayout());

contentPanel.setBorder(**new** EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.*CENTER*);

contentPanel.setLayout(**null**);

JLabel lblLogin = **new** JLabel(context.get("lblLogin"));

lblLogin.setBounds(10, 11, 105, 14);

contentPanel.add(lblLogin);

JLabel lblCategory = **new** JLabel(context.get("lblCategory"));

lblCategory.setBounds(10, 34, 105, 14);

contentPanel.add(lblCategory);

JLabel lblContactinformation = **new** JLabel(context.get("lblContactinformation"));

lblContactinformation.setVerticalAlignment(SwingConstants.*TOP*);

lblContactinformation.setHorizontalAlignment(SwingConstants.*LEFT*);

lblContactinformation.setBounds(10, 59, 414, 14);

contentPanel.add(lblContactinformation);

btnBlockUser = **new** JButton(context.get("btnBlockUser"));

btnBlockUser.setBounds(168, 182, 121, 23);

contentPanel.add(btnBlockUser);

btnUnblockUser = **new** JButton(context.get("btnUnblockUser"));

btnUnblockUser.setBounds(37, 182, 121, 23);

contentPanel.add(btnUnblockUser);

lblLoginlabel = **new** JLabel("LoginLabel");

lblLoginlabel.setBounds(125, 11, 299, 14);

contentPanel.add(lblLoginlabel);

lblCategorylabel = **new** JLabel("CategoryLabel");

lblCategorylabel.setBounds(125, 34, 299, 14);

contentPanel.add(lblCategorylabel);

JLabel lblTelephone = **new** JLabel(context.get("lblTelephone"));

lblTelephone.setBounds(37, 84, 78, 14);

contentPanel.add(lblTelephone);

lblTelephonelabel = **new** JLabel("TelephoneLabel");

lblTelephonelabel.setBounds(125, 84, 299, 14);

contentPanel.add(lblTelephonelabel);

JLabel lblEmail = **new** JLabel(context.get("lblEmail"));

lblEmail.setBounds(37, 109, 78, 14);

contentPanel.add(lblEmail);

lblEmaillable = **new** JLabel("EmailLable");

lblEmaillable.setBounds(125, 109, 299, 14);

contentPanel.add(lblEmaillable);

JLabel lblAddress = **new** JLabel(context.get("lblAddress"));

lblAddress.setBounds(37, 132, 78, 14);

contentPanel.add(lblAddress);

lblAddresslable = **new** JLabel("AddressLable");

lblAddresslable.setBounds(125, 134, 299, 14);

contentPanel.add(lblAddresslable);

JLabel lblState = **new** JLabel(context.get("lblState"));

lblState.setBounds(37, 157, 78, 14);

contentPanel.add(lblState);

lblStatelabel = **new** JLabel("StateLabel");

lblStatelabel.setBounds(125, 157, 299, 14);

contentPanel.add(lblStatelabel);

setDefaultCloseOperation(JDialog.*DISPOSE\_ON\_CLOSE*);

setVisible(**true**);

btnBlockUser.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

ContextForCommand context = **new** ContextForCommand();

context.put("login", lblLoginlabel.getText());

context.put("state", 3);

context.put("proprietor", User.**this**);

Controller.*getController*().execute(context, CommandFactory.*CHANGE\_STATE*);

}

});

btnUnblockUser.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

ContextForCommand context = **new** ContextForCommand();

context.put("login", lblLoginlabel.getText());

context.put("state", 1);

context.put("proprietor", User.**this**);

Controller.*getController*().execute(context, CommandFactory.*CHANGE\_STATE*);

}

});

}

/\*\*

\* Метод для встановлення інформації для відображення.

\* **@param** login логін

\* **@param** category категорія

\* **@param** state стан

\* **@param** contInfo контактна інформація

\*/

**public** **void** setInformation(String login, String category, **int** state, ContactInformation contInfo){

lblLoginlabel.setText(login);

lblCategorylabel.setText(category);

lblAddresslable.setText(contInfo.getAdress());

lblEmaillable.setText(contInfo.getEmail());

lblTelephonelabel.setText(Integer.*toString*(contInfo.getTelepfone()));

lblStatelabel.setText(getState(state));

**if**(!category.equals(context.get("Client"))){

btnBlockUser.setEnabled(**false**);

btnUnblockUser.setEnabled(**false**);

}

}

/\*\*

\* Метод для отримання назви стану за переданою цифрою.

\* **@param** id числовий індентифікатор стану

\* **@return** назву стану

\*/

**private** String getState(**int** id){

String result = "null";

**switch** (id) {

**case** 1:

result = context.get("Active");

btnUnblockUser.setEnabled(**false**);

btnBlockUser.setEnabled(**true**);

**break**;

**case** 2:

result = context.get("NonActive");

btnBlockUser.setEnabled(**false**);

btnUnblockUser.setEnabled(**false**);

**break**;

**case** 3:

result = context.get("Banned");

btnBlockUser.setEnabled(**false**);

btnUnblockUser.setEnabled(**true**);

**break**;

}

**return** result;

}

ДОДАТОК Б. ТАБЛИЦІ ВІДПОВІДНОСТІ.

Табл. Б.1 Відповідність елементів вікна авторизації до класів

|  |  |
| --- | --- |
| Кнопка «Ввійти» | javax.swing.JButton |
| Кнопка «Забули пароль?» | javax.swing.JButton |
| Інформаційна зона | javax.swing.JLabel |
| Напис «Логін:» | javax.swing.JLabel |
| Напис «Пароль:» | javax.swing.JLabel |
| Поле для введення логіну | javax.swing.JTextField |
| Поле для введення паролю | javax.swing.JPasswordField |
| Вікно авторизації | javax.swing.JFrame |

Табл. Б.2 Відповідність елементів головного вікна до класів.

|  |  |
| --- | --- |
| Головне вікно | javax.swing.JFrame |
| Роздільник | javax.swing.JSeparator |
| Поле назви каталогу | javax.swing.JLabel |
| Інформаційне поле | javax.swing.JLabel |
| Зона з бігунками | javax.swing.JScrollPane |
| Таблиця відображення інформації | javax.swing.JTable |
| Кнопка «Створити каталог» | javax.swing.JButton |
| Кнопка «Перейменувати каталог» | javax.swing.JButton |
| Кнопка «Видалити каталог» | javax.swing.JButton |
| Кнопка «Користувачі» | javax.swing.JButton |
| Кнопка «Товари» | javax.swing.JButton |
| Кнопка «Відкрити товар» | javax.swing.JButton |
| Кнопка «Створити товар» | javax.swing.JButton |
| Кнопка «Видалити товар» | javax.swing.JButton |
| Кнопка «На головну» | javax.swing.JButton |
| Кнопка «Детальна інформація» | javax.swing.JButton |
| Панель меню | javax.swing.JMenuBar |
| Меню «Довідка» | javax.swing.JMenu |
| Меню «Налаштування» | javax.swing.JMenu |
| Меню «Мова» | javax.swing.JMenu |
| Меню «Стиль» | javax.swing.JMenu |
| Пункт «Про програму» | javax.swing.JMenuItem |
| Пункт «Допомога» | javax.swing.JMenuItem |
| Пункт «Англійська» | javax.swing.JMenuItem |
| Пункт «Українська» | javax.swing.JMenuItem |
| Пункт «Стандартний» | javax.swing.JMenuItem |
| Пункт «Системний» | javax.swing.JMenuItem |
| Пункт «Німб» | javax.swing.JMenuItem |

Табл. Б.3 Відповідність елементів вікна створення каталогу до класів.

|  |  |
| --- | --- |
| Вікно створення каталогу | javax.swing.JDialog |
| Напис «Назва каталогу:» | javax.swing.JLabel |
| Поле для введення назви | javax.swing.JTextField |
| Кнопка «Створити» | javax.swing.JButton |
| Кнопка «Скасувати» | javax.swing.JButton |

Табл. Б.4 Відповідність елементів вікна створення товару до класів.

|  |  |
| --- | --- |
| Вікно створення товару | javax.swing.JDialog |
| Інформаційна зона | javax.swing.JLabel |
| Кнопка «Створити» | javax.swing.JButton |
| Кнопка «Скасувати» | javax.swing.JButton |
| Напис «Назва:» | javax.swing.JLabel |
| Напис «Ціна:» | javax.swing.JLabel |
| Напис «Виробник:» | javax.swing.JLabel |
| Напис «Кількість:» | javax.swing.JLabel |
| Напис «Опис:» | javax.swing.JLabel |
| Поле для введення назви | javax.swing.JTextField |
| Поле для введення ціни | javax.swing.JTextField |
| Поле для введення виробника | javax.swing.JTextField |
| Поле для введення кількості | javax.swing.JTextField |
| Поле для введення опису | javax.swing.JTextField |

Табл. Б.5 Відповідність елементів вікна перейменування каталогу до класів.

|  |  |
| --- | --- |
| Вікно перейменування каталогу | javax.swing.JDialog |
| Напис «Нова назва:» | javax.swing.JLabel |
| Поле для введення назви | javax.swing.JTextField |
| Кнопка «Перейменувати» | javax.swing.JButton |
| Кнопка «Скасувати» | javax.swing.JButton |

Табл. Б.6 Відповідність елементів вікна редагування товару до класів.

|  |  |
| --- | --- |
| Вікно редагування товару | javax.swing.JDialog |
| Інформаційна зона | javax.swing.JLabel |
| Кнопка «Редагувати» | javax.swing.JButton |
| Кнопка «Скасувати» | javax.swing.JButton |
| Напис «Назва:» | javax.swing.JLabel |
| Напис «Ціна:» | javax.swing.JLabel |
| Напис «Виробник:» | javax.swing.JLabel |
| Напис «Кількість:» | javax.swing.JLabel |
| Напис «Опис:» | javax.swing.JLabel |
| Поле для введення назви | javax.swing.JTextField |
| Поле для введення ціни | javax.swing.JTextField |
| Поле для введення виробника | javax.swing.JTextField |
| Поле для введення кількості | javax.swing.JTextField |
| Поле для введення опису | javax.swing.JTextField |
| Поле для введення популярності | javax.swing.JTextField |

Табл. Б.7 Відповідність елементів вікна перегляду товару до класів.

|  |  |
| --- | --- |
| Вікно перегляду товару | javax.swing.JDialog |
| Поле відображення коментарів | javax.swing.JTextPane |
| Кнопка «Редагувати» | javax.swing.JButton |
| Кнопка «Видалити» | javax.swing.JButton |
| Напис «Назва:» | javax.swing.JLabel |
| Напис «Ціна:» | javax.swing.JLabel |
| Напис «Виробник:» | javax.swing.JLabel |
| Напис «Кількість:» | javax.swing.JLabel |
| Напис «Опис:» | javax.swing.JLabel |
| Поле відображення назви | javax.swing.JLabel |
| Поле відображення ціни | javax.swing.JLabel |
| Поле відображення виробника | javax.swing.JLabel |
| Поле відображення кількості | javax.swing.JLabel |
| Поле відображення опису | javax.swing.JTextPane |
| Поле відображення популярності | javax.swing.JLabel |

Табл. Б.8 Відповідність елементів вікна перегляду користувача до класів.

|  |  |
| --- | --- |
| Вікно перегляду користувача | javax.swing.JDialog |
| Кнопка «Розблокувати» | javax.swing.JButton |
| Кнопка «Заблокувати» | javax.swing.JButton |
| Поле логіну | javax.swing.JLabel |
| Поле категорії | javax.swing.JLabel |
| Поле контактної інформації | javax.swing.JLabel |
| Поле телефону | javax.swing.JLabel |
| Поле електронної пошти | javax.swing.JLabel |
| Поле адреси | javax.swing.JLabel |
| Поле стану | javax.swing.JLabel |

ДАТАТОК Г. ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА.

Користувач на початку роботи програм и повинен авторизуватися, заповнивши форми «Логін», «Пароль» і натиснувши кнопку «Ввійти».

Якщо користувач забув логін або пароль то він може дізнатися про необхідні дії натиснувши на кнопці «Забули пароль?».

Після авторизації користувач може виконувати наступні дії:

1). Переглядати каталоги.

2). Створювати каталоги.

3). Редагувати вже існуючі каталоги.

4). Видаляти каталоги.

5). Переглядати товари.

6) Редагувати товари.

7)Видаляти товари.

8) Створювати нові товари.

9)Переглядати список користувачів.

10). Переглядати конкретного користувача.

11). Блокувати користувача.

12). Розблоковувати користувача.

13). Змінювати мову інтерфейсу.

14). Змінювати стиль інтерфейсу.

Ці дії користувач може виконати наступним чином:

1. Для перегляду змісту конкретного каталогу користувач може натиснути в таблиці на рядок з потрібним каталогом. Після цієї операції користувач може побачити зміст обраного каталогу.
2. Для створення нового каталогу користувач повинен натиснути на кнопку «Створити каталог», що розташована в головному меню програми. Після цього з’явиться діалогове вікно, в якому потрібно буде вказати назву створюваного каталогу і назву каталога, в якому потрібно провести створення. Після цього користувач повинен натиснути на кнопку «Ок». Програма створить каталог, якщо це можливо для каталога-кореня. В іншому разі користувач отримає повідомлення про неможливість створення каталогу в цій точці.
3. Для редагування користувач повинен натиснути на кнопці «Редагувати каталог» в головному меню програми. Після цієї дії з’явиться діалогове вікно, в якому користувач зможе змінити назву редагованого каталогу. Після введення нової назви, користувач повинен натиснути кнопку «Ок», щоб підтвердити зміни.
4. Користувач може видалити каталог натиснувши на кнопку «Видалити каталог», що знаходиться в головному меню програми. Після натиснення кнопки, у користувача буде запрошене підтвердження на видалення. Якщо користувач підтвердить свій вибір, то каталог разом з усіма товарами, які належать йому, буде видалений.
5. Користувач може детально переглядати товари, які його зацікавили, обравши таблиці потрібний товар і натиснувши кнопку в головному меню «Відкрити товар». Після цієї дії користувачу в окремому вікні буде представлена детальна інформація про даний товар.
6. Користувач може редагувати товар, натиснувши під час детального перегляду конкретного товару на кнопку «Редагувати». Після цієї дії користувачу буде запропоновано в окремому вікні ввести нову інформацію. Для підтвердження редагування користувач повинен після завершення внесення нової інформації натиснути кнопку «Ок».
7. Користувач може видалити конкретний товар, натиснувши на кнопку «Видалити» при детальному перегляді товару у вікні детального перегляду товару. Після цієї дії у користувача буде запрошене підтвердження видалення. Якщо користувач підтвердить видалення, товар буде видалено.
8. Користувач може створити новий товар натиснувши в головному меню на кнопку «Створити товар». Після цього користувач у окремому вікні повинен внести інформацію про створюваний товар і натиснути в тому вікні кнопку «Ок». Після натиснення цієї кнопки товар буде створено.
9. Користувач може переглядати список користувачів, натиснувши на кнопку «Користувачі». Результат буде відображено у вигляді таблиці в головному вікні програми. Користувач може взаємодіяти з результатом.
10. Користувач може переглядати конкретного користувача натиснувши в таблиці на рядок з інформацією в таблиці користувачів. Після цієї дії відкриється вікно з детальною інформацією про обраного користувача.
11. Користувач може заблокувати обраного користувача натиснувши у вікні детального перегляду інформації про користувача кнопку «Блокувати».
12. Користувач може розблокувати користувача, натиснувши у вікні детального перегляду інформації про користувача кнопку «Розблокувати».
13. Користувач може змінити мову інтерфейсу, натиснувши на меню «Налаштування» і обравши в підпункті «Мови» потрібну мову. Обрана мова буде встановлена після перезапуску програми.
14. Користувач може змінювати стиль інтерфейсу, натиснувши на меню «Налаштування» і обравши в підпункті «Стилі», потрібний стиль. Обраний стиль буде встановлений після перезапуску програми.

ДОДАТОК Д. UML-ДІАГРАМИ.

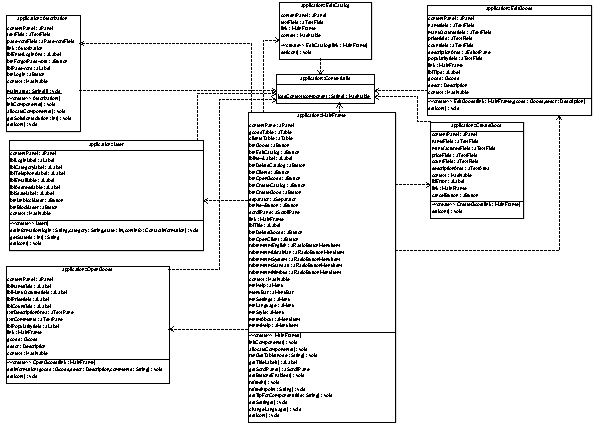


Рис. Д.1 UML-діаграма пакету application.

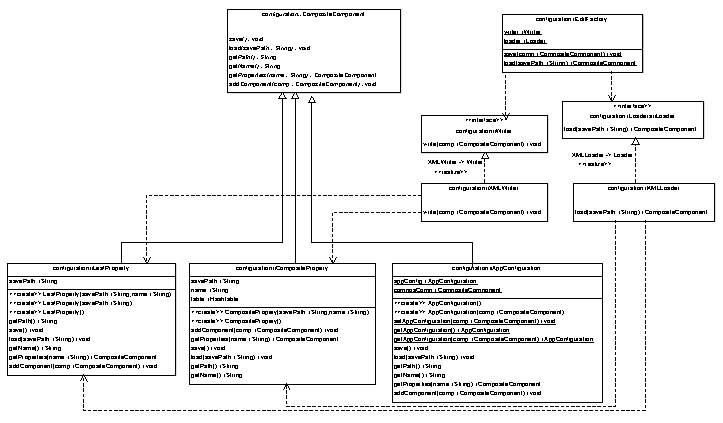


Рис. Д.2 UML-діаграма пакету configuration.

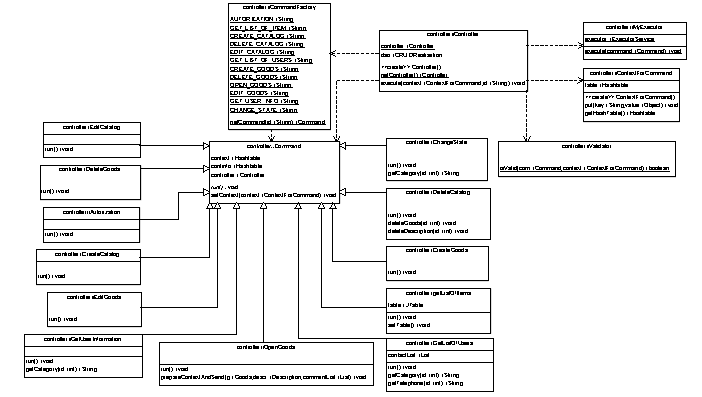


Рис. Д.3 UML-діаграма пакету controller.

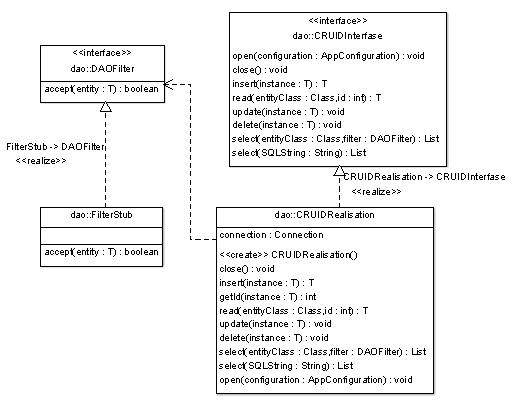


Рис. Д.4 UML-діаграма пакету dao.

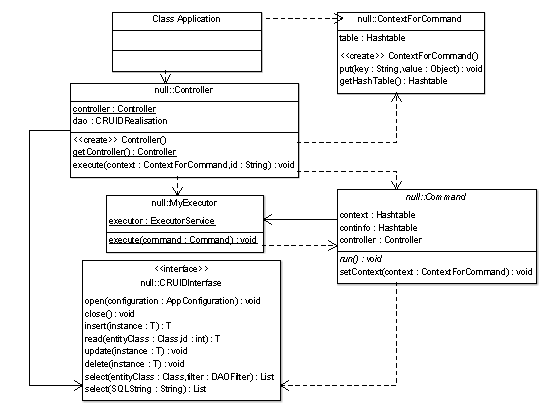


Рис. Д.5 UML-діаграма взаємодії пакетів.