НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

КУРСОВА РОБОТА

з інженерії програмного забезпечення
на тему: «Система замовлень послуг по розміщенню зовнішньої реклами. Інтерфейс менеджера»

Студента 2 курсу групи ІО-22
напряму підготовки 6.050102
«Комп'ютерна інженерія»
спеціальності 7.8.05010201
«Комп'ютерні системи та мережі»
Стася Д.О.
Керівник: к.т.н. доцент Абу-Усбах О.Н.

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

ЗМІСТ

ВСТУП

Метою курсової роботи є закріплення теоретичних знань і практичних навичок студентів з проектування, моделювання, розробки та тестування програмного забезпечення, здобутих у курсі інженерії програмного забезпечення, вивчення шаблону проектування MVC. Змістом курсової роботи є розробка програмного додатку з графічним інтерфейсом користувача «Система замовлень послуг по розміщенню зовнішньої реклами. Інтерфейс менеджера». Для виконання завдання я використав мову програмування Java, бібліотеку Swing, середовище розробки Eclipse.

1. ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Загальне завдання

Потрібно розробити інтерфейс менеджера для системи замовлень послуг по розміщенню зовнішньої реклами. Програмний додаток повинен бути реалізований за допомогою шаблону проектування MVC.

1.2 Огляд шаблону MVC

1.2.1 Загальні відомості

Model-view-controller (MVC) — це схема використання деяких шаблонів проектування, за допомогою яких модель даних додатку, інтерфейс користувача і взаємодія з користувачем розділені на три окремі компонента таким чином, щоб модифікація одного з компонентів мінімально впливала на інші. Ця схема проектування часто використовується для побудови архітектурного каркасу, коли переходять від теорії до реалізації в конкретній предметній області.

1.2.2 Призначення

Основна ціль використання шаблону MVC складається в відокремленні бізнес-логіки (моделі) від її візуалізації (представлення). Внаслідок цього збільшується можливість повторного використання. Найбільш корисно застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах. Тоді виконуються такі задачі:

1. До одної моделі можливо приєднати декілька видів, при цьому не змінюючи реалізацію моделі. Наприклад, деякі дані можуть бути представлені одночасно у вигляді електронної таблиці, гістограми і круговою діаграмою.

2. Не змінюючи реалізацію видів, можна змінити реакції на дії користувачів (натискання на кнопки, ввід даних), для цього достатньо використати інший контролер.
3. З'являється можливість розділення праці програмістів, коли програміст що займається бізнес-логікою, не повинен знати про те, яке представлення буде використовуватись.

1.2.3 Концепція MVC

Концепція MVC дозволяє розділити дані, представлення й обробку дій користувача на три окремих компонента (рис. 1.1):

- Модель. Модель надає знання: дані та методи роботи з цими даними, реагує на запити, змінюючи свій стан. Не містить інформації, як ці знання можливо візуалізувати.
- Представлення, вид. Відповідає за відтворення інформації (візуалізацію). Часто в якості представлення виступає форма (вікно) із графічними елементами.
- Контролер. Забезпечує зв'язок між користувачем та системою: контролює ввід даних користувачем і використовує модель і представлення для реалізації необхідної реакції.

Важливо відмітити, що як представлення, так і контролер залежать від моделі. Однак модель не залежить ні від представлення, ні від контролера. Цим осягається призначення такого відокремлення: воно дозволяє будувати модель незалежно від візуального представлення, а також створювати декілька різних представлень для однієї моделі. Для реалізації схеми MVC використовується достатньо велика кількість шаблонів проектування (в залежності від складності архітектурного рішення), основні з яких

«спостережник», «стратегія», «компонувальник», про яких можна дізнатися в праці [1].

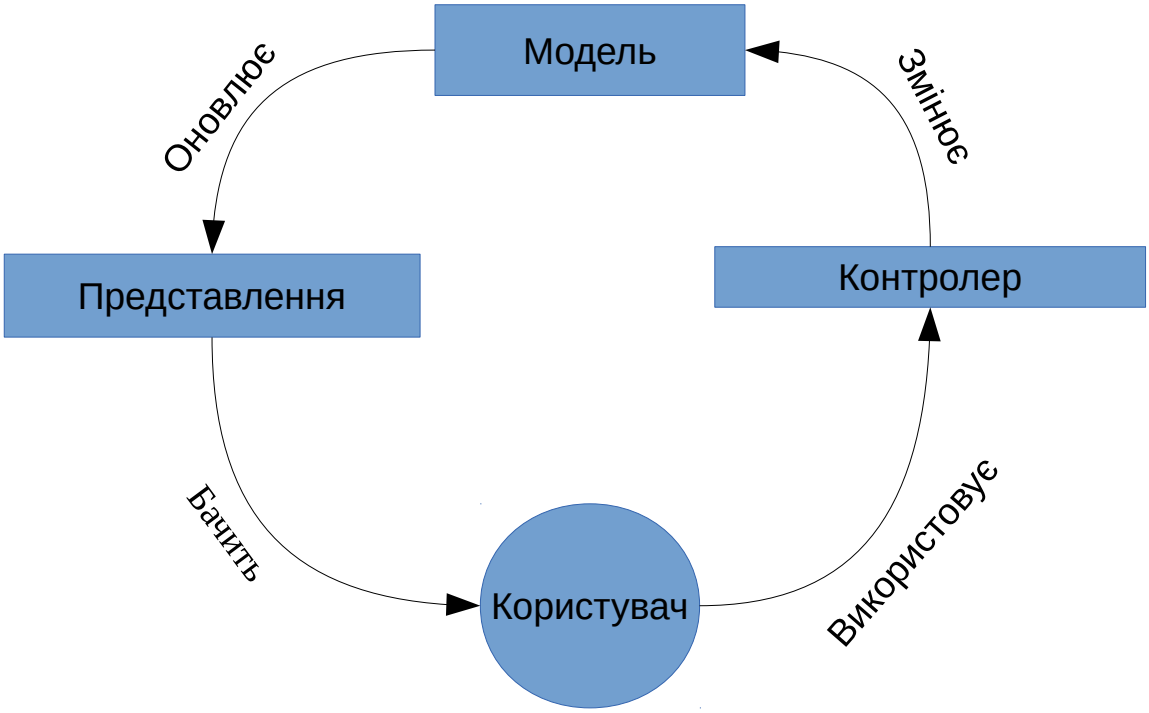


Рис. 1.1 Концепція MVC

Найбільш типова реалізація відокремлює вид від моделі шляхом встановлення між ними протоколу взаємодії, використовуючи апарат подій (підписка/повідомлення). При кожній зміні внутрішніх даних в моделі вона повідомляє всі представлення, які залежать від неї, і представлення оновлюється. Для цього використовується шаблон «спостережник». При обробці реакції користувача вид вибирає, в залежності от потрібної реакції, потрібний контролер, який забезпечує той чи інший зв'язок із моделлю. Для цього використовується шаблон «стратегія», або замість цього може бути модифікація з використанням шаблона «команда». А для можливості однотипного використання

компонентів складеного ієрархічного виду може використовуватись шаблон «компонувальник». Крім того, можуть використовуватись і інші шаблони проектування, наприклад, «фабричний метод», який дозволить задавати за замовчуванням тип контролера для відповідного виду.

У моїй програмі моделлю є класи ,які представляють таблиці бази даних, контролером є класи, які відповідають за обробку дій користувача, а видом — класи, які створюють графічний інтерфейс користувача.

2.3 Функціональність

Програма повинна мати наступну функціональність:

1. можливість перегляду інформації о клієнтах;
2. керування замовленнями;
3. можливість надавати й приймати документи від працівників фірми.

2.4 Вимоги до реалізації

1. Використання мови програмування Java;
2. Проект має бути повністю задокументований за допомогою JavaDoc;

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Прецеденти

Користувач має можливість зафіксувати виконану дію у базі даних, натиснувши на пункт “New action” в меню “Tools” (рис. 2.1).

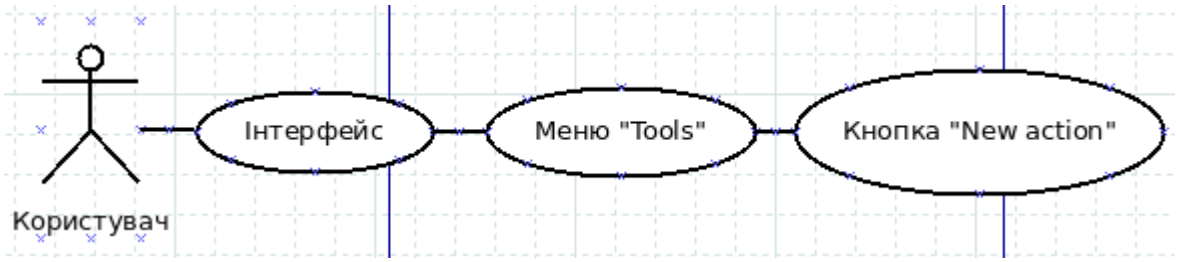


Рис. 2.1. Прецедент фіксування нової дії у БД

Користувач може отримати інформацію про користувача системи чи замовлення, виділивши відповідний елемент у дереві клієнтів чи робітників (рис. 2.2).

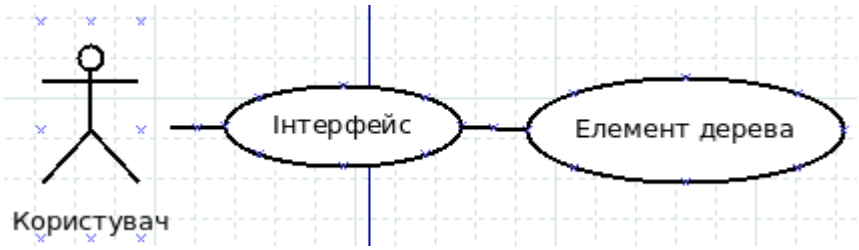


Рис.2.2. Прецедент отримання інформації про користувача чи замовлення

Користувач має можливість отримати інформацію про версію та автора програми, натиснувши на пункт “About” меню “Help” (рис. 2.3).

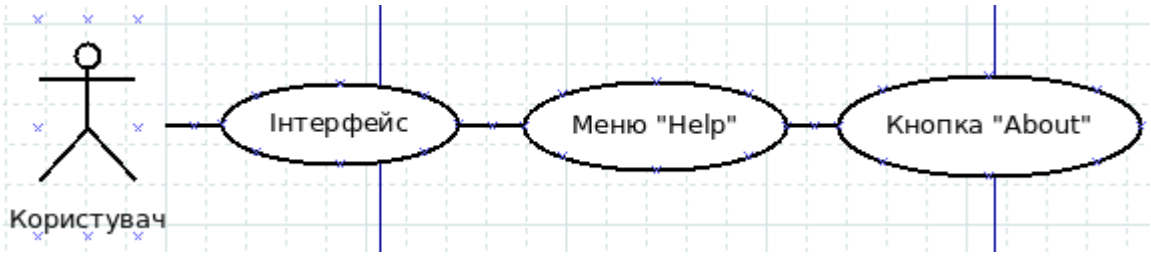


Рис. 2.3. Прецедент отримання загальної інформації про програму

Користувач може видаляти необхідні сутності з бази даних, натиснувши на пункт “Remove” контекстного меню відповідного елементу дерева робітників чи клієнтів (рис. 2.4).

Рис. 2.4. Прецедент видалення сутності з БД

Користувач має можливість змінювати налаштування додатку. Для цього йому потрібно натиснути на пункт “Settings” меню “Tools” і заповнити необхідні поля (Рис. 3.5).

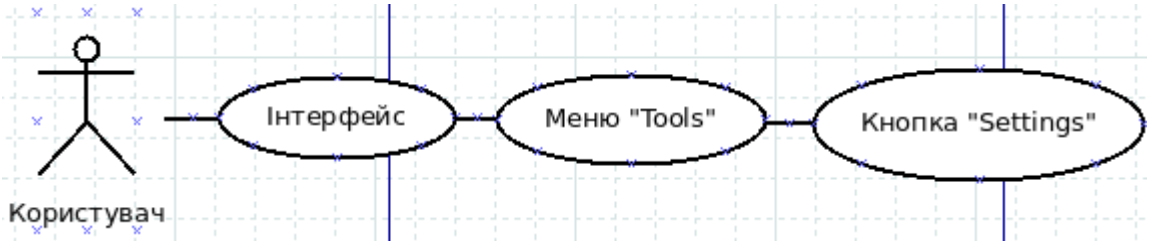


Рис. 2.5. Прецедент налаштування програми

2.2 Сценарії використання програми

Отримання інформації про клієнта

Передумова: програма запущена, користувач пройшов аутентифікацію.

1. Користувач переходить на вкладку “Клієнти”.
2. Програма надає список клієнтів відсортованих по категоріям.
3. Користувач виділяє певного клієнта.
4. Програма відображає інформацію про нього.

Фіксування нової дії у БД.

Передумова: програма запущена, користувач пройшов аутентифікацію.

1. Користувач натискає на меню “Tools”.
2. Програма відображає пункти меню “Tools”.
3. Користувач натискує на пункт меню “New action”.
4. Програма відображає діалогове вікно.
5. Користувач заповнює всі необхідні поля, натискає на кнопку “Choose files”.
6. Програма надає вікно для вибору файлу.
7. Користувач вибирає файли та натискає на кнопку “OK”.

Налаштування програми.

Передумова: програма запущена, користувач пройшов аутентифікацію.

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

Отримання загальної інформації про програму.

Передумова: програма запущена, користувач пройшов аутентифікацію.

1. Користувач натискає на меню “Help”.
2. Програма відображає пункти меню “Help”.

3. Користувач натискає на пункт “About”.
4. Програма відображає загальну інформацію про додаток.

2.3 Проектування структури бази даних

На рис. 2 зображена структура бази даних програмної системи. Для реалізації бази даних я використав СУБД MySQL.

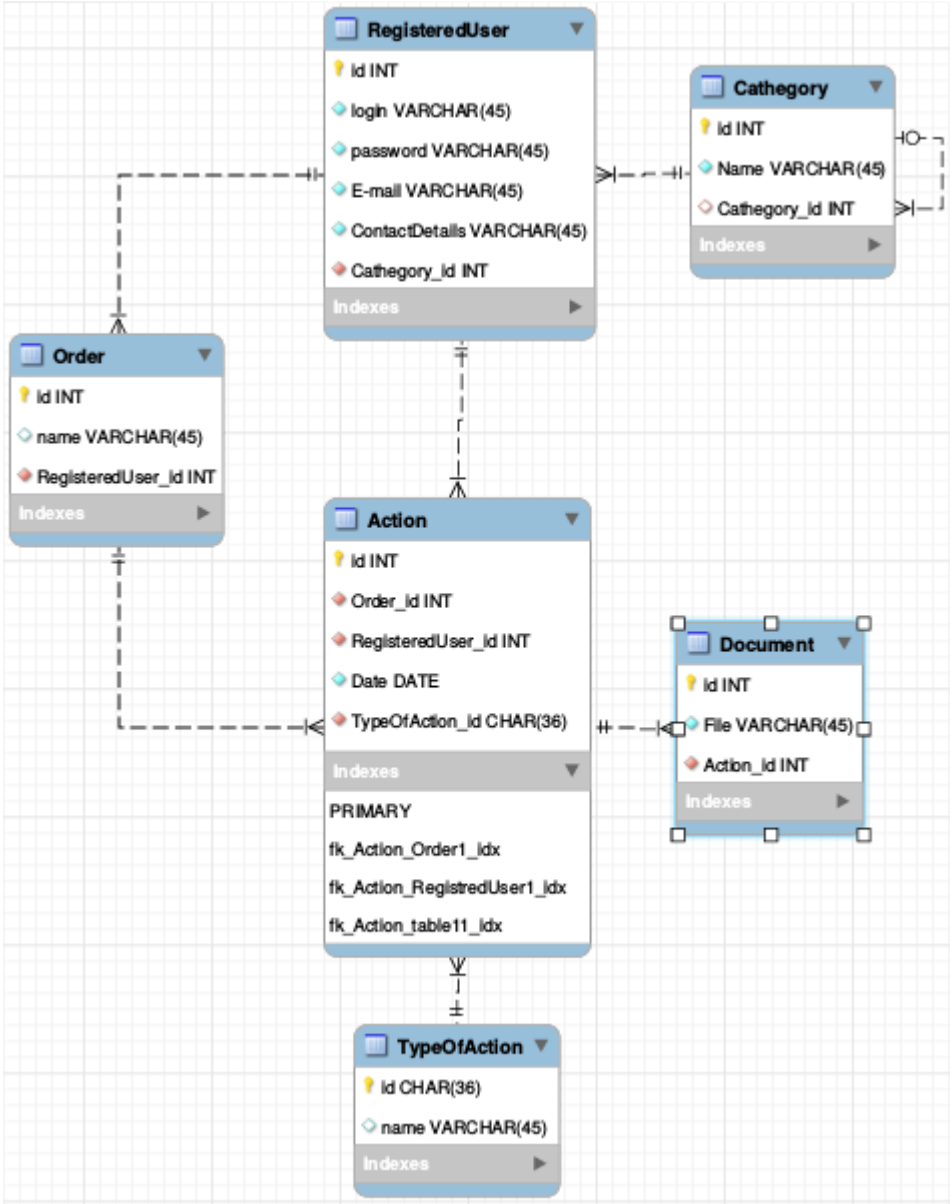


Рис.2.6 Структура бази даних

2.4 Проектування граничних класів додатку

На рис. 2.7 зображені діаграма граничних класів, які складають графічний інтерфейс користувача.

Структура граничних класів має саме такий вигляд для того, щоб можна було легко та швидко знайти необхідний елемент управління.

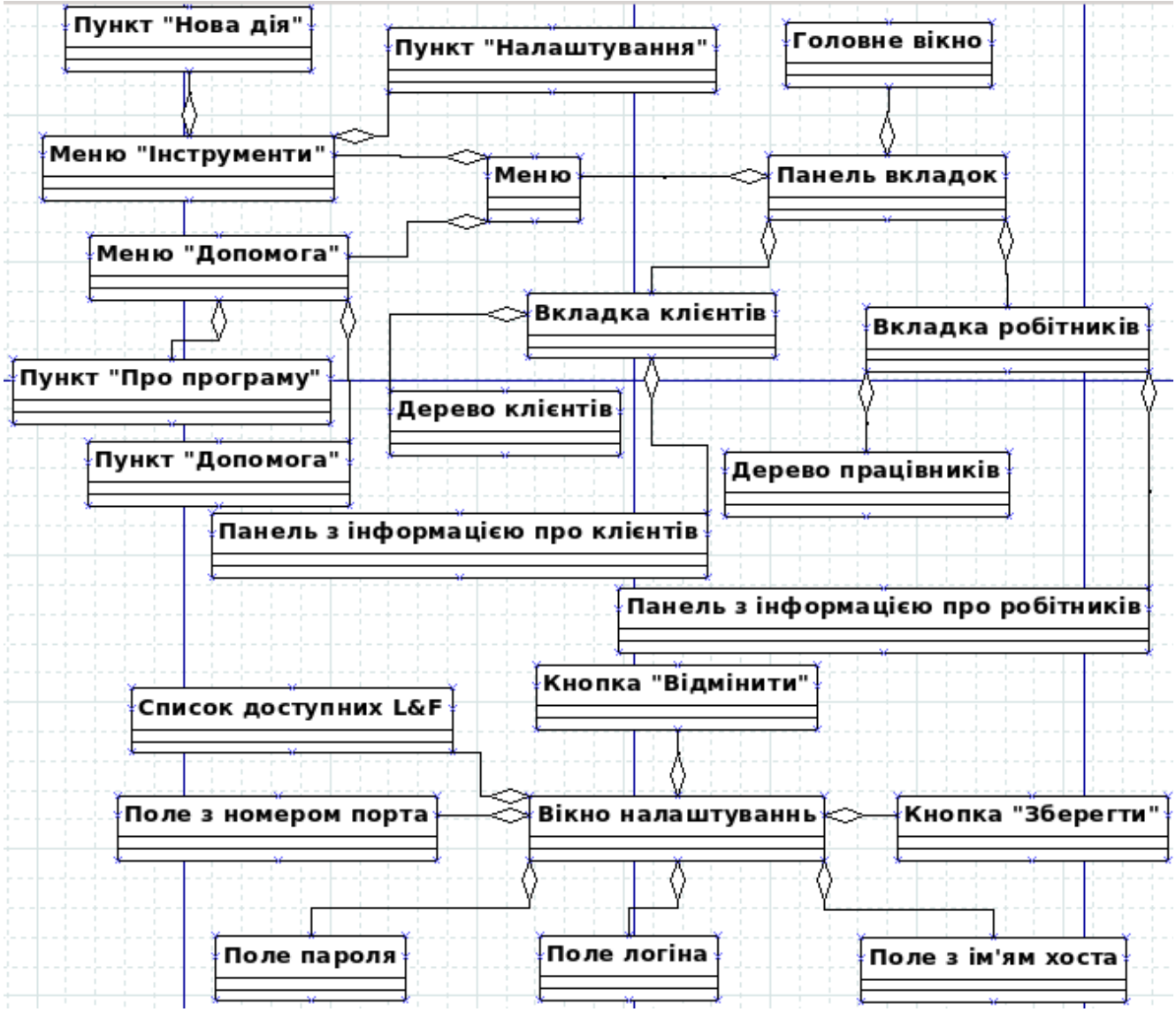


Рис.2.7. Діаграма граничних класів додатку

Таблиця1. Відповідність граничних класів класам Java

Граничний клас	Java клас
Головне вікно	javax.swing.JFrame
Панель вкладок	javax.swing.JTabedPane
Вкладка клієнтів	javax.swing.JPane
Вкладка робітників	javax.swing.JPane
Дерево клієнтів	javax.swing.JTree
Панель з інформацією про клієнтів	javax.swing.JPane
Дерево робітників	javax.swing.JTree
Панель з інформацією про робітників	javax.swing.JPane
Меню	javax.swing.JMenuBar
Вікно налаштувань	javax.swing.JDialog
Поле логіна	javax.swing.JTextField
Поле пароля	javax.swing.JPasswordField
Поле хоста БД	javax.swing.JTextField
Поле порта БД	javax.swing.JSpinner
Меню “Інструменти”	javax.swing.JMenu
Меню “Допомога”	javax.swing.JMenu
Кнопка “Зберегти”	javax.swing.JButton
Кнопка “Відмінити”	javax.swing.JButton
Пункт “Налаштування”	javax.swing.JMenuItem
Пункт “Допомога”	javax.swing.JMenuItem
Пункт “Нова дія”	javax.swing.JMenuItem
Пункт “Про програму”	javax.swing.JMenuItem
Вікно авторизації	javax.swing.JFrame

2.5 Проектування інтерфейсу користувача

На рис. 2.8 — 2.10 зображені ескізи графічного інтерфейсу користувача. Графічний інтерфейс розроблявся таким чином, щоб

користувач мав можливість швидко і легко знайти потрібну йому інформацію чи вибрати операцію.

На рис. 2.8 зображена вкладка клієнтів. Вона розділена на дві частини. Зліва розташоване дерево клієнтів, в якому клієнти для зручності розташовані по категоріям, а також зображені їхні дії пов'язані з замовленням з відповідними документами. На правій частині розташована область з текстовою інформацією про вбраний елемент дерева.

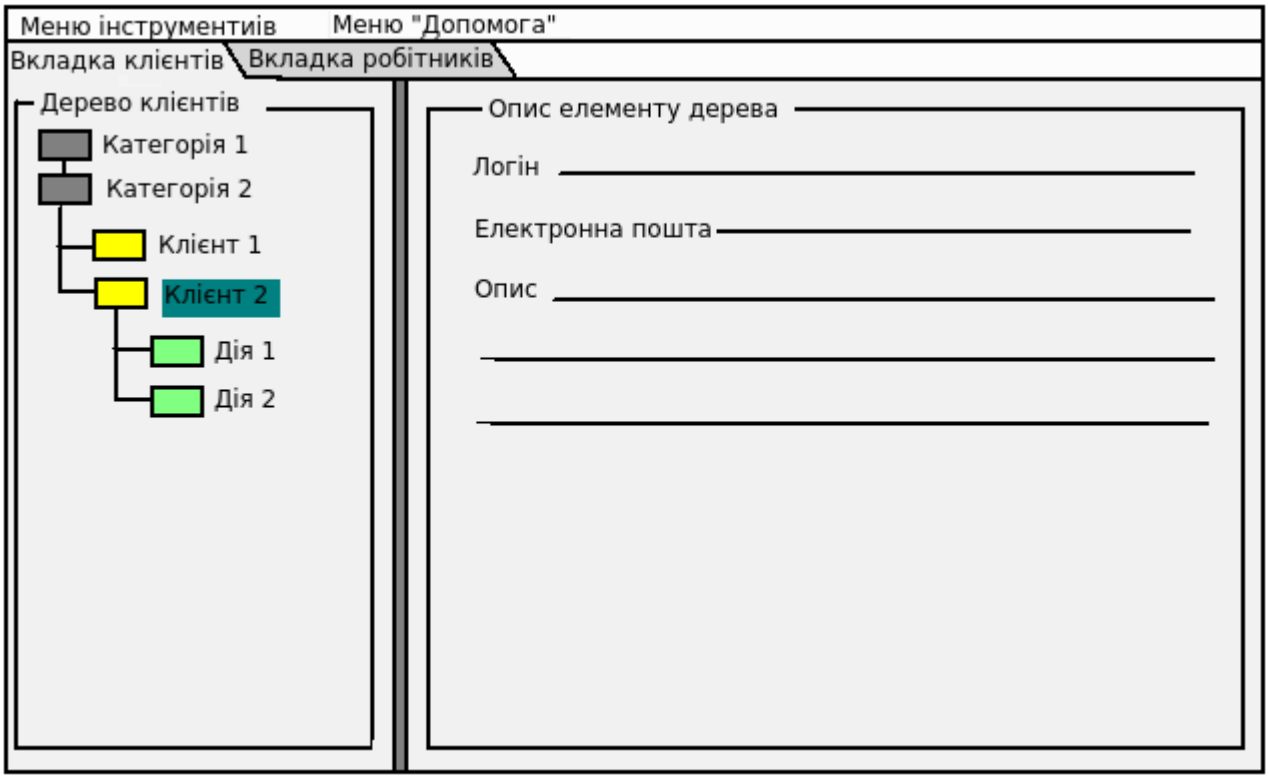


Рис. 2.8 Основне вікно. Вкладка клієнтів

На рис. 2.9 зображено вікно з відкритою вкладкою робітників. Вона розділена на дві частини, які можуть змінювати свій розмір. В лівій частині розташовано дерево робітників, в якому робітники для зручності розташовані по категоріям, а також зображені їхні дії пов'язані з реалізацією замовлення з прикладеними документами. В правій частині

розташована текстова інформація про вибраний елемент дерева.

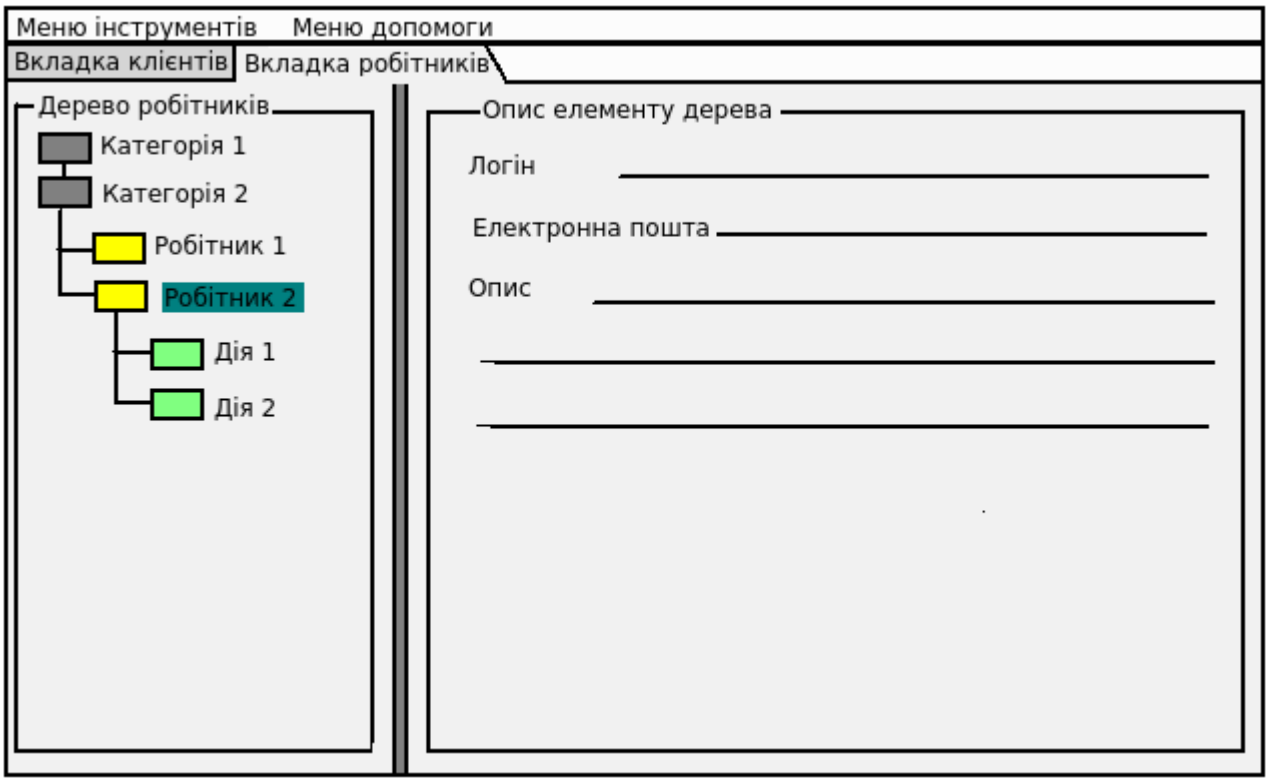


Рис. 2.9. Основне вікно. Вкладка працівників

На рис. 2.10 зображено вікно налаштувань. На ньому розташовані поля з необхідними параметрами додатку.

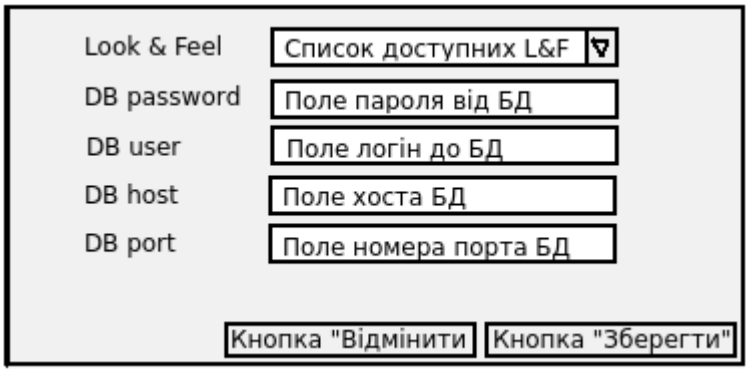


Рис. 2.10 Вікно налаштувань

3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1 Структура програмного додатку

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

3.2 Документація

dao.filters

Клас ActionsByOrderAndUser

public class **ActionsByOrderAndUser**

extends java.lang.Object

implements DAOFilter

Фільтр, що фільтрує Дії за Замовленням та Користувачем.

Перелік контсрукторів

Ім'я і опис
ActionsByOrderAndUser(java.util.UUID order, java.util.UUID regUser)

Перелік методів

Модифікатор і тип	Ім'я і опис
<T> boolean	**accept**(T entity) визначає, чи задовільняє сутність даній умові

Детальний опис конструкторів

ActionsByOrderAndUser

public ActionsByOrderAndUser(java.util.UUID order,

java.util.UUID regUser)

Детальний опис методів

accept - визначає, чи задовільняє сутність даній умові.

public <T> boolean accept(T entity)

Оголошений як:

accept у інтерфейсі DAOFilter

Повертає:

true, якщо сутність задовільняє даному крітерію.

configuration

Клас AppConfig

public class **AppConfig**

extends java.lang.Object

implements IConfig

Цей клас надає доступ до конфігурації додадку, надає методи до упраління нею.

Перелік методів

Модифікатор і тип	Ім'я і опис
static **AppConfig**	**getInstance**()
java.lang.String	**getProperty**(java.lang.String name)
java.util.Iterator<java.util.Map.Entry<java.lang.String,java.lang.String>>	**iterator**()
void	**load**() Зчитує кофігурацію із файлу.
void	**save**() Записує конфігурацію у файл.
void	**setProperty**(java.lang.String property, java.lang.String value)
void	**setReader**(**Reader** reader)
void	**setWriter**(**Writer** writer)

java.lang.String	**toString**()

Детальний опис методів

public void save()

throws javax.xml.stream.XMLStreamException,

java.io.IOException

Записує конфігурацію у файл.

Викидає:

javax.xml.stream.XMLStreamException

java.io.IOException

load

public void load()

throws javax.xml.parsers.ParserConfigurationException,

org.xml.sax.SAXException,

java.io.IOException

Зчитує кофігурацію із файлу.

Викидає:

javax.xml.parsers.ParserConfigurationException

org.xml.sax.SAXException

java.io.IOException

toString

public java.lang.String toString()

Перевизначає:

toString у класі java.lang.Object

iterator

public

java.util.Iterator<java.util.Map.Entry<java.lang.String,java.lang.String>>

iterator()

Оголошений як:

iterator у інтерфейсі

java.lang.Iterable<java.util.Map.Entry<java.lang.String,java.lang.String>>

dao

Interface BlobHandler

public interface **BlobHandler**

Клас який зберігає інформацію про бінарний файл, необхідну для його лінивого завантаження з бази даних у локальне сховище.

Перелік методів

Модифікатор і тип	Ім'я і опис
java.lang.String	**getField**()
java.io.File	**getFile**()
java.lang.String	**getFilename**()
java.util.UUID	**getId**()
java.lang.String	**getTable**()

Клас BlobHandlerFactory

public abstract class **BlobHandlerFactory**

extends java.lang.Object

Клас, який надає необхідний екземпляр BlobHandler відповідно до виду бази даних.

Перелік контсрукторів

Ім'я і опис
BlobHandlerFactory()

Перелік методів

Модифікатор і тип	Ім'я і опис
abstract **BlobHandler**	**createBlobHandler**(java.lang.String table, java.util.UUID id, java.lang.String field, java.lang.String filename)

	Створює об'єкт класу BlobHandler.
static **BlobHandlerFactory**	**getInstance**() - повертає екземпляр цього класу.

Детальний опис конструкторів

BlobHandlerFactory

public BlobHandlerFactory()

Детальний опис методів

createBlobHandler

public abstract BlobHandler createBlobHandler(java.lang.String table,
java.util.UUID id,
java.lang.String field,
java.lang.String filename)

Створює об'єкт класу BlobHandler.

Параметри:

table - таблиця

id - ідентифікатор

field - поле

filename — ім'я файлу

Повертає: об'єкт BlobHandler.

app.controller.annotation

Аннотація COMMAND

@Retention(value=RUNTIME)

@Target(value=TYPE)

public @interface **COMMAND**

Ця аннотація містить ключ спадкоємця класу Command.

Перелік необхідних елементів

Модифікатор і тип	Required Element and Description

java.lang.String	**key** — ключ команди.

app.controller.commands

Клас Command

public abstract class **Command**

extends java.lang.Object

implements java.lang.Runnable

Інтерфейс для класів команд, що виконує контролер.

Перелік контсрукторів

Ім'я і опис
Command()

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**addParameter**(java.lang.String key, java.lang.Object value) Додає параметри до команди.
void	**deleteParametr**(java.lang.String key) видаляє параметр по ключу.
Context	**getParameters**() Отримати парааметри.
void	**setParameters**(**Context** parameters) Змінює параметри команди.

Детальний опис конструкторів

Command

public Command()

Детальний опис методів

setParameters

public void setParameters(Context parameters)

Змінює параметри команди.

Параметри:

parameters - - параметры

getParameters

public Context getParameters()

Отримати парааметри.

Повертає:

- параметри команди.

addParameter

public void addParameter(java.lang.String key,
java.lang.Object value)

Додає параметри до команди.

Параметри:

key - - ключ параметра

value - - значення параметра.

deleteParametr

public void deleteParametr(java.lang.String key)

видаляє параметр по ключу.

Параметри:

key - - ключ параметра.

configuration

Клас ConfWriterFactory

public class **ConfWriterFactory**

extends java.lang.Object

Цей клас надає можливість отримати необхідний об'єкт спадкоємеця класу Writer, залежно від файлу конфіґурації.

Перелік контсрукторів

Ім'я і опис
ConfWriterFactory()

Перелік методів

Модифікатор і тип	Ім'я і опис
static **Writer**	**getWriter**(java.lang.String file)

Детальний опис конструкторів

ConfWriterFactory

public ConfWriterFactory()

Детальний опис методів

getWriter

public static Writer getWriter(java.lang.String file)

throws java.io.IOException

Викидає:

app.controller.annotation

Аннотація CONTEXT

@Retention(value=RUNTIME)

@Target(value=TYPE)

public @interface **CONTEXT**

Містить в собі список параметрів для команди (Command).

Перелік необхідних елементів

Модифікатор і тип	Required Element and Description
PARAMETER[]	**list** - список параметрів.

Element Detail

list — список параметрів.

app.controller.commands

app.controller

Клас Controller

public class **Controller**

extends java.lang.Object

Клас, що приймає і виконує команди.

Перелік методів

Модифікатор і тип	Ім'я і опис
boolean	**awaitTermination**(long timeout, java.util.concurrent.TimeUnit unit) Чекати завершення виконання команд певну кількість часу.
void	**executeCommand**(**Command** command) Виконує команду.
static **Controller**	**getInstance**()
void	**shutdown**() Завершити приймати команди.

Детальний опис методів

executeCommand

public void executeCommand(Command command)
throws IncorrectParameterListException

Виконує команду. Команди складаються у чергу і виконуються у окремому потоці.

Викидає:

IncorrectParameterListException

java.lang.InterruptedException

awaitTermination

public boolean awaitTermination(long timeout,
java.util.concurrent.TimeUnit unit)

throws java.lang.InterruptedException

Чекати завершення виконання команд певну кількість часу. Якщо час вийшов команди завершуються передчасно. Необхідно виконувати після виконання методу shutdown().

Параметри:

timeout - - кількість часу.

unit - - Одиниці виміру часу.

Повертає:

true ,якщо виконання команд завершилось за заданий час, і false, якщо команди завершились передчасно.

Викидає:

java.lang.InterruptedException

shutdown

public void shutdown()

Завершити приймати команди.

getInstance

public static Controller getInstance()

app

Клас CourseWork

public class **CourseWork**

extends java.lang.Object

Перелік контсрукторів

Ім'я і опис
CourseWork()

Перелік методів

Модифікатор і тип	Ім'я і опис
static void	**main**(java.lang.String[] args)

	Точка входу в програму.

Детальний опис конструкторів

CourseWork

public CourseWork()

Детальний опис методів

main

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

throws java.io.IOException,

javax.xml.parsers.ParserConfigurationException,

org.xml.sax.SAXException,

IncorrectParameterListException,

DAOException,

java.sql.SQLException,

java.lang.InterruptedException,

java.lang.ClassNotFoundException,

java.lang.InstantiationException,

java.lang.IllegalAccessException,

javax.swing.UnsupportedLookAndFeelException

Точка входу в програму.

Параметри:

args — аргументи командної строки.

Викидає:

java.io.IOException

javax.xml.parsers.ParserConfigurationException

org.xml.sax.SAXException

IncorrectParameterListException

DAOException

java.sql.SQLException

java.lang.InterruptedException

java.lang.ClassNotFoundException

java.lang.InstantiationException

java.lang.IllegalAccessException

javax.swing.UnsupportedLookAndFeelException

dao

public interface **CRUDInterface**

Інтерфес, для доступу і маніпулювання базою данних.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**close**() Закриває з'єднання з базою даних
<T> void	**delete**(T instance) Видалити об'єкт-сутність із БД.
<T> T	**insert**(T instance) Зберігає об'єкт-сутність у БД.
void	**open**() Відкриває з'єднання з базою даних
<T> T	**read**(java.lang.Class entityClass, java.util.UUID id) Зчитує об'экт певного класу-сутності з БД з певним id.
<T> java.util.List<T>	**select**(java.lang.Class entityClass, **DAOFilter** filter) Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.
<T> java.util.List<T>	**select**(java.lang.String SQLString, java.lang.Class<T> intitiesClass)

	Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.
<T> void	**update**(T instance) Поновлює об'єкт-сутність у БД.

Детальний опис методів

open

void open()

throws DAOException

Відкриває з'єднання з базою даних

Викидає:

DAOException

close

void close()

throws DAOException

Закриває з'єднання з базою даних

Викидає:

DAOException

insert

<T> T insert(T instance)

throws DAOException

Зберігає об'єкт-сутність у БД.

Параметри:

instance -

Повертає:

Викидає:

DAOException

read

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

java.util.UUID id)
throws DAOException

Зчитує об'экт певного класу-сутності з БД з певним id.

Параметри:

entityClass -

id -

Повертає:

Викидає:

DAOException

update

<T> void update(T instance)
throws DAOException

Поновлює об'єкт-сутність у БД.

Параметри:

instance -

Викидає:

DAOException

delete

<T> void delete(T instance)
throws DAOException

Видалити об'єкт-сутність із БД.

Параметри:

instance -

Викидає:

DAOException

select

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

throws DAOException

Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.

Параметри:

entityClass -

filter - - фільтр, за допомогою якого фільтруються сутності.

Повертає:

Викидає:

DAOException

select

<T> java.util.List<T> select(java.lang.String SQLString,
java.lang.Class<T> intitiesClass)
throws DAOException

Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.

Параметри:

SQLString -

intitiesClass -

Повертає:

Викидає:

DAOException

dao

public interface **DAOFilter**

Экземпляри класів, що реалізують цей інтерфейс дозволяють відфільтровувати класи-сутності.

Перелік методів

Модифіка	Ім'я і опис

тор і тип	
<T> boolean	**accept**(T entity)

Детальний опис методів

accept

<T> boolean accept(T entity)

Параметри:

entity — сутність.

Повертає:

true, якщо сутність задовільняє деяким крітеріям.

app.view.defaultview

Клас DefaultView

public class **DefaultView**

extends View

Клас, що представляє інтерфейс користувача за замовчуванням. Реагує на події, що генеруються іншими частинами додатку, і змінює вигляд необхідним чином або виконує необхідні дії.

Модифікатор і тип	Клас and Description
static class	**DefaultView.ViewContext**

Перелік контсрукторів

Ім'я і опис
DefaultView()

Перелік методів

Модифікатор і тип	Ім'я і опис
MainWindow	**getMainWindow**()
void	**onEvent**(**AppEvent** e) Реагує на події створені іншими частинами додатку

void	**start**() Створює інтерфейс користувача.

Детальний опис конструкторів

DefaultView

public DefaultView()

Детальний опис методів

start

public void start()

Створює інтерфейс користувача.

Оголошений як:

start у класі View

onEvent

public void onEvent(AppEvent e)

Реагує на події створені іншими частинами додатку

Оголошений як:

onEvent у класі View

Параметри:

e - - екземпляр класу AppEvent. Зараз підримуються події CathegoryClientsReceived, CathegoryWorkersReceived, EntitiesReceived, ErrorEvent, MassageEvent, ChangingLaFNeeded.

getMainWindow

public MainWindow getMainWindow()

app.view.defaultview

Клас EntityMenu

public class **EntityMenu**

extends javax.swing.JPopupMenu

implements java.awt.event.ActionListener

Клас, що представляє контекстне меню елемента дерева. Реагує на події викликані

елементами цього меню.

Дивись також:

Serialized Form

Перелік контсрукторів

Ім'я і опис
EntityMenu(javax.swing.JTree tree, javax.swing.tree.TreePath selPath)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**actionPerformed**(java.awt.event.ActionEvent e) Реагує на натискання пунктів контекстного меню.

Детальний опис конструкторів

EntityMenu

public EntityMenu(javax.swing.JTree tree,
javax.swing.tree.TreePath selPath)

Детальний опис методів

actionPerformed

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

Реагує на натискання пунктів контекстного меню.

Оголошений як:

actionPerformed у інтерфейсі java.awt.event.ActionListener

Параметри:

e - - об'єкт подія, що створюється пунктом меню.

Дивись також:

ActionListener

app.view.defaultview

Клас EntityNode

public class **EntityNode**

extends javax.swing.tree.DefaultMutableTreeNode

Клас, що представляє елемент дерева сутностей.

Перелік контсрукторів

Ім'я і опис
EntityNode(java.lang.Object object) Конструктор, що створює вузел дерева, що представляє об'єкт object.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**addDummy**() Додає до данго елемента елемент-заглушку.
void	**removeDummy**() Видаляє елемент-заглушку даного елемента.

Детальний опис конструкторів

EntityNode

public EntityNode(java.lang.Object object)

Конструктор, що створює вузел дерева, що представляє об'єкт object.

Параметри:

object - - об'єкт, що представляється цим вузлом.

Дивись також:

DefaultMutableTreeNode

Детальний опис методів

removeDummy

public void removeDummy()

Видаляє елемент-заглушку даного елемента. Якщо елемента-заглушки немає, нічого не робить.

addDummy

public void addDummy()

Додає до данго елемента елемент-заглушку.

app.model.events

Клас ErrorEvent

public class **ErrorEvent**

extends java.lang.Object

implements AppEvent

Клас, що сповіщає про виникнення помилки в программі.

Перелік контсрукторів

Ім'я і опис
ErrorEvent(java.lang.Exception exception)

Перелік методів

Модифікатор і тип	Ім'я і опис
java.lang.Exception	**getException**() повертає виключення пов'язане з цією помилкою.

Детальний опис конструкторів

ErrorEvent

public ErrorEvent(java.lang.Exception exception)

Детальний опис методів

getException

public java.lang.Exception getException()

dao.annotation

Аннотація FilenameColumn

@Retention(value=RUNTIME)

public @interface **FilenameColumn**

Містить інформацію про ім'я поля таблиці, в якому міститься ім'я данного файлу.

Перелік необхідних елементів

Модифікатор і тип	Required Element and Description

java.lang.String	**filenameColumn** — ім'я колонки з ім'ям імені файлу.

Element Detail

filenameColumn - ім'я колонки з ім'ям імені файлу.

public abstract java.lang.String filenameColumn

app.controller.commands.commandset

Клас MainWindow

public class **MainWindow**

extends javax.swing.JFrame

Головне вікно інтерфейсу за замовчуванням. Містить всі основні елементи управління.

Перелік контсрукторів

Ім'я і опис
MainWindow() Виконує роботу по побудові головного вікна.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**fillClientTree**() Виконує початкове заповнення дерева клієнтів.
void	**fillWorkerTree**() Виконує початкове заповнення дерева працівників.
javax.swing.JTree	**getClientsTree**()
NewActionDialog	**getNewActionDialog**() Створює діалог фіксування нової дії у БД.
javax.swing.JTree	**getWorkersTree**()

Детальний опис конструкторів

MainWindow

public MainWindow()

Виконує роботу по побудові головного вікна.

Детальний опис методів

fillClientTree

public void fillClientTree()

Виконує початкове заповнення дерева клієнтів.

fillWorkerTree

public void fillWorkerTree()

Виконує початкове заповнення дерева працівників.

getNewActionDialog

public NewActionDialog getNewActionDialog()

Створює діалог фіксування нової дії у БД.

Повертає:

getClientsTree

public javax.swing.JTree getClientsTree()

getWorkersTree

public javax.swing.JTree getWorkersTree()

app.view.defaultview

Клас NewActionDialog

public class **NewActionDialog**

extends javax.swing.JDialog

implements java.awt.event.ActionListener

Перелік контсрукторів

Ім'я і опис
NewActionDialog() Виконує роботу по побудові даного діалоу.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**actionPerformed**(java.awt.event.ActionEvent e)
void	**fillOrders**(**Order**[] orders) Заповнює список замовлень.
void	**setChoosedFiles**(java.io.File[] files) Змінює набір файлів, що вибрав користувач.

Детальний опис конструкторів

NewActionDialog

public NewActionDialog()

Виконує роботу по побудові даного діалоу.

Детальний опис методів

fillOrders

public void fillOrders(Order[] orders)

Заповнює список замовлень.

Параметри:

orders - - замовлення, якими заповнюється список заказів діалогового вікна.

setChoosedFiles

public void setChoosedFiles(java.io.File[] files)

Змінює набір файлів, що вибрав користувач.

Параметри:

files -

actionPerformed

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

Оголошений як:

actionPerformed у інтерфейсі java.awt.event.ActionListener

Дивись також:

ActionListener

dao.annotation

Аннотація Primary

@Retention(value=RUNTIME)

@Target(value=FIELD)

public @interface **Primary**

Вказує, що дане поле є первинним ключем таблиці у БД.

configuration.confutils.io

Клас PropertiesReader

public class **PropertiesReader**

extends java.lang.Object

implements Reader

Надає можливість зчитувати конфігурацію із файлу типу properties.

Перелік контсрукторів

Ім'я і опис
PropertiesReader(java.io.File file)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**load**(**Config** config) Завантажує конфігурацію.

Детальний опис конструкторів

PropertiesReader

public PropertiesReader(java.io.File file)

Детальний опис методів

load

public void load(Config config)

throws java.io.FileNotFoundException,

java.io.IOException

Завантажує конфігурацію.

Оголошений як:

load у інтерфейсі Reader

Параметри:

config - - конфіг, який модифікується.

Викидає:

java.io.IOException

java.io.FileNotFoundException

configuration.confutils.io

Клас PropertiesWriter

public class **PropertiesWriter**

extends java.lang.Object

implements Writer

Надає можливість запису конфігурації у файл типу properties.

Перелік контсрукторів

Ім'я і опис
PropertiesWriter(java.io.File file)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**save**(**Config** config) Записує конфігурацію у файл.

Детальний опис конструкторів

PropertiesWriter

public PropertiesWriter(java.io.File file)

Детальний опис методів

save

public void save(Config config)

throws java.io.IOException

Записує конфігурацію у файл.

Оголошений як:

save у інтерфейсі Writer

Параметри:

config - - конфіг, який записується.

Викидає:

java.io.IOException

utils

Клас R

public final class **R**

extends java.lang.Object

Клас, що надає доступ до ресурсів програми.

Перелік методів

Модифікатор і тип	Ім'я і опис
static java.lang.Object	**getResource**(java.lang.String key) Надає ресурс за ключем.

Детальний опис методів

getResource

public static java.lang.Object getResource(java.lang.String key)

Надає ресурс за ключем.

Параметри:

key - - унікальний ключ ресурса.

Повертає:

об'єкт-ресурс.

configuration.confutils.io

Interface Reader

Всі відомі класи, що реалізують цей інтерфейс:

PropertiesReader, XMLReader

---

public interface **Reader**

Надає можливість зчитувати конфігурацію із файлу.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**load**(**Config** config) Завантажує конфігурацію.

Детальний опис методів

load

void load(Config config)

throws javax.xml.parsers.ParserConfigurationException,
org.xml.sax.SAXException,
java.io.IOException

Завантажує конфігурацію.

Параметри:

config - - конфіг, який модифікується.

Викидає:

javax.xml.parsers.ParserConfigurationException

org.xml.sax.SAXException

java.io.IOException

app.controller.commands.commandset

Клас SettingsDialog

public class **SettingsDialog**

extends javax.swing.JDialog

implements java.awt.event.ActionListener

Діалог налаштувань програми.

Перелік контсрукторів

Ім'я і опис
SettingsDialog()
SettingsDialog(javax.swing.JFrame frame)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**actionPerformed**(java.awt.event.ActionEvent e)
void	**getCurrentValues**() Ініціалізує даний діалог значеннями із поточної конфігурації додатку.

Детальний опис конструкторів

SettingsDialog

public SettingsDialog()

Дивись також:

JDialog

SettingsDialog

public SettingsDialog(javax.swing.JFrame frame)

Дивись також:

JDialog

Детальний опис методів

getCurrentValues

public void getCurrentValues()

Ініціалізує даний діалог значеннями із поточної конфігурації додатку.

actionPerformed

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

Оголошений як:

actionPerformed у інтерфейсі java.awt.event.ActionListener

Дивись також:

ActionListener

dao.sqlcrud

Клас SQLBlobHandlerFactory

public class **SQLBlobHandlerFactory**

extends BlobHandlerFactory

Фабрика SQLBlobHandler.

Перелік контсрукторів

Ім'я і опис
SQLBlobHandlerFactory()

Перелік методів

Модифікатор і тип	Ім'я і опис
BlobHandler	**createBlobHandler**(java.lang.String table, java.util.UUID id, java.lang.String field, java.lang.String filename) Створює об'єкт класу BlobHandler.

Детальний опис конструкторів

SQLBlobHandlerFactory

public SQLBlobHandlerFactory()

Детальний опис методів

createBlobHandler

public BlobHandler createBlobHandler(java.lang.String table,
java.util.UUID id,
java.lang.String field,
java.lang.String filename)

Description copied from class: BlobHandlerFactory

Створює об'єкт класу BlobHandler.

Оголошений як:

createBlobHandler у класі BlobHandlerFactory

Повертає: об'єкт BlobHandler

dao.sqlcrud

Клас SQLCloser

public class **SQLCloser**

extends TransparentCRUD

Перелік контсрукторів

Ім'я і опис
SQLCloser()

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**close**() Закриває з'єднання з базою даних

Детальний опис конструкторів

SQLCloser

public SQLCloser()

Детальний опис методів

close

public void close()

throws DAOException

Закриває з'єднання з базою даних

Оголошений як:

close у інтерфейсі CRUDInterface

Перевизначає:

close у класі TransparentCRUD

Викидає:

DAOException

dao.sqlcrud.utils

Клас SQLCRUDUtils

public class **SQLCRUDUtils**

extends java.lang.Object

Перелік контсрукторів

Ім'я і опис
SQLCRUDUtils()

Перелік методів

Модифікатор і тип	Ім'я і опис
static java.lang.String	**createBlobString**(java.lang.String table, java.lang.String id, java.lang.String column, java.lang.String filename) Створює строку, яка представляє об'єкт BlobHandler і може бути перетворена в об'єкт BlobHandler методом DAOAnotationUtils.mapToEntity().
static <T> T	**getEntity**(java.sql.ResultSet resSet, java.lang.Class entityClass) Отримати об'єкт повного класу-сутності із ResultSet.
static <T> void	**preparedStSet**(java.lang.reflect.Field field, T instance, java.sql.PreparedStatement st, int stIndex) Виконує необхідну операцію

	PreparedStatement.set..() для значення поля field об'єкта instance.

Детальний опис конструкторів

SQLCRUDUtils

public SQLCRUDUtils()

Детальний опис методів

preparedStSet

public static <T> void preparedStSet(java.lang.reflect.Field field,
T instance,
java.sql.PreparedStatement st,
int stIndex)
throws java.lang.IllegalAccessException,
java.lang.IllegalArgumentException,
java.lang.reflect.InvocationTargetException,
java.sql.SQLException,
java.beans.IntrospectionException,
java.io.FileNotFoundException,
DAOException

Виконує необхідну операцію PreparedStatement.set..() для значення поля field об'єкта instance.

Параметри:

field - поле

instance - екземпляр

st -

stIndex - індекс

Викидає:

java.lang.IllegalAccessException

java.lang.IllegalArgumentException

java.lang.reflect.InvocationTargetException

java.sql.SQLException

java.beans.IntrospectionException

java.io.FileNotFoundException

DAOException

createBlobString

public static java.lang.String createBlobString(java.lang.String table,
java.lang.String id,
java.lang.String column,
java.lang.String filename)

Створює строку, яка представляє об'єкт BlobHandler і може бути перетворена в об'єкт BlobHandler методом DAOAnotationUtils.mapToEntity().

Параметри:

table -

id -

column -

filename -

Повертає:

getEntity

public static <T> T getEntity(java.sql.ResultSet resSet,
java.lang.Class entityClass)
throws java.sql.SQLException

Отримати об'єкт повного класу-сутності із ResultSet.

Параметри:

resSet -

entityClass -

Повертає:

Викидає:

java.sql.SQLException

dao.sqlcrud

Клас SQLDeleter

public class **SQLDeleter**

extends TransparentCRUD

Перелік контсрукторів

Ім'я і опис
SQLDeleter()

Перелік методів

Модифікатор і тип	Ім'я і опис
<T> void	**delete**(T instance) Видалити об'єкт-сутність із БД.

Детальний опис конструкторів

SQLDeleter

public SQLDeleter()

Детальний опис методів

delete

public <T> void delete(T instance)

throws DAOException

Видалити об'єкт-сутність із БД.

Оголошений як:

delete у інтерфейсі CRUDInterface

Перевизначає:

delete у класі TransparentCRUD

Викидає:

DAOException

dao.sqlcrud

Клас SQLSelector

public class **SQLSelector**

extends TransparentCRUD

Перелік контсрукторів

Ім'я і опис
SQLSelector()

Перелік методів

Модифікатор і тип	Ім'я і опис
<T> java.util.List<T>	**select**(java.lang.Class entityClass, **DAOFilter** filter) Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.
<T> java.util.List<T>	**select**(java.lang.String SQLString, java.lang.Class<T> entitiesClass) Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.

Детальний опис конструкторів

SQLSelector

public SQLSelector()

Детальний опис методів

select

public <T> java.util.List<T> select(java.lang.Class entityClass,
DAOFilter filter)
throws DAOException

Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.

Оголошений як:

select у інтерфейсі CRUDInterface

Перевизначає:

select у класі TransparentCRUD

filter - - фільтр, за допомогою якого фільтруються сутності.

Повертає:

Викидає:

DAOException

select

public <T> java.util.List<T> select(java.lang.String SQLString,
java.lang.Class<T> entitiesClass)
throws DAOException

Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.

Оголошений як:

select у інтерфейсі CRUDInterface

Перевизначає:

select у класі TransparentCRUD

Повертає:

Викидає:

DAOException

dao.sqlcrud

Клас SQLUpdater

public class **SQLUpdater**

extends TransparentCRUD

Клас, який виконує операцію update() інтерфейсу CRUDIntarface.

Перелік контсрукторів

Ім'я і опис
SQLUpdater()

Перелік методів

Модифікатор і тип	Ім'я і опис
<T> void	**update**(T instance) Поновлює об'єкт-сутність у БД.

Детальний опис конструкторів

SQLUpdater

public SQLUpdater()

Детальний опис методів

update

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

Поновлює об'єкт-сутність у БД.

Оголошений як:

update у інтерфейсі CRUDInterface

Перевизначає:

update у класі TransparentCRUD

Викидає:

DAOException

dao

Клас TransparentCRUD

public abstract class **TransparentCRUD**
extends java.lang.Object
implements CRUDInterface

Клас, який дозволяє організувати список виконавців операцій CRUDInterface.

Перелік контсрукторів

Ім'я і опис
TransparentCRUD()

TransparentCRUD(**CRUDInterface** nextCRUD)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**close**() Закриває з'єднання з базою даних
<T> void	**delete**(T instance) Видалити об'єкт-сутність із БД.
CRUDInterface	**getNextCRUD**()
<T> T	**insert**(T instance) Зберігає об'єкт-сутність у БД.
void	**open**() Відкриває з'єднання з базою даних
<T> T	**read**(java.lang.Class entityClass, java.util.UUID id) Зчитує об'экт певного класу-сутності з БД з певним id.
<T> java.util.List<T>	**select**(java.lang.Class entityClass, **DAOFilter** filter) Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.
<T> java.util.List<T>	**select**(java.lang.String SQLString, java.lang.Class<T> entitiesClass) Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.
void	**setNextCRUD**(**TransparentCRUD** nextCRUD)
<T> void	**update**(T instance) Поновлює об'єкт-сутність у БД.

Детальний опис конструкторів

TransparentCRUD

public TransparentCRUD()

TransparentCRUD

public TransparentCRUD(CRUDInterface nextCRUD)

Детальний опис методів

getNextCRUD

public CRUDInterface getNextCRUD()

setNextCRUD

public void setNextCRUD(TransparentCRUD nextCRUD)

insert

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

Зберігає об'єкт-сутність у БД.

Оголошений як:

insert у інтерфейсі CRUDInterface

Повертає:

Викидає:

DAOException

read

public <T> T read(java.lang.Class entityClass,
java.util.UUID id)
throws DAOException

Зчитує об'экт певного класу-сутності з БД з певним id.

Оголошений як:

read у інтерфейсі CRUDInterface

Повертає:

Викидає:

DAOException

update

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

Поновлює об'єкт-сутність у БД.

Оголошений як:

update у інтерфейсі CRUDInterface

Викидає:

DAOException

delete

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

Видалити об'єкт-сутність із БД.

Оголошений як:

delete у інтерфейсі CRUDInterface

Викидає:

DAOException

select

public <T> java.util.List<T> select(java.lang.Class entityClass,
DAOFilter filter)
throws DAOException

Description copied from interface: CRUDInterface

Зчитує всі об'экти певного класу-сутності з БД, які задовільняють певну умову.

Оголошений як:

select у інтерфейсі CRUDInterface

filter - - фільтр, за допомогою якого фільтруються сутності.

Повертає:

Викидає:

DAOException

select

public <T> java.util.List<T> select(java.lang.String SQLString,
java.lang.Class<T> entitiesClass)
throws DAOException

Отримує всі об'єкти певної сутності, які отримані за допомогою SQLSring.

Оголошений як:

select у інтерфейсі CRUDInterface

Повертає:

Викидає:

DAOException

open

public void open()
throws DAOException

Відкриває з'єднання з базою даних

Оголошений як:

open у інтерфейсі CRUDInterface

Викидає:

DAOException

close

public void close()
throws DAOException

Закриває з'єднання з базою даних

Оголошений як:

close у інтерфейсі CRUDInterface

Викидає:

DAOException

app.view

Клас View

public abstract class **View**

extends java.lang.Object

Абстрактний клас, що задає інтерфейс для представлень додатку. Має фабричний метод, що створює представлення.

Перелік контсрукторів

Ім'я і опис
View()

Перелік методів

Модифікатор і тип	Ім'я і опис
static **View**	**getInstance**() Надає екземпляр класу View.
abstract void	**onEvent**(**AppEvent** e)
abstract void	**start**() Виконує операції пов'язані зі створенням та початковою ініціалізації представлення.

Детальний опис конструкторів

View

public View()

Детальний опис методів

onEvent

public abstract void onEvent(AppEvent e)

getInstance

public static View getInstance()

Надає екземпляр класу View. Цей метод створює екземпляр тільки один раз, а потім повертає вже створений екземпляр.

Повертає:

об'єкт View.

start

public abstract void start()

Виконує операції пов'язані зі створенням та початковою ініціалізації представлення.

configuration.confutils.io

Interface Writer

public interface **Writer**

Надає можливість запису конфігурації у файл.

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**save**(**Config** config) Записує конфігурацію у файл.

Детальний опис методів

save

void save(Config config)
throws javax.xml.stream.XMLStreamException,
java.io.IOException

Записує конфігурацію у файл.

Параметри:

config - - конфіг, який записується.

Викидає:

javax.xml.stream.XMLStreamException

java.io.IOException

configuration.confutils.io

Клас XMLReader

public class **XMLReader**

extends java.lang.Object

implements Reader

Надає можливість зчитувати конфігурацію із файлу типу xml.

Перелік контсрукторів

Ім'я і опис
XMLReader(java.io.File file)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**load**(**Config** config) Завантажує конфігурацію.

Детальний опис конструкторів

XMLReader

public XMLReader(java.io.File file)

Детальний опис методів

load

public void load(Config config)

throws javax.xml.parsers.ParserConfigurationException,

org.xml.sax.SAXException,

java.io.IOException

Завантажує конфігурацію.

Оголошений як:

load у інтерфейсі Reader

Параметри:

config - - конфіг, який модифікується.

Викидає:

javax.xml.parsers.ParserConfigurationException

org.xml.sax.SAXException

java.io.IOException

configuration.confutils.io

Клас XMLWriter

public class **XMLWriter**

extends java.lang.Object

implements Writer

Надає можливість запису конфігурації у файл типу xml.

Перелік контсрукторів

Ім'я і опис
XMLWriter(java.io.File file)

Перелік методів

Модифікатор і тип	Ім'я і опис
void	**save**(**Config** config) Записує конфігурацію у файл.

Детальний опис конструкторів

XMLWriter

public XMLWriter(java.io.File file)

Детальний опис методів

save

public void save(Config config)

throws javax.xml.stream.XMLStreamException,

java.io.IOException

Записує конфігурацію у файл.

Оголошений як:

save у інтерфейсі Writer

Параметри:

config - - конфіг, який записується.

Викидає:

javax.xml.stream.XMLStreamException

java.io.IOException

3.3 Інструкція для користувача

Для отримання інформації про клієнта необхідно знайти його у дереві клієнтів, що знаходиться на вкладці “Клієнти”. Інформація з'явиться праворуч у текстовому полі. Для того щоб відкрити документ необхідно знайти його у дереві клієнтів чи працівників і клацнути по ньому два рази. Для того щоб змінити налаштування програми необхідно натиснуты на пункт меню “Інструменти” -> “Налаштування”. Потім заповнити необхідні поля та натиснути на кнопку “Зберегти”. Для того щоб зафіксувати певну дію у базі даних необхідно натиснути на пункт меню “Інструменти” -> “Нова дія” та заповнити необхідні поля. Потім треба натиснути кнопку “Ок”.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

Результат виконання прецеденту запуску програми (див рис. 2.1) зображений на рисунку 4.1.

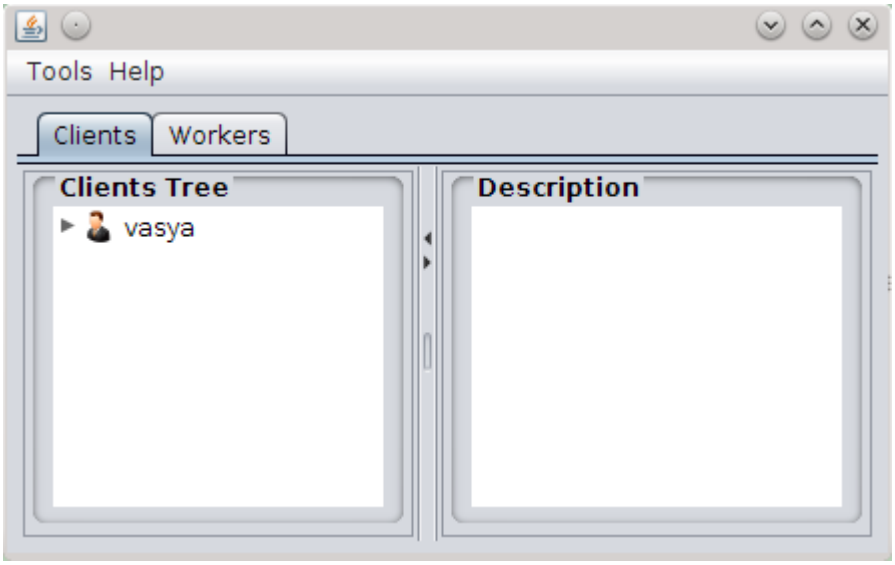


Рис. 4.1 Вікно програми відразу після запуску

На рисунку 4.2 зображено виконання прецеденту фіксування виконання дії по замовленню у БД (див рис. 2.2).

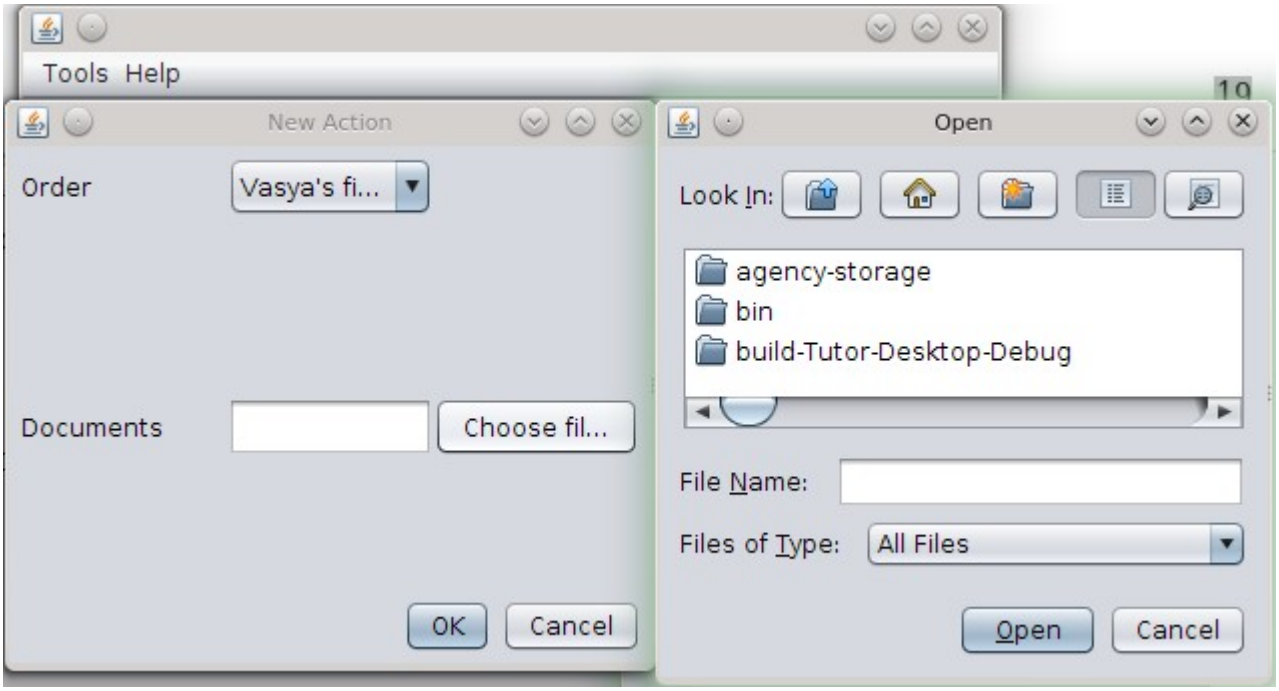


Рис. 4.2 Виконання прецеденти фіксування виконаної дії у БД

На рисунку 4.3 зображено виконання прецеденту отримання інформації про користувачів і замовлення (див рис. 2.3) .

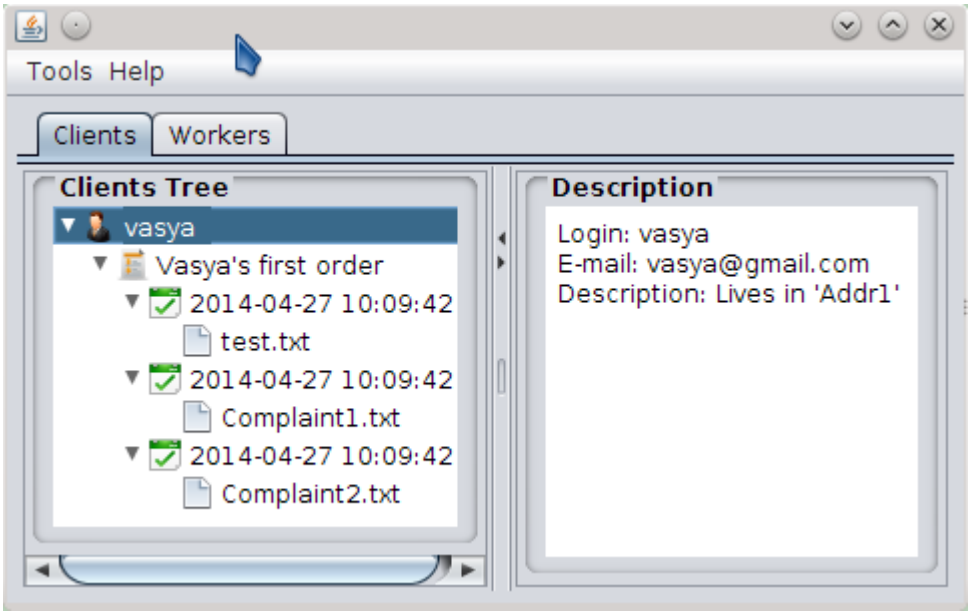


Рис. 4.3 Отримання інформації про користувачів і замовлення

На рис. 4.4 зображено вміст меню “Tools”.

На рис. 4.5 зображено вміст меню “Help”.

В меню “Tools” знаходяться пункти меню “New action” і “Settings”. Пункт “New action” призначений для виконання прецеденти фіксування виконаної дії по замовленню у базі даних. Пункт “Settings” призначений для виконання налаштування додатку.

В меню “Help” знаходяться пункти “Help” і “About”. Пункт “Help” призначений для отримання інструкції по використання додатком. Пункт “About” призначений для отримання інформації про версію і автора програми.

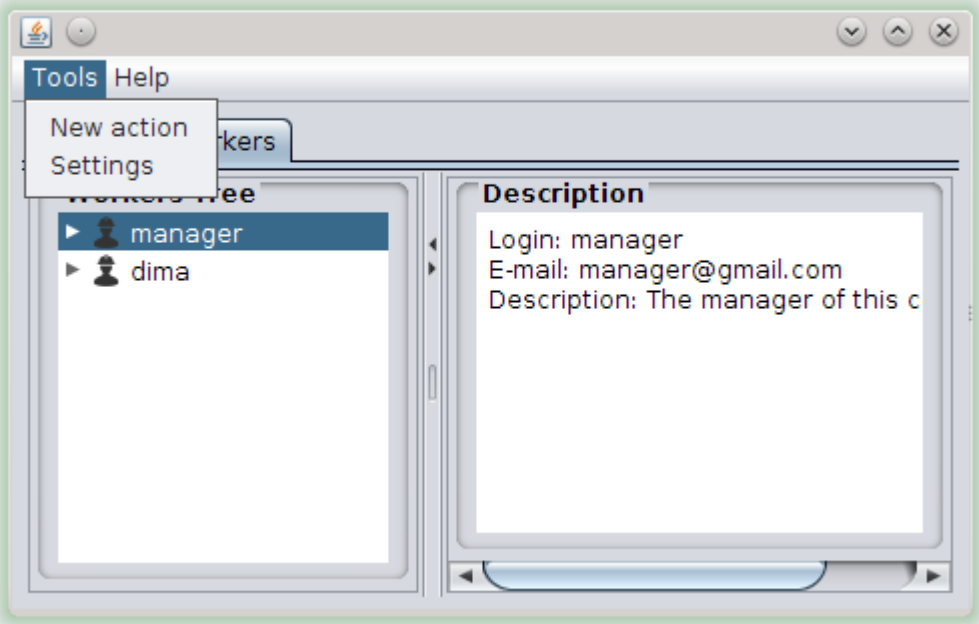


Рис. 4.4 Меню “Tools”

На рисунку 4.6 зображено результат виконання прецеденту отримання інформації про версію і автора програми (див рис. 2.4).

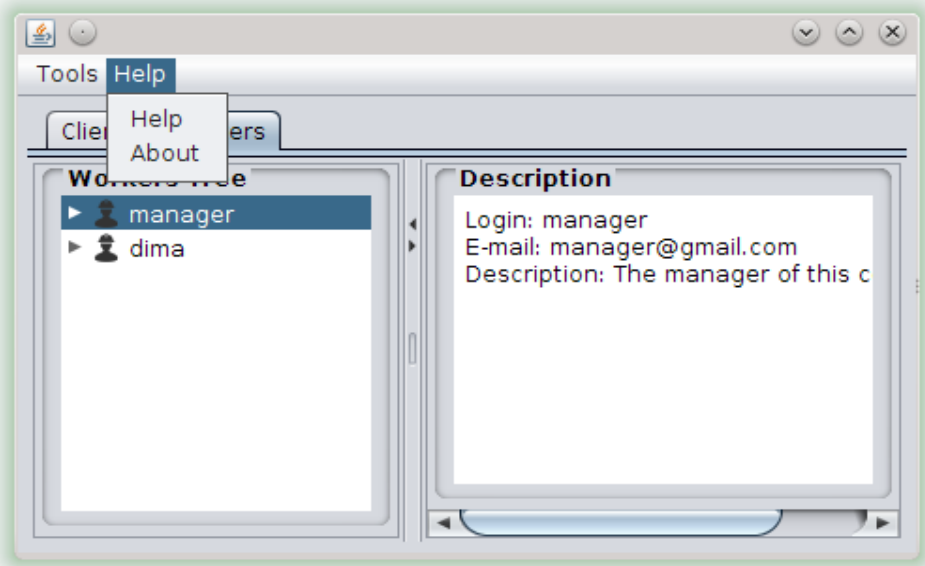


Рис. 4.5 Меню “Help”

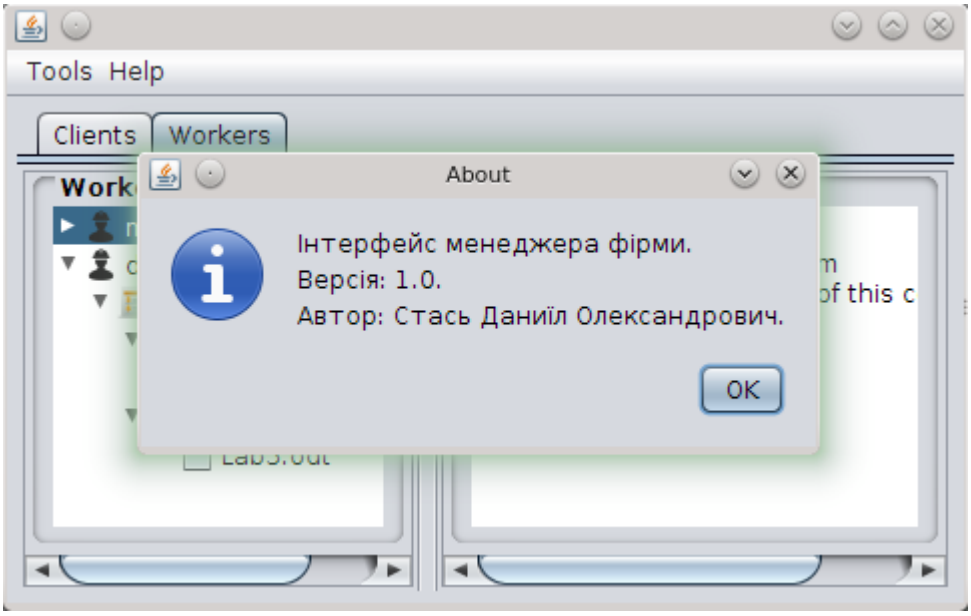


Рис. 4.6. Отримання інформації про версію і автора програми

На рисунку 4.7 зображено результат виконання прецеденту видалення елементу з бази даних (див рис.2.5).

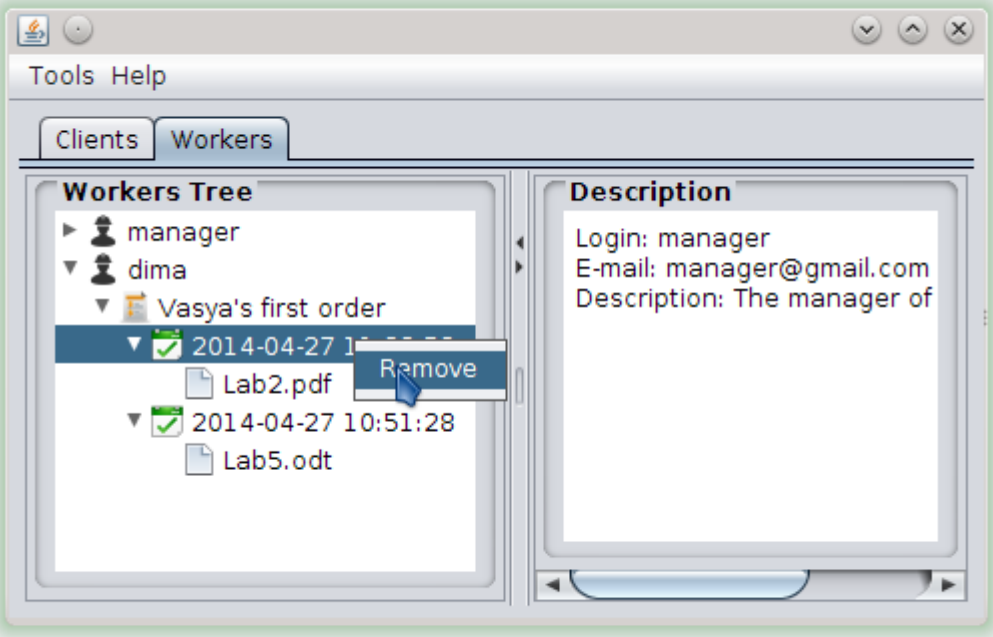


Рис. 4.7. Видалення елементу з БД

На рисунку 4.8 зображено виконання прецеденту налаштування

додатку (див рис. 2.6). Користувач має можливість змінювати логін, пароль і адресу БД, адресу тимчасового сховища файлів, а також вигляд програми.

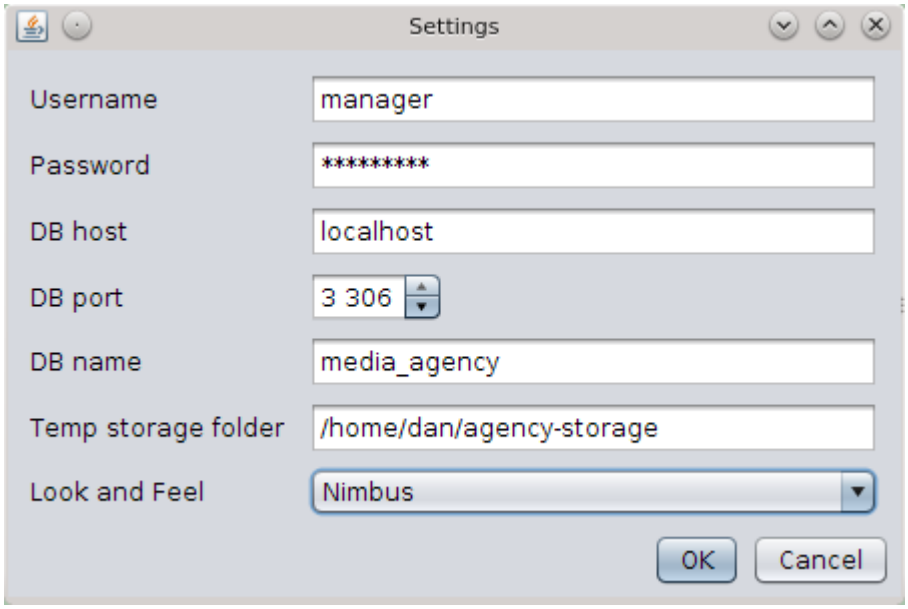


Рис. 4.8. Вікно налаштування додатку

Програмний додаток, що розробляється у даній роботі, реалізований згідно всіх вимог зазначених у технічному завданні. Під час тестування додатку не виявлено помилок чи зависань. На всіх етапах роботи він працює коректно.

ВИСНОВКИ

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

Всі вимоги, які були зазначені в технічному завданні, в програмному додатку було повністю виконано. Інтерфейс користувача розроблено так, щоб можна було швидко і зручно оперувати базою даних замовлень фірми.

В розробленому додатку реалізовано наступний функціонал: можливість перегляду бази даних, отримання інформації про користувачів і замовлень, зберігання документів по виконаній роботі у базі даних, можливість відкривати документи у програмі за замовчуванням системи та видалення елементів з бази даних.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Приемы объектно-ориентированого проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2011 – 368 с.: ил. –ISBN 978-5-469-01136-1.

2. Герберт Шилдт Java. Полное руководство, 8-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 20012. – 1104 с. – ISBN 978-5-8459-1759-1 (рус.)

3. Эккель Б. Философия Java / Эккель Брюс; Пер.с англ. Е.Матвеев.– 4-е изд.–СПб.: Питер, 2010. – 640с.: ил. – (Библиотека программиста). – Алф.указ.:с.631. – ISBN 978-5-388-00003-3.

4. Хорстманн Кей С. Java 2. Том 1. Основы / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 816 с.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1378-4 (рус.).

5. Хорстманн Кей С. Java 2. Том 2. Тонкости программирования / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 992 с.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1482-8 (рус.).

6. Стелтинг Стивен Применение шаблонов Java /Стелтинг Стивен, Маасен Олав; Пер. с англ. –М.: Издательский дом “Вильямс”, 2002. – 576 с.: ил. – Парал. тит. англ. – (Библиотека профессионала). – ISBN 5-8459-0339-4 (рус.).

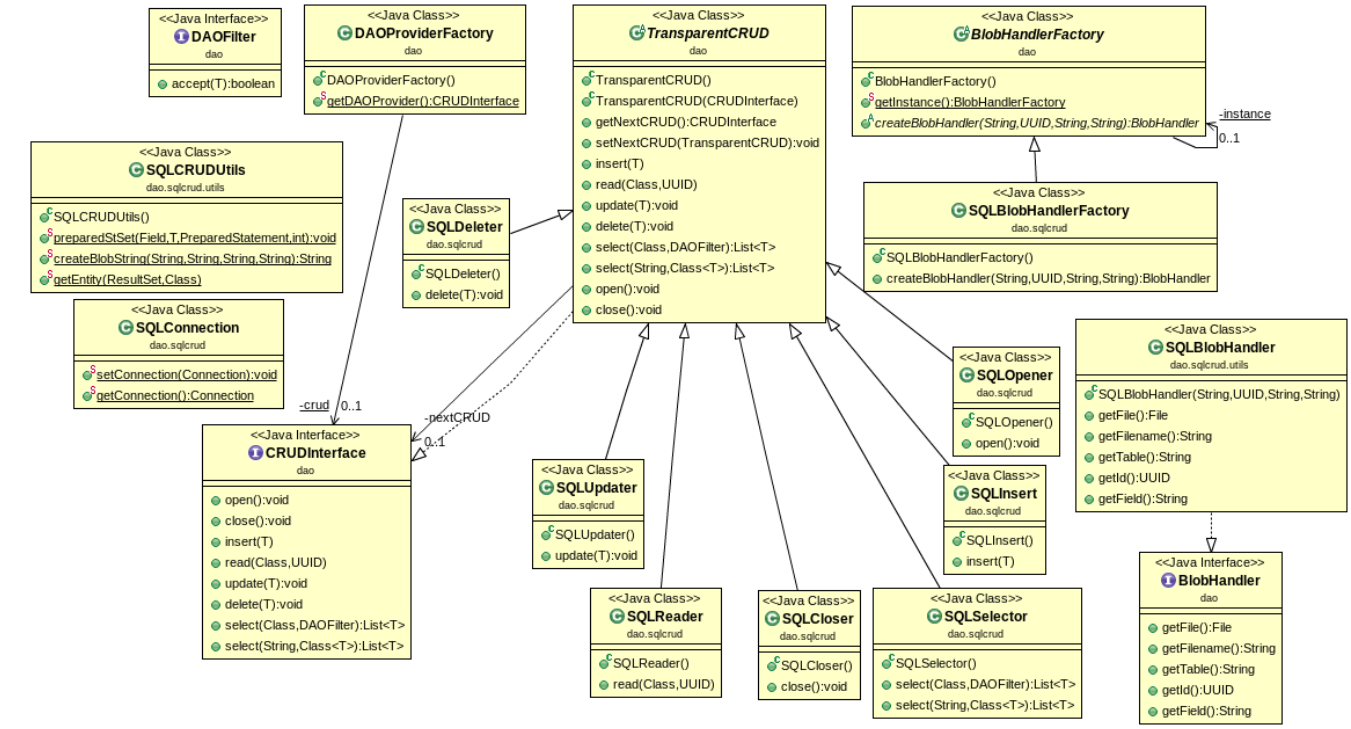
7. Дженифер Тидвелл Разработка пользовательских интрефейсов; Пер. с англ. –Е. Шикарева: Издательский дом “Питер”, 2008. – 416 с. – ISBN 978-5-91180-073-4 (рус.).

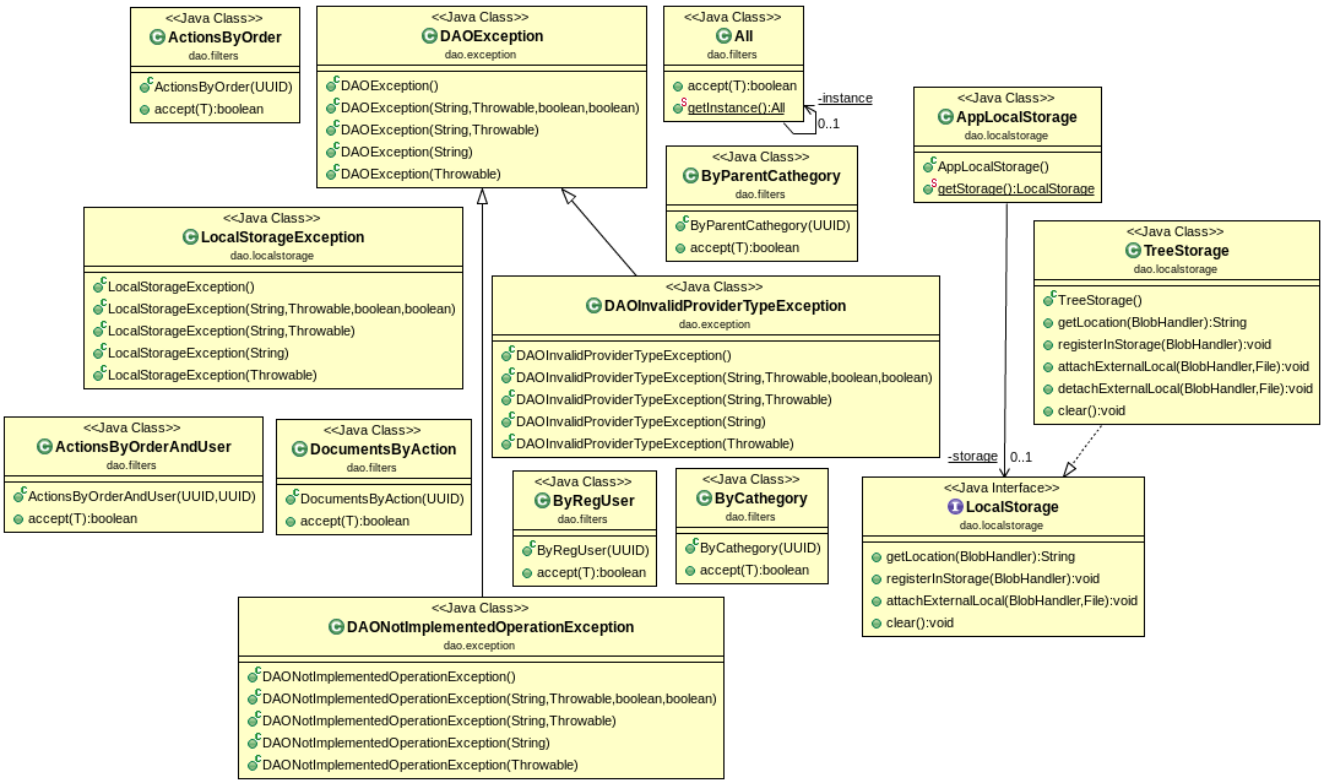
8. А. К. Гультяев Проектирование и дизайн пользовательского интерфейса / В. А. Машин; Пер. с англ. : Издательский дом “ Корона-Принт ”, 2010. – 350 с. – ISBN 978-5-7931-0814-0 (рус.).

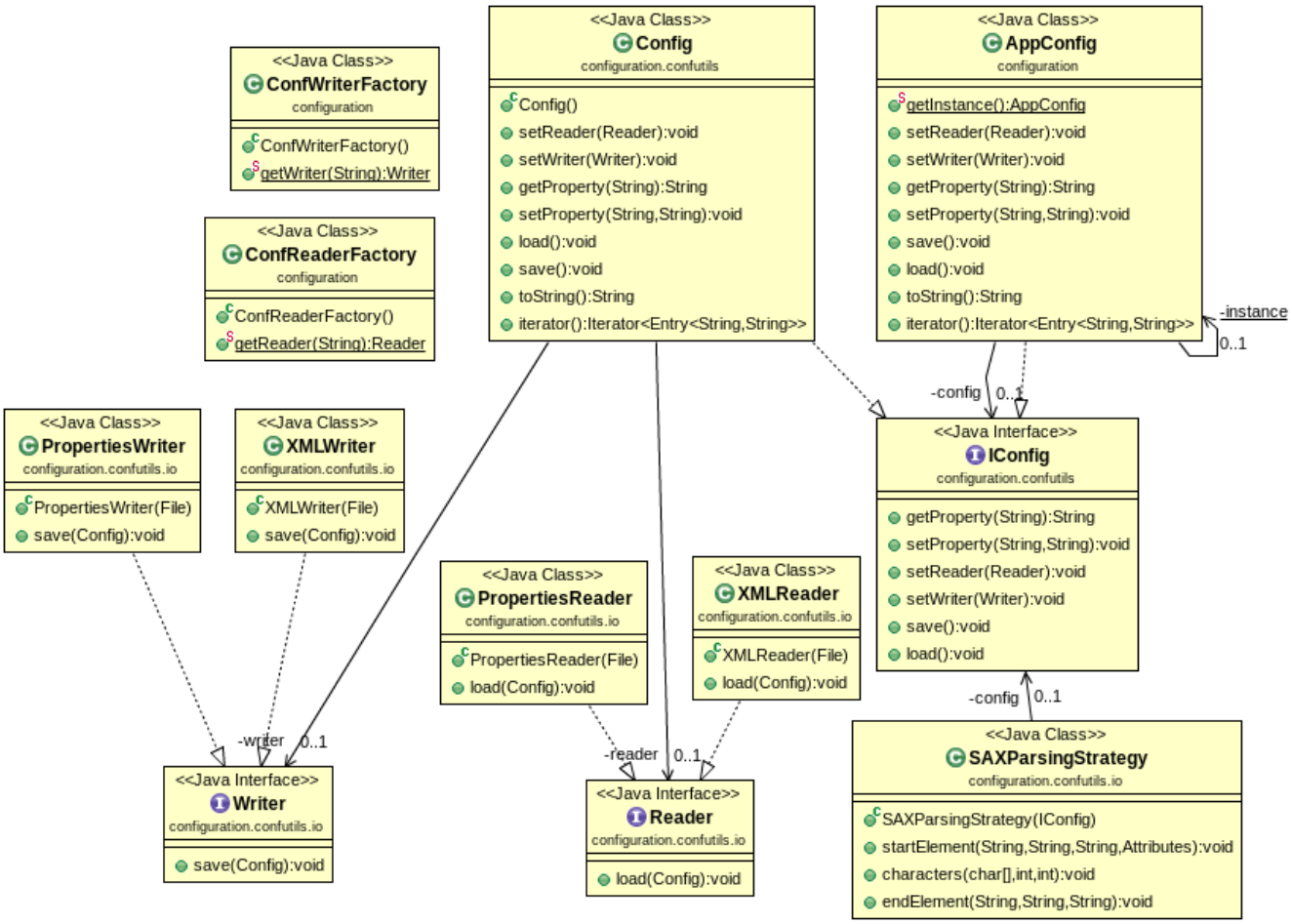
9. Герберт Шилдт. Библиотека SWING для Java: руководство для начинающих - Вильямс , 2007 — С. 704 - ISBN 978-5-8459-1162-9, 0-07-226314-8.

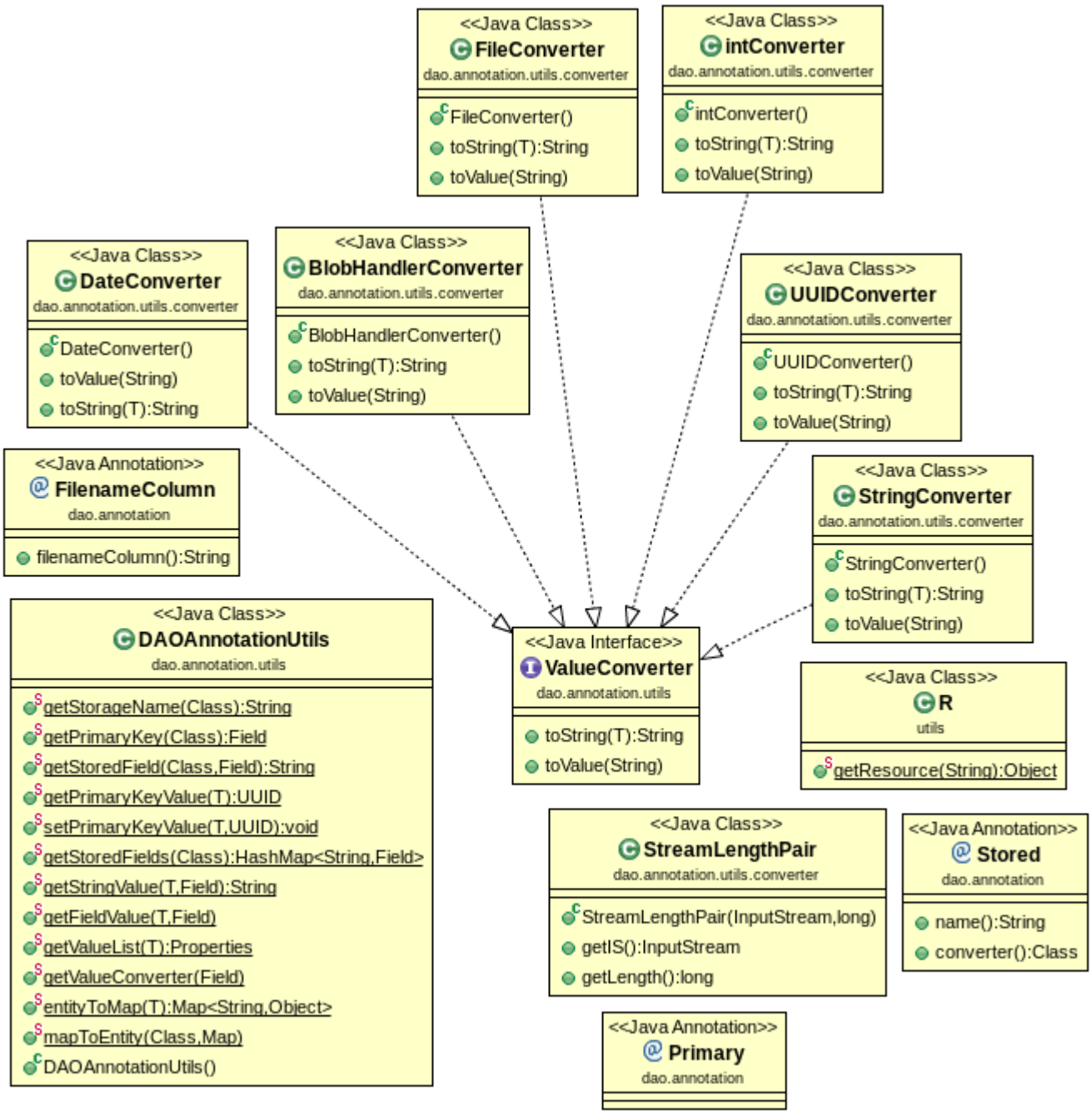
10. Марк Гранд Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560. — ISBN 5-94735-047-5

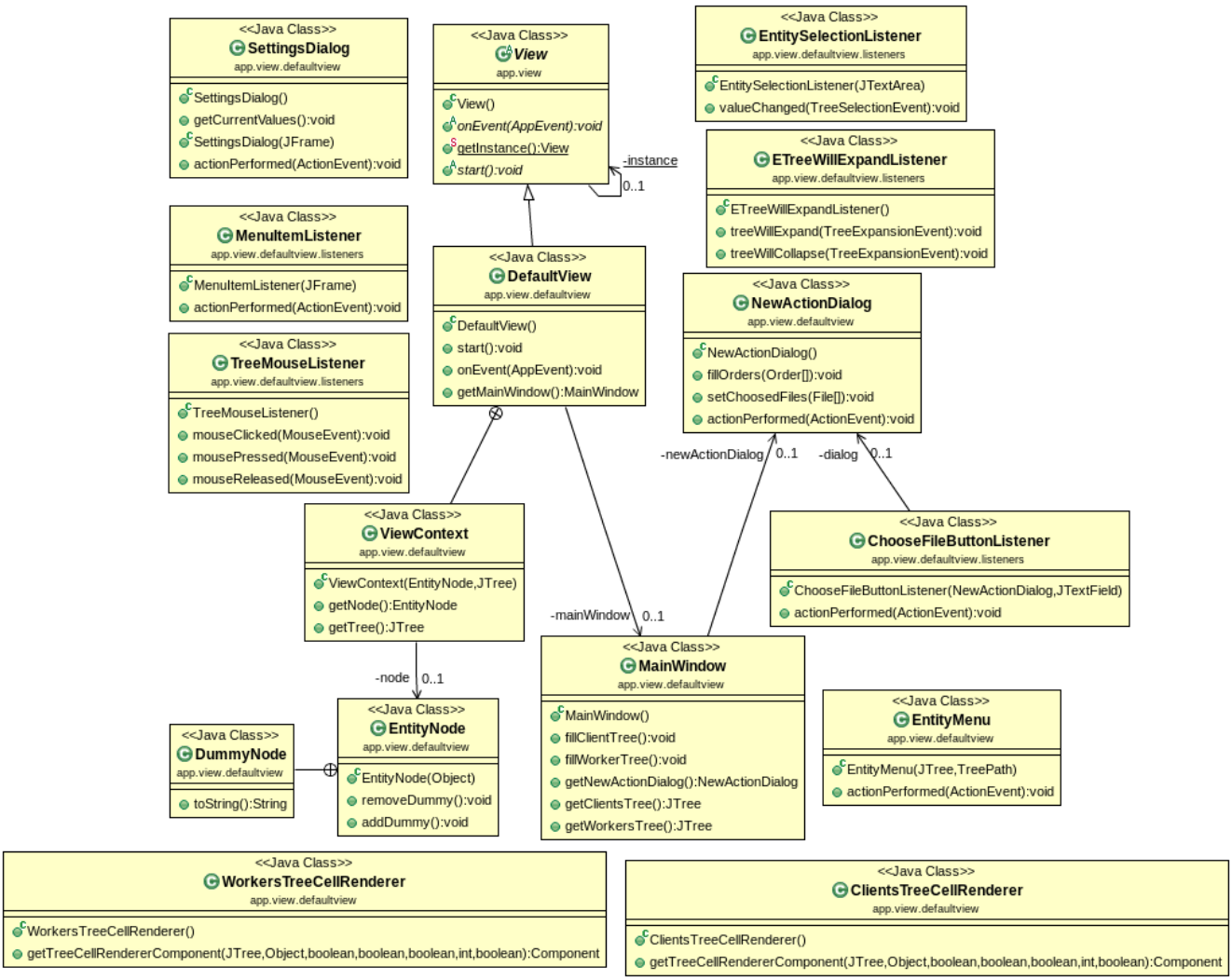
ДОДАТОК А. ДІАГРАМИ КЛАСІВ ДОДАТКУ

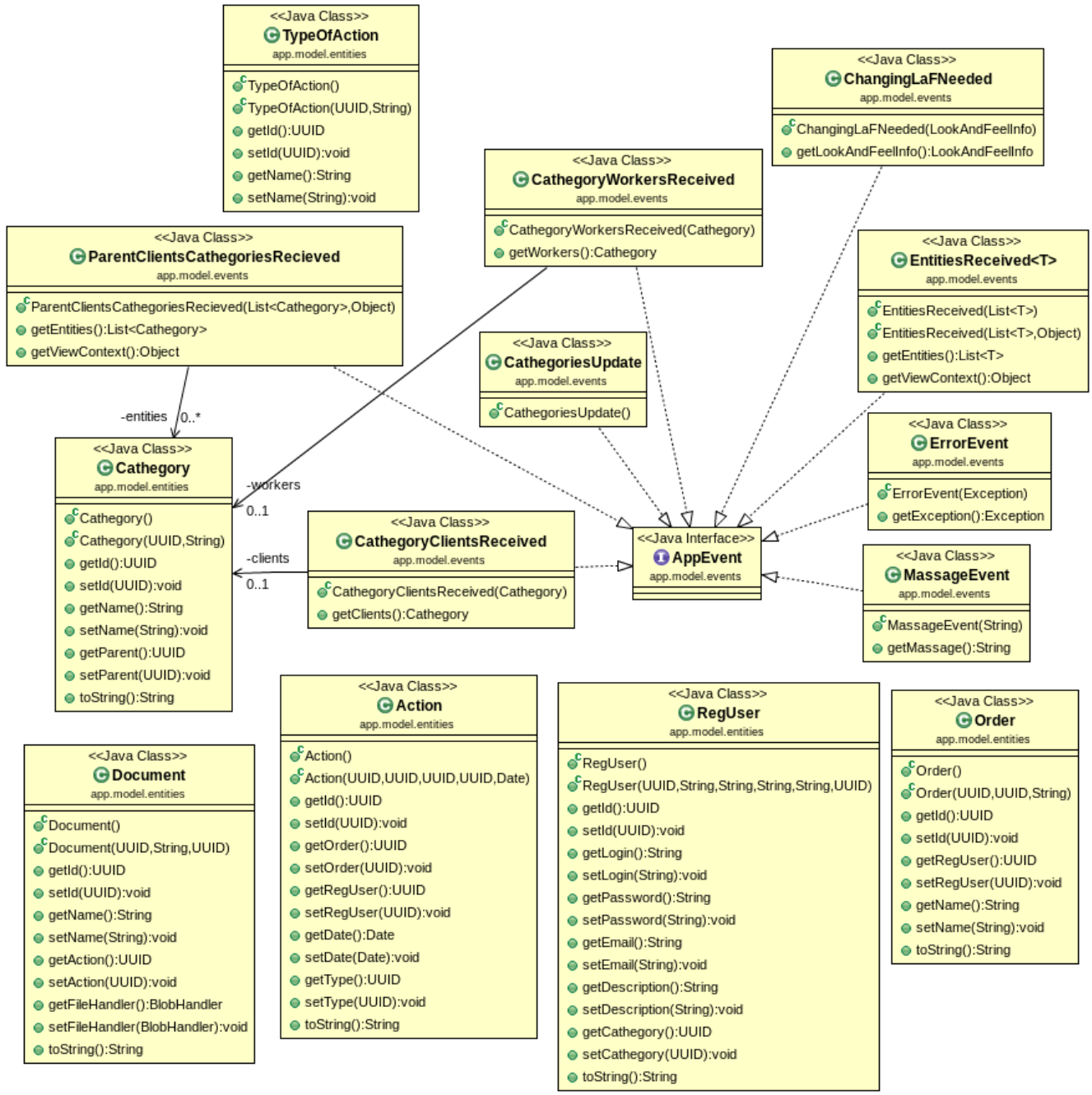


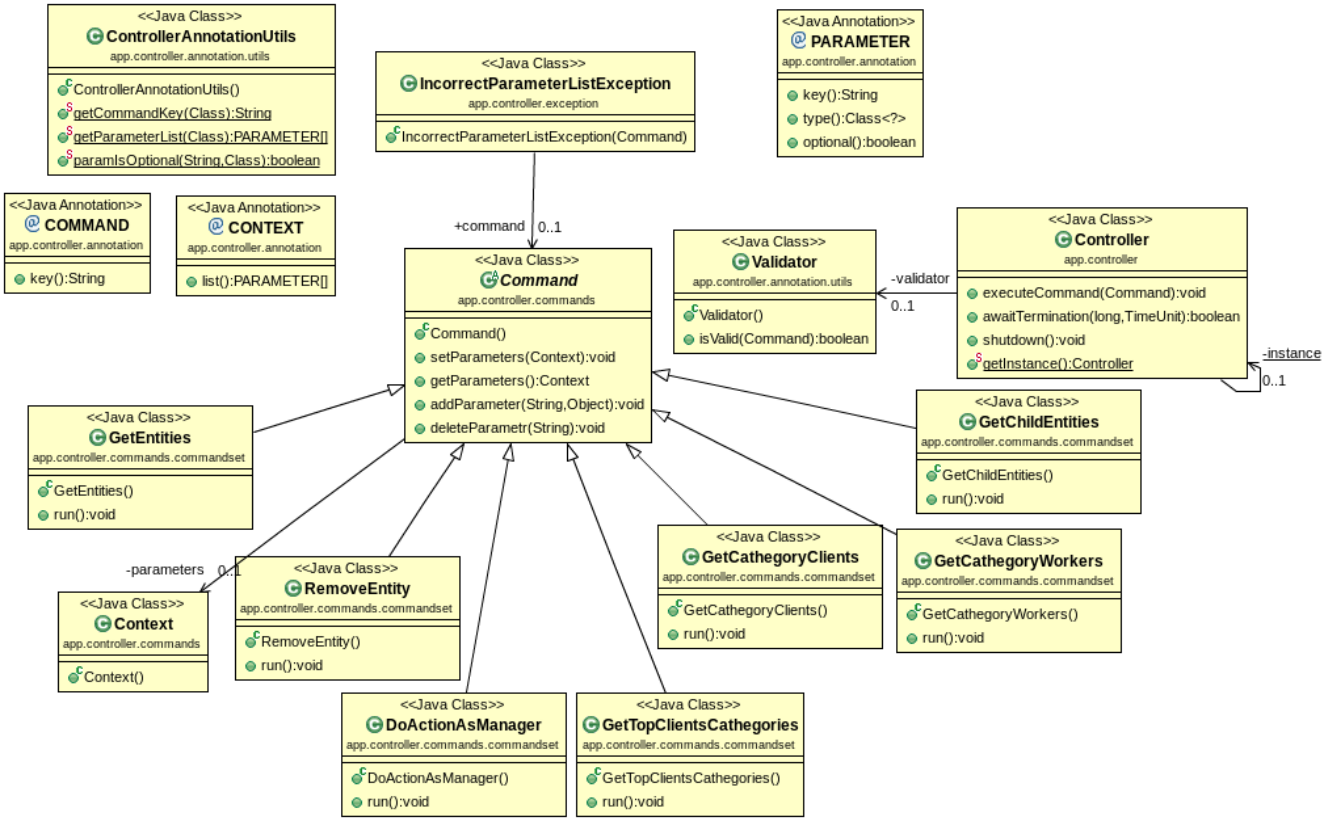



<<Java Class>>
ConfWriterFactory
configuration
ConfWriterFactory()
getWriter(String):Writer
<<Java Class>>
ConfReaderFactory
configuration
ConfReaderFactory()
getReader(String):Reader
<<Java Class>>
Config
configuration.confutils
Config()
setReader(Reader):void
setWriter(Writer):void
getProperty(String):String
setProperty(String,String):void
load():void
save():void
toString():String
iterator():Iterator<Entry<String,String>>
<<Java Class>>
AppConfig
configuration
getInstance():AppConfig
setReader(Reader):void
setWriter(Writer):void
getProperty(String):String
setProperty(String,String):void
save():void
load():void
toString():String
iterator():Iterator<Entry<String,String>>
-instance
0..1
-config
0..1
<<Java Class>>
PropertiesWriter
configuration.confutils.io
PropertiesWriter(File)
save(Config):void
<<Java Class>>
XMLWriter
configuration.confutils.io
XMLWriter(File)
save(Config):void
<<Java Interface>>
IConfig
configuration.confutils
getProperty(String):String
setProperty(String,String):void
setReader(Reader):void
setWriter(Writer):void
save():void
load():void
<<Java Class>>
PropertiesReader
configuration.confutils.io
PropertiesReader(File)
load(Config):void
<<Java Class>>
XMLReader
configuration.confutils.io
XMLReader(File)
load(Config):void
-writer
0..1
-reader
0..1
-config
0..1
<<Java Interface>>
Writer
configuration.confutils.io
save(Config):void
<<Java Interface>>
Reader
configuration.confutils.io
load(Config):void
<<Java Class>>
SAXParsingStrategy
configuration.confutils.io
SAXParsingStrategy(IConfig)
startElement(String,String,String,Attributes):void
characters(char[],int,int):void
endElement(String,String,String):void


<<Java Class>>
FileConverter
dao.annotation.utils.converter
FileConverter()
toString(T):String
toValue(String)
<<Java Class>>
intConverter
dao.annotation.utils.converter
intConverter()
toString(T):String
toValue(String)
<<Java Class>>
DateConverter
dao.annotation.utils.converter
DateConverter()
toValue(String)
toString(T):String
<<Java Class>>
BlobHandlerConverter
dao.annotation.utils.converter
BlobHandlerConverter()
toString(T):String
toValue(String)
<<Java Class>>
UUIDConverter
dao.annotation.utils.converter
UUIDConverter()
toString(T):String
toValue(String)
<<Java Annotation>>
FilenameColumn
dao.annotation
filenameColumn():String
<<Java Class>>
StringConverter
dao.annotation.utils.converter
StringConverter()
toString(T):String
toValue(String)
<<Java Class>>
DAOAnnotationUtils
dao.annotation.utils
getStorageName(Class):String
getPrimaryKey(Class):Field
getStoredField(Class,Field):String
getPrimaryKeyValue(T):UUID
setPrimaryKeyValue(T,UUID):void
getStoredFields(Class):HashMap<String,Field>
getStringValue(T,Field):String
getFieldValue(T,Field)
getValueList(T):Properties
getValueConverter(Field)
entityToMap(T):Map<String,Object>
mapToEntity(Class,Map)
DAOAnnotationUtils()
<<Java Interface>>
ValueConverter
dao.annotation.utils
toString(T):String
toValue(String)
<<Java Class>>
R
utils
getResource(String):Object
<<Java Class>>
StreamLengthPair
dao.annotation.utils.converter
StreamLengthPair(InputStream,long)
getIS():InputStream
getLength():long
<<Java Annotation>>
Stored
dao.annotation
name():String
converter():Class
<<Java Annotation>>
Primary
dao.annotation


<<Java Class>>
SettingsDialog
app.view.defaultview
SettingsDialog()
getCurrentValues():void
SettingsDialog(JFrame)
actionPerformed(ActionEvent):void
<<Java Class>>
View
app.view
View()
onEvent(AppEvent):void
getInstance():View
start():void
-instance
0..1
<<Java Class>>
EntitySelectionListener
app.view.defaultview.listeners
EntitySelectionListener(JTextArea)
valueChanged(TreeSelectionEvent):void
<<Java Class>>
ETreeWillExpandListener
app.view.defaultview.listeners
ETreeWillExpandListener()
treeWillExpand(TreeExpansionEvent):void
treeWillCollapse(TreeExpansionEvent):void
<<Java Class>>
MenuItemListener
app.view.defaultview.listeners
MenuItemListener(JFrame)
actionPerformed(ActionEvent):void
<<Java Class>>
DefaultView
app.view.defaultview
DefaultView()
start():void
onEvent(AppEvent):void
getMainWindow():MainWindow
<<Java Class>>
NewActionDialog
app.view.defaultview
NewActionDialog()
fillOrders(Order[]):void
setChoosedFiles(File[]):void
actionPerformed(ActionEvent):void
<<Java Class>>
TreeMouseListener
app.view.defaultview.listeners
TreeMouseListener()
mouseClicked(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void
-newActionDialog
0..1
-dialog
0..1
<<Java Class>>
ViewContext
app.view.defaultview
ViewContext(EntityNode,JTree)
getNode():EntityNode
getTree():JTree
<<Java Class>>
ChooseFileButtonListener
app.view.defaultview.listeners
ChooseFileButtonListener(NewActionDialog,JTextField)
actionPerformed(ActionEvent):void
-mainWindow
0..1
-node
0..1
<<Java Class>>
MainWindow
app.view.defaultview
MainWindow()
fillClientTree():void
fillWorkerTree():void
getNewActionDialog():NewActionDialog
getClientsTree():JTree
getWorkersTree():JTree
<<Java Class>>
EntityMenu
app.view.defaultview
EntityMenu(JTree,TreePath)
actionPerformed(ActionEvent):void
<<Java Class>>
DummyNode
app.view.defaultview
toString():String
<<Java Class>>
EntityNode
app.view.defaultview
EntityNode(Object)
removeDummy():void
addDummy():void
<<Java Class>>
WorkersTreeCellRenderer
app.view.defaultview
WorkersTreeCellRenderer()
getTreeCellRendererComponent(JTree,Object,boolean,boolean,boolean,int,boolean):Component
<<Java Class>>
ClientsTreeCellRenderer
app.view.defaultview
ClientsTreeCellRenderer()
getTreeCellRendererComponent(JTree,Object,boolean,boolean,boolean,int,boolean):Component


<<Java Class>>
TypeOfAction
app.model.entities
TypeOfAction()
TypeOfAction(UUID,String)
getId():UUID
setId(UUID):void
getName():String
setName(String):void
<<Java Class>>
ChangingLaFNeeded
app.model.events
ChangingLaFNeeded(LookAndFeelInfo)
getLookAndFeelInfo():LookAndFeelInfo
<<Java Class>>
CathegoryWorkersReceived
app.model.events
CathegoryWorkersReceived(Cathegory)
getWorkers():Cathegory
<<Java Class>>
ParentClientsCathegoriesRecieved
app.model.events
ParentClientsCathegoriesRecieved(List<Cathegory>,Object)
getEntities():List<Cathegory>
getViewContext():Object
<<Java Class>>
EntitiesReceived<T>
app.model.events
EntitiesReceived(List<T>)
EntitiesReceived(List<T>,Object)
getEntities():List<T>
getViewContext():Object
<<Java Class>>
CathegoriesUpdate
app.model.events
CathegoriesUpdate()
-entities
0..*
<<Java Class>>
ErrorEvent
app.model.events
ErrorEvent(Exception)
getException():Exception
<<Java Class>>
Cathegory
app.model.entities
Cathegory()
Cathegory(UUID,String)
getId():UUID
setId(UUID):void
getName():String
setName(String):void
getParent():UUID
setParent(UUID):void
toString():String
-workers
0..1
-clients
0..1
<<Java Class>>
CathegoryClientsReceived
app.model.events
CathegoryClientsReceived(Cathegory)
getClients():Cathegory
<<Java Interface>>
AppEvent
app.model.events
<<Java Class>>
MassageEvent
app.model.events
MassageEvent(String)
getMassage():String
<<Java Class>>
Document
app.model.entities
Document()
Document(UUID,String,UUID)
getId():UUID
setId(UUID):void
getName():String
setName(String):void
getAction():UUID
setAction(UUID):void
getFileHandler():BlobHandler
setFileHandler(BlobHandler):void
toString():String
<<Java Class>>
Action
app.model.entities
Action()
Action(UUID,UUID,UUID,UUID,Date)
getId():UUID
setId(UUID):void
getOrder():UUID
setOrder(UUID):void
getRegUser():UUID
setRegUser(UUID):void
getDate():Date
setDate(Date):void
getType():UUID
setType(UUID):void
toString():String
<<Java Class>>
RegUser
app.model.entities
RegUser()
RegUser(UUID,String,String,String,String,UUID)
getId():UUID
setId(UUID):void
getLogin():String
setLogin(String):void
getPassword():String
setPassword(String):void
getEmail():String
setEmail(String):void
getDescription():String
setDescription(String):void
getCathegory():UUID
setCathegory(UUID):void
toString():String
<<Java Class>>
Order
app.model.entities
Order()
Order(UUID,UUID,String)
getId():UUID
setId(UUID):void
getRegUser():UUID
setRegUser(UUID):void
getName():String
setName(String):void
toString():String


<<Java Class>>
ControllerAnnotationUtils
app.controller.annotation.utils
ControllerAnnotationUtils()
getCommandKey(Class):String
getParameterList(Class):PARAMETER[]
paramIsOptional(String,Class):boolean
<<Java Class>>
IncorrectParameterListException
app.controller.exception
IncorrectParameterListException(Command)
<<Java Annotation>>
PARAMETER
app.controller.annotation
key():String
type():Class<?>
optional():boolean
<<Java Annotation>>
COMMAND
app.controller.annotation
key():String
<<Java Annotation>>
CONTEXT
app.controller.annotation
list():PARAMETER[]
+command
0..1
<<Java Class>>
Command
app.controller.commands
Command()
setParameters(Context):void
getParameters():Context
addParameter(String,Object):void
deleteParametr(String):void
<<Java Class>>
Validator
app.controller.annotation.utils
Validator()
isValid(Command):boolean
-validator
0..1
<<Java Class>>
Controller
app.controller
executeCommand(Command):void
awaitTermination(long,TimeUnit):boolean
shutdown():void
getInstance():Controller
-instance
0..1
<<Java Class>>
GetEntities
app.controller.commands.commandset
GetEntities()
run():void
<<Java Class>>
GetChildEntities
app.controller.commands.commandset
GetChildEntities()
run():void
-parameters
0..1
<<Java Class>>
Context
app.controller.commands
Context()
<<Java Class>>
RemoveEntity
app.controller.commands.commandset
RemoveEntity()
run():void
<<Java Class>>
GetCathegoryClients
app.controller.commands.commandset
GetCathegoryClients()
run():void
<<Java Class>>
GetCathegoryWorkers
app.controller.commands.commandset
GetCathegoryWorkers()
run():void
<<Java Class>>
DoActionAsManager
app.controller.commands.commandset
DoActionAsManager()
run():void
<<Java Class>>
GetTopClientsCathegories
app.controller.commands.commandset
GetTopClientsCathegories()
run():void

ДОДАТОК Б. ВИХІДНИЙ КОД ДОДАТКУ

```
package app.controller.annotation.utils;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.annotation.PARAMETER;

public class ControllerAnnotationUtils {

	public static String getCommandKey(Клас c)
{
		COMMAND t = (COMMAND)
c.getAnnotation(COMMAND.class);
		return (t != null) ? t.key() :
null;
	}

	public static PARAMETER[]
getParameterList(Клас c) {
		CONTEXT t = (CONTEXT)
c.getAnnotation(CONTEXT.class);
		if (t == null)
			return null;
		return t.list();
	}

	public static boolean
paramIsOptional(String paramKey, Клас command) {
		PARAMETER[] pl =
getParameterList(command);
		for (PARAMETER p : pl) {
			if
(p.key().equals(paramKey))
				return p.optional();
		}
		return true;
	}
}
package app.controller.annotation.utils;

import app.controller.annotation.PARAMETER;
import app.controller.commands.Command;
import app.controller.commands.Context;

/**
 * Проверяет корректность сигнатуры вызываемого
метода
 *
 */
public class Validator {
	/**
	 * Проверить метод
	 *
	 * @param comand
	 *            - вызываемый метод
	 * @return - true - если правильная
сигнатура
	 */
	public boolean isValid(Command comand) {
		Context context =
comand.getParameters();
		PARAMETER[] parameters =
ControllerAnnotationUtils
				.getParameterList(com
and.getClass());
		if (parameters == null)
			return false;

		for (PARAMETER parameter :
parameters) {
			String key =
parameter.key();
			boolean optional =
parameter.optional();
			Object ObjectContext =
context.get(key);
			if (ObjectContext == null) {
				if (!optional) {
					return false;
				}
			} else {
				Клас<? extends
Object> classCurrentParameter = ObjectContext
.getCl
ass();
				if (!
parameter.type().isAssignableFrom(classCurrentPara
meter)) {
					return false;
				}
			}
		}
		return true;
	}
}
package app.controller.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Ця аннотація містить ключ спадкоємця класу
Command.
 *
 * @author dan
 *
 */
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.TYPE)
public @interface COMMAND {
	public String key();
}
package app.controller.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Містить в собі список параметрів для команди
(Command).
 *
 * @author dan
 *
 */
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.TYPE)
public @interface CONTEXT {
	PARAMETER[] list();
}
package app.controller.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
```

```
/**
 * Містить в собі інформацію про параметр команди
(Command).
 *
 * @author dan
 *
 */
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = ElementType.TYPE)
public @interface PARAMETER {
	public String key();

	public Клас<?> type() default Object.class;

	public boolean optional() default false;
}
```

```
package app.controller.commands.commandset;

import java.util.List;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.annotation.PARAMETER;
import app.controller.commands.Command;
import app.controller.commands.Context;
import app.model.entities.Action;
import app.model.entities.Document;
import app.model.entities.RegUser;
import app.model.entities.TypeOfAction;
import app.model.events.ErrorEvent;
import app.view.View;
import dao.CRUDInterface;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

/**
 * Записує нову дію у БД
 *
 * @author dan
 *
 */
@COMMAND(key = "do-action")
@CONTEXT(list = { @PARAMETER(key = "Action", type
= Action.class),
		@PARAMETER(key = "Documents", type
= Document[].class) })
public class DoActionAsManager extends Command {

	@Override
	public void run() {
		Context context = getParameters();
		Action action = (Action)
context.get("Action");
		try {
			CRUDInterface crud =
DAOProviderFactory.getDAOProvider();
			List<RegUser> users =
crud.select(
					"SELECT *
FROM RegisteredUser WHERE login = 'manager';",

	RegUser.class);
			List<TypeOfAction>
actionTypes = crud.select(
					"SELECT *
FROM TypeOfAction WHERE name = 'regular';",

	TypeOfAction.class);
			if (!(users.isEmpty() ||
actionTypes.isEmpty())) {
				TypeOfAction
actionType = actionTypes.get(0);
				RegUser manager =
users.get(0);

	action.setRegUser(manager.getId());

	action.setType(actionType.getId());
				crud.insert(action);
				Document[] docs =
(Document[]) context.get("Documents");
				for (Document
document : docs) {

	crud.insert(document);
				}
			}
		} catch (DAOException e) {

	View.getInstance().onEvent(new
ErrorEvent(e));
		}
	}

}
```

```
package app.controller.commands.commandset;

import java.util.List;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.commands.Command;
import app.model.entities.Cathegory;
import app.model.events.CathegoryClientsReceived;
import app.model.events.ErrorEvent;
import app.view.View;
import dao.CRUDInterface;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

/**
 * Команда, що отримує категорію "Clients" з БД.
 *
 * @author dan
 *
 */
@COMMAND(key = "get-category-clients")
@CONTEXT(list = {})
public class GetCathegoryClients extends Command {

	@Override
	public void run() {
		try {
			CRUDInterface crud =
DAOProviderFactory.getDAOProvider();
			String sql = "SELECT * FROM
Cathegory WHERE name = 'Clients';";
			List<Cathegory> cathegories
= crud.select(sql, Cathegory.class);
			if (!cathegories.isEmpty())
{

	CathegoryClientsReceived event = new
CathegoryClientsReceived(

	cathegories.get(0));

	View.getInstance().onEvent(event);
			}
		} catch (DAOException e) {

	View.getInstance().onEvent(new
ErrorEvent(e));
```

```
                }

        }
}
package app.controller.commands.commandset;

import java.util.List;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.commands.Command;
import app.model.entities.Cathegory;
import app.model.events.CathegoryWorkersReceived;
import app.model.events.ErrorEvent;
import app.view.View;
import dao.CRUDInterface;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

/**
 * Команда, що отримує категорію "Workers" з БД.
 *
 * @author dan
 *
 */
@COMMAND(key = "get-category-workers")
@CONTEXT(list = {})
public class GetCathegoryWorkers extends Command {
        @Override
        public void run() {
                try {
                        CRUDInterface crud =
DAOProviderFactory.getDAOProvider();
                        String sql = "SELECT * FROM
Cathegory WHERE name = 'Workers';";
                        List<Cathegory> cathegories
= crud.select(sql, Cathegory.class);
                        if (!cathegories.isEmpty())
{

        CathegoryWorkersReceived event = new
CathegoryWorkersReceived(

        cathegories.get(0));

        View.getInstance().onEvent(event);
                        }
                } catch (DAOException e) {

        View.getInstance().onEvent(new
ErrorEvent(e));
                }

        }

}
package app.controller.commands.commandset;

import java.util.List;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.annotation.PARAMETER;
import app.controller.commands.Command;
import app.model.entities.Cathegory;
import app.model.events.ErrorEvent;
import
app.model.events.ParentClientsCathegoriesRecieved;
import app.view.View;
import dao.CRUDInterface;
import dao.DAOFilter;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

@COMMAND(key = "get-top-clients-categories")
@CONTEXT(list = { @PARAMETER(key = "filter", type
= DAOFilter.class),
                @PARAMETER(key = "viewContext",
type = Object.class, optional = true) })
public class GetTopClientsCathegories extends
Command {

        @Override
        public void run() {
                try {
                        CRUDInterface crud =
DAOProviderFactory.getDAOProvider();
                        DAOFilter filter =
(DAOFilter) getParameters().get("filter");
                        List<Cathegory> cathegories
= crud.select(Cathegory.class, filter);
                        Object viewContext =
getParameters().get("viewContext");

        ParentClientsCathegoriesRecieved event =
new ParentClientsCathegoriesRecieved(
                                        cathegories,
viewContext);

        View.getInstance().onEvent(event);
                        //
DB.setCathegories(cathegories);
                } catch (DAOException e) {

        View.getInstance().onEvent(new
ErrorEvent(e));
                }
        }

}
package app.controller.commands.commandset;

import app.controller.annotation.COMMAND;
import app.controller.annotation.CONTEXT;
import app.controller.annotation.PARAMETER;
import app.controller.commands.Command;
import app.model.events.ErrorEvent;
import app.view.View;
import dao.CRUDInterface;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

/**
 * Команда, що видаляє сутність з БД.
 *
 * @author dan
 *
 */
@COMMAND(key = "remove-entity")
@CONTEXT(list = { @PARAMETER(key = "entity", type
= Object.class) })
public class RemoveEntity extends Command {

        @Override
        public void run() {
                Object entity =
getParameters().get("entity");
                try {
                        CRUDInterface crud =
DAOProviderFactory.getDAOProvider();
                        crud.delete(entity);
                } catch (DAOException e) {
```

```
		View.getInstance().onEvent(new
ErrorEvent(e));
			}

		}

}
package app.controller.commands;

/**
 * Інтерфейс для класів команд, що виконує
контролер.
 *
 */
public abstract class Command implements Runnable
{
		private Context parameters = new Context();

		/**
		 * Змінює параметри команди.
		 *
		 * @param parameters
		 *			- параметры
		 */
		public void setParameters(Context
parameters) {
			this.parameters = parameters;
		}

		/**
		 * Отримати парааметри.
		 *
		 * @return - параметри команди.
		 */
		public Context getParameters() {
			return parameters;
		}

		/**
		 * Додає параметри до команди.
		 *
		 * @param key
		 *			- ключ параметра
		 * @param value
		 *			- значення параметра.
		 */
		public void addParameter(String key, Object
value) {
			parameters.put(key, value);
		}

		/**
		 * видаляє параметр по ключу.
		 *
		 * @param key
		 *			- ключ параметра.
		 */
		public void deleteParametr(String key) {
			parameters.remove(key);
		}
}
package app.controller.commands;

import java.util.HashMap;

/**
 * Клас, що утримує параметри команди.
 *
 */
@SuppressWarnings("serial")
public class Context extends HashMap<String,
Object> {
}
package app.controller.exception;

import app.controller.commands.Command;

/**
 * Викидується, якщо список параметрів команди не
правельний.
 *
 */
@SuppressWarnings("serial")
public class IncorrectParameterListException
extends Exception {
		public Command command;

		public
IncorrectParameterListException(Command command) {
			this.command = command;
		}
}
package app.controller;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import app.controller.annotation.utils.Validator;
import app.controller.commands.Command;
import
app.controller.exception.IncorrectParameterListExc
eption;

/**
 * Клас, що приймає і виконує команди.
 *
 */
public class Controller {
		private static Controller instance;
		private Validator validator = new
Validator();
		private ExecutorService executor =
Executors.newSingleThreadExecutor();

		private Controller() {

		}

		/**
		 * Виконує команду. Команди складаються у
чергу і виконуються у окремому
		 * потоці.
		 *
		 * @throws IncorrectParameterListException
		 * @throws InterruptedException
		 */
		public void executeCommand(Command command)
				throws
IncorrectParameterListException {
			if (validator.isValid(command)) {
				executor.execute(command);
			} else {
				throw new
IncorrectParameterListException(command);
			}
		}

		/**
		 * Чекати завершення виконання команд певну
кількість часу. Якщо час вийшов
		 * команди завершуються передчасно.
Необхідно виконувати після виконання
```

```
         * методу shutdown().
         *
         * @param timeout
         *            - кількість часу.
         * @param unit
         *            - Одиниці виміру часу.
         * @return true ,якщо виконання команд
завершилось за заданий час, і false,
         *         якщо команди завершились
передчасно.
         * @throws InterruptedException
         */
        public boolean awaitTermination(long
timeout, TimeUnit unit)
                        throws InterruptedException
{
                return
executor.awaitTermination(timeout, unit);
        }

        /**
         * Завершити приймати команди.
         */
        public void shutdown() {
                executor.shutdown();
        }

        public static synchronized Controller
getInstance() {
                if (instance == null) {
                        instance = new Controller();
                }
                return instance;
        }
}
package app.model.entities;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.UUID;

import dao.annotation.Primary;
import dao.annotation.Stored;
import
dao.annotation.utils.converter.DateConverter;
import
dao.annotation.utils.converter.UUIDConverter;

/**
 * Клас, що представляє таблицю Action із БД.
 *
 * @author dan
 *
 */
@Stored(name = "Action")
public class Action {
        private static DateFormat dateFormat = new
SimpleDateFormat(
                        "yyyy-MM-dd hh:mm:ss");

        @Stored(name = "id", converter =
UUIDConverter.class)
        @Primary
        private UUID id;

        @Stored(name = "Order_id", converter =
UUIDConverter.class)
        private UUID order;

        @Stored(name = "RegisteredUser_id",
converter = UUIDConverter.class)
        private UUID regUser;

        @Stored(name = "TypeOfAction_id", converter
= UUIDConverter.class)
        private UUID type;

        @Stored(name = "datetime", converter =
DateConverter.class)
        private Date date;

        public Action() {
        }

        public Action(UUID id, UUID order, UUID
regUser, UUID type, Date date) {
                this.id = id;
                this.order = order;
                this.regUser = regUser;
                this.date = date;
                this.type = type;
        }

        public UUID getId() {
                return id;
        }

        public void setId(UUID id) {
                this.id = id;
        }

        public UUID getOrder() {
                return order;
        }

        public void setOrder(UUID order) {
                this.order = order;
        }

        public UUID getRegUser() {
                return regUser;
        }

        public void setRegUser(UUID regUser) {
                this.regUser = regUser;
        }

        public Date getDate() {
                return date;
        }

        public void setDate(Date date) {
                this.date = date;
        }

        public UUID getType() {
                return type;
        }

        public void setType(UUID type) {
                this.type = type;
        }

        /**
         * @return строку - дату виконання дії.
         */
        @Override
        public String toString() {
                return
dateFormat.format(getDate());
        }
}
package app.model.entities;
```

```
import java.util.UUID;

import dao.annotation.Primary;
import dao.annotation.Stored;
import
dao.annotation.utils.converter.UUIDConverter;

/**
 * Клас, що представляє таблицю Cathegory із БД.
 *
 * @author dan
 *
 */
@Stored(name = "Cathegory")
public class Cathegory {
	@Stored(name = "id", converter =
UUIDConverter.class)
	@Primary
	private UUID id;

	@Stored(name = "name")
	private String name;

	@Stored(name = "Parent_id", converter =
UUIDConverter.class)
	private UUID parent;

	public Cathegory() {
	}

	public Cathegory(UUID id, String name) {
		this.id = id;
		this.name = name;
	}

	public UUID getId() {
		return id;
	}

	public void setId(UUID id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public UUID getParent() {
		return parent;
	}

	public void setParent(UUID parent) {
		this.parent = parent;
	}

	@Override
	public String toString() {
		return name;
	}
}
package app.model.entities;

import java.util.UUID;

import dao.BlobHandler;
import dao.BlobHandlerFactory;
import dao.annotation.FilenameColumn;
import dao.annotation.Primary;
import dao.annotation.Stored;
import
dao.annotation.utils.converter.BlobHandlerConverte
r;
import
dao.annotation.utils.converter.UUIDConverter;

/**
 * Клас, що представляє таблицю Document із БД.
 *
 * @author dan
 *
 */
@Stored(name = "Document")
public class Document {
	private final String filenameColumn =
"name";
	private final String fileColumn = "file";
	@Stored(name = "id", converter =
UUIDConverter.class)
	@Primary
	private UUID id;

	@Stored(name = filenameColumn)
	private String name;

	@Stored(name = fileColumn, converter =
BlobHandlerConverter.class)
	@FilenameColumn(filenameColumn =
filenameColumn)
	private BlobHandler fileHandler;

	@Stored(name = "Action_id", converter =
UUIDConverter.class)
	private UUID action;

	public Document() {
	}

	public Document(UUID id, String name, UUID
action) {
		this.id = id;
		this.name = name;
		this.action = action;
		String table =
getClass().getAnnotation(Stored.class).name();
		BlobHandlerFactory bhFactory =
BlobHandlerFactory.getInstance();
		this.fileHandler =
bhFactory.createBlobHandler(table, id, fileColumn,
				name);

	}

	public UUID getId() {
		return id;
	}

	public void setId(UUID id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public UUID getAction() {
```

```
		return action;
	}

	public void setAction(UUID action) {
		this.action = action;
	}

	public BlobHandler getFileHandler() {
		return fileHandler;
	}

	public void setFileHandler(BlobHandler
file) {
		this.fileHandler = file;
	}

	@Override
	public String toString() {
		return getName();
	}
}
package app.model.entities;

import java.util.UUID;

import dao.annotation.Primary;
import dao.annotation.Stored;
import
dao.annotation.utils.converter.UUIDConverter;

/**
 * Клас, що представляє таблицю Order із БД.
 *
 * @author dan
 *
 */
@Stored(name = "Order")
public class Order {
	@Stored(name = "id", converter =
UUIDConverter.class)
	@Primary
	private UUID id;

	@Stored(name = "name")
	private String name;

	@Stored(name = "RegisteredUser_id",
converter = UUIDConverter.class)
	private UUID regUser;

	public Order() {
	}

	public Order(UUID id, UUID regUser, String
name) {
		this.id = id;
		this.regUser = regUser;
		this.name = name;
	}

	public UUID getId() {
		return id;
	}

	public void setId(UUID id) {
		this.id = id;
	}

	public UUID getRegUser() {
		return regUser;
	}

	public void setRegUser(UUID regUser) {
		this.regUser = regUser;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@Override
	public String toString() {
		return getName();
	}
}
package app.model.entities;

import java.util.UUID;

import dao.annotation.Primary;
import dao.annotation.Stored;
import
dao.annotation.utils.converter.UUIDConverter;

/**
 * Клас, що представляє таблицю RegisteredUser із
БД.
 *
 * @author dan
 *
 */
@Stored(name = "RegisteredUser")
package app.model.events;

/**
 * Інтерфейс для класів-подій додатку.
 *
 * @author dan
 *
 */
public interface AppEvent {

}
package app.model.events;

/**
 * Клас, що сповіщає про оновлення категорій.
 *
 * @author dan
 *
 */
public class CathegoriesUpdate implements AppEvent
{
}
package app.model.events;

import app.model.entities.Cathegory;

/**
 * Клас, що представляє подію отримання категорії
"Клієнти".
 *
 * @author dan
 *
 */
public class CathegoryClientsReceived implements
AppEvent {
	private Cathegory clients;

	public CathegoryClientsReceived(Cathegory
```

```
clients) {
		this.clients = clients;
	}

	public Cathegory getClients() {
		return clients;
	}
}
package app.model.events;

import app.model.entities.Cathegory;

/**
 * Клас, що представляє подію отримання категорії
"Робітники".
 *
 * @author dan
 *
 */
public class CathegoryWorkersReceived implements
AppEvent {
	private Cathegory workers;

	public CathegoryWorkersReceived(Cathegory
workers) {
		this.workers = workers;
	}

	public Cathegory getWorkers() {
		return workers;
	}
}
package app.model.events;

import javax.swing.UIManager.LookAndFeelInfo;

/**
 * Клас, що сповіщає про необхідність змінення
Look and Feel інтерфейсу.
 *
 * @author dan
 *
 */
public class ChangingLaFNeeded implements AppEvent
{

	private LookAndFeelInfo lookAndFeelInfo;

	public ChangingLaFNeeded(LookAndFeelInfo
lookAndFeelInfo) {
		this.lookAndFeelInfo =
lookAndFeelInfo;
	}

	public LookAndFeelInfo getLookAndFeelInfo()
{
		return lookAndFeelInfo;
	}
}
package app.model.events;

import java.util.List;

/**
 * Клас, що представляє подію отримання сутностей
повного типу.
 *
 * @author dan
 *
 * @param <T>
 */
public class EntitiesReceived<T> implements
AppEvent {
	private List<T> entities;
	private Object viewContext;

	public EntitiesReceived(List<T> entities) {
		this.entities = entities;
	}

	public EntitiesReceived(List<T> entities,
Object viewContext) {
		this.entities = entities;
		this.viewContext = viewContext;
	}

	public List<T> getEntities() {
		return entities;
	}

	public Object getViewContext() {
		return viewContext;
	}

}
package app.model.events;

/**
 * Клас, що сповіщає про виникнення помилки в
программі.
 *
 * @author dan
 *
 */
public class ErrorEvent implements AppEvent {
	private Exception exception;

	public ErrorEvent(Exception exception) {
		this.exception = exception;
	}

	public Exception getException() {
		return exception;
	}
}
package app.model.events;

/**
 * Калс, що сповіщує View про необхідність вивести
певне повідомлення на екран.
 *
 * @author dan
 *
 */
public class MassageEvent implements AppEvent {
	private String massage;

	public MassageEvent(String massage) {
		this.massage = massage;
	}

	public String getMassage() {
		return massage;
	}
}
package app.model.events;

import java.util.List;

import app.model.entities.Cathegory;

/**
 *
 * @author dan
```

```
 *
 */
public class ParentClientsCathegoriesRecieved
implements AppEvent {
	private List<Cathegory> entities;
	private Object viewContext;

	public
ParentClientsCathegoriesRecieved(List<Cathegory>
entities,
			Object viewContext) {
		this.entities = entities;
		this.viewContext = viewContext;
	}

	public List<Cathegory> getEntities() {
		return entities;
	}

	public Object getViewContext() {
		return viewContext;
	}
}
package app.view.defaultview.listeners;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JTextField;

import app.view.defaultview.NewActionDialog;
import configuration.AppConfig;

/**
 * Слухач, що случає кнопку "Choose File" і
відкриває окно вибору файлу, коли
 * кнопка натискається.
 *
 * @author dan
 *
 */
public class ChooseFileButtonListener implements
ActionListener {
	private JTextField choosedFilesField;
	private NewActionDialog dialog;

	public
ChooseFileButtonListener(NewActionDialog dialog,
			JTextField
choosedFilesField) {
		this.choosedFilesField =
choosedFilesField;
		this.dialog = dialog;
	}

	@Override
	public void actionPerformed(ActionEvent e)
{
		JButton button = (JButton)
e.getSource();
		AppConfig config =
AppConfig.getInstance();
		String lastWorkingDir =
config.getProperty("last-working-directory");
		JFileChooser fc = new
JFileChooser(lastWorkingDir);
		fc.setMultiSelectionEnabled(true);
		int returnedVal =
fc.showOpenDialog(button);
		if (returnedVal ==
JFileChooser.APPROVE_OPTION) {
			File[] files =
fc.getSelectedFiles();
			config.setProperty("last-
working-directory", files[0].toString());
			String filesString =
buildFilesString(files);

	choosedFilesField.setText(filesString);

	dialog.setChoosedFiles(files);
		}
	}

	private String buildFilesString(File[]
files) {
		StringBuilder strBuilder = new
StringBuilder();
		for (File file : files) {
			strBuilder.append('"');
			strBuilder.append(file);
			strBuilder.append(" \"");
		}
		return strBuilder.toString();
	}
}
package app.view.defaultview.listeners;

import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;

import app.model.entities.Order;
import app.model.entities.RegUser;
import app.view.defaultview.EntityNode;

/**
 * Клас, що реагує на вибір елементу списку
сутностей.
 *
 * @author dan
 *
 */
public class EntitySelectionListener implements
TreeSelectionListener {

	private JTextArea textArea;

	public EntitySelectionListener(JTextArea
textArea) {
		this.textArea = textArea;
	}

	@Override
	public void valueChanged(TreeSelectionEvent
e) {
		JTree tree = (JTree) e.getSource();
		Object c =
tree.getLastSelectedPathComponent();
		if (!(c instanceof EntityNode)) {
			return;
		}
		EntityNode node = (EntityNode) c;
		Object entity =
node.getUserObject();
		if (entity instanceof RegUser) {
			RegUser user = (RegUser)
entity;
			ifRegUser(user);
		} else if (entity instanceof Order)
```

```
{
                        Order order = (Order)
entity;
                        ifOrder(order);
                }
        }

        private void ifOrder(Order order) {
                textArea.setText(order.getName());
        }

        private void ifRegUser(RegUser user) {
                StringBuilder builder = new
StringBuilder();
                String description =
user.getDescription();
                String email = user.getEmail();
                String login = user.getLogin();
                builder.append("Login: ");
                builder.append(login);
                builder.append('\n');
                builder.append("E-mail: ");
                builder.append(email);
                builder.append('\n');
                builder.append("Description: ");
                builder.append(description);
                builder.append('\n');

        textArea.setText(builder.toString());
        }

}
package app.view.defaultview.listeners;

import javax.swing.JTree;
import javax.swing.event.TreeExpansionEvent;
import javax.swing.event.TreeWillExpandListener;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.ExpandVetoException;
import javax.swing.tree.TreeNode;
import javax.swing.tree.TreePath;

import app.controller.Controller;
import
app.controller.commands.commandset.GetChildEntitie
s;
import
app.controller.exception.IncorrectParameterListExc
eption;
import app.model.entities.Order;
import app.model.events.ErrorEvent;
import app.view.View;
import
app.view.defaultview.DefaultView.ViewContext;
import app.view.defaultview.EntityNode;

/**
 * Клас, що реагує на розкриття елементу дерева.
 *
 * @author dan
 *
 */
public class ETreeWillExpandListener implements
TreeWillExpandListener {

        @Override
        public void
treeWillExpand(TreeExpansionEvent event)
                        throws ExpandVetoException {
                JTree tree = (JTree)
event.getSource();
                TreePath path = event.getPath();
                EntityNode currentNode =
(EntityNode) path.getLastPathComponent();
                if (currentNode.isRoot()) {
                        return;
                }
                Object entity =
currentNode.getUserObject();
                Object user = null;
                if (entity instanceof Order) {
                        user = ((EntityNode)
currentNode.getParent()).getUserObject();
                }
                GetChildEntities getChildEntities =
new GetChildEntities();

        getChildEntities.addParameter("parent",
entity);
                ViewContext context = new
ViewContext(currentNode, tree);


getChildEntities.addParameter("viewContext",
context);
                if
(tree.getName().equalsIgnoreCase("workers")) {

        getChildEntities.addParameter("type", 2);
                }
                if (user != null) {
                        Object[] arr = { user };


getChildEntities.addParameter("requiredEntities",
arr);
                }
                try {


Controller.getInstance().executeCommand(getChildEn
tities);
                } catch
(IncorrectParameterListException ex) {

        View.getInstance().onEvent(new
ErrorEvent(ex));
                }

        }

        @Override
        public void
treeWillCollapse(TreeExpansionEvent event)
                        throws ExpandVetoException {
                JTree tree = (JTree)
event.getSource();
                TreePath path = event.getPath();
                EntityNode currentNode =
(EntityNode) path.getLastPathComponent();
                TreeNode[] removedNodes = new
TreeNode[currentNode.getChildCount()];
                int[] indexes = new
int[removedNodes.length];
                for (int i = 0; i <
removedNodes.length; i++) {
                        removedNodes[i] =
currentNode.getChildAt(i);
                        indexes[i] = i;
                }
                currentNode.removeAllChildren();
                DefaultTreeModel model =
(DefaultTreeModel) tree.getModel();
                model.nodesWereRemoved(currentNode,
```

```
indexes, removedNodes);
                currentNode.addDummy();
                int[] index = { 0 };

        model.nodesWereInserted(currentNode,
index);
        }

}
package app.view.defaultview;

import java.awt.Component;
import java.io.File;

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;

import utils.R;
import app.model.entities.Action;
import app.model.entities.Cathegory;
import app.model.entities.Document;
import app.model.entities.Order;
import app.model.entities.RegUser;

/**
 * Клас, що відповідає за відрисовку елементів
дерева клієнтів.
 *
 * @author dan
 *
 */
public class ClientsTreeCellRenderer extends
DefaultTreeCellRenderer {
        private ImageIcon regUserIcon;
        private ImageIcon orderIcon;
        private ImageIcon actionIcon;
        private JFileChooser fileChooser = new
JFileChooser();
        private ImageIcon cathegoryIcon;

        public ClientsTreeCellRenderer() {
                cathegoryIcon = (ImageIcon)
R.getResource("cathegory-icon");
                regUserIcon = (ImageIcon)
R.getResource("reg-user-icon");
                orderIcon = (ImageIcon)
R.getResource("order-icon");
                actionIcon = (ImageIcon)
R.getResource("action-icon");
        }

        @Override
        public Component
getTreeCellRendererComponent(JTree tree, Object
value,
                        boolean sel, boolean
expanded, boolean leaf, int row,
                        boolean hasFocus) {

        super.getTreeCellRendererComponent(tree,
value, sel, expanded, leaf,
                                row, hasFocus);
                DefaultMutableTreeNode node =
(DefaultMutableTreeNode) value;
                Object entity =
node.getUserObject();
                if (entity instanceof Cathegory &&
cathegoryIcon != null) {
                        setIcon(cathegoryIcon);
                } else if (entity instanceof
RegUser && regUserIcon != null) {
                        setIcon(regUserIcon);
                } else if (entity instanceof Order
&& orderIcon != null) {
                        setIcon(orderIcon);
                } else if (entity instanceof Action
&& actionIcon != null) {
                        setIcon(actionIcon);
                } else if (entity instanceof
Document) {
                        Document doc = (Document)
entity;
                        Icon documentIcon;
                        File file = new
File(doc.getName());
                        documentIcon =
fileChooser.getIcon(file);
                        setIcon(documentIcon);
                }
                return this;
        }
}
package app.view.defaultview;

import java.awt.EventQueue;
import java.awt.Toolkit;
import java.io.PrintWriter;
import java.io.StringWriter;
import
java.lang.reflect.InvocationTargetException;
import java.util.List;

import javax.swing.JOptionPane;
import javax.swing.JTree;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import
javax.swing.UnsupportedLookAndFeelException;
import javax.swing.UIManager.LookAndFeelInfo;
import javax.swing.tree.DefaultTreeModel;

import app.controller.Controller;
import
app.controller.commands.commandset.GetChildEntitie
s;
import
app.controller.exception.IncorrectParameterListExc
eption;
import app.model.entities.Cathegory;
import app.model.entities.Document;
import app.model.entities.Order;
import app.model.events.AppEvent;
import app.model.events.CathegoryClientsReceived;
import app.model.events.CathegoryWorkersReceived;
import app.model.events.ChangingLaFNeeded;
import app.model.events.EntitiesReceived;
import app.model.events.ErrorEvent;
import app.model.events.MassageEvent;
import app.view.View;

/**
 * Клас, що представляє інтерфейс користувача за
замовчуванням. Реагує на події,
 * що генеруються іншими частинами додатку, і
змінює вигляд необхідним чином або
 * виконує необхідні дії.
 *
 * @author dan
 *
 */
```

```
public class DefaultView extends View {

	private MainWindow mainWindow;

	/**
	 * Створює інтерфейс користувача.
	 */
	@Override
	public void start() {
		EventQueue.invokeLater(new
Runnable() {
			public void run() {
				try {
					mainWindow =
new MainWindow();

	mainWindow.fillClientTree();

	mainWindow.fillWorkerTree();

	mainWindow.setVisible(true);
				} catch (Exception e)
{

	e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Реагує на події створені іншими
частинами додатку
	 *
	 * @param e
	 *            - екземпляр класу AppEvent.
Зараз підримуються події
	 *            CathegoryClientsReceived,
CathegoryWorkersReceived,
	 *            EntitiesReceived, ErrorEvent,
MassageEvent, ChangingLaFNeeded.
	 */
	@Override
	public void onEvent(AppEvent e) {
		if (e instanceof
CathegoryClientsReceived) {
			Cathegory cathegory =
((CathegoryClientsReceived) e).getClients();


ifReceivedRootCathegory(mainWindow.getClientsTree(
), cathegory);
		} else if (e instanceof
CathegoryWorkersReceived) {
			Cathegory cathegory =
((CathegoryWorkersReceived) e).getWorkers();


ifReceivedRootCathegory(mainWindow.getWorkersTree(
), cathegory);
		} else if (e instanceof
EntitiesReceived) {

	ifReceivedEntities((EntitiesReceived<?>)
e);
		} else if (e instanceof ErrorEvent)
{
			ifError(e);
		} else if (e instanceof
MassageEvent) {
			MassageEvent errorEvent =
(MassageEvent) e;
System.out.println(errorEvent.getMassage());
		} else if (e instanceof
ChangingLaFNeeded) {
			ChangingLaFNeeded ev =
(ChangingLaFNeeded) e;
			LookAndFeelInfo info =
ev.getLookAndFeelInfo();
			try {


UIManager.setLookAndFeel(info.getClassName());


SwingUtilities.updateComponentTreeUI(mainWindow);
				// mainWindow.pack();
			} catch
(ClassNotFoundException | InstantiationException
				|
IllegalAccessException |
UnsupportedLookAndFeelException ex) {
				onEvent(new
ErrorEvent(ex));
			}
		}
	}

	private void ifReceivedRootCathegory(JTree
tree, Cathegory cathegory) {
		final EntityNode root = new
EntityNode(cathegory);
		final DefaultTreeModel
clientsTreeModel = new DefaultTreeModel(root);
		tree.setModel(clientsTreeModel);
		tree.setRootVisible(false);
		GetChildEntities getChildEntities =
new GetChildEntities();

	getChildEntities.addParameter("parent",
cathegory);
		ViewContext context = new
ViewContext(root, tree);


getChildEntities.addParameter("viewContext",
context);
		try {


Controller.getInstance().executeCommand(getChildEn
tities);
		} catch
(IncorrectParameterListException ex) {
			onEvent(new ErrorEvent(ex));
		}
		mainWindow.revalidate();
	}

	private void ifError(AppEvent e) {
		ErrorEvent errorEvent =
(ErrorEvent) e;
		StringWriter sw = new
StringWriter();
		PrintWriter pw = new
PrintWriter(sw);
		Exception ex =
errorEvent.getException();
		ex.printStackTrace(pw);
		Toolkit.getDefaultToolkit().beep();

	JOptionPane.showMessageDialog(mainWindow,
```

```
sw, "Error!",
                JOptionPane.ERROR_MESSAGE);
        }

        @SuppressWarnings("unchecked")
        private <T> void
ifReceivedEntities(EntitiesReceived<T> e) {
                List<Object> entities =
(List<Object>) e.getEntities();
                if (entities.isEmpty()) {
                        return;
                }
                Object viewContext =
e.getViewContext();
                if
(viewContext.equals(NewActionDialog.class)) {
                        ifNewAction(entities);
                } else {
                        placeEntities(entities,
(ViewContext) e.getViewContext());
                }
        }

        private void ifNewAction(List<Object>
entities) {
                NewActionDialog dialog =
mainWindow.getNewActionDialog();

        dialog.fillOrders(entities.toArray(new
Order[0]));
                dialog.setVisible(true);
        }

        private void placeEntities(List<Object>
entities, final ViewContext context) {
                final EntityNode node =
context.getNode();
                node.removeDummy();
                final JTree tree =
context.getTree();
                final DefaultTreeModel treeModel =
(DefaultTreeModel) tree.getModel();
                for (final Object entity :
entities) {
                        final EntityNode newNode =
new EntityNode(entity);
                        try {

        SwingUtilities.invokeAndWait(new Runnable()
{
                                        public void
run() {
                                                int n
= node.getChildCount();

        node.insert(newNode, n);
                                                if (!
(entity instanceof Document)) {

        newNode.addDummy();
                                                }
                                        }
                                });
                        } catch
(InvocationTargetException | InterruptedException
e) {

        View.getInstance().onEvent(new
ErrorEvent(e));
                        }
                }
                treeModel.reload(node);
        }

        public MainWindow getMainWindow() {
                return mainWindow;
        }

        public static class ViewContext {
                private EntityNode node;
                private JTree tree;

                public ViewContext(EntityNode node,
JTree tree) {
                        this.node = node;
                        this.tree = tree;
                }

                public EntityNode getNode() {
                        return node;
                }

                public JTree getTree() {
                        return tree;
                }
        }
}
package app.view.defaultview;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPopupMenu;
import javax.swing.JTree;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;

import app.controller.Controller;
import
app.controller.commands.commandset.RemoveEntity;
import
app.controller.exception.IncorrectParameterListExc
eption;
import app.model.events.ErrorEvent;
import app.view.View;

/**
 * Клас, що представляє контекстне меню елемента
дерева. Реагує на події
 * викликані елементами цього меню.
 *
 * @author dan
 *
 */
public class EntityMenu extends JPopupMenu
implements ActionListener {
        private JMenuItem remove;
        private JTree tree;
        private TreePath selPath;

        public EntityMenu(JTree tree, TreePath
selPath) {
                this.selPath = selPath;
                this.tree = tree;
                remove = new JMenuItem("Remove");
                remove.addActionListener(this);
                add(remove);
        }

        /**
```

```
         * Реагує на натискання пунктів
контекстного меню.
         *
         * @param e
         *            - об'єкт подія, що
створюється пунктом меню.
         * @see ActionListener
         */
        @Override
        public void actionPerformed(ActionEvent e)
{
                EntityNode node = (EntityNode)
selPath.getLastPathComponent();
                Object entity =
node.getUserObject();
                MainWindow mw = ((DefaultView)
View.getInstance()).getMainWindow();
                Toolkit.getDefaultToolkit().beep();
                int option =
JOptionPane.showConfirmDialog(mw,
                                "Remove " + entity +
'?', "Confirm removing",

        JOptionPane.OK_CANCEL_OPTION);
                if (option == 2) {
                        return;
                }
                RemoveEntity removeEntity = new
RemoveEntity();
                removeEntity.addParameter("entity",
entity);
                EntityNode parent = (EntityNode)
node.getParent();
                int nodeIndex =
parent.getIndex(node);
                node.removeFromParent();
                DefaultTreeModel model =
(DefaultTreeModel) tree.getModel();
                if (parent != null) {
                        int[] indexes = {
nodeIndex };
                        Object[] removedChildren = {
node };

        model.nodesWereRemoved(parent, indexes,
removedChildren);
                        if (parent.getChildCount()
<= 0) {
                                parent.addDummy();
                                int[] index = { 0 };

        model.nodesWereInserted(parent, index);
                        }
                }
                try {


Controller.getInstance().executeCommand(removeEnti
ty);
                } catch
(IncorrectParameterListException ex) {

        View.getInstance().onEvent(new
ErrorEvent(ex));
                }
        }
}
package app.view.defaultview;

import javax.swing.tree.DefaultMutableTreeNode;

/**
 * Клас, що представляє елемент дерева сутностей.
 *
 * @author dan
 *
 */
public class EntityNode extends
DefaultMutableTreeNode {
        private int dummyNodeIndex = -1;

        /**
         * Конструктор, що створює вузел дерева, що
представляє об'єкт object.
         *
         * @param object
         *            - об'єкт, що представляється
цим вузлом.
         * @see DefaultMutableTreeNode
         */
        public EntityNode(Object object) {
                super(object);
        }

        /**
         * Видаляє елемент-заглушку даного
елемента. Якщо елемента-заглушки немає,
         * нічого не робить.
         */
        public void removeDummy() {
                if (dummyNodeIndex >= 0) {
                        remove(dummyNodeIndex);
                        dummyNodeIndex = -1;
                }
        }

        /**
         * Додає до данго елемента елемент-
заглушку.
         */
        public void addDummy() {
                dummyNodeIndex = getChildCount();
                insert(new
DefaultMutableTreeNode(new DummyNode()),
dummyNodeIndex);
        }

        private class DummyNode {
                @Override
                public String toString() {
                        return "[Empty]";
                }
        }
}
package app.view.defaultview;

import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.concurrent.TimeUnit;

import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.border.EmptyBorder;
import javax.swing.border.TitledBorder;
```

```
import app.controller.Controller;
import
app.controller.commands.commandset.GetCathegoryCli
ents;
import
app.controller.commands.commandset.GetCathegoryWor
kers;
import
app.controller.exception.IncorrectParameterListExc
eption;
import app.model.events.ErrorEvent;
import app.view.View;
import
app.view.defaultview.listeners.ETreeWillExpandList
ener;
import
app.view.defaultview.listeners.EntitySelectionList
ener;
import
app.view.defaultview.listeners.MenuItemListener;
import
app.view.defaultview.listeners.TreeMouseListener;
import dao.DAOProviderFactory;
import dao.exception.DAOException;

/**
 * Головне вікно інтерфейсу за замовчуванням.
Містить всі основні елементи
 * управління.
 *
 * @author dan
 *
 */
public class MainWindow extends JFrame {
        private JTree workersTree;
        private JTextArea workerDescription;
        private JTree clientsTree;
        private JTextArea clientDescription;
        private NewActionDialog newActionDialog;

        /**
         * Виконує роботу по побудові головного
вікна.
         */
        public MainWindow() {


setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 600, 458);

                JMenuBar menuBar = new JMenuBar();
                setJMenuBar(menuBar);

                JMenu toolsMenu = new
JMenu("Tools");
                menuBar.add(toolsMenu);

                JMenuItem newActionItem = new
JMenuItem("New action");
                MenuItemListener menuItemListener =
new MenuItemListener(this);


newActionItem.addActionListener(menuItemListener);
                toolsMenu.add(newActionItem);

                JMenuItem settingsItem = new
JMenuItem("Settings");
                toolsMenu.add(settingsItem);
settingsItem.addActionListener(menuItemListener);

                JMenu helpMenu = new JMenu("Help");
                menuBar.add(helpMenu);

                JMenuItem helpItem = new
JMenuItem("Help");
                helpMenu.add(helpItem);

                JMenuItem aboutItem = new
JMenuItem("About");
                helpMenu.add(aboutItem);


aboutItem.addActionListener(menuItemListener);

                JPanel contentPane = new JPanel();
                contentPane.setBorder(new
EmptyBorder(5, 5, 5, 5));
                contentPane.setLayout(new
BorderLayout(0, 0));
                setContentPane(contentPane);

                JTabbedPane tabbedPane = new
JTabbedPane(JTabbedPane.TOP);
                contentPane.add(tabbedPane,
BorderLayout.CENTER);

                JSplitPane workersPane = new
JSplitPane();

        workersPane.setOneTouchExpandable(true);

        workersPane.setContinuousLayout(true);

        workersPane.setDividerLocation(200);

                JScrollPane scrollPane_1 = new
JScrollPane();
                scrollPane_1.setViewportBorder(new
TitledBorder(null, "Workers Tree",
                                TitledBorder.LEADING,
TitledBorder.TOP, null, null));

        workersPane.setLeftComponent(scrollPane_1);

                workersTree = new JTree();
                WorkersTreeCellRenderer
workersRenderer = new WorkersTreeCellRenderer();


workersTree.setCellRenderer(workersRenderer);
                workersTree.setName("workers");

        scrollPane_1.setViewportView(workersTree);

                JScrollPane scrollPane_3 = new
JScrollPane();
                scrollPane_3.setViewportBorder(new
TitledBorder(null, "Description",
                                TitledBorder.LEADING,
TitledBorder.TOP, null, null));


workersPane.setRightComponent(scrollPane_3);

                workerDescription = new
JTextArea();
                EntitySelectionListener
workersSelectionListener = new
EntitySelectionListener(
                                workerDescription);
```

```
                //
workerDescription.setPreferredSize(new
Dimension(450, 100));


workersTree.addTreeSelectionListener(workersSelect
ionListener);


scrollPane_3.setViewportView(workerDescription);

                JSplitPane clientsPane = new
JSplitPane();

        clientsPane.setContinuousLayout(true);

        clientsPane.setOneTouchExpandable(true);

        clientsPane.setDividerLocation(200);

                tabbedPane.addTab("Clients", null,
clientsPane, null);
                tabbedPane.addTab("Workers", null,
workersPane, null);

                JScrollPane scrollPane = new
JScrollPane();
                scrollPane.setViewportBorder(new
TitledBorder(null, "Clients Tree",
                                TitledBorder.LEADING,
TitledBorder.TOP, null, null));

        clientsPane.setLeftComponent(scrollPane);

                clientsTree = new JTree();
                ClientsTreeCellRenderer
clientsRenderer = new ClientsTreeCellRenderer();


clientsTree.setCellRenderer(clientsRenderer);
                clientsTree.addMouseListener(new
TreeMouseListener());
                ETreeWillExpandListener
expandationListener = new
ETreeWillExpandListener();
                clientsTree.setName("clients");


clientsTree.addTreeWillExpandListener(expandationL
istener);


workersTree.addTreeWillExpandListener(expandationL
istener);
                workersTree.addMouseListener(new
TreeMouseListener());

        scrollPane.setViewportView(clientsTree);

                JScrollPane scrollPane_2 = new
JScrollPane();
                scrollPane_2.setViewportBorder(new
TitledBorder(null, "Description",
                                TitledBorder.LEADING,
TitledBorder.TOP, null, null));


clientsPane.setRightComponent(scrollPane_2);

                clientDescription = new
JTextArea();
                EntitySelectionListener
clientSelectionListener = new
EntitySelectionListener(
                                clientDescription);
                //
clientDescription.setPreferredSize(new
Dimension(450, 100));


clientsTree.addTreeSelectionListener(clientSelecti
onListener);


scrollPane_2.setViewportView(clientDescription);

                addWindowListener(new
WindowAdapter() {
                        @Override
                        public void
windowClosing(WindowEvent e) {
                                try {

        Controller.getInstance().shutdown();


Controller.getInstance().awaitTermination(5,

        TimeUnit.SECONDS);


DAOProviderFactory.getDAOProvider().close();
                                } catch (DAOException
| InterruptedException ex) {

        View.getInstance().onEvent(new
ErrorEvent(ex));
                                }
                        }
                });
                setLocationRelativeTo(null);

        }

        /**
         * Виконує початкове заповнення дерева
клієнтів.
         */
        public void fillClientTree() {
                GetCathegoryClients
getCathegoryClients = new GetCathegoryClients();
                try {


Controller.getInstance().executeCommand(getCathego
ryClients);
                } catch
(IncorrectParameterListException e) {

        View.getInstance().onEvent(new
ErrorEvent(e));
                }
        }

        /**
         * Виконує початкове заповнення дерева
працівників.
         */
        public void fillWorkerTree() {
                GetCathegoryWorkers
getCathegoryWorkers = new GetCathegoryWorkers();
                try {
```

```
Controller.getInstance().executeCommand(getCathegoryWorkers);
                } catch (IncorrectParameterListException e) {

                View.getInstance().onEvent(new ErrorEvent(e));
                }
        }

        /**
         * Створює діалог фіксування нової дії у БД.
         *
         * @return
         */
        public NewActionDialog getNewActionDialog() {
                newActionDialog = new NewActionDialog();
                newActionDialog.setTitle("New Action");


newActionDialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
                return newActionDialog;
        }

        public JTree getClientsTree() {
                return clientsTree;
        }

        public JTree getWorkersTree() {
                return workersTree;
        }
}
package app.view.defaultview;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.util.Date;
import java.util.UUID;

import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

import app.controller.Controller;
import app.controller.commands.commandset.DoActionAsManager;
import app.controller.exception.IncorrectParameterListException;
import app.model.entities.Action;
import app.model.entities.Document;
import app.model.entities.Order;
import app.model.events.ErrorEvent;
import app.view.View;
import app.view.defaultview.listeners.ChooseFileButtonListener;
import dao.localstorage.AppLocalStorage;
import dao.localstorage.LocalStorageException;

public class NewActionDialog extends JDialog implements ActionListener {

        private final JPanel contentPanel = new JPanel();
        private JTextField textField;
        private JComboBox<Order> ordersComboBox;
        private File[] choosedFiles;

        /**
         * Виконує роботу по побудові даного діалоу.
         */
        public NewActionDialog() {
                setBounds(100, 100, 461, 288);
                getContentPane().setLayout(new BorderLayout());
                contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
                getContentPane().add(contentPanel, BorderLayout.CENTER);
                contentPanel.setLayout(new GridLayout(0, 3, 0, 30));
                {
                        JLabel lblOrder = new JLabel("Order");
                        contentPanel.add(lblOrder);
                }
                {
                        ordersComboBox = new JComboBox();

        contentPanel.add(ordersComboBox);
                }
                contentPanel.add(Box.createGlue());
                {
                        Component glue = Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        Component glue = Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        Component glue = Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        JLabel lblDocuments = new JLabel("Documents");

        contentPanel.add(lblDocuments);
                }
                {
                        textField = new JTextField();
                        contentPanel.add(textField);
                        textField.setColumns(10);
                }
                {
                        JButton btnChooseFiles = new JButton("Choose files...");

        btnChooseFiles.addActionListener(new
```

```
ChooseFileButtonListener(this,
                                        textField));
        contentPanel.add(btnChooseFiles);
                }
                {
                        Component glue =
Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        Component glue =
Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        Component glue =
Box.createGlue();
                        contentPanel.add(glue);
                }
                {
                        JPanel buttonPane = new
JPanel();
                        FlowLayout fl_buttonPane =
new FlowLayout(FlowLayout.RIGHT);
        buttonPane.setLayout(fl_buttonPane);
        getContentPane().add(buttonPane,
BorderLayout.SOUTH);
                        {
                                JButton okButton =
new JButton("OK");
        okButton.setActionCommand("OK");
        buttonPane.add(okButton);
        getRootPane().setDefaultButton(okButton);
        okButton.addActionListener(this);
                        }
                        {
                                JButton cancelButton
= new JButton("Cancel");
        cancelButton.setActionCommand("Cancel");
        buttonPane.add(cancelButton);
        cancelButton.addActionListener(this);
                        }
                }
        }

        /**
         * Заповнює список замовлень.
         *
         * @param orders
         *             - замовлення, якими
заповнюється список заказів діалогового
         *             вікна.
         */
        public void fillOrders(Order[] orders) {
                ordersComboBox.removeAllItems();
                for (Order order : orders) {
        ordersComboBox.addItem(order);
                }
        }

        /**
         * Змінює набір файлів, що вибрав
користувач.
         *
         * @param files
         */
        public void setChoosedFiles(File[] files) {
                this.choosedFiles = files;
        }

        /**
         * @see ActionListener
         */
        @Override
        public void actionPerformed(ActionEvent e)
{
                String command =
e.getActionCommand();
                if (command.equalsIgnoreCase("OK"))
{
                        try {
                                try {
                                        ifOk();
                                } catch (IOException
| LocalStorageException ex) {
        View.getInstance().onEvent(new
ErrorEvent(ex));
                                }
                        } catch
(IncorrectParameterListException ex) {
        View.getInstance().onEvent(new
ErrorEvent(ex));
                        }
                } else if
(command.equalsIgnoreCase("Cancel")) {
                        dispose();
                }
        }

        private void ifOk() throws
IncorrectParameterListException, IOException,
                        LocalStorageException {
                Order order = (Order)
ordersComboBox.getSelectedItem();
                UUID id = UUID.randomUUID();
                Date date = new
Date(System.currentTimeMillis());
                Action newAction = new Action(id,
order.getId(), null, null, date);
                Document[] documents = new
Document[choosedFiles.length];
                for (int i = 0; i <
choosedFiles.length; i++) {
                        File file = choosedFiles[i];
                        UUID docId =
UUID.randomUUID();
                        String name =
file.getName();
                        documents[i] = new
Document(docId, name, newAction.getId());

AppLocalStorage.getStorage().attachExternalLocal(
        documents[i].getFileHandler(), file);
                }
                DoActionAsManager doActionAsManager
= new DoActionAsManager();
        doActionAsManager.addParameter("Action",
newAction);
```

```
        doActionAsManager.addParameter("Documents",
documents);


Controller.getInstance().executeCommand(doActionAs
Manager);
                dispose();// TODO mb make reusable
        }
}
package app.view.defaultview;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.SpinnerNumberModel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;
import javax.swing.border.EmptyBorder;
import javax.xml.stream.XMLStreamException;

import net.miginfocom.swing.MigLayout;
import app.model.events.ChangingLaFNeeded;
import app.model.events.ErrorEvent;
import app.view.View;
import configuration.AppConfig;
import configuration.confutils.IConfig;

/**
 * Діалог налаштувань програми.
 *
 * @author dan
 *
 */
public class SettingsDialog extends JDialog
implements ActionListener {

        private final JPanel contentPanel = new
JPanel();
        private JTextField usernameField;
        private JPasswordField passwordField;
        private JTextField hostField;
        private JTextField dbNameField;
        private JTextField storageFolderField;
        private JSpinner portSpinner;
        private JComboBox<String> lookNFeels;

        /**
         * @see JDialog
         */
        public SettingsDialog() {
                createGUI();
        }

        private void createGUI() {

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
                setBounds(100, 100, 450, 300);
                setTitle("Settings");
                getContentPane().setLayout(new
BorderLayout());
                contentPanel.setBorder(new
EmptyBorder(5, 5, 5, 5));
                getContentPane().add(contentPanel,
BorderLayout.CENTER);
                contentPanel
                                .setLayout(new
MigLayout("", "[][][grow]", "[][][][][][][][]"));
                {
                        JLabel lblUsername = new
JLabel("Username");

        contentPanel.add(lblUsername, "cell 0 0");
                }
                {
                        usernameField = new
JTextField();

        contentPanel.add(usernameField, "cell 2
0,growx");

        usernameField.setColumns(10);
                }
                {
                        JLabel lblPassword = new
JLabel("Password");

        contentPanel.add(lblPassword, "cell 0 1");
                }
                {
                        passwordField = new
JPasswordField();

        contentPanel.add(passwordField, "cell 2
1,growx");
                }
                {
                        JLabel lblDbHost = new
JLabel("DB host");
                        contentPanel.add(lblDbHost,
"cell 0 2");
                }
                {
                        hostField = new
JTextField();
                        contentPanel.add(hostField,
"cell 2 2,growx");
                        hostField.setColumns(10);
                }
                {
                        JLabel lblDbPort = new
JLabel("DB port");
                        contentPanel.add(lblDbPort,
"cell 0 3");
                }
                {
                        portSpinner = new
JSpinner();
                        portSpinner.setModel(new
SpinnerNumberModel(new Integer(3306),
                                        null, null,
new Integer(1)));

        contentPanel.add(portSpinner, "cell 2 3");
                }
                {
                        JLabel lblDbName = new
JLabel("DB name");
                        contentPanel.add(lblDbName,
"cell 0 4");
                }
                {
```

```
                        dbNameField = new
JTextField();

        contentPanel.add(dbNameField, "cell 2
4,growx");
                        dbNameField.setColumns(10);
                }
                {
                        JLabel lblTempStorageFolder
= new JLabel("Temp storage folder");

        contentPanel.add(lblTempStorageFolder,
"cell 0 5");
                }
                {
                        storageFolderField = new
JTextField();

        contentPanel.add(storageFolderField, "cell
2 5,growx");

        storageFolderField.setColumns(10);
                }
                {
                        JLabel lblLookAndFeel = new
JLabel("Look and Feel");

        contentPanel.add(lblLookAndFeel, "cell 0
6");
                }
                {
                        lookNFeels = new
JComboBox<>();
                        contentPanel.add(lookNFeels,
"cell 2 6,growx");
                }
                {
                        JPanel buttonPane = new
JPanel();
                        buttonPane.setLayout(new
FlowLayout(FlowLayout.RIGHT));

        getContentPane().add(buttonPane,
BorderLayout.SOUTH);
                        {
                                JButton okButton =
new JButton("OK");

        okButton.setActionCommand("OK");

        buttonPane.add(okButton);

        getRootPane().setDefaultButton(okButton);

        okButton.addActionListener(this);

        okButton.setActionCommand("apply");
                        }
                        {
                                JButton cancelButton
= new JButton("Cancel");

        cancelButton.setActionCommand("Cancel");

        buttonPane.add(cancelButton);

        cancelButton.addActionListener(this);

        cancelButton.setActionCommand("cancel");
                        }
                }
        }
```

```
        /**
         * Ініціалізує даний діалог значеннями із
поточної конфігурації додатку.
         */
        public void getCurrentValues() {
                IConfig config =
AppConfig.getInstance();


usernameField.setText(config.getProperty("username
"));
passwordField.setText(config.getProperty("password
"));
hostField.setText(config.getProperty("host"));dbNa
meField.setText(config.getProperty("db"));
storageFolderField.setText(config.getProperty("loc
al-storage"));
portSpinner.setValue(Integer.parseInt(config.getPr
operty("port")));
                LookAndFeelInfo[] lafInfos =
package dao;
import java.io.File;
import java.util.UUID;

import dao.exception.DAOException;

/**
 * Клас який зберігає інформацію про бінарний
файл, необхідну для його ленивого
 * завантаження з бази даних у локальне сховище.
 *
 * @author dan
 *
 */
public interface BlobHandler {
        File getFile() throws DAOException;

        String getFilename();

        String getTable();

        UUID getId();

        String getField();
}
package dao;

import java.util.UUID;

import configuration.AppConfig;
import dao.sqlcrud.SQLBlobHandlerFactory;

/**
 * Клас, який надає необхідний екземпляр
BlobHandler відповідно до виду бази
 * даних.
 *
 * @author dan
 *
 */
public abstract class BlobHandlerFactory {
        private static BlobHandlerFactory instance;

        public static synchronized
BlobHandlerFactory getInstance() {
                if (instance == null) {
                        AppConfig appConfig =
AppConfig.getInstance();
                        String daoProvider =
appConfig.getProperty("dao-provider");
                        if
```

```
(daoProvider.toLowerCase().equals("sql")) {
                                instance = new
SQLBlobHandlerFactory();
                        }
                }
                return instance;
        }

        /**
         * Створює об'єкт класу BlobHandler.
         * @param table
         * @param id
         * @param field
         * @param filename
         * @return
         */
        public abstract BlobHandler
createBlobHandler(String table, UUID id,
                        String field, String
filename);
}
package dao;

import java.util.List;
import java.util.UUID;

import dao.exception.DAOException;

/**
 * Інтерфес, для доступу і маніпулювання базою
данних.
 *
 * @author dan
 *
 */
public interface CRUDInterface {
        /**
         * Відкриває з'єднання з базою даних
         *
         * @throws DAOException
         */
        void open() throws DAOException;

        /**
         * Закриває з'єднання з базою даних
         *
         * @throws DAOException
         */
        void close() throws DAOException;

        /**
         * Зберігає об'єкт-сутність у БД.
         *
         * @param instance
         * @return
         * @throws DAOException
         */
        <T> T insert(T instance) throws
DAOException;

        /**
         * Зчитує об'экт певного класу-сутності з
БД з певним id.
         *
         * @param entityClass
         * @param id
         * @return
         * @throws DAOException
         */
        <T> T read(Клас entityClass, UUID id)
throws DAOException;

        /**
         * Поновлює об'єкт-сутність у БД.
         *
         * @param instance
         * @throws DAOException
         */
        <T> void update(T instance) throws
DAOException;

        /**
         * Видалити об'єкт-сутність із БД.
         *
         * @param instance
         * @throws DAOException
         */
        <T> void delete(T instance) throws
DAOException;

        /**
         * Зчитує всі об'экти певного класу-
сутності з БД, які задовільняють певну
         * умову.
         *
         * @param entityClass
         * @param filter
         *            - фільтр, за допомогою якого
фільтруються сутності.
         * @return
         * @throws DAOException
         */
        <T> List<T> select(Клас entityClass,
DAOFilter filter) throws DAOException;

        /**
         * Отримує всі об'єкти певної сутності, які
отримані за допомогою SQLSring.
         *
         * @param SQLString
         * @param intitiesClass
         * @return
         * @throws DAOException
         */
        <T> List<T> select(String SQLString,
Клас<T> intitiesClass)
                        throws DAOException;

}
package dao;

/**
 * Экземпляри класів, що реалізують цей інтерфейс
дозволяють відфільтровувати
 * класи-сутності.
 *
 * @author dan
 *
 */
public interface DAOFilter {
        /**
         *
         * @param entity
         * @return true, якщо сутність задовільняє
деяким крітеріям.
         */
        public <T> boolean accept(T entity);
}
package dao;

import java.sql.DriverManager;
import java.sql.SQLException;

import configuration.AppConfig;
```

```
import dao.exception.DAOException;
import
dao.exception.DAOInvalidProviderTypeException;
import dao.sqlcrud.SQLCloser;
import dao.sqlcrud.SQLDeleter;
import dao.sqlcrud.SQLInsert;
import dao.sqlcrud.SQLOpener;
import dao.sqlcrud.SQLReader;
import dao.sqlcrud.SQLSelector;
import dao.sqlcrud.SQLUpdater;

/**
 * Цей клас надає можливість отримати необхідний
екземпляр CRUDIntarface
 * відповідно до конфіґура ції додатку.
 *
 * @author dan
 *
 */
public class DAOProviderFactory {
	private static CRUDInterface crud;

	public static synchronized CRUDInterface
getDAOProvider()
			throws DAOException {
		if (crud != null) {
			return crud;
		}
		AppConfig appConfig =
AppConfig.getInstance();
		String daoProvider =
appConfig.getProperty("dao-provider");
		if (daoProvider.equals("sql")) {
			try {

	DriverManager.registerDriver(new
org.mariadb.jdbc.Driver());
				SQLOpener opener =
new SQLOpener();
				SQLReader reader =
new SQLReader();
				SQLInsert inserter =
new SQLInsert();
				SQLSelector selector
= new SQLSelector();
				SQLUpdater updater =
new SQLUpdater();
				SQLDeleter deleter =
new SQLDeleter();
				SQLCloser closer =
new SQLCloser();

	opener.setNextCRUD(reader);

	reader.setNextCRUD(inserter);

	inserter.setNextCRUD(selector);

	selector.setNextCRUD(updater);

	updater.setNextCRUD(deleter);

	deleter.setNextCRUD(closer);
				crud = opener;
				return crud;
			} catch (SQLException e) {
				throw new
DAOException(e.getMessage());
			}
		}
		throw new
DAOInvalidProviderTypeException();
	}
}
package dao;

import java.util.List;
import java.util.UUID;

import dao.exception.DAOException;

/**
 * Клас, який дозволяє організувати список
виконавців операцій CRUDInterface.
 *
 * @author dan
 *
 */
public abstract class TransparentCRUD implements
CRUDInterface {
	private CRUDInterface nextCRUD = null;

	public TransparentCRUD() {
	}

	public TransparentCRUD(CRUDInterface
nextCRUD) {
		super();
		this.nextCRUD = nextCRUD;
	}

	public CRUDInterface getNextCRUD() {
		return nextCRUD;
	}

	public void setNextCRUD(TransparentCRUD
nextCRUD) {
		this.nextCRUD = nextCRUD;
	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public <T> T insert(T instance) throws
DAOException {
		if (nextCRUD != null) {
			return
nextCRUD.insert(instance);
		}
		return null;
	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public <T> T read(Клас entityClass, UUID
id) throws DAOException {
		if (nextCRUD != null) {
			return
nextCRUD.read(entityClass, id);
		}
		return null;
	}

	/**
	 * {@inheritDoc}
	 */
	@Override
	public <T> void update(T instance) throws
DAOException {
		if (nextCRUD != null) {
			nextCRUD.update(instance);
```

```
                }
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public <T> void delete(T instance) throws
DAOException {
                if (nextCRUD != null) {
                        nextCRUD.delete(instance);
                }
        }

        @Override
        public <T> List<T> select(Клас entityClass,
DAOFilter filter)
                        throws DAOException {
                if (nextCRUD != null) {
                        return
nextCRUD.select(entityClass, filter);
                }
                return null;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public <T> List<T> select(String SQLString,
Клас<T> entitiesClass)
                        throws DAOException {
                if (nextCRUD != null) {
                        return
nextCRUD.select(SQLString, entitiesClass);
                }
                return null;
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public void open() throws DAOException {
                if (nextCRUD != null) {
                        nextCRUD.open();
                }
        }

        /**
         * {@inheritDoc}
         */
        @Override
        public void close() throws DAOException {
                if (nextCRUD != null) {
                        nextCRUD.close();
                }
        }

}
package utils;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.HashMap;

import javax.swing.ImageIcon;

/**
 * Клас, що надає доступ до ресурсів програми.
 *
 * @author dan
 *
 */
public final class R {

        private R() {
        }

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

        static {
                String config = null;
                try {
                        config = getConfig();
                } catch (IOException e) {
                        e.printStackTrace();
                        System.exit(1);
                }
                res.put("config", config);
                putIcons();
        }

        /**
         * Надає ресурс за ключем.
         *
         * @param key
         *            - унікальний ключ ресурса.
         * @return об'єкт-ресурс.
         */
        public static Object getResource(String
key) {
                return res.get(key);
        }

        private static void putIcons() {
                URL url =
R.class.getClassLoader().getResource("category-
small.png");
                if (url != null) {
                        res.put("cathegory-icon",
new ImageIcon(url));
                }
                url =
R.class.getClassLoader().getResource("client-
small.png");
                if (url != null) {
                        res.put("reg-user-icon", new
ImageIcon(url));
                }
                url =
R.class.getClassLoader().getResource("order-
small.png");
                if (url != null) {
                        res.put("order-icon", new
ImageIcon(url));
                }
                url =
R.class.getClassLoader().getResource("action-
small.png");
                if (url != null) {
                        res.put("action-icon", new
ImageIcon(url));
                }
                url =
R.class.getClassLoader().getResource("worker.png")
;
                if (url != null) {
                        res.put("worker-icon", new
ImageIcon(url));
```

```
            }

        }

        private static String getConfig() throws
IOException {
            String path =
ClassLoader.getSystemClassLoader().getResource("."
)
                            .getPath();
            File config = new File(new
File(path).getAbsoluteFile() + "/config.xml");
            if (!config.exists()) {
                    URL url =
R.class.getClassLoader().getResource("config.xml")
;
                    InputStream is =
url.openStream();
                    FileOutputStream fos = new
FileOutputStream(config);
                    byte[] buffer = new
byte[512];
                    int len;
                    while ((len =
is.read(buffer)) != -1) {
                            fos.write(buffer, 0,
len);
                    }
                    is.close();
                    fos.close();
            }
            return config.getPath();
        }
}
```