МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

автоматизованих систем обробки інформації та управління

**Пояснювальна записка**

до курсової роботи

з дисципліни

“Об’єктно-орієнтовне програмування 2 ”

на тему

"Редактор будівельних креслень"

|  |  |  |
| --- | --- | --- |
| Керівник : Подрубайло О. О. |  | Виконав : Канівець Дмитро Володимирович |
| Допущений до захисту |  | студент гр. ІП-41, ФІОТ |
| І\_\_\_І \_\_\_\_\_\_\_\_ 201\_ \_\_\_\_\_\_  підпис |  | 2 курс  № IП-4111 |
| Захистив з оцінкою  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_  оцінка підпис  І\_\_\_І \_\_\_\_\_\_\_\_\_ 201\_ |  | Підпис: |

Київ 2016

**ЗМІСТ**

[ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 4](#_Toc450957316)

[ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ 7](#_Toc450957317)

[Робота з файлами та XML 10](#_Toc450957318)

[Робота з фігурами 10](#_Toc450957319)

[Робота з правилами 13](#_Toc450957320)

[Робота з кресленнями 13](#_Toc450957321)

[User interface 14](#_Toc450957322)

[Загальна структура 15](#_Toc450957323)

[Висновок 17](#_Toc450957324)

[РЕАЛІЗАЦИЯ ПРОГРАМНОГО КОМПЛЕКСУ 18](#_Toc450957325)

[Допоміжні класи 18](#_Toc450957326)

[PriorityQueue 18](#_Toc450957327)

[Представлення малюнка 18](#_Toc450957328)

[Command 18](#_Toc450957329)

[KeyboardState 18](#_Toc450957330)

[Drawable 19](#_Toc450957331)

[Scene 19](#_Toc450957332)

[SceneManager 20](#_Toc450957333)

[Фігури 21](#_Toc450957334)

[BuilderFomXML 21](#_Toc450957335)

[Figure 21](#_Toc450957336)

[ImmutableFigure 22](#_Toc450957337)

[LineFigure 22](#_Toc450957338)

[FiguresLoader 23](#_Toc450957339)

[FiguresManager 23](#_Toc450957340)

[Правила 24](#_Toc450957341)

[Rule 24](#_Toc450957342)

[ContainerRule 24](#_Toc450957343)

[PriorityRule 24](#_Toc450957344)

[RulesLoader 25](#_Toc450957345)

[RulesManager 25](#_Toc450957346)

[Файли 25](#_Toc450957347)

[XMLTag 25](#_Toc450957348)

[XMLWriter 26](#_Toc450957349)

[XMLReader 26](#_Toc450957350)

[XMLBuilder 26](#_Toc450957351)

[User interface 26](#_Toc450957352)

[Canvas 26](#_Toc450957353)

[MainWindow 27](#_Toc450957354)

[AboutWindow 28](#_Toc450957355)

[Main 28](#_Toc450957356)

[Додавання фігури до креслення 29](#_Toc450957357)

[Завантаження креслення 31](#_Toc450957358)

[Завантаження пакету фігур 32](#_Toc450957359)

[Висновок 33](#_Toc450957360)

[ОПИС ЗАСТОСУВАННЯ СИСТЕМИ 34](#_Toc450957361)

[Висновок 40](#_Toc450957362)

[ВИСНОВОК ДО КУРСОВОЇ РОБОТИ 41](#_Toc450957363)

[СПИСОК ЛІТЕРАТУРИ 42](#_Toc450957364)

[ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ 43](#_Toc450957365)

# ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

При проектуванні програмного забезпечення потрібно розуміти основні принципи проектування архітектури системи. Ключові питання проектування – це декомпозиція програм на функціональні компоненти для незалежного і одночасного їхнього виконання, розподіл компонентів у середовищі функціонування і їх взаємодія між собою, забезпечення якості і живучості системи й ін. Проектування архітектури ПЗ проводиться архітектурним стилем, заснованим на визначенні основних елементів структури – підсистем, компонентів, об'єктів і зв'язків між ними.

Однією з перших проблем, які постають перед розробником є правильне складання вимог, які потребує замовник, та правильна декомпозиція системи. Розробник повинен зрозуміти, який функціонал повинен бути розроблений, та спроектувати прототип кінцевого продукту.

Давайте розглянемо основні етапи розробки програмного забезпечення:

1. Аналіз вимог до програмного забезпечення

Аналіз вимог полягає в визначенні потреб та умов які висуваються щодо продукту, враховуючи можливо конфліктні вимоги. Вимоги мають бути задокументованими, вимірними, тестовними, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи. Вимоги можуть бути архітектурними, структурними, поведінковими, функціональними, та не функціональними.

1. Проектування програмного забезпечення

Проектування програмного забезпечення – це процес вирішення задач та планування для створення програмного рішення. На цьому етапі створюється архітектура програмного забезпечення, де описується структура програми, властивості компонентів та відносини між ними.

1. Інженерія програмного забезпечення

На цьому найдовшому етапі відбувається розробка продукта, враховуючи архітектуру та структури даних, які були виявлені та вибрані на попередніх кроках. Розроблена система має покривати всі виявлені вимоги.

Програми для створення креслень базуються на використанні векторної графіки. Векторні зображення формуються з примітивів (таких як крапка, лінія чи коло), які описуються не набором пікселів та їх кольорів (як в растрових зображеннях), а математичними формулами. Внаслідок цього вони мають можливість нескінченного масштабування, адже математичні формули можна обчислити для будь-якох точності аргументів. Тому векторні графічні зображення є оптимальним засобом для зберігання високоточних графічних об'єктів (креслення, схеми і т. д.), для яких має значення наявність чітких і ясних контурів. Всі компоненти векторного зображення описуються математично, а виходить – абсолютно точно. Векторні зображення, як правило, будуються вручну, однак у деяких випадках вони можуть бути також отримані з растрових за допомогою програм трасування. Векторні зображення не в змозі забезпечити близьку до оригіналу реалістичність та точність передачі кольорів, але плюсом векторної графіки є те, що файли, що зберігають векторні графічні зображення, мають порівняно невеликий обсяг. Важливо також, що векторні графічні зображення можуть бути збільшені або зменшені без втрати якості.

Розглянемо існуючі програми для створення будівельних креслень:

**Sweet Home 3D**

Ця програма була розроблена французькою компанією eTeks.

Основні її можливості:

1. Створення стін.
2. Розміщення меблів.
3. Збереження креслення до файлу та його завантаження.
4. Завантаження пакетів з додатковими меблями

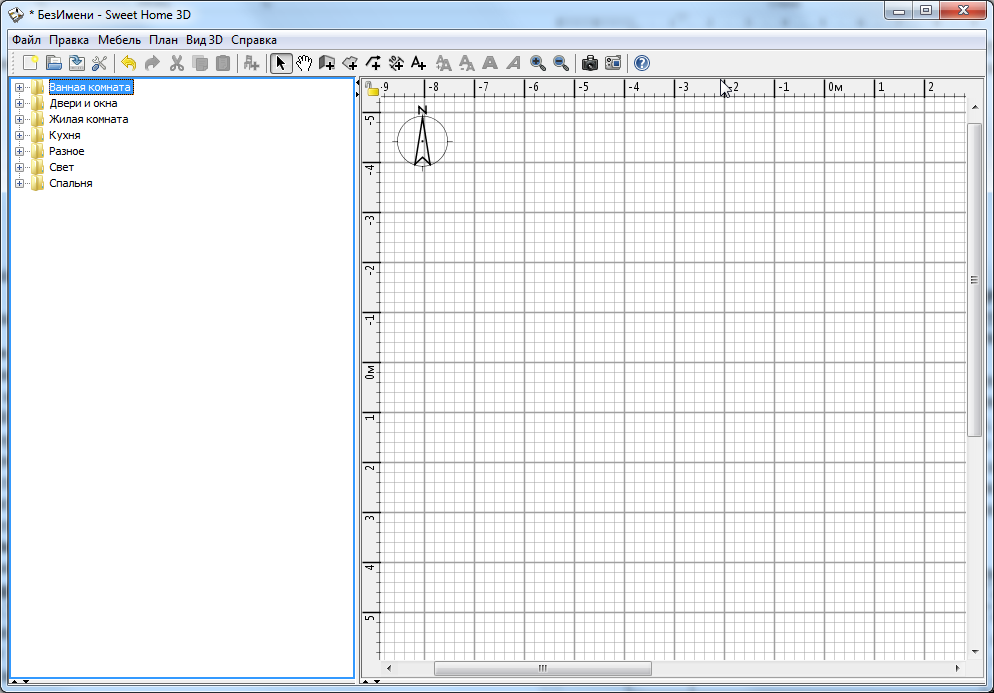


Рисунок 1 - інтерфейс програми Sweet Home

**Planner5d**

Ця програма представлена у вигляді веб додатку.

Основні функції:

1. Створення загатованих кімнат.
2. Розміщення меблів.
3. Створені плани зберігаються в cookies у браузері.

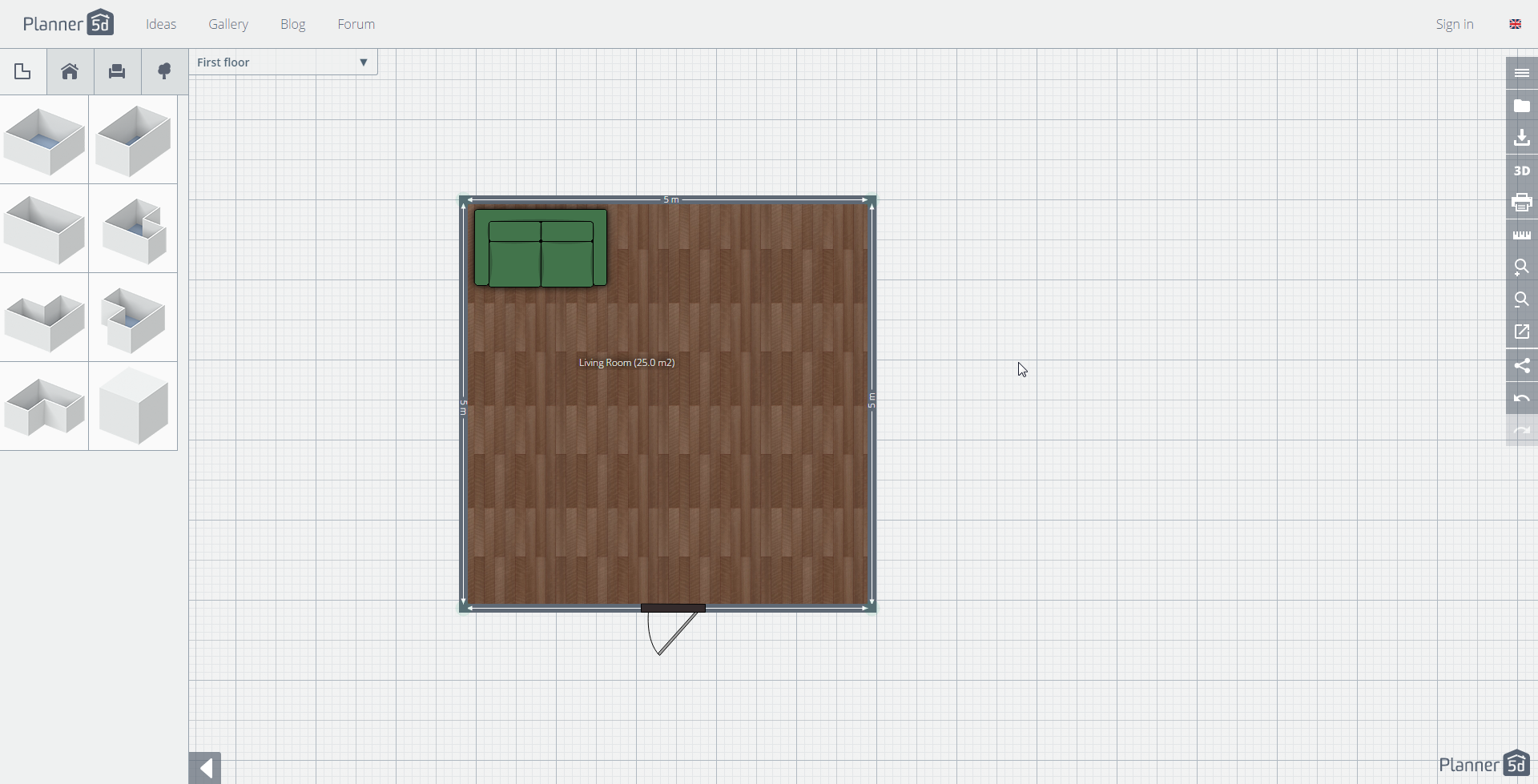


Рисунок 2 - інтерфейс програми Planner5d

**Висновок**

У цьому розділі було розглянуто існуючі рішення у проектуванні редактора будівельних креслень та визначено основні етапи проектування обраної системи. На основі розглянутих питань було визначено основну функціональність створюваного редактора:

* Можливість створення креслень.
* Можливість додавання стін та меблів до креслення.
* Можливість завантаження додаткових пакетів з меблями.
* Можливість збереження та завантаження креслення.
* Можливість відміни та повторення зроблених дій.

# ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

Перш за все визначимо призначення системи.

Програма повинна вміти редагувати створювати, завантажувати та зберігати креслення. Також повинна бути присутня можливість завантаження пакетів з маблями. Має бути присутня можливість відміни та повторення зроблених дій. Також мають бути наявні правила розміщення фігур на кресленні.

Після визначення призначення системи можемо виконати функціональну декомпозицію системи. Для проведення декомпозиції системи було обрано метод визначення прецедентів функціональної моделі. Визначимо основні прецеденти функціональної моделі нашої системи. Інформацію подано у вигляді таблиці:

|  |  |  |
| --- | --- | --- |
| № | Прецедент | Опис |
| 1 | Завантаження та вибір фігур | Система надає можливість завантажувати пакети з фігурами та вибирати конкретну для додання на креслення. |
| 2 | Редагування поточного креслення | Система надає можливість додавати та видаляти фігури. |
| 3 | Збереження та завантаження креслень | Користувач може зберегти та завантажити вреслення. |

Наступним кроком є декомпозиція прецедентів.

|  |  |  |
| --- | --- | --- |
| № | Прецедент | Опис |
| **1** | **Завантаження та вибір фігур** | |
| 1.1 | Вибір фігури зі списку | Система надає можливість вибрати завантажену фігуру для додавання на креслення |
| 1.2 | Завантаження пакету фігур | Система надає можливість завантажити пакет фігур |
| 1.2.1 | Вибір файлу з фігурами | Система надає можливість обрати файл для завантаження |
| 1.2.2 | Зчитування та розбір XML | Система зчитує файл та будує дерево розбору XML |
| 1.2.3 | Створення фігур з XML | Система створює фігури із завантаженими параментрами |
| 1.2.4 | Додавання завантажених фігур до списку | Система додає фігури до списку доступних |
| 1.2.5 | Створення правил з XML | Система створює правила із завантаженими параметрами |
| 1.2.6 | Додавання правил до списку | Система додає заванажені правила до списку діючих |
| **2** | **Редагування поточного креслення** | |
| 2.1 | Додванна вибраної фігури на креслення | Система додає обрану фігуру на креслення |
| 2.2 | Вибір фігури на кресленні | Система надає можливість видалити фігуру на кресленні |
| 2.2.1 | Видалення обраної фігури | Система надає можливість видадити обрану фігуру |
| 2.3 | Можливість рухати креслення | Система надає можливість рухати креслення |
| 2.4 | Історія змін | Система надає можливість відміняти та повторювати дії |
| 2.4.1 | Можливість відмінити дію | Система надає можливість відмінити дію |
| 2.4.2 | Можливість повторити дію | Система надає можливість повторити дію |
| **3** | **Створення креслень** | |
| 3.1 | Створення нового креслення | Система надає можливість створити нове креслення |
| 3.2 | Збереження креслень | Система зберігає креслення у вибраний файл |
| 3.2.1 | Вибір файлу | Система надає можливість обрати файл |
| 3.2.2 | Запис фігур у вигляді XML | Система зберігає у файл фігури, що знаходяться на поточному креслення |
| 3.3 | Завантаження креслень | Система надає можливість завантажити креслення |
| 3.3.1 | Вибір файлу | Надає можливість борати файл для завантаження |
| 3.3.2 | Зчитування та розбір XML | Система зчитує файл та будує дерево розбору XML |
| 3.3.3 | Додавання на креслення отриманих фігур | Система додає завантажені фігури на креслення |

Наведемо отримані прецеденти у вигляді Use-Case діаграми:

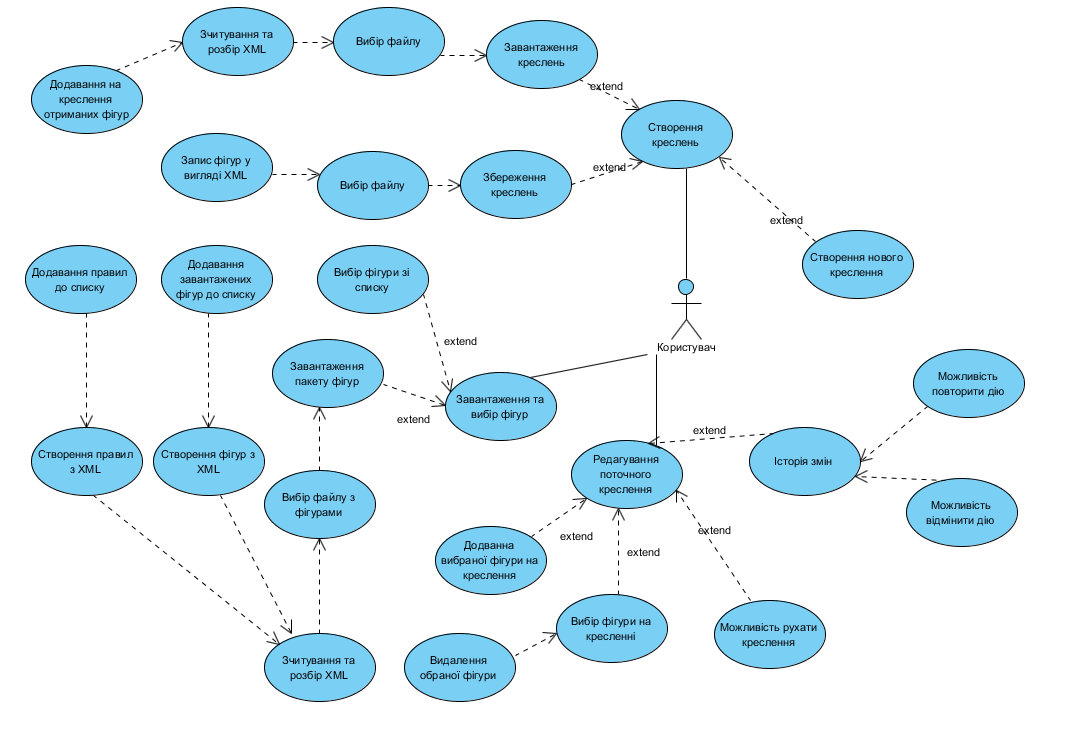


Рисунок 3 - Use case діаграма

Створивши функціональну модель, можна провести декомпозицію системи на підсистеми та визначити зв’язки між ними. Всього в редакторі векторних зображень можна виділити 5 підсистем:

* Робота з файлами та XML.
* Робота з правилами.
* Робота з фігурами.
* Робота з кресленням.
* User interface.

Детально розглянемо кожну підсистему.

## Робота з файлами та XML

Підсистема, що відповідає за читання та запис файлів XML.

Містить клас XMLTag, що представляє собою одиничний тег XML. Його полями є назва, внутрішній текст, батьківський тег та список дочірніх тегів. Кожен XML документ представляється у вигляді дерева цих класів.

Для читання XML створений клас XMLReader. На основі заданого імені файла він будує дерево тегів, що містяться в цьому файлі. Після зчитування повертає кореневий тег документа.

Для запису XML документів створений клас XMLWriter. На основі переданого кореневого тега він формує текстове представлення документа та записує його у файл.

Для спрощення запису є клас XMLBuilder, який примає пари ім'я та значення і на їх основі будує дерево тегів, а потім використовуючи XMLWriter записує документ. Застосування цього класу дозволяє іншим класам документи XML не знаючи про їх структуру. Вони лише вказують пари ключ-значення. які треба зберегти.

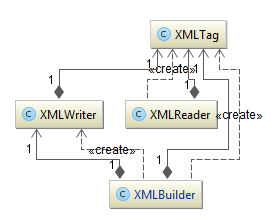


Рисунок 4 - підсистема роботи з файлами

## Робота з фігурами

Основні задачі цієї підсистеми – керування фігурами.

Основні функції класу фігура – завантаження фігури з файлу, додавання на креслення та розміщення на кресленні при його завантаженні. Також фігура має малювати передперегляд перед додаванням на креслення (слідуючи принципу «WYSIWYG» – що бачиш, те і отримуєш), містити своє ім’я та ім’я пакету, якому вона належить. Також фігура має набір тегів, які використовуються для застосування правил.

Для спрощення завантаження з файлів створено інтерфейс BuilderFromXML, який приймає кореневий тег та створює відповідну фігуру. Для зовнішніх класів він дозволяє абстрагуватись від внутрішньої структури фігури, а отже з’являється можливість їх уніфікованого створення.

Кожна фігура має функції, що отримують поточний стан клавіатури та миші а також фінкцію, фка викликається коли необхідно намалювати передперегляд.

Для елементів, що додаються на креслення створений окремий клас Drawable, який має функцію для відмальовки на полотні та може за допомогою інтерфейсу XMLBuilder записати відомості про себе у файл.

Створено дві реалізації фігур. Для можливості завантаження з XML створено внутрішні класи що реалізуть інтерфейс BuilderFomXML, для додавання до креслення створена реалізація класу Drawable.

Перша реалізація класу фігура – ImmutableFigure. Вона представляє собою фігуру, що не змінює своїх розмірів та конфігурації (прикладом може бути стіл чи двері).

Друга реалізація – LineFigure. Вона представляє собою лінію заданої ширини (наприклад стіну).

Також в цій підсистемі присутній контейнер для фігур – FiguresManager. Він відповідає за завантаження пакетів та вибір поточної фігури, що зв’язаний зі списком фігур у UI.

Завантаження фігур відюувається за допомогою класу FiguresLoader. Він зчитує XML файл та для кожного тегу з фігурою отримує відповідний BuilderFromXML та за допомогою нього завантажує фігуру.

Наведемо діаграму класів для даної підсистеми.

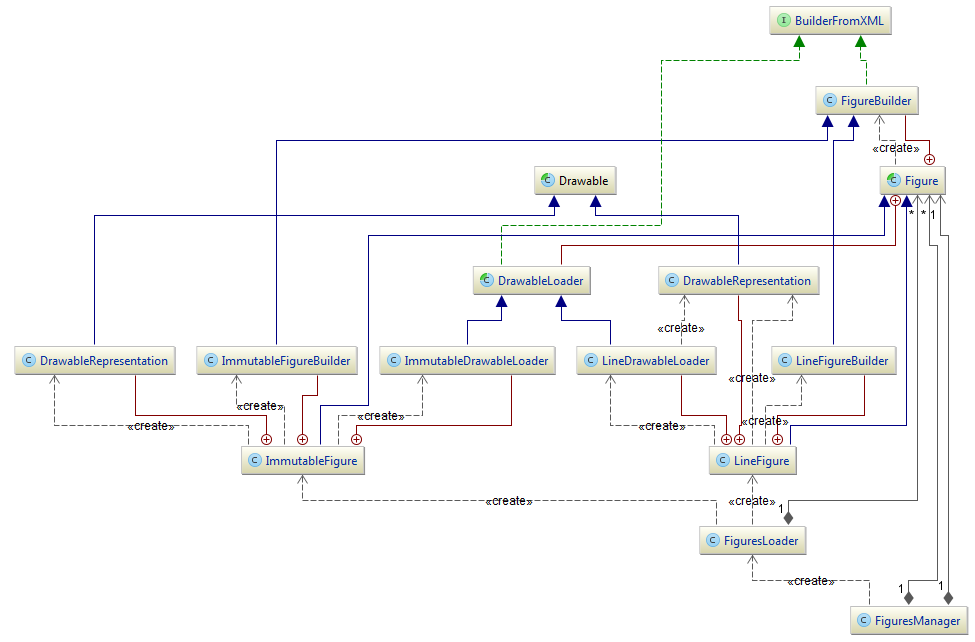


Рисунок 5 - Підсистема фігур

## Робота з правилами

Основна задача цієї підсистеми – зберігання та застосування правил.

Функції класу правило – збереження типів фігур до яких можна застосувати (тегів), та метод, що перетворює задану фігуру по правилу, що міститься в цьому класі. Реалізовано два типи правил – PlacemetRule, що дозволяє задати порядок малювання об’єктів на полотні (наприклад меблі малюються пісдя підлоги), та ContainterRule, що дозволяє заборонити перетин певних об'єктів, або навпаки, дозволити розміщувати один об'єкт тільки всередині іншого.

Також в цій підсистемі присутній контейнер – RulesManager, який зберінає всі правила та при запиті послідовно обробляє об’єкт всіма правилами. Для спрощення доступу він виконаний з застосування паттерна Singletone, що дозволяє отримувати доступ до нього з будь-якого місця програми.

Для завантаження правил використовується клас RulesLoader. Він зчитує заданий XML та для кожного правила в ньому конструює правило з відповідними параметрами.

Наведемо діаграму класів для даної підсистеми.

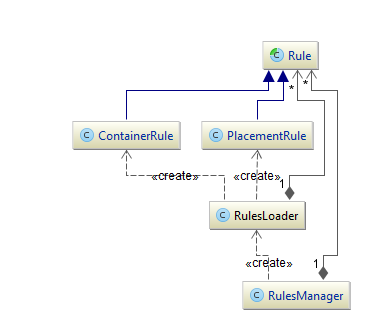


Рисунок 6 - підсистема правил

## Робота з кресленнями

У цій підсистемі задекларований інтерфес для команд, що містить два методи – виконати та відмінити.

Тут міститься основний клас що представляє креслення – Scene. У ньому зберігаються всі об’єкти що були намальовані. Також є метод для малювання всіх об’єктів на полотні. Для додавання нових об’єктів використовуються команди, для їх отримання є публічний метод. Також за допомогою команд реалізоване видалення. Це дозволило легко реалізувати історію змінта операції відміни і повторення дій.

Також частиною цієї підсистеми є клас SceneManager, який зберігає всі відкриті креслення та слугує посередником між поточним кресленням та іншими елементами системи. За допомогою нього поточне креслення може отримати доступ до правил та інтерфейсу, а зовнішні класи – до команд для додавання нових фігур на креслення. Також SceneManager займається завантаженням існуючих креслень із файлів та збереженням поточного до файлу.

Наведемо діаграму класів для даної підсистеми.

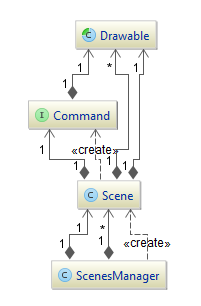


Рисунок 7 - підсистема креслень

## User interface

Ця підсистема створена для взаємодії з користувачем і складається з трьох вікон.

Перше і основне – MainWindow. Саме для нього встановлюються всі обробники клавіатури та миші. Містить в собі список доступних для малювання фігур, кнопки для операцій видалення, відміни останньої дії та її повторення, а також клас Canvas – полотно, на якому малюється креслення. Для кожного оновлення креслення використовується метод repaint(), який передаэ керування графікою класу SceneManager.

Друге вікно – AboutWindow, я якому виводиться інформація про програму та її автора.

Третє – DebugWindow, містить текстове поле та використовується для виводу логів прирозробці та відладці програми.

Наведемо діаграму класів для даної підсистеми.

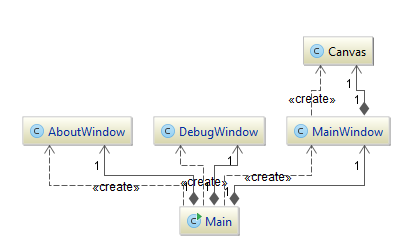


Рисунок 8 - підсистема UI

## Загальна структура

Основна обробка подій програмою відбувається так – класи Scene та FiguresManager підписуються на оновлення клавіатури та миші; клас FiguresManager підписується на зміну стану списку фігур. Якщо виконуються умови додання фігури з класу SceneManager отримується та виконується команда на додавання фігури до поточного креслення. Піля виконання команди відбувається перевірка на відповідність креслення правилам. Якщо поточний стан заборонений правилами то відбувається відміна останньої команди.

Якщо користувач натискає кнопки відміни чи повторення дії ці події через клас SceneManager направляються до поточного креслення, де за можливості виконується відповідна дія. При виборі завантаження з файлу чи збереження до файлу SceneManager виконнує вінповідну операцію. За рахунок використання паттерну Singletone доступ до глобальних компронентів, таких як SceneManager чи FiguresManager значно спрощено.

Для досягнення легкої модифікованості системи був використаний паттерн Mediator, що дозволило позбавитись від сильної зв’язності системи. За рахунок використання паттерну Buidler та створення узагальнених класів та інтерфейсів досягнута можливість однотипно маніпулювати різними об’єктами що значно спрощує створення нових типів, адже для них не доведеться перероблювати інші класи.

Наведемо загальну діаграму класів.

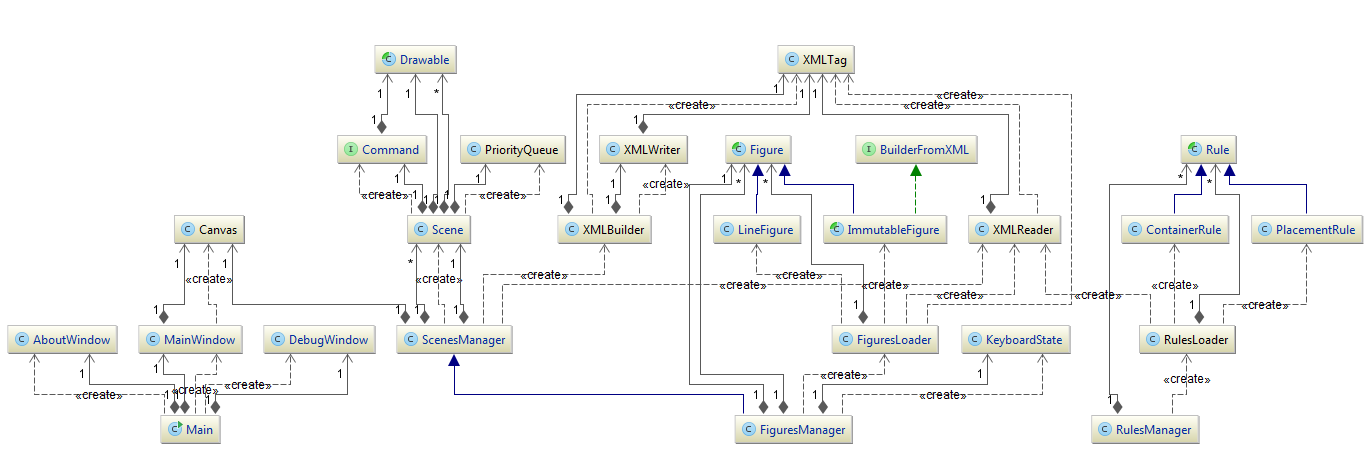


Рисунок 9 - загальна діаграма класів

## Висновок

У даному розділі нами була виконана функціональна декомпозиція системи. Також, ми спроектували наступні підсистеми, та визначили взаємозв’язки між ними:

- Підсистема “Взаємозв’язок складових системи”;

- Підсистема “Безпечна робота із зображенням”;

- Підсистема “Історія змін”;

- Підсистема “Редагування”;

- Підсистема “Робота із шарами”.

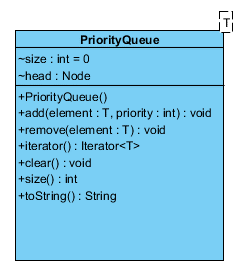
Для кожної із них ми побудували діаграму класів.

При проектуванні системи були використані

# РЕАЛІЗАЦИЯ ПРОГРАМНОГО КОМПЛЕКСУ

## Допоміжні класи

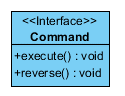
### PriorityQueue



Представляє собою чергу з пріоритетом, засновану на однозв’язному лінійному списку. Реалізує інтерфейс «Iterable». Використовується для зберігання фігур в порядку їх відмальовки в класі Scene.

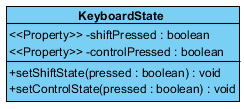
## Представлення малюнка

### Command



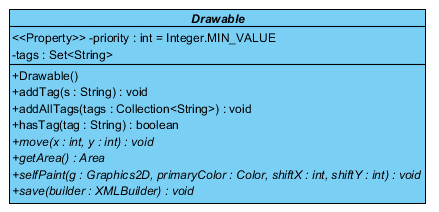
Представляє собою інтерфейс для паттерна Command. Містить два методи – execute та reverse. Використовується у класі Scene для додавання та видалення фігур.

### KeyboardState



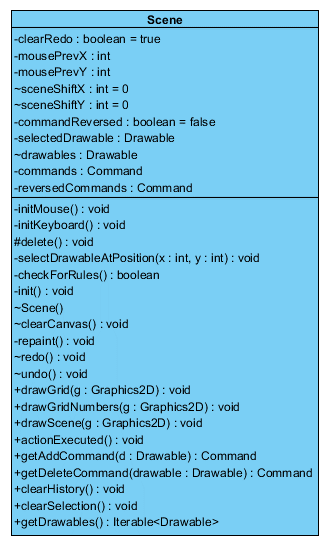
Призначення – збереження в собі стану клавіатури для передачі до інших класів. В поточній реалізації зберігається лише стан клавіш Shift та Ctrl.

### Drawable



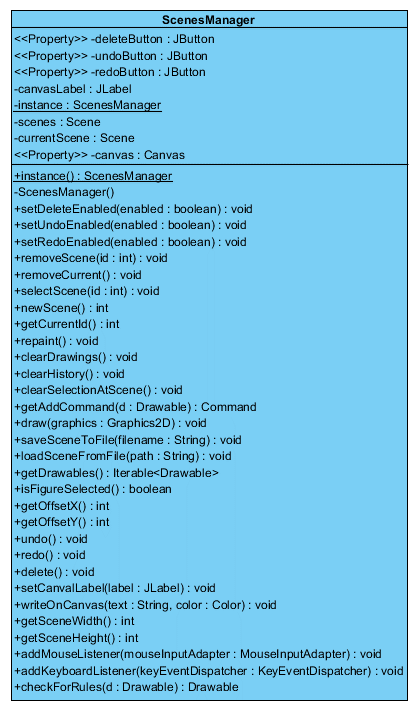
Це абстрактний клас, що представляє собою намальовану фігуру. Містить в собі теги (використовуються для застосування правил), пріоритет (використовується для визначення порядку малювання), абстрактні методи для малювання та збереження в файл.

### Scene



Представляє собою поточний малюнок. Для взаємодії з системою використовує посередник SceneManager. Зберігає та малює створені фігури (Drawable), представляє інтерфейс для отримання команд на додавання фігур, відміни дій, очистки малюнка, відслідковує події мишки та клавіатури для можливості рухати малюнок та видаляти фігури.

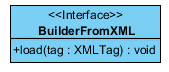
### SceneManager



Зберігає в собі малюнки (Scene) та дозволяє переключатись між ними. Виконує роль посередника між малюнками та системою. Для спрощення доступу виконаний у вигляді Singletone. Додає до функціоналу малюнка методи роботи з файлами – збереження та завантаження. Представляє інтерфейс по роботі з UI та полотном для малюнка.

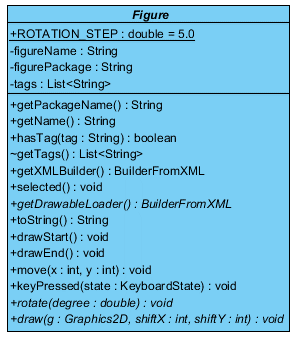
## Фігури

### BuilderFomXML



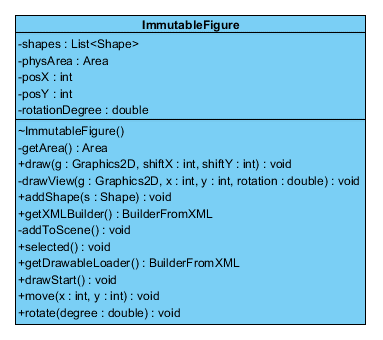
Представляє собою інтерфейс для конструювання об’єкта з XML. Містить один метод, що приймає XML тег.

### Figure



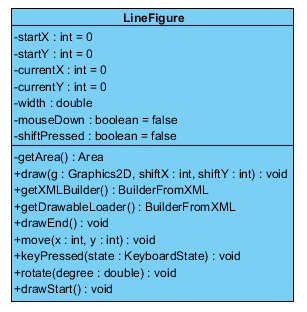
Це абстрактний клас, представдяє собою фігуру, яку можна намалювати. Містить в собі ім’я, назву модуля, теги, надає інтерфей для отримання BuilderFromXML, який завантажує цю фігуру з файлу, та BuilderFromXML для завантаження намальованої фігури (Drawable) та додання її до поточного малюнка, метод для малювання попереднього перегляду фігури та методи, що отримують повідомлення про зміну стану клавіатури та миші. За рахунок використання абстрактних класів та паттерну Builder дозволяє зовнішнім класам однотипно обробляти всі фігури.

### ImmutableFigure



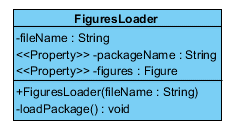
Фігура, що предcтавляє собою незмінний об’єкт (наприклад стіл). реалізовує функціонал класу фігури. Використовує RulesManager для відображення можливості установки фігури при попередньому перегляді. Для додавання до поточного малюнка використовує внутрішній клас DrawableImplementation (що реалізовує функціонал класу Drawable), для завантаження з файлу використовує внутрішній клас ImmutableFigureBuilder, який реалізовує інтерфейс BuilderFromXML.

### LineFigure



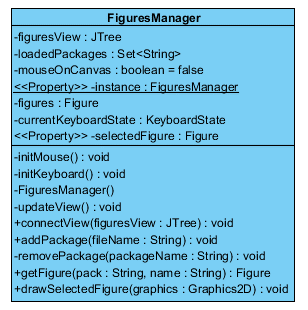
Фігура, що представляє собою лінію (наприклад стіну). Використовує RulesManager для відображення можливості установки фігури при попередньому перегляді. Для додавання до поточного малюнка використовує внутрішній клас DrawableImplementation (що реалізовує функціонал класу Drawable), для завантаження з файлу використовує внутрішній клас LineFigureBuilder, який реалізовує інтерфейс BuilderFromXML.

### FiguresLoader



Призначений для завантаження фігур із вказаного файлу. Для читання файлу використовуює XMLReader, для створення фігури використовує XMLBuilder. За рахунок використання паттерну Builder у класі Figure дозволяє легко реаліщувати заваннтаження будь-яких фігур незалежно від їх внутрішньої архітектури.

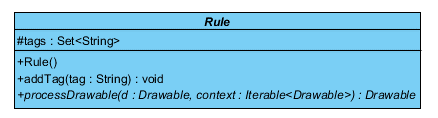
### FiguresManager



Містить в собі всі завантажені фігури. Для спрощення доступу виконаний у вигляді Singletone. Має методи для завантаження нових фігур. відслідковує події мишки та клавіатури та перенаправляє їх до вибраної фігури. З’єднаний із JTree в UI для вибору фігури.

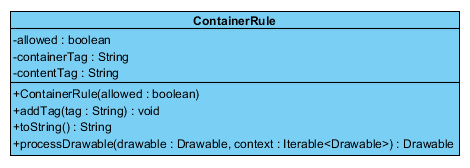
## Правила

### Rule



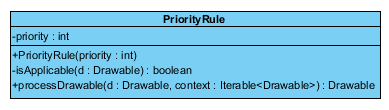
Представляє інтерфейс для обробки фігури. Приймає фігуру та обробляє її залежно від правила.

### ContainerRule



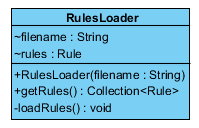
Це правило дозволяє заборонити перетин певних об'єктів, або навпаки, дозволити розміщувати один об'єкт тільки всередині іншого. Реалізовує функціонал класу Rule. При неможливості встановлення об’єкту повертає значення null.

### PriorityRule



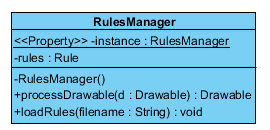
Правило яке встановлює пріоритет відмальовки. Реалізовує функціонал класу Rule.

### RulesLoader



Використовується для завантаження правил. Зчитує заданий файл та повертає список правил.

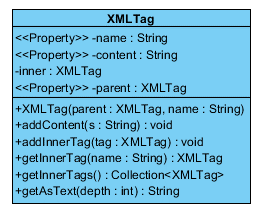
### RulesManager



Контейнер для правил. Зберігає усі діючі правила. Має методи для застосування правил до конкретної фігури та завантаження правил з файлу (за допомогою RulesLoader). Реалізований із використанням патерну Singletone, що дозволяє отримати доступ до правил з будь-якого місця програми та гарантувати, що всі правила будуть зберігатись в одному місці.

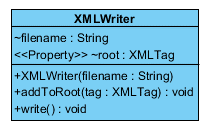
## Файли

### XMLTag



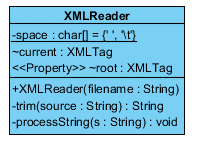
Представляє собою простий XML тег. Зберігає ім’я, внутрішній текст, батьківський тег та список нащадків. Має метод для конвертації в текст для запису в файл.

### XMLWriter



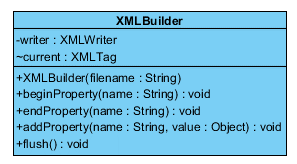
Використовується для запису XML. Записує заданий кореневий тег у файл із заданим ім’ям.

### XMLReader



Зчитує XML файл із заданим ім’ям та перетвоює його у дерево. Повертає кореневий тег.

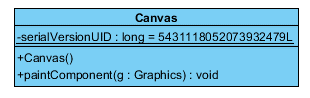
### XMLBuilder



За допомогою XMLWriter записує XML файл. Має спрощений інтерфейс створення XML дерева на основі пар ключ-значення.

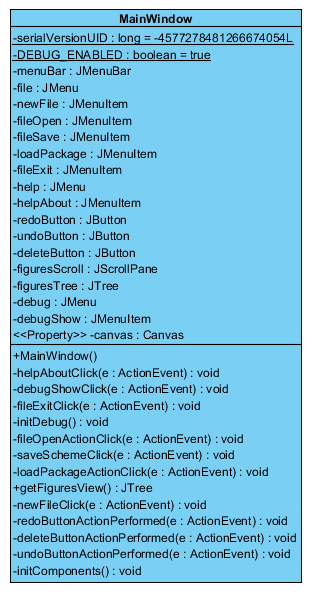
## User interface

### Canvas



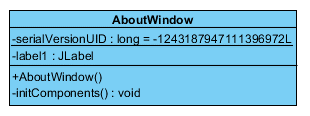
Представляє собоє полотно для малювання, розміщується у головному вікні. Налідується від JPanel та перевантажує метод малювання, що дозволяє передавати контроль над створенням зображення до інших класів.

### MainWindow



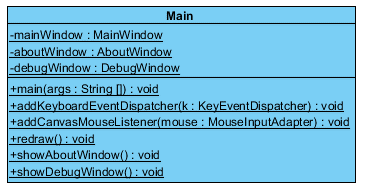
Головне вікно програми. Містить в собі список доступних для малювання фігур, кнопки для операцій видалення, відміни останньої дії та її повторення, а також клас Canvas – полотно, на якому малюється креслення. Для кожного оновлення креслення використовується метод repaint(), який передає керування графікою класу SceneManager.

### AboutWindow



Вікно для виводу інформації про програму. Містить текстове поле в якому вказана назва програми та її автор.

## Main



Точка входу програми. Створює та ініціалізує вікна. Слугує посередником між UI та іншими компонентами.

## Додавання фігури до креслення

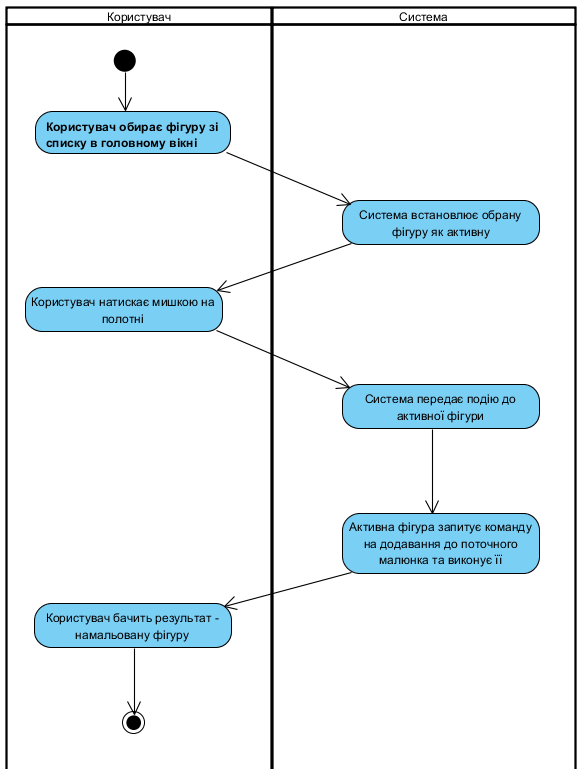


Рисунок 10 - activity diagram

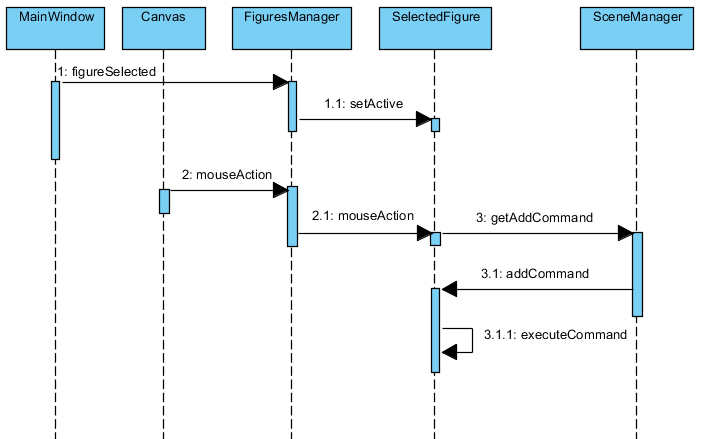


Рисунок 11 - sequence diagram

## Завантаження креслення

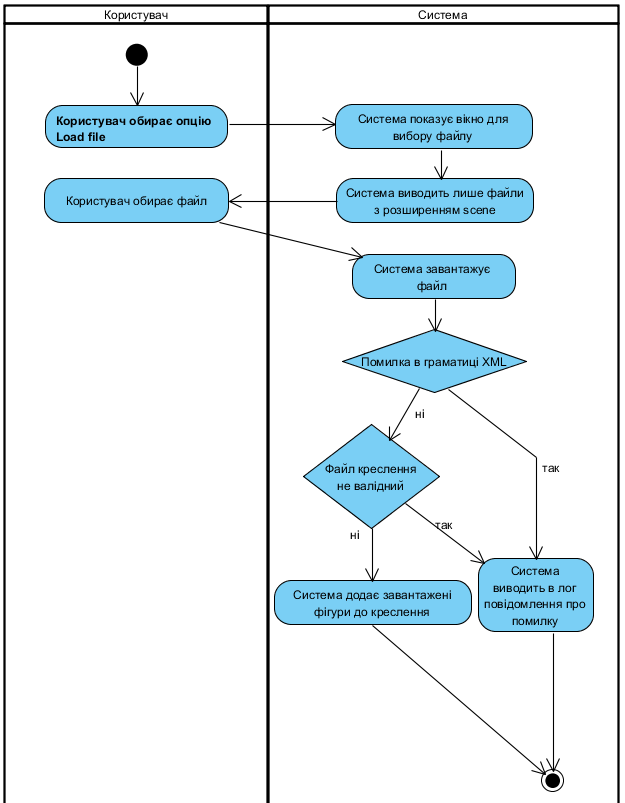


Рисунок 12 - activity diagram

## Завантаження пакету фігур

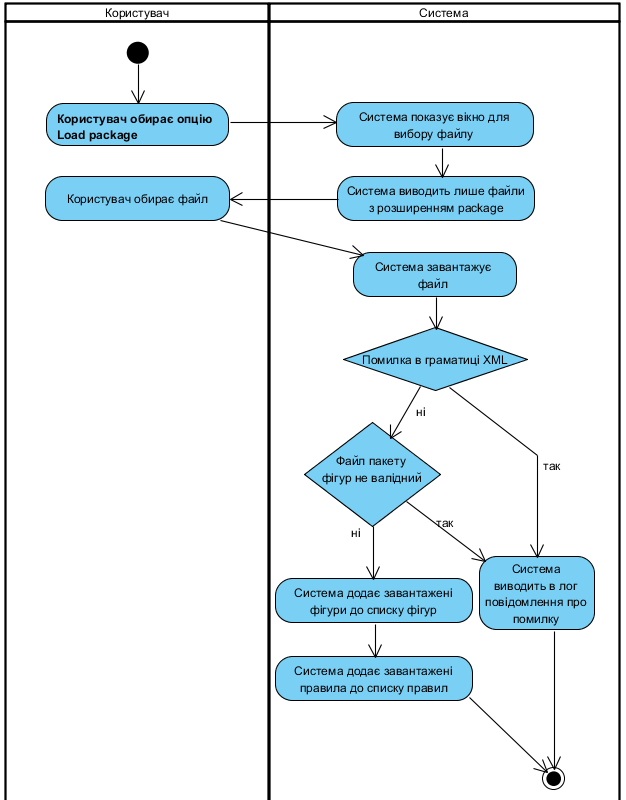


Рисунок 13 - activity diagram

## Висновок

Найважливішим завданням під час проектування та реалізації класів було забезпечення взаємодії між класами, а також програмна реалізація визначеного у першому розділі функціоналу.

Найбільш функціоналоємним класом цієї системи є Scene. Цей клас відповідає за логіку роботи з кресленням. Також ємними є класи SceneManager, RulesManager та FiguresManager, які реалізовують інтерфейси доступу до малюнків, правил та фігур відповідно.

За рахунок використання інтерфейсів, абстрактних класів та паттерну Builder було досягнуто можливість однотипно обробляти різні об’єкти, що значно спрощує подальше розширення системи.

# ОПИС ЗАСТОСУВАННЯ СИСТЕМИ

Редактор креслень застосовується для проектування будівель. При запуску програми автоматично завантажуєтся пакет фігур за замовчанням. На головному екрані знаходяться:

* список доступних для розміщення фігур
* кнопки для відміни та повтору дії (також для виконання цих операцій можна використовувати комбінації клавіш Ctrl+C та Ctrl+V відповідно)
* кнопка для видаленя обраної фігури
* полотно для встановлення фігур

Також зверху доступне меню за допомогою якого можна зберігати та завантажувати креслення, а також заванотажувати додаткові пакети із фігурами.

Для обертання фігур можна використовувати кнопки Q та E.

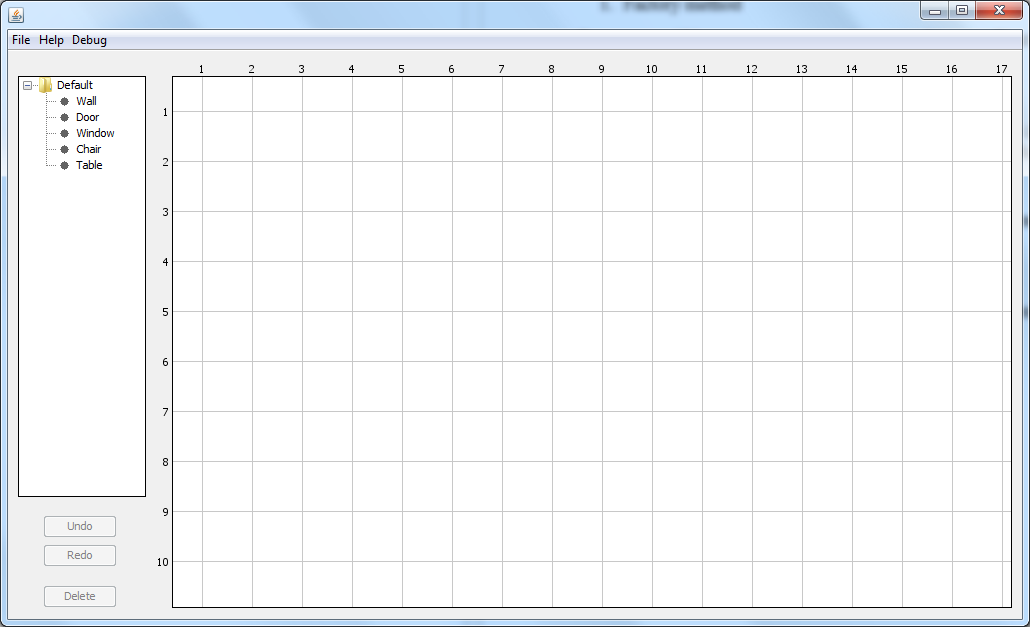


Рисунок 14 - стартове вікно програми

Користувач може завантажити додаткові пакети з меблями. Дозволяється завантажувати лише файли з розширенням .figures.

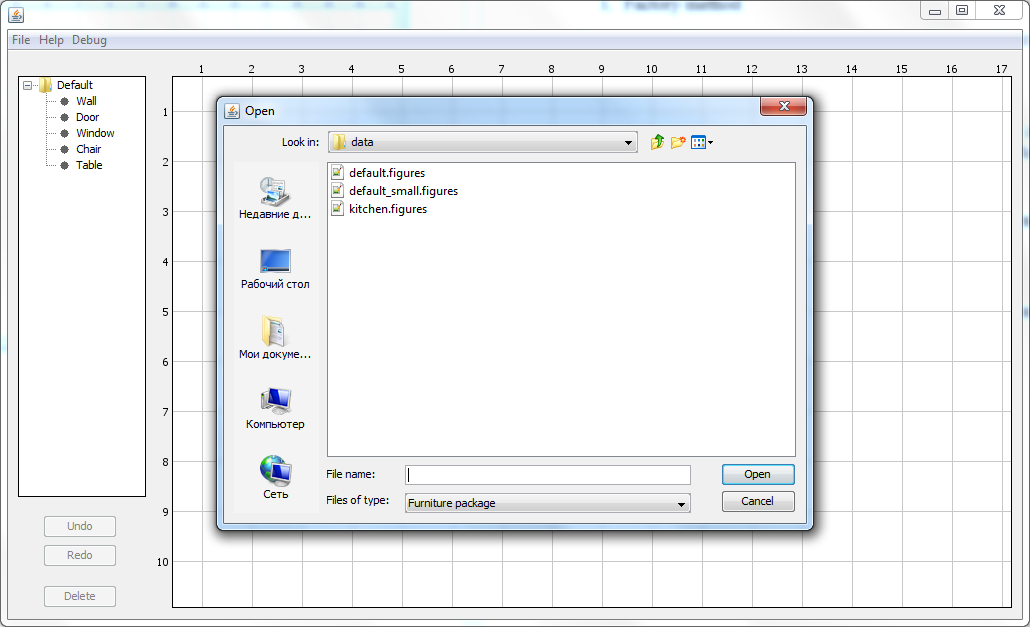


Рисунок 15 - завантаження додаткових пакетів

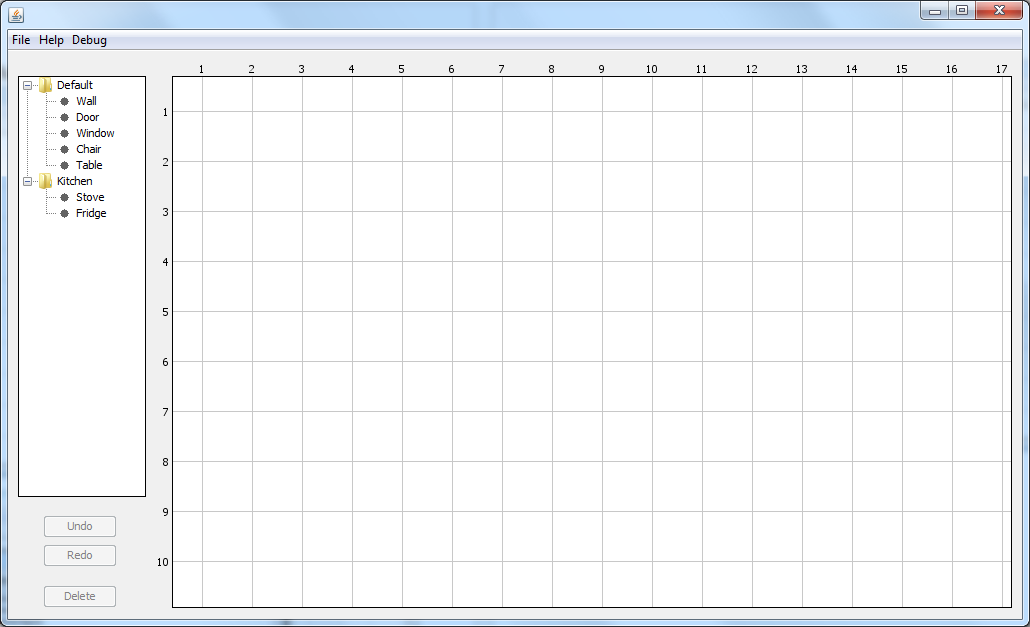


Рисунок 16 - програма після завантаження пакету

Після завантаження пакету фігури, що в ньому знаходяться відоюражаються в списку зліва.

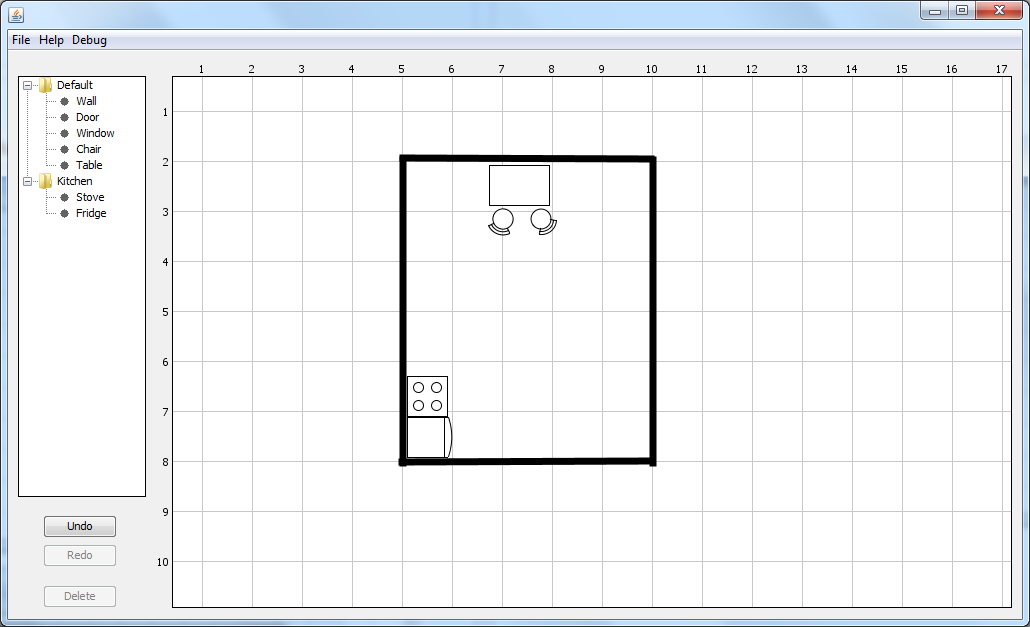


Рисунок 17 - розмыщення меблів

Користувач може малювати стіни та розміщувати меблі.

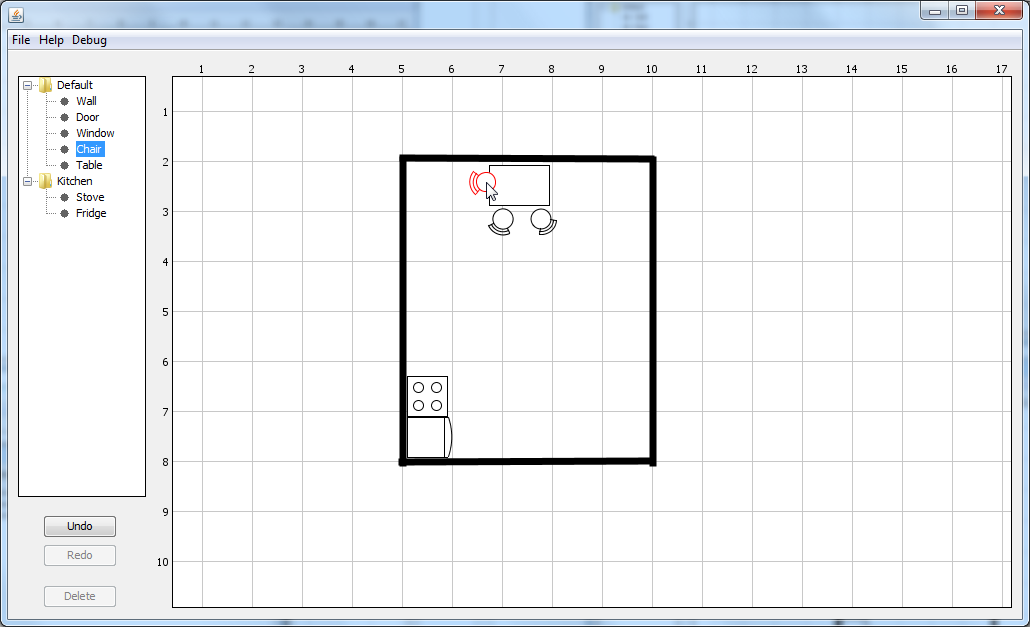


Рисунок 18 - робота правил

Внаслідок застосування правил користувач не може ставити меблі один на одного

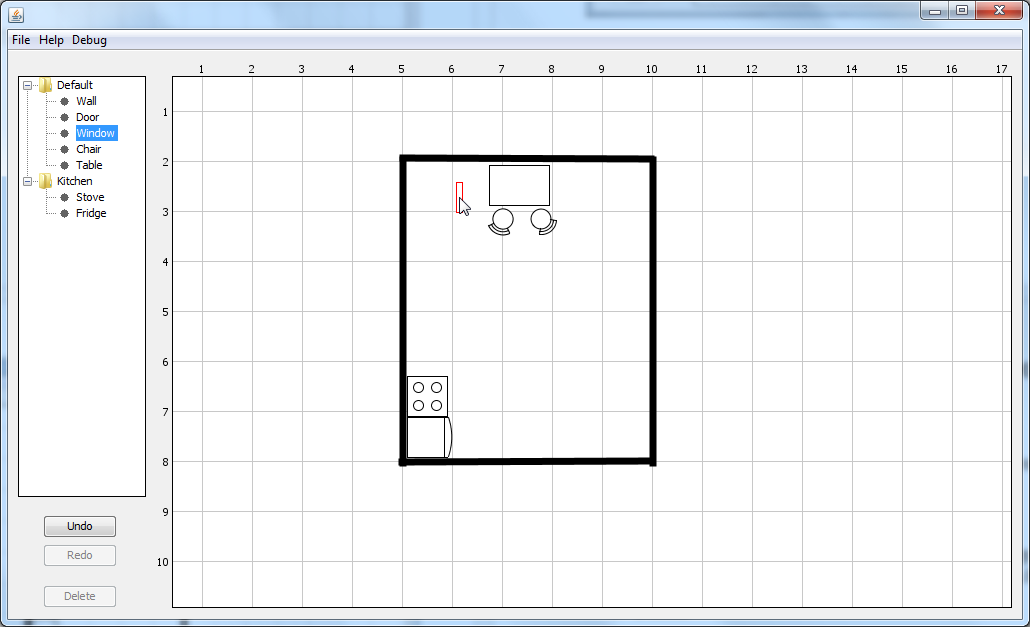


Рисунок 19 - застосування правил

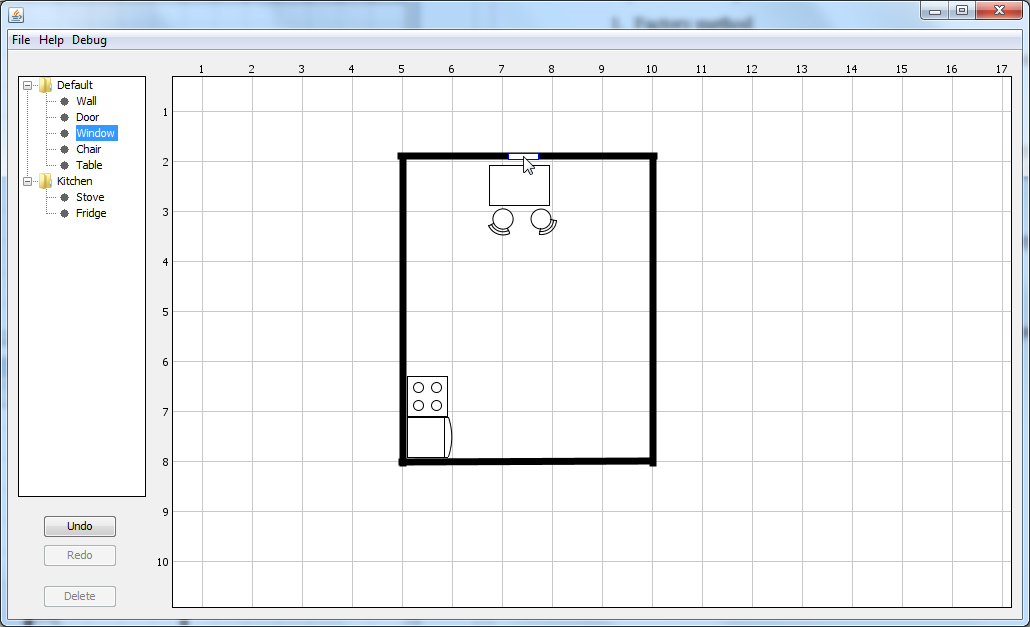


Рисунок 20 - застосування правил

Також застосування правил не дає змоги встановити вікно без стіни. Його можна розмістити тільки якщо воно буде повністю знаходитись в стіні.

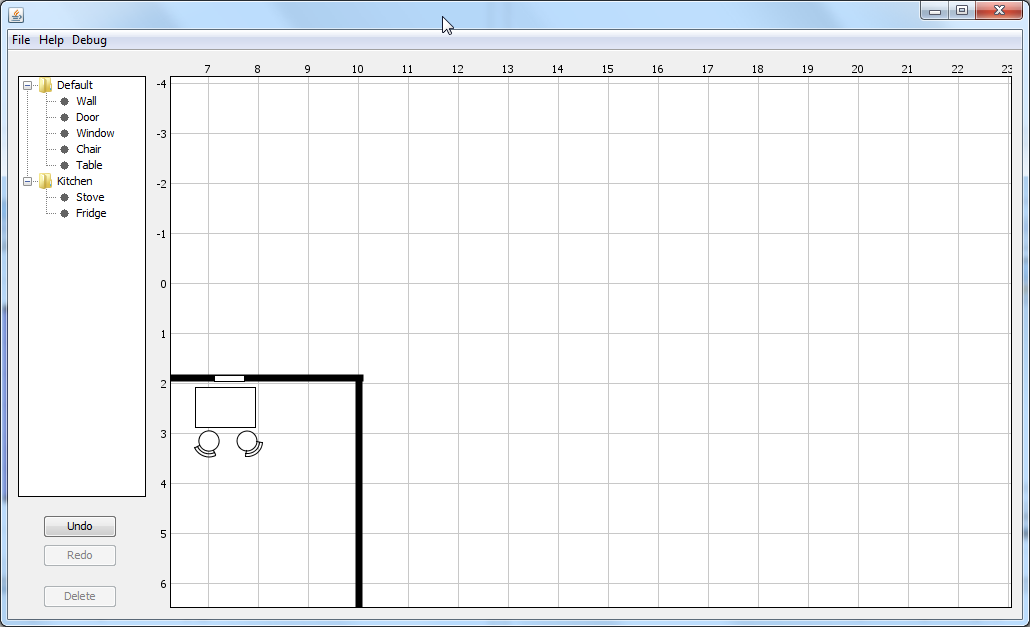


Рисунок 21 - рух креслення

Користувач також може рухати креслення.

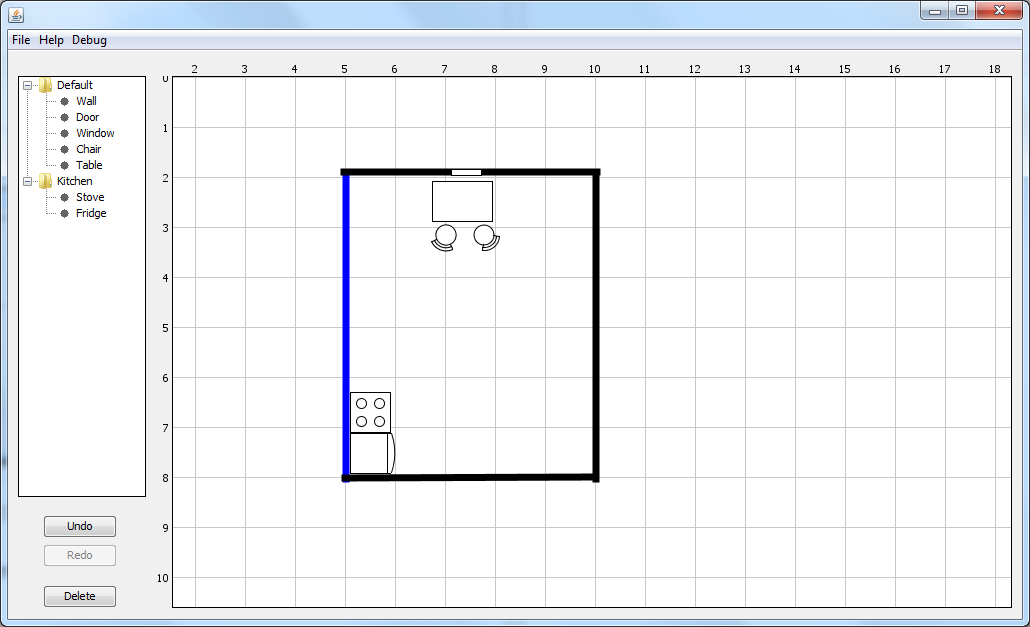


Рисунок 22 - вибір встановлених фігур

Користувач може вибирати вже встановлені фігури. Для цього треба очистити вибрану фігуру (це можна зробити натиснувши клавышу Esc) і клікнути мишкою на них.

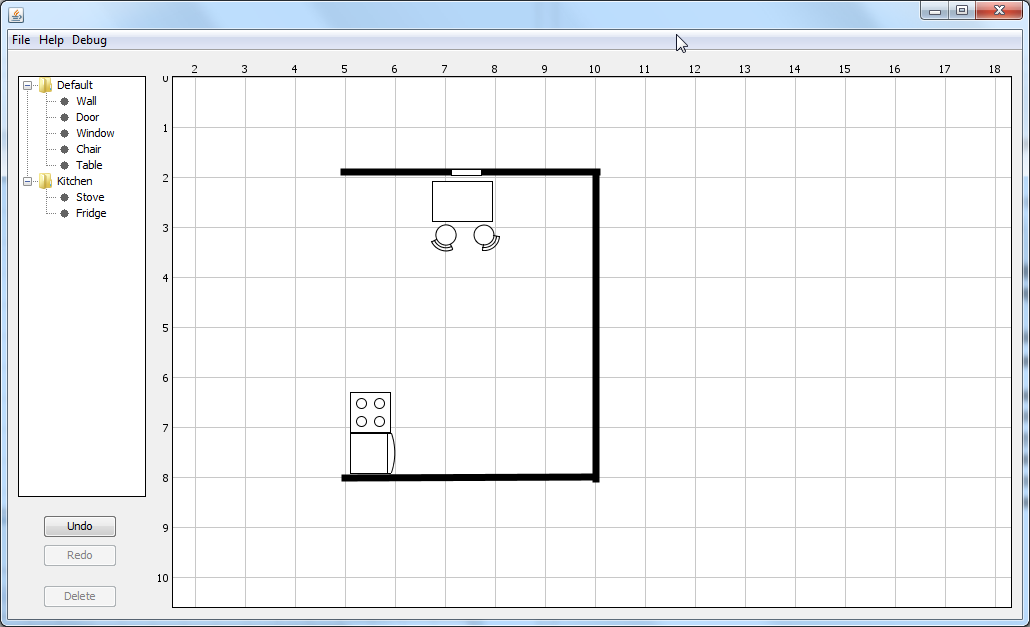


Рисунок 23 - видалення фігур

Також користувач може вибирати встановлені фігури та видаляти їх.

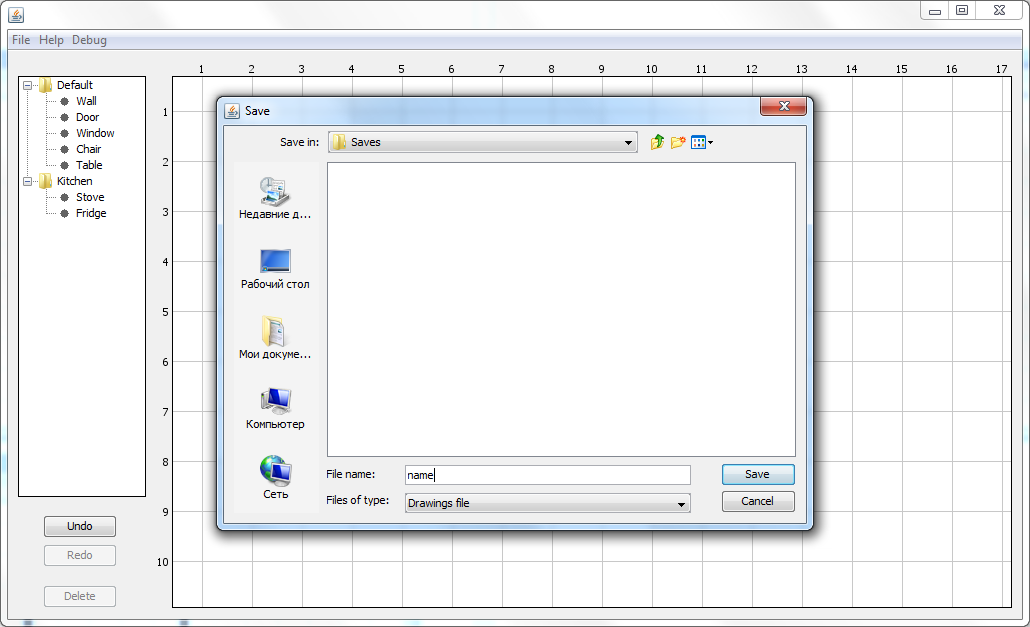


Рисунок 24 - збереження файлу

Користувач може вберегти створене креслення. Вони зберігаються у форматі .scene.

## Висновок

Створений редактор будівельних креслень повністю реалізував запланований функціонал з першого розділу:

* Можливість створення креслень.
* Можливість додавання стін та меблів до креслення.
* Можливість завантаження додаткових пакетів з меблями.
* Можливість збереження та завантаження креслення.
* Можливість відміни та повторення зроблених дій.

# ВИСНОВОК ДО КУРСОВОЇ РОБОТИ

Розроблена програма для редагування будівельних креслень повністю задовольнила потреби, описані в першому розділі, а саме:

* Можливість створення креслень.
* Можливість додавання стін та меблів до креслення.
* Можливість завантаження додаткових пакетів з меблями.
* Можливість збереження та завантаження креслення.
* Можливість відміни та повторення зроблених дій.

Під час створення архітектури даної системи, що описана у другому і третьому розділи цієї роботи, були викориистані наступні шаблони проектування:

1. Factory method

Дозволив уніфікувати створення різних графічних об’єктів – кожен об’єкт Figure створює специфічний об’єкт Drawable.

1. Command

Застосування команд для додавання та видалення фігур дозволило легко зберігати історію змін та виконувати команди Undo та Redo.

1. Builder

У комбінації з Factory method дозволив легко завантажувати фігури з файлу незалежно від їх внутрішньої структури.

1. Mediator

Дозволив уникнути сильної зв’язності системи, що спрощує подальшу модифікацію.

1. Template method

Використання цього паттерну дозволило уникнути дублювання коду, що спрощує його наступну підтримку.

# СПИСОК ЛІТЕРАТУРИ

1. Гранд Марк . « Шаблоны проектирования в JAVA» .: Пер. с англ. - М.: Издательский дом "Вильямс", 2011. - 205 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. «Приемы объектно-ориентированного проектирования. Паттерны проектирования.» - СПб: Питер, 2001. - 368 с.
3. Кейс «Проектирование графического редактора» : - Software Design Популярно о проектировании программ - 4.01.2011 - Режим доступу: http://askofen.blogspot.com/2011/01/blog-post.html. – Назва з екрана
4. Тидвелл Дж. Разработка пользовательских интерфейсов.— Питер, 2007. Jenifer Tidwell. Designing Interfaces, 2nd Edition. Patterns for Effective Interaction Design. — O’Reilly, 2010.
5. Йенер М., Фидом А. «Паттерны проектирования» .: Пер. с англ. - М.: Издательский дом "Вильямс", 2010. - 421 с.
6. Углубляясь в Graphics2D : - Хабрахабр - Режим доступу: https://habrahabr.ru/company/alee/blog/127518/ – Назва з екрана
7. Йенер М., Фидом А. «Паттерны проектирования» .: Пер. с англ. - М.: Издательский дом "Вильямс", 2010. - 421 с.
8. Мак-Колм Смит Джейсон. «Элементарные шаблоны проектирования» .: Пер. с англ. - М.: Издательский дом "Вильямс", 2012. - 221 с.
9. И. В. Кузьмин «Основы моделирования сложных систем». –.: Вища школа, 1981. – 360 с.
10. Ларман Крэг. «Применение UML и шаблоны проектирования» - СПб: Питер, 2009. - 341 с.

# ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Command.java

**package** com.coursework.editor;

**public** **interface** Command {

**void** execute();

**void** reverse();

}

KeyboardState.java

**package** com.coursework.editor;

**public** **class** KeyboardState {

**private** **boolean** shiftPressed;

**private** **boolean** controlPressed;

**public** **boolean** isShiftPressed() {

**return** shiftPressed;

}

**public** **void** setShiftState(**boolean** pressed) {

shiftPressed = pressed;

}

**public** **boolean** isControlPressed() {

**return** controlPressed;

}

**public** **void** setControlState(**boolean** pressed) {

controlPressed = pressed;

}

}

Scene.java

**package** com.coursework.editor;

**import** java.awt.Color;

**import** java.awt.Graphics2D;

**import** java.awt.event.KeyEvent;

**import** java.awt.KeyEventDispatcher;

**import** java.awt.event.MouseEvent;

**import** java.util.Stack;

**import** javax.swing.event.MouseInputAdapter;

**import** com.coursework.editor.figures.Drawable;

**import** com.coursework.main.Debug;

**import** com.coursework.util.PriorityQueue;

**class** Scene {

**private** **boolean** clearRedo = **true**;

**private** **int** mousePrevX;

**private** **int** mousePrevY;

**int** sceneShiftX = 0;

**int** sceneShiftY = 0;

**private** Drawable selectedDrawable;

PriorityQueue<Drawable> drawables;

**private** Stack<Command> commands;

**private** Stack<Command> reversedCommands;

**private** **boolean** commandReversed = **false**;

**private** **void** initMouse() {

ScenesManager.*instance*().addMouseListener(**new** MouseInputAdapter() {

@Override

**public** **void** mouseDragged(MouseEvent e) {

**if** (!ScenesManager.*instance*().isFigureSelected()) {

**if** (selectedDrawable == **null**) {

sceneShiftX += e.getX() - mousePrevX;

sceneShiftY += e.getY() - mousePrevY;

mousePrevX = e.getX();

mousePrevY = e.getY();

}

}

actionExecuted();

}

@Override

**public** **void** mousePressed(MouseEvent e) {

**if** (e.getButton() == MouseEvent.***BUTTON1***) {

**if** (!ScenesManager.*instance*().isFigureSelected()) {

selectDrawableAtPosition(e.getX(), e.getY());

mousePrevX = e.getX();

mousePrevY = e.getY();

}

}

actionExecuted();

}

});

}

**private** **void** initKeyboard() {

ScenesManager.*instance*().addKeyboardListener(**new** KeyEventDispatcher() {

@Override

**public** **boolean** dispatchKeyEvent(KeyEvent e) {

**if** (e.getID() == KeyEvent.***KEY\_PRESSED***) {

**if** ((e.getKeyCode() == KeyEvent.***VK\_Z***) && ((e.getModifiers() & KeyEvent.***CTRL\_MASK***) != 0)) {

undo();

}

**if** ((e.getKeyCode() == KeyEvent.***VK\_Y***) && ((e.getModifiers() & KeyEvent.***CTRL\_MASK***) != 0)) {

redo();

}

**if** (e.getKeyCode() == KeyEvent.***VK\_ESCAPE***) {

selectedDrawable = **null**;

actionExecuted();

}

**if** (e.getKeyCode() == KeyEvent.***VK\_DELETE***) {

delete();

}

}

**return** **false**;

}

});

}

**protected** **void** delete() {

**if** (selectedDrawable != **null**) {

getDeleteCommand(selectedDrawable).execute();

selectedDrawable = **null**;

actionExecuted();

}

}

**private** **void** selectDrawableAtPosition(**int** x, **int** y) {

x -= sceneShiftX;

y -= sceneShiftY;

Drawable last = **null**, newSelected = **null**;

**boolean** found = **false**;

**for** (Drawable d : drawables) {

**if** (d.getArea().contains(x, y)) {

last = d;

**if** (d == selectedDrawable) {

found = **true**;

}

**if** (!found) {

newSelected = d;

}

}

}

selectedDrawable = newSelected;

**if** (selectedDrawable == **null**) {

selectedDrawable = last;

}

}

**private** **boolean** checkForRules() {

**boolean** unchanged = **true**;

**for** (Drawable d : drawables) {

Drawable newD = ScenesManager.*instance*().checkForRules(d);

**if** (newD == **null** || !newD.getArea().equals(d.getArea())) {

unchanged = **false**;

}

}

**if** (!unchanged) {

System.***out***.println("Can not perform action");

undo();

Debug.*log*("Can not perform action");

}

**return** unchanged;

}

**private** **void** init() {

commands = **new** Stack<>();

reversedCommands = **new** Stack<>();

drawables = **new** PriorityQueue<Drawable>();

initMouse();

initKeyboard();

}

Scene() {

init();

actionExecuted();

}

**void** clearCanvas() {

drawables.clear();

repaint();

sceneShiftX = 0;

sceneShiftY = 0;

}

**private** **void** repaint() {

ScenesManager.*instance*().repaint();

}

**void** redo() {

**if** (reversedCommands.size() > 0) {

commandReversed = **true**;

Command c = reversedCommands.pop();

c.execute();

}

}

**void** undo() {

**if** (commands.size() > 0) {

System.***out***.println("reverse");

clearRedo = **false**;

Command c = commands.peek();

reversedCommands.add(c);

c.reverse();

}

}

**public** **void** drawGrid(Graphics2D g) {

g.setColor(**new** Color(200, 200, 200));

**int** width = ScenesManager.*instance*().getSceneWidth();

**int** height = ScenesManager.*instance*().getSceneHeight();

**for** (**int** i = sceneShiftX % 50 -50;i < width; i+=50) {

g.drawLine(i, 0, i, height);

}

**for** (**int** i = sceneShiftY % 50 -50;i < height; i+=50) {

g.drawLine(0, i, width, i);

}

}

**public** **void** drawGridNumbers(Graphics2D g) {

**int** width = ScenesManager.*instance*().getSceneWidth();

**int** height = ScenesManager.*instance*().getSceneHeight();

Color bgColor = **new** Color(240, 240, 240);

**int** maxTextWidth = 0;

**for** (**int** i = sceneShiftY % 50 -50;i < height; i+=50) {

String s = Integer.*toString*((i - sceneShiftY) / 50);

maxTextWidth = Math.*max*(maxTextWidth, g.getFontMetrics().stringWidth(s));

}

maxTextWidth += 8;

g.setColor(bgColor);

g.fillRect(0, 0, width, 15);

g.fillRect(0, 0, maxTextWidth, height);

g.setColor(Color.***BLACK***);

**for** (**int** i = sceneShiftX % 50 -50;i < width; i+=50) {

String s = Integer.*toString*((i - sceneShiftX) / 50);

**int** size = g.getFontMetrics().stringWidth(s);

g.drawString(s, i - size/2, 12);

}

**int** textWidth = 0;

**for** (**int** i = sceneShiftY % 50 -50;i < height; i+=50) {

String s = Integer.*toString*((i - sceneShiftY) / 50);

textWidth = g.getFontMetrics().stringWidth(s);

g.drawString(Integer.*toString*((i - sceneShiftY) / 50), maxTextWidth - textWidth - 3, i + 5);

}

g.setColor(Color.***BLACK***);

g.drawLine(maxTextWidth, 15, width, 15);

g.drawLine(maxTextWidth, 15, maxTextWidth, height);

g.drawLine(maxTextWidth, height-1, width, height-1);

g.drawLine(width-1, 15, width-1, height);

g.setColor(bgColor);

g.fillRect(0, 0, maxTextWidth, 15);

}

**public** **void** drawScene(Graphics2D g) {

drawGrid(g);

**for** (Drawable d : drawables) {

d.selfPaint(g, ((d == selectedDrawable) ? Color.***BLUE*** : Color.***BLACK***), sceneShiftX, sceneShiftY);

}

drawGridNumbers(g);

}

**public** **void** actionExecuted() {

checkForRules();

**if** (clearRedo && !commandReversed) {

reversedCommands.clear();

} **else** {

commandReversed = **false**;

}

repaint();

ScenesManager.*instance*().setUndoEnabled(commands.size() > 0);

ScenesManager.*instance*().setRedoEnabled(reversedCommands.size() > 0);

ScenesManager.*instance*().setDeleteEnabled(selectedDrawable != **null**);

}

**public** Command getAddCommand(Drawable d) {

Drawable result = ScenesManager.*instance*().checkForRules(d);

**return** **new** Command() {

**private** Drawable drawable;

@Override

**public** **void** reverse() {

drawables.remove(drawable);

commands.remove(**this**);

clearRedo = **false**;

actionExecuted();

}

@Override

**public** **void** execute() {

**this**.drawable = result;

**if** (drawable != **null**) {

commands.add(**this**);

drawables.add(drawable, -(drawable.getPriority()+1));

clearRedo = **true**;

actionExecuted();

} **else** {

System.***out***.println("Deny " + d.toString());

}

}

};

}

**public** Command getDeleteCommand(Drawable drawable) {

**return** **new** Command() {

@Override

**public** **void** reverse() {

drawables.add(drawable, -(drawable.getPriority()+1));

commands.remove(**this**);

clearRedo = **false**;

actionExecuted();

}

@Override

**public** **void** execute() {

**if** (drawable != **null**) {

commands.add(**this**);

drawables.remove(drawable);

clearRedo = **true**;

actionExecuted();

}

}

};

}

**public** **void** clearHistory() {

commands.clear();

reversedCommands.clear();

}

**public** **void** clearSelection() {

selectedDrawable = **null**;

actionExecuted();

}

**public** Iterable<Drawable> getDrawables() {

**return** drawables;

}

}

ScenesManager.java

**package** com.coursework.editor;

**import** java.awt.Color;

**import** java.awt.Graphics2D;

**import** java.awt.KeyEventDispatcher;

**import** java.io.FileNotFoundException;

**import** java.util.LinkedList;

**import** java.util.List;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.event.MouseInputAdapter;

**import** com.coursework.editor.figures.Drawable;

**import** com.coursework.editor.figures.Figure;

**import** com.coursework.editor.figures.FiguresManager;

**import** com.coursework.editor.rules.RulesManager;

**import** com.coursework.files.XMLBuilder;

**import** com.coursework.files.XMLReader;

**import** com.coursework.files.XMLTag;

**import** com.coursework.main.Debug;

**import** com.coursework.main.Main;

**import** com.coursework.windows.Canvas;

**public** **class** ScenesManager {

**private** **static** ScenesManager *instance*;

**public** **static** ScenesManager instance() {

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

*instance* = **new** ScenesManager();

**return** *instance*;

}

**private** ScenesManager() {

scenes = **new** LinkedList<>();

}

**private** List<Scene> scenes;

**private** Scene currentScene;

**private** JButton deleteButton, undoButton, redoButton;

**private** Canvas canvas;

**private** JLabel canvasLabel;

**public** **void** setCanvas(Canvas canvas) {

**this**.canvas = canvas;

}

**public** **void** setDeleteButton(JButton button) {

**this**.deleteButton = button;

}

**public** **void** setRedoButton(JButton button) {

**this**.redoButton = button;

}

**public** **void** setUndoButton(JButton button) {

**this**.undoButton = button;

}

**public** **void** setDeleteEnabled(**boolean** enabled) {

deleteButton.setEnabled(enabled);

}

**public** **void** setUndoEnabled(**boolean** enabled) {

undoButton.setEnabled(enabled);

}

**public** **void** setRedoEnabled(**boolean** enabled) {

redoButton.setEnabled(enabled);

}

**public** **void** removeScene(**int** id) {

scenes.remove(id);

}

**public** **void** removeCurrent() {

scenes.remove(currentScene);

}

**public** **void** selectScene(**int** id) {

currentScene = scenes.get(id);

}

**public** **int** newScene() {

scenes.add(**new** Scene());

**return** scenes.size() - 1;

}

**public** **int** getCurrentId() {

**return** scenes.indexOf(currentScene);

}

**public** **void** repaint() {

canvas.repaint();

}

**public** **void** clearDrawings() {

currentScene.clearCanvas();

}

**public** **void** clearHistory() {

currentScene.clearHistory();

}

**public** **void** clearSelectionAtScene() {

currentScene.clearSelection();

}

**public** Command getAddCommand(Drawable d) {

**return** currentScene.getAddCommand(d);

}

**public** **void** draw(Graphics2D graphics) {

currentScene.drawScene(graphics);

FiguresManager.*getInstance*().drawSelectedFigure(graphics);

}

**public** **void** saveSceneToFile(String filename) {

XMLBuilder builder = **new** XMLBuilder(filename);

builder.beginProperty("scene");

**for** (Drawable d : currentScene.drawables) {

d.save(builder);

}

builder.endProperty("scene");

builder.flush();

}

**public** **void** loadSceneFromFile(String path) **throws** FileNotFoundException {

currentScene.clearCanvas();

XMLReader reader = **new** XMLReader(path);

**for** (XMLTag tag : reader.getRoot().getInnerTags()) {

**if** (tag.getName().equals("figure")) {

Figure f = FiguresManager.*getInstance*().getFigure(

tag.getInnerTag("figurePackage").getContent(),

tag.getInnerTag("figureName").getContent());

**if** (f != **null**) {

f.getDrawableLoader().load(tag);

} **else** {

Debug.*log*("Can not load figure " + tag.getInnerTag("figurePackage").getContent() + '.' + tag.getInnerTag("figureName").getContent());

}

} **else** {

Debug.*error*("Unrecognised tag " + tag.getName());

}

}

currentScene.clearHistory();

}

**public** Iterable<Drawable> getDrawables() {

**return** currentScene.getDrawables();

}

**public** **boolean** isFigureSelected() {

**return** FiguresManager.*getInstance*().getSelectedFigure() != **null**;

}

**public** **int** getOffsetX() {

**return** currentScene.sceneShiftX;

}

**public** **int** getOffsetY() {

**return** currentScene.sceneShiftY;

}

**public** **void** undo() {

currentScene.undo();

}

**public** **void** redo() {

currentScene.redo();

}

**public** **void** delete() {

currentScene.delete();

}

**public** **void** setCanvalLabel(JLabel label) {

**this**.canvasLabel = label;

}

**public** **void** writeOnCanvas(String text, Color color) {

canvasLabel.setText(text);

canvasLabel.setForeground(color);

}

**public** **int** getSceneWidth() {

**return** canvas.getWidth();

}

**public** **int** getSceneHeight() {

**return** canvas.getHeight();

}

**public** **void** addMouseListener(MouseInputAdapter mouseInputAdapter) {

Main.*addCanvasMouseListener*(mouseInputAdapter);

}

**public** **void** addKeyboardListener(KeyEventDispatcher keyEventDispatcher) {

Main.*addKeyboardEventDispatcher*(keyEventDispatcher);

}

**public** Drawable checkForRules(Drawable d) {

**return** RulesManager.*getInstance*().processDrawable(d);

}

}

BuilderFromXML.java

**package** com.coursework.editor.figures;

**import** com.coursework.files.XMLTag;

**public** **interface** BuilderFromXML {

**void** load(XMLTag tag);

}

Drawable.java

package com.coursework.editor.figures;

import java.awt.Color;

import java.awt.Graphics2D;

import java.awt.geom.Area;

import java.util.Collection;

import java.util.HashSet;

import java.util.Set;

import com.coursework.files.XMLBuilder;

public abstract class Drawable {

private int priority = Integer.MIN\_VALUE;

private Set<String> tags;

public Drawable() {

tags = new HashSet<>();

}

public void addTag(String s) {

tags.add(s);

}

public void addAllTags(Collection<String> tags) {

this.tags.addAll(tags);

}

public boolean hasTag(String tag) {

return tags.contains(tag);

}

public void setPriority(int p) {

priority = p;

}

public int getPriority() {

return priority;

}

public abstract void move(int x, int y);

public abstract Area getArea();

public abstract void selfPaint(Graphics2D g, Color primaryColor, int shiftX, int shiftY);

public abstract void save(XMLBuilder builder);

}

Figure.java

package com.coursework.editor.figures;

import com.coursework.editor.KeyboardState;

import com.coursework.files.XMLTag;

import java.awt.\*;

import java.util.Collection;

import java.util.LinkedList;

import java.util.List;

public abstract class Figure {

public static final double ROTATION\_STEP = 5.0;

private String figureName;

private String figurePackage;

private List<String> tags;

public String getPackageName() {

return figurePackage;

}

public String getName() {

return figureName;

}

public boolean hasTag(String tag) {

return tags.contains(tag);

}

List<String> getTags() {

return tags;

}

public BuilderFromXML getXMLBuilder() {

return new FigureBuilder();

}

public void selected() {}

protected class FigureBuilder implements BuilderFromXML {

@Override

public void load(XMLTag tag) {

figurePackage = tag.getInnerTag("package").getContent();

figureName = tag.getInnerTag("name").getContent();

tags = new LinkedList<>();

Collection<XMLTag> inner = tag.getInnerTags();

for (XMLTag t : inner) {

if (t.getName().equals("tag")) {

tags.add(t.getContent());

}

}

}

}

protected abstract class DrawableLoader implements BuilderFromXML {

}

public abstract BuilderFromXML getDrawableLoader();

@Override

public String toString() {

return getName();

}

public void drawStart() {}

public void drawEnd() {}

public void move(int x, int y) {}

public void keyPressed(KeyboardState state) {}

public abstract void rotate(double degree);

public abstract void draw(Graphics2D g, int shiftX, int shiftY);

}

FiguresLoader.java

package com.coursework.editor.figures;

import com.coursework.editor.rules.RulesManager;

import com.coursework.files.XMLReader;

import com.coursework.files.XMLTag;

import com.coursework.main.Debug;

import java.io.FileNotFoundException;

import java.util.Collection;

import java.util.LinkedList;

import java.util.List;

class FiguresLoader {

private String fileName;

private String packageName;

private List<Figure> figures;

public FiguresLoader(String fileName) throws FileNotFoundException {

this.fileName = fileName;

figures = new LinkedList<Figure>();

Debug.log("Loading figures from " + fileName);

loadPackage();

RulesManager.getInstance().loadRules(fileName);

}

public String getPackageName() {

return packageName;

}

public List<Figure> getFigures() {

return figures;

}

private void loadPackage() throws FileNotFoundException {

XMLReader reader = new XMLReader(fileName);

XMLTag root = reader.getRoot();

packageName = root.getInnerTag("name").getContent();

Collection<XMLTag> figs = root.getInnerTag("figures").getInnerTags();

for (XMLTag tag: figs) {

if (tag.getName().equals("figure")) {

if (tag.getInnerTag("package") == null) {

XMLTag pack = new XMLTag(tag, "package");

pack.addContent(packageName);

tag.addInnerTag(pack);

}

Figure f = null;

switch (tag.getInnerTag("type").getContent()) {

case "immutable":

f = new ImmutableFigure();

break;

case "line":

f = new LineFigure();

break;

default:

Debug.log("Cant load figure of type " + tag.getInnerTag("type").getContent());

break;

}

if (f != null) {

f.getXMLBuilder().load(tag);

figures.add(f);

}

}

}

}

}

FiguresManager.java

package com.coursework.editor.figures;

import com.coursework.editor.KeyboardState;

import com.coursework.editor.ScenesManager;

import com.coursework.main.Main;

import javax.swing.\*;

import javax.swing.event.MouseInputAdapter;

import javax.swing.event.TreeSelectionEvent;

import javax.swing.event.TreeSelectionListener;

import javax.swing.tree.DefaultMutableTreeNode;

import javax.swing.tree.DefaultTreeModel;

import javax.swing.tree.TreeSelectionModel;

import java.awt.\*;

import java.awt.event.KeyEvent;

import java.awt.event.MouseEvent;

import java.awt.event.MouseWheelEvent;

import java.io.FileNotFoundException;

import java.util.\*;

import java.util.List;

public class FiguresManager {

private static FiguresManager instance;

private JTree figuresView;

private Set<String> loadedPackages;

private List<Figure> figures;

private KeyboardState currentKeyboardState;

private boolean mouseOnCanvas = false;

private Figure selectedFigure;

public static FiguresManager getInstance() {

if (instance == null)

instance = new FiguresManager();

return instance;

}

private void initMouse() {

Main.addCanvasMouseListener(new MouseInputAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

mouseOnCanvas = true;

ScenesManager.instance().repaint();

}

@Override

public void mouseExited(MouseEvent e) {

mouseOnCanvas = false;

ScenesManager.instance().repaint();

}

@Override

public void mouseDragged(MouseEvent e){

if (selectedFigure != null) {

selectedFigure.move(e.getX() - ScenesManager.instance().getOffsetX(), e.getY() - ScenesManager.instance().getOffsetY());

}

ScenesManager.instance().repaint();

}

@Override

public void mouseMoved(MouseEvent e){

if (selectedFigure != null) {

selectedFigure.move(e.getX() - ScenesManager.instance().getOffsetX(), e.getY() - ScenesManager.instance().getOffsetY());

}

ScenesManager.instance().repaint();

}

@Override

public void mousePressed(MouseEvent e) {

if (e.getButton() == MouseEvent.BUTTON1) {

if (selectedFigure != null) {

selectedFigure.drawStart();

}

}

ScenesManager.instance().repaint();

}

@Override

public void mouseReleased(MouseEvent e) {

if (selectedFigure != null) {

if (e.getButton() == MouseEvent.BUTTON1 )

selectedFigure.drawEnd();

}

ScenesManager.instance().repaint();

}

public void mouseWheelMoved(MouseWheelEvent e) {

if (selectedFigure != null) {

selectedFigure.rotate(e.getWheelRotation() \* Figure.ROTATION\_STEP);

ScenesManager.instance().repaint();

}

}

});

}

private void initKeyboard() {

currentKeyboardState = new KeyboardState();

Main.addKeyboardEventDispatcher(new KeyEventDispatcher() {

@Override

public boolean dispatchKeyEvent(KeyEvent e) {

if (e.getID() == KeyEvent.KEY\_PRESSED) {

if (selectedFigure != null) {

if (e.getKeyCode() == KeyEvent.VK\_Q) {

selectedFigure.rotate(-Figure.ROTATION\_STEP);

}

if (e.getKeyCode() == KeyEvent.VK\_E) {

selectedFigure.rotate(Figure.ROTATION\_STEP);

}

ScenesManager.instance().repaint();

}

}

if (e.getKeyCode() == KeyEvent.VK\_SHIFT) {

if (e.getID() == KeyEvent.KEY\_PRESSED){

currentKeyboardState.setShiftState(true);

}

if (e.getID() == KeyEvent.KEY\_RELEASED) {

currentKeyboardState.setShiftState(false);

}

if (selectedFigure != null) {

selectedFigure.keyPressed(currentKeyboardState);

ScenesManager.instance().repaint();

}

}

return false;

}

});

}

private FiguresManager() {

initMouse();

initKeyboard();

loadedPackages = new HashSet<String>();

figures = new LinkedList<Figure>();

Main.addKeyboardEventDispatcher(new KeyEventDispatcher() {

@Override

public boolean dispatchKeyEvent(KeyEvent e) {

if (e.getID() == KeyEvent.KEY\_PRESSED && e.getKeyCode() == KeyEvent.VK\_ESCAPE) {

figuresView.clearSelection();

}

return false;

}

});

}

private void updateView() {

DefaultMutableTreeNode root = (DefaultMutableTreeNode)figuresView.getModel().getRoot();

root.removeAllChildren();

DefaultMutableTreeNode currentNode = null;

String currentPackage = "";

for (Figure f : figures) {

if (!f.getPackageName().equals(currentPackage)) {

currentNode = new DefaultMutableTreeNode(f.getPackageName());

root.add(currentNode);

currentPackage = f.getPackageName();

}

currentNode.add(new DefaultMutableTreeNode(f));

}

DefaultTreeModel model = new DefaultTreeModel(root);

figuresView.setModel(model);

}

public void connectView(JTree figuresView) {

this.figuresView = figuresView;

figuresView.getSelectionModel().setSelectionMode (TreeSelectionModel.SINGLE\_TREE\_SELECTION);

figuresView.setRootVisible(false);

figuresView.setShowsRootHandles(true);

figuresView.addTreeSelectionListener(new TreeSelectionListener() {

@Override

public void valueChanged(TreeSelectionEvent e) {

DefaultMutableTreeNode node = (DefaultMutableTreeNode) figuresView.getLastSelectedPathComponent();

if (node == null) {

selectedFigure = null;

} else {

if (node.isLeaf())

selectedFigure = (Figure) node.getUserObject();

selectedFigure.selected();

}

ScenesManager.instance().clearSelectionAtScene();

}

});

}

/\*

\* Load package from specified file

\* \*/

public void addPackage(String fileName) throws FileNotFoundException {

FiguresLoader loader;

loader = new FiguresLoader(fileName);

String name = loader.getPackageName();

if (loadedPackages.contains(name)) {

removePackage(name);

} else {

loadedPackages.add(name);

}

figures.addAll(loader.getFigures());

updateView();

//renewFactories();

}

/\*

\* Remove package with specified name

\*/

private void removePackage(String packageName) {

Iterator<Figure> it = figures.iterator();

while (it.hasNext()) {

Figure f = it.next();

if (f.getPackageName().equals(packageName)) {

it.remove();

}

}

}

public Figure getFigure(String pack, String name) {

for (Figure f : figures) {

if (f.getPackageName().equals(pack) && f.getName().equals(name)) {

return f;

}

}

return null;

}

public Figure getSelectedFigure() {

return selectedFigure;

}

public void drawSelectedFigure(Graphics2D graphics) {

if (selectedFigure != null && mouseOnCanvas) {

selectedFigure.draw(graphics, ScenesManager.instance().getOffsetX(), ScenesManager.instance().getOffsetY());

}

}

/\*

public void renewFactories() {

for (Figure f: figures)

f.setAddToSceneOperation(ScenesManager.instance().getAddCommands());

}\*/

}

ImmutableFigure.java

package com.coursework.editor.figures;

import com.coursework.editor.ScenesManager;

import com.coursework.editor.rules.RulesManager;

import com.coursework.files.XMLBuilder;

import com.coursework.files.XMLTag;

import com.coursework.main.Debug;

import java.awt.\*;

import java.awt.geom.\*;

import java.util.LinkedList;

import java.util.List;

class ImmutableFigure extends Figure {

ImmutableFigure() {

super();

physArea = new Area();

shapes = new LinkedList<>();

rotationDegree = 0;

}

private List<Shape> shapes;

private Area physArea;

private int posX;

private int posY;

private double rotationDegree;

private Area getArea() {

Area a = new Area(physArea);

AffineTransform transform = new AffineTransform();

transform.translate(posX, posY);

a.transform(transform);

transform.setToIdentity();

transform.rotate(Math.toRadians(rotationDegree), posX, posY);

a.transform(transform);

return a;

}

@Override

public void draw(Graphics2D g, int shiftX, int shiftY) {

Area current = getArea();

Drawable newArea = RulesManager.getInstance().processDrawable(new DrawableRepresentation(current, posX, posY));

boolean allowed = false;

if (newArea != null) {

current = newArea.getArea();

allowed = true;

}

AffineTransform transform = new AffineTransform();

transform.translate(shiftX, shiftY);

current.transform(transform);

g.setColor(Color.WHITE);

g.fill(current);

g.setColor(allowed ? Color.BLUE : Color.RED);

drawView(g, posX + shiftX, posY + shiftY, rotationDegree);

}

private void drawView(Graphics2D g, int x, int y, double rotation) {

AffineTransform t = new AffineTransform();

t.translate(x, y);

t.rotate(Math.toRadians(rotation));

for (Shape s: shapes) {

Shape sx = t.createTransformedShape(s);

g.draw(sx);

}

}

public void addShape(Shape s) {

if (s != null)

shapes.add(s);

}

private class ImmutableFigureBuilder extends FigureBuilder {

private void loadPhysics(XMLTag root) {

for (XMLTag areaTag: root.getInnerTags()) {

if (areaTag.getName().equals("area")) {

Area a = new Area();

switch (areaTag.getInnerTag("type").getContent()) {

case "rect":

a = new Area(new Rectangle2D.Double(

Double.parseDouble(areaTag.getInnerTag("x").getContent()),

Double.parseDouble(areaTag.getInnerTag("y").getContent()),

Double.parseDouble(areaTag.getInnerTag("w").getContent()),

Double.parseDouble(areaTag.getInnerTag("h").getContent())));

break;

case "arc":

a = new Area(new Arc2D.Double(

new Rectangle2D.Double(

Double.parseDouble(areaTag.getInnerTag("x").getContent()),

Double.parseDouble(areaTag.getInnerTag("y").getContent()),

Double.parseDouble(areaTag.getInnerTag("w").getContent()),

Double.parseDouble(areaTag.getInnerTag("h").getContent())),

Double.parseDouble(areaTag.getInnerTag("start").getContent()),

Double.parseDouble(areaTag.getInnerTag("size").getContent()),

Arc2D.CHORD));

break;

case "poly":

LinkedList<Integer> xCoord = new LinkedList<>(), yCoord = new LinkedList<>();

for (XMLTag t : areaTag.getInnerTags()) {

if (t.getName().equals("x"))

xCoord.add(Integer.parseInt(t.getContent()));

if (t.getName().equals("y"))

yCoord.add(Integer.parseInt(t.getContent()));

}

int[] x = new int[xCoord.size()], y = new int[yCoord.size()];

int counter = 0;

for (Integer i : xCoord) {

x[counter++] = i;

}

counter = 0;

for (Integer i : yCoord) {

y[counter++] = i;

}

Polygon p = new Polygon(x, y, xCoord.size());

a = new Area(p);

break;

default:

//Cant load

break;

}

if (areaTag.getInnerTag("action").getContent().equals("add")) {

physArea.add(a);

}

if (areaTag.getInnerTag("action").getContent().equals("sub")) {

physArea.subtract(a);

}

} else {

Debug.log("Non area tag in areas zone");

}

}

}

private void loadGraphics(XMLTag root) {

for (XMLTag areaTag: root.getInnerTags()) {

if (areaTag.getName().equals("area")) {

Shape s = null;

switch (areaTag.getInnerTag("type").getContent()) {

case "line":

s = new Line2D.Double(

Double.parseDouble(areaTag.getInnerTag("xStart").getContent()),

Double.parseDouble(areaTag.getInnerTag("yStart").getContent()),

Double.parseDouble(areaTag.getInnerTag("xEnd").getContent()),

Double.parseDouble(areaTag.getInnerTag("yEnd").getContent()));

break;

case "poly":

LinkedList<Integer> xCoord = new LinkedList<>(), yCoord = new LinkedList<>();

for (XMLTag t : areaTag.getInnerTags()) {

if (t.getName().equals("x"))

xCoord.add(Integer.parseInt(t.getContent()));

if (t.getName().equals("y"))

yCoord.add(Integer.parseInt(t.getContent()));

}

int[] x = new int[xCoord.size()], y = new int[yCoord.size()];

int counter = 0;

for (Integer i : xCoord) {

x[counter++] = (int)i;

}

counter = 0;

for (Integer i : yCoord) {

y[counter++] = (int)i;

}

s = new Polygon(x, y, xCoord.size());

break;

case "arc":

s = new Arc2D.Double(

new Rectangle2D.Double(

Double.parseDouble(areaTag.getInnerTag("x").getContent()),

Double.parseDouble(areaTag.getInnerTag("y").getContent()),

Double.parseDouble(areaTag.getInnerTag("w").getContent()),

Double.parseDouble(areaTag.getInnerTag("h").getContent())),

Double.parseDouble(areaTag.getInnerTag("start").getContent()),

Double.parseDouble(areaTag.getInnerTag("size").getContent()),

Arc2D.OPEN);

break;

default:

Debug.log("Can not load shape " + areaTag.getInnerTag("type").getContent());

break;

}

addShape(s);

}

}

}

@Override

public void load(XMLTag root) {

super.load(root);

loadPhysics(root.getInnerTag("physAreas"));

loadGraphics(root.getInnerTag("graphAreas"));

}

}

@Override

public BuilderFromXML getXMLBuilder() {

return new ImmutableFigureBuilder();

}

private class DrawableRepresentation extends Drawable {

int x;

int y;

double rotation;

Area a;

DrawableRepresentation(Area area, int xPos, int yPos) {

//super();

a = new Area(area);

x = xPos;

y = yPos;

rotation = rotationDegree;

addAllTags(getTags());

}

@Override

public void selfPaint(Graphics2D g, Color primaryColor, int shiftX, int shiftY) {

Area area = new Area(a);

AffineTransform transform = new AffineTransform();

transform.translate(shiftX, shiftY);

area.transform(transform);

g.setColor(Color.WHITE);

g.fill(area);

g.setColor(primaryColor);

drawView(g, x + shiftX, y + shiftY, rotation);

}

@Override

public void save(XMLBuilder builder) {

builder.beginProperty("figure");

builder.addProperty("figureName", getName());

builder.addProperty("figurePackage", getPackageName());

builder.addProperty("x", x);

builder.addProperty("y", y);

builder.addProperty("rotation", rotation);

builder.endProperty("figure");

}

@Override

public Area getArea() {

return a;

}

@Override

public void move(int x, int y) {

AffineTransform t = new AffineTransform();

t.translate(x, y);

a.transform(t);

}

}

private void addToScene() {

ScenesManager.instance().getAddCommand(new DrawableRepresentation(getArea(), posX, posY)).execute();

}

@Override

public void selected() {

rotationDegree = 0;

}

private class ImmutableDrawableLoader extends DrawableLoader {

@Override

public void load(XMLTag tag) {

rotationDegree = Double.parseDouble(tag.getInnerTag("rotation").getContent());

move(

Integer.parseInt(tag.getInnerTag("x").getContent()),

Integer.parseInt(tag.getInnerTag("y").getContent()));

addToScene();

}

}

@Override

public BuilderFromXML getDrawableLoader() {

return new ImmutableDrawableLoader();

}

@Override

public void drawStart() {

this.addToScene();

}

@Override

public void move(int x, int y) {

this.posX = x;

this.posY = y;

}

@Override

public void rotate(double degree) {

rotationDegree += degree;

}

}

LineFigure.java

package com.coursework.editor.figures;

import com.coursework.editor.KeyboardState;

import com.coursework.editor.ScenesManager;

import com.coursework.editor.rules.RulesManager;

import com.coursework.files.XMLBuilder;

import com.coursework.files.XMLTag;

import java.awt.\*;

import java.awt.geom.AffineTransform;

import java.awt.geom.Area;

import java.awt.geom.Rectangle2D;

public class LineFigure extends Figure {

private int startX = 0;

private int startY = 0;

private int currentX = 0;

private int currentY = 0;

private double width;

private boolean mouseDown = false;

private boolean shiftPressed = false;

private Area getArea() {

double deltaX = currentX - startX;

double deltaY = currentY - startY;

double length = Math.sqrt(Math.pow(deltaX, 2) + Math.pow(deltaY, 2));

Area a = new Area(new Rectangle2D.Double(startX - width/2, startY - width/2, length + width, width));

double theta = Math.atan2(deltaY, deltaX);

if (shiftPressed) {

double step = Math.toRadians(Figure.ROTATION\_STEP);

double proportion = theta/step;

theta = Math.round(proportion) \* step;

}

AffineTransform transform = new AffineTransform();

transform.rotate(theta, startX, startY);

a.transform(transform);

return a;

}

@Override

public void draw(Graphics2D g, int shiftX, int shiftY) {

if (mouseDown) {

Area current = getArea();

Drawable newArea = RulesManager.getInstance().processDrawable(new DrawableRepresentation());

boolean allowed = false;

if (newArea != null) {

current = newArea.getArea();

allowed = true;

}

AffineTransform transform = new AffineTransform();

transform.translate(shiftX, shiftY);

current.transform(transform);

g.setColor(allowed ? Color.GRAY : new Color(128, 0, 0));

g.fill(current);

}

}

private class LineFigureBuilder extends FigureBuilder {

@Override

public void load(XMLTag tag) {

super.load(tag);

width = Double.parseDouble(tag.getInnerTag("width").getContent());

}

}

@Override

public BuilderFromXML getXMLBuilder() {

return new LineFigureBuilder();

}

private class LineDrawableLoader extends DrawableLoader {

@Override

public void load(XMLTag tag) {

startX = Integer.parseInt(tag.getInnerTag("startX").getContent());

startY = Integer.parseInt(tag.getInnerTag("startY").getContent());

currentX = Integer.parseInt(tag.getInnerTag("endX").getContent());

currentY = Integer.parseInt(tag.getInnerTag("endY").getContent());

shiftPressed = Boolean.parseBoolean(tag.getInnerTag("isStraighten").getContent());

//Drawable d = ;

ScenesManager.instance().getAddCommand(new DrawableRepresentation()).execute();

//addCommandFactory.getCommand(d).execute();

}

}

@Override

public BuilderFromXML getDrawableLoader() {

return new LineDrawableLoader();

}

private class DrawableRepresentation extends Drawable {

int startX;

int startY;

int endX;

int endY;

Area area;

boolean isStraighten;

public DrawableRepresentation() {

area = new Area(LineFigure.this.getArea());

startX = LineFigure.this.startX;

startY = LineFigure.this.startY;

endX = LineFigure.this.currentX;

endY = LineFigure.this.currentY;

isStraighten = LineFigure.this.shiftPressed;

addAllTags(LineFigure.this.getTags());

}

@Override

public void selfPaint(Graphics2D g, Color primaryColor, int shiftX, int shiftY) {

Area a = new Area(area);

AffineTransform transform = new AffineTransform();

transform.translate(shiftX, shiftY);

a.transform(transform);

g.setColor(primaryColor);

g.fill(a);

}

@Override

public void save(XMLBuilder builder) {

builder.beginProperty("figure");

builder.addProperty("figureName", getName());

builder.addProperty("figurePackage", getPackageName());

builder.addProperty("startX", startX);

builder.addProperty("startY", startY);

builder.addProperty("endX", endX);

builder.addProperty("endY", endY);

builder.addProperty("isStraighten", isStraighten);

builder.endProperty("figure");

}

@Override

public Area getArea() {

return area;

}

@Override

public void move(int x, int y) {

AffineTransform t = new AffineTransform();

t.translate(x, y);

area.transform(t);

}

}

@Override

public void drawEnd() {

ScenesManager.instance().getAddCommand(new DrawableRepresentation()).execute();

mouseDown = false;

}

@Override

public void move(int x, int y) {

if (!mouseDown) {

startX = x;

startY = y;

}

currentX = x;

currentY = y;

}

@Override

public void keyPressed(KeyboardState state) {

shiftPressed = state.isShiftPressed();

}

@Override

public void rotate(double degree) {

//Ignore, rotated by mouse dragging

}

@Override

public void drawStart() {

mouseDown = true;

}

}

ContainerRule.java

package com.coursework.editor.rules;

import java.awt.geom.Area;

import com.coursework.editor.figures.Drawable;

public class ContainerRule extends Rule{

private boolean allowed;

private String containerTag;

private String contentTag;

public ContainerRule(boolean allowed) {

this.allowed = allowed;

}

@Override

public void addTag(String tag) {

if (containerTag == null)

containerTag = tag;

else

contentTag = tag;

}

@Override

public String toString() {

String result = super.toString();

return result + ':' + (allowed ? "allowed" : "disallowed " + containerTag + " " + contentTag);

}

@Override

public Drawable processDrawable(Drawable drawable, Iterable<Drawable> context) {

if (drawable != null) {

if (allowed) {

if (drawable.hasTag(contentTag)) {

boolean canPlace = false;

for (Drawable d : context) {

if (d.hasTag(containerTag)) {

Area a = new Area(drawable.getArea());

a.subtract(d.getArea());

if (a.isEmpty()) {

canPlace = true;

}

}

}

return canPlace ? drawable : null;

}

} else {

String pair = null;

if (drawable.hasTag(containerTag)) {

pair = contentTag;

}

if (drawable.hasTag(contentTag)) {

pair = containerTag;

}

if (pair != null) {

boolean hasIntersection = false;

for (Drawable d : context) {

if (d.hasTag(pair) && d != drawable) {

Area a = new Area(drawable.getArea());

a.intersect(d.getArea());

if (!a.isEmpty()) {

hasIntersection = true;

}

}

}

return hasIntersection ? null : drawable;

}

}

}

return drawable;

}

}

PriorityRule.java

package com.coursework.editor.rules;

import com.coursework.editor.figures.Drawable;

public class PriorityRule extends Rule {

private int priority;

public PriorityRule(int priority) {

this.priority = priority;

}

private boolean isApplicable(Drawable d) {

if (d == null)

return false;

boolean result = false;

for (String tag : tags) {

result |= d.hasTag(tag);

}

return result;

}

@Override

public Drawable processDrawable(Drawable d, Iterable<Drawable> context) {

if (isApplicable(d) && priority > d.getPriority()) {

d.setPriority(priority);

}

return d;

}

}

Rule.java

package com.coursework.editor.rules;

import java.util.HashSet;

import java.util.Set;

import com.coursework.editor.figures.Drawable;

public abstract class Rule {

protected Set<String> tags;

public Rule() {

tags = new HashSet<>();

}

public void addTag(String tag) {

tags.add(tag);

}

public abstract Drawable processDrawable(Drawable d, Iterable<Drawable> context);

}

RulesLoader.java

package com.coursework.editor.rules;

import java.io.FileNotFoundException;

import java.util.Collection;

import java.util.LinkedList;

import java.util.List;

import com.coursework.files.XMLReader;

import com.coursework.files.XMLTag;

import com.coursework.main.Debug;

public class RulesLoader {

String filename;

List<Rule> rules;

public RulesLoader(String filename) throws FileNotFoundException {

this.filename = filename;

loadRules();

Debug.log("Loading rules from " + filename);

}

public Collection<Rule> getRules() {

return rules;

}

private void loadRules() throws FileNotFoundException {

rules = new LinkedList<>();

XMLReader reader = new XMLReader(filename);

Collection<XMLTag> rulesTags = reader.getRoot().getInnerTag("rules").getInnerTags();

for (XMLTag tag : rulesTags) {

if (tag.getName().equals("rule")) {

Rule rule = null;

switch (tag.getInnerTag("type").getContent()) {

case "priority":

rule = new PriorityRule(Integer.parseInt(tag.getInnerTag("value").getContent()));

break;

case "container":

rule = new ContainerRule(tag.getInnerTag("allowed").getContent().equals("true"));

default:

Debug.log("Can not load rule of type " + tag.getInnerTag("type").getContent());

break;

}

Collection<XMLTag> innerTags = tag.getInnerTags();

for (XMLTag inner : innerTags) {

if (inner.getName().equals("tag")) {

rule.addTag(inner.getContent());

}

}

rules.add(rule);

}

}

}

}

RulesManager.java

package com.coursework.editor.rules;

import java.io.FileNotFoundException;

import java.util.LinkedList;

import java.util.List;

import com.coursework.editor.figures.Drawable;

import com.coursework.editor.ScenesManager;

public class RulesManager {

private static RulesManager instance;

private RulesManager() {

rules = new LinkedList<>();

}

private List<Rule> rules;

public Drawable processDrawable(Drawable d) {

Iterable<Drawable> context = ScenesManager.instance().getDrawables();

Drawable result = d;

for (Rule rule : rules) {

result = rule.processDrawable(result, context);

}

return result;

}

public static RulesManager getInstance() {

if (instance == null)

instance = new RulesManager();

return instance;

}

public void loadRules(String filename) throws FileNotFoundException {

RulesLoader loader = new RulesLoader(filename);

rules.addAll(loader.getRules());

}

}

XMLBuilder.java

**package** com.coursework.files;

**import** com.coursework.main.Debug;

**public** **class** XMLBuilder {

**private** XMLWriter writer;

XMLTag current;

**public** XMLBuilder(String filename) {

writer = **new** XMLWriter(filename);

}

**public** **void** beginProperty(String name) {

**if** (current == **null**) {

current = **new** XMLTag(**null**, name);

writer.setRoot(current);

} **else** {

XMLTag tag = **new** XMLTag(current, name);

current.addInnerTag(tag);

current = tag;

}

}

**public** **void** endProperty(String name) {

**if** (name == current.getName()) {

current = current.getParent();

} **else** {

//Something done wrong

Debug.*log*("Incorrect XML use");

}

}

**public** **void** addProperty(String name, Object value) {

beginProperty(name);

current.addContent(value.toString());

endProperty(name);

}

**public** **void** flush() {

writer.write();

}

}

XMLReader.java

package com.coursework.files;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class XMLReader {

public XMLReader(String filename) throws FileNotFoundException {

Scanner s = new Scanner(new File(filename));

current = new XMLTag(null, "root");

while (s.hasNextLine()) {

processString(s.nextLine());

}

s.close();

}

XMLTag current;

XMLTag root;

private static final char[] space = {' ', '\t'};

private String trim(String source) {

if (source == null || source.length() == 0) {

return source;

}

int startIndex = -1, lastIndex = source.length();

boolean spacing;

do {

startIndex++;

spacing = false;

for (int i = 0;i < space.length; i++) {

if (space[i] == source.charAt(startIndex)) {

spacing = true;

}

}

} while (spacing && startIndex < source.length() -1);

do {

lastIndex--;

spacing = false;

for (int i = 0;i < space.length; i++) {

if (space[i] == source.charAt(lastIndex)) {

spacing = true;

}

}

} while (spacing && lastIndex > 0);

return source.substring(startIndex, lastIndex + 1);

}

private void processString(String s) {

s = trim(s);

int tagStart = s.indexOf('<');

int tagEnd = s.indexOf('>');

if (tagStart != -1 && tagEnd > tagStart) {

if (tagStart != 0) {

current.addContent(s.substring(0, tagStart));

}

if (s.charAt(tagStart + 1) == '/') {

current = current.getParent();

} else {

XMLTag newTag = new XMLTag(current, s.substring(1, tagEnd));

current.addInnerTag(newTag);

current = newTag;

if (root == null)

root = current;

}

if (tagEnd != s.length() - 1) {

processString(s.substring(tagEnd + 1, s.length()));

}

} else {

current.addContent(s);

}

/\*

if (s.charAt(0) == '<' && s.charAt(s.length()-1) == '>') {

if (s.charAt(1) == '/') {

current = current.getParent();

} else {

XMLTag newTag = new XMLTag(current, s.substring(1, s.length()-1));

current.addInnerTag(newTag);

current = newTag;

if (root == null) {

root = current;

}

}

} else {

current.addContent(s);

}\*/

}

public XMLTag getRoot() {

//printTag(root, 0);

return root;

}

}

XMLTag.java

package com.coursework.files;

import java.util.Collection;

import java.util.LinkedList;

import java.util.List;

public class XMLTag {

private List<XMLTag> inner;

private XMLTag parent;

private String name;

private String content;

public XMLTag(XMLTag parent, String name) {

this.parent = parent;

this.name = name;

inner = new LinkedList<XMLTag>();

}

public XMLTag getParent() {

return parent;

}

public String getName() {

return name;

}

public String getContent() {

return content;

}

public void addContent(String s) {

if (content == null)

content = s;

else

content += '\n' + s;

}

public void addInnerTag(XMLTag tag) {

inner.add(tag);

}

public XMLTag getInnerTag(String name) {

for (XMLTag tag : inner) {

if (tag.name.equals(name)) {

return tag;

}

}

return null;

}

public Collection<XMLTag> getInnerTags() {

return inner;

}

public String getAsText(int depth) {

String prefix = "";

for (int i = 0;i < depth; i++)

prefix += '\t';

String result = "";

result += prefix + "<" + name + ">\n";

if (content != null) {

result += prefix + '\t' + content + '\n';

}

for (XMLTag t : inner) {

result += t.getAsText(depth + 1);

}

result += prefix + "</" + name + ">\n";

return result;

}

}

XMLWriter.java

package com.coursework.files;

import java.io.FileNotFoundException;

import java.io.PrintWriter;

import com.coursework.main.Debug;

public class XMLWriter {

XMLTag root;

String filename;

public XMLWriter(String filename) {

this.filename = filename;

}

public void setRoot(XMLTag root) {

this.root = root;

}

public XMLTag getRoot() {

return root;

}

public void addToRoot(XMLTag tag) {

root.addInnerTag(tag);

}

public void write() {

PrintWriter writer;

try {

writer = new PrintWriter(filename);

writer.write(root.getAsText(0));

writer.close();

} catch (FileNotFoundException e) {

Debug.log("Can not find file " + filename);

}

}

}

Debug.java

package com.coursework.main;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Calendar;

public class Debug {

private static final String logFilename = "debug.log";

private static final DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");

public static void clear() {

try {

FileWriter out = new FileWriter(logFilename);

out.append("");

out.close();

} catch (IOException e) {

// TODO Auto-generated catch block

//cant clear

e.printStackTrace();

}

}

private static void writeMessage(String message, String prefix) {

FileWriter out;

try {

out = new FileWriter(logFilename, true);

out.append(prefix + message + "\n");

out.close();

} catch (IOException e) {

// TODO Auto-generated catch block

//cant write

e.printStackTrace();

}

}

public static void error(String message) {

Calendar now = Calendar.getInstance();

String time = "[" + dateFormat.format(now.getTime()) + "]";

writeMessage(message, time + "[ERR] ");

}

public static void log(String message) {

Calendar now = Calendar.getInstance();

String time = "[" + dateFormat.format(now.getTime()) + "]";

writeMessage(message, time + "[LOG] ");

}

public static String getAll() {

StringBuilder sb = new StringBuilder();

try {

BufferedReader br = new BufferedReader(new FileReader(logFilename));

String line = br.readLine();

while (line != null) {

sb.append(line);

sb.append(System.getProperty("line.separator"));

line = br.readLine();

}

br.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

//open

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

//read

e.printStackTrace();

}

return sb.toString();

}

}

Main.java

package com.coursework.main;

import java.awt.KeyEventDispatcher;

import java.awt.KeyboardFocusManager;

import java.io.FileNotFoundException;

import javax.swing.SwingUtilities;

import javax.swing.UIManager;

import javax.swing.WindowConstants;

import javax.swing.event.MouseInputAdapter;

import com.coursework.editor.ScenesManager;

import com.coursework.editor.figures.FiguresManager;

import com.coursework.windows.AboutWindow;

import com.coursework.windows.DebugWindow;

import com.coursework.windows.MainWindow;

public class Main {

private static MainWindow mainWindow;

private static AboutWindow aboutWindow;

private static DebugWindow debugWindow;

public static void main(String[] args) {

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception e) {

Debug.log("Can not set system UI style");

e.printStackTrace();

}

//Init settings and debug and start main window

SwingUtilities.invokeLater(new Runnable() {

public void run() {

Debug.clear();

debugWindow = new DebugWindow();

aboutWindow = new AboutWindow();

mainWindow = new MainWindow();

Debug.log("Windows created");

ScenesManager.instance().setCanvas(mainWindow.getCanvas());

int id = ScenesManager.instance().newScene();

ScenesManager.instance().selectScene(id);

Debug.log("Scene manager initialised");

FiguresManager.getInstance().connectView(mainWindow.getFiguresView());

try {

FiguresManager.getInstance().addPackage("data/default.figures");

} catch (FileNotFoundException e) {

Debug.log("Can not load default pack");

System.out.println("Can not load default pack");

}

Debug.log("Default figures loaded");

mainWindow.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

mainWindow.setVisible(true);

Debug.log("Application started");

}

});

}

public static void addKeyboardEventDispatcher(KeyEventDispatcher k) {

KeyboardFocusManager manager = KeyboardFocusManager.getCurrentKeyboardFocusManager();

manager.addKeyEventDispatcher(k);

}

public static void redraw() {

mainWindow.getCanvas().repaint();

}

public static void showAboutWindow() {

if (aboutWindow != null) {

aboutWindow.setVisible(true);

} else {

Debug.log("About window");

}

}

public static void showDebugWindow() {

if (debugWindow != null) {

debugWindow.renew();

debugWindow.setVisible(true);

}

}

public static void addCanvasMouseListener(MouseInputAdapter mouse) {

mainWindow.getCanvas().addMouseListener(mouse);

mainWindow.getCanvas().addMouseMotionListener(mouse);

mainWindow.getCanvas().addMouseWheelListener(mouse);

}

}

AboutWindow.java

/\*

\* Created by JFormDesigner on Tue Apr 12 14:49:59 EEST 2016

\*/

package com.coursework.windows;

import java.awt.\*;

import javax.swing.\*;

import com.coursework.main.Debug;

/\*\*

\* @author D PUpkin

\*/

public class AboutWindow extends JFrame {

/\*\*

\*

\*/

private static final long serialVersionUID = -1243187947111396972L;

public AboutWindow() {

initComponents();

setResizable(false);

Debug.log("About window initialized");

}

private void initComponents() {

// JFormDesigner - Component initialization - DO NOT MODIFY //GEN-BEGIN:initComponents

// Generated using JFormDesigner Evaluation license - D PUpkin

label1 = new JLabel();

//======== this ========

Container contentPane = getContentPane();

//---- label1 ----

label1.setText("<html>\n\tBuildings planner\n\t<br><br>\n\tCreated by Kanivets Dmytro\n</html>");

GroupLayout contentPaneLayout = new GroupLayout(contentPane);

contentPane.setLayout(contentPaneLayout);

contentPaneLayout.setHorizontalGroup(

contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addContainerGap()

.addComponent(label1, GroupLayout.PREFERRED\_SIZE, 144, GroupLayout.PREFERRED\_SIZE)

.addContainerGap(GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

contentPaneLayout.setVerticalGroup(

contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addContainerGap()

.addComponent(label1, GroupLayout.PREFERRED\_SIZE, 67, GroupLayout.PREFERRED\_SIZE)

.addContainerGap(GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

pack();

setLocationRelativeTo(getOwner());

// JFormDesigner - End of component initialization //GEN-END:initComponents

}

// JFormDesigner - Variables declaration - DO NOT MODIFY //GEN-BEGIN:variables

// Generated using JFormDesigner Evaluation license - D PUpkin

private JLabel label1;

// JFormDesigner - End of variables declaration //GEN-END:variables

}

Canvas.java

package com.coursework.windows;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.RenderingHints;

import javax.swing.JPanel;

import com.coursework.editor.ScenesManager;

public class Canvas extends JPanel {

private static final long serialVersionUID = 5431118052073932479L;

public Canvas() {

super();

setOpaque(true);

setBackground(Color.white);

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D graphics = (Graphics2D)g;

graphics.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

graphics.setRenderingHint(RenderingHints.KEY\_RENDERING, RenderingHints.VALUE\_RENDER\_QUALITY);

graphics.setRenderingHint(RenderingHints.KEY\_INTERPOLATION, RenderingHints.VALUE\_INTERPOLATION\_BICUBIC);

ScenesManager.instance().draw(graphics);

}

/\*

private void initComponents() {

// JFormDesigner - Component initialization - DO NOT MODIFY //GEN-BEGIN:initComponents

// Generated using JFormDesigner Evaluation license - D Pupkin

//======== this ========

setBackground(Color.white);

// JFormDesigner evaluation mark

setBorder(new javax.swing.border.CompoundBorder(

new javax.swing.border.TitledBorder(new javax.swing.border.EmptyBorder(0, 0, 0, 0),

"JFormDesigner Evaluation", javax.swing.border.TitledBorder.CENTER,

javax.swing.border.TitledBorder.BOTTOM, new java.awt.Font("Dialog", java.awt.Font.BOLD, 12),

java.awt.Color.red), getBorder())); addPropertyChangeListener(new java.beans.PropertyChangeListener(){public void propertyChange(java.beans.PropertyChangeEvent e){if("border".equals(e.getPropertyName()))throw new RuntimeException();}});

GroupLayout layout = new GroupLayout(this);

setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup()

.addGap(0, 400, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

layout.createParallelGroup()

.addGap(0, 300, Short.MAX\_VALUE)

);

// JFormDesigner - End of component initialization //GEN-END:initComponents

}\*/

}

MainWindow.java

/\*

\* Created by JFormDesigner on Tue Apr 12 14:34:20 EEST 2016

\*/

package com.coursework.windows;

import com.coursework.editor.ScenesManager;

import com.coursework.editor.figures.FiguresManager;

import com.coursework.main.Debug;

import com.coursework.main.Main;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import javax.swing.filechooser.FileNameExtensionFilter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

/\*\*

\* @author D PUpkin

\*/

public class MainWindow extends JFrame {

/\*\*

\*

\*/

private static final long serialVersionUID = -4577278481266674054L;

private static final boolean DEBUG\_ENABLED = true;

public MainWindow() {

initComponents();

if (DEBUG\_ENABLED) {

initDebug();

}

ScenesManager.instance().setDeleteButton(deleteButton);

ScenesManager.instance().setUndoButton(undoButton);

ScenesManager.instance().setRedoButton(redoButton);

Debug.log("Main window initialized");

}

private void helpAboutClick(ActionEvent e) {

Main.showAboutWindow();

}

private void debugShowClick(ActionEvent e) {

Main.showDebugWindow();

}

private void fileExitClick(ActionEvent e) {

Frame[] frames = Frame.getFrames();

for (int i = 0;i < frames.length; i++) {

frames[i].dispatchEvent(new WindowEvent(frames[i], WindowEvent.WINDOW\_CLOSING));

}

}

private void initDebug() {

debug = new JMenu();

debugShow = new JMenuItem();

debug.setText("Debug");

debugShow.setText("Show");

debugShow.addActionListener((e) -> debugShowClick(e));

debug.add(debugShow);

menuBar.add(debug);

}

private void fileOpenActionClick(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(System.getProperty("user.home"));

FileNameExtensionFilter filter = new FileNameExtensionFilter("Drawings file", "scene");

fileChooser.addChoosableFileFilter(filter);

fileChooser.setAcceptAllFileFilterUsed(false);

int result = fileChooser.showOpenDialog(this);

if (result == JFileChooser.APPROVE\_OPTION) {

try {

ScenesManager.instance().loadSceneFromFile(fileChooser.getSelectedFile().getAbsolutePath());

} catch (FileNotFoundException e1) {

Debug.log("File " + fileChooser.getSelectedFile().getAbsolutePath() + " not found");

}

Main.redraw();

}

}

private void saveSchemeClick(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(System.getProperty("user.home"));

FileNameExtensionFilter filter = new FileNameExtensionFilter("Drawings file", "scene");

fileChooser.addChoosableFileFilter(filter);

fileChooser.setAcceptAllFileFilterUsed(false);

int result = fileChooser.showSaveDialog(this);

if (result == JFileChooser.APPROVE\_OPTION) {

String filename = fileChooser.getSelectedFile().getAbsolutePath();

ScenesManager.instance().saveSceneToFile(filename + (filename.endsWith(".scene") ? "" : ".scene"));

}

}

private void loadPackageActionClick(ActionEvent e) {

String path = System.getProperty("user.home");

try {

path = new File(".").getCanonicalPath() + "/data";

} catch (IOException ex) {

ex.printStackTrace();

}

JFileChooser fileChooser = new JFileChooser(path);

FileNameExtensionFilter filter = new FileNameExtensionFilter("Furniture package", "figures");

fileChooser.addChoosableFileFilter(filter);

fileChooser.setAcceptAllFileFilterUsed(false);

int result = fileChooser.showOpenDialog(this);

if (result == JFileChooser.APPROVE\_OPTION) {

File selectedFile = fileChooser.getSelectedFile();

Debug.log("Figures file chosen: " + selectedFile.getAbsolutePath());

try {

FiguresManager.getInstance().addPackage(selectedFile.getAbsolutePath());

} catch (FileNotFoundException e1) {

//Main.showMessage("File not found");

}

}

}

/\*

public JList<String> getFiguresList() {

return figuresList;

}

\*/

public JTree getFiguresView() {

return figuresTree;

}

private void newFileClick(ActionEvent e) {

ScenesManager.instance().removeCurrent();

int id = ScenesManager.instance().newScene();

ScenesManager.instance().selectScene(id);

}

private void redoButtonActionPerformed(ActionEvent e) {

ScenesManager.instance().redo();

}

private void deleteButtonActionPerformed(ActionEvent e) {

ScenesManager.instance().delete();

}

private void undoButtonActionPerformed(ActionEvent e) {

ScenesManager.instance().undo();

}

private void initComponents() {

// JFormDesigner - Component initialization - DO NOT MODIFY //GEN-BEGIN:initComponents

// Generated using JFormDesigner Evaluation license - D Pupkin

menuBar = new JMenuBar();

file = new JMenu();

newFile = new JMenuItem();

fileOpen = new JMenuItem();

fileSave = new JMenuItem();

loadPackage = new JMenuItem();

fileExit = new JMenuItem();

help = new JMenu();

helpAbout = new JMenuItem();

canvas = new Canvas();

redoButton = new JButton();

undoButton = new JButton();

deleteButton = new JButton();

figuresScroll = new JScrollPane();

figuresTree = new JTree();

//======== this ========

Container contentPane = getContentPane();

//======== menuBar ========

{

//======== file ========

{

file.setText("File");

//---- newFile ----

newFile.setText("New file");

newFile.addActionListener(e -> newFileClick(e));

file.add(newFile);

file.addSeparator();

//---- fileOpen ----

fileOpen.setText("Open file");

fileOpen.addActionListener(e -> fileOpenActionClick(e));

file.add(fileOpen);

//---- fileSave ----

fileSave.setText("Save file");

fileSave.addActionListener(e -> saveSchemeClick(e));

file.add(fileSave);

file.addSeparator();

//---- loadPackage ----

loadPackage.setText("Load figures package");

loadPackage.addActionListener(e -> loadPackageActionClick(e));

file.add(loadPackage);

file.addSeparator();

//---- fileExit ----

fileExit.setText("Exit");

fileExit.addActionListener(e -> fileExitClick(e));

file.add(fileExit);

}

menuBar.add(file);

//======== help ========

{

help.setText("Help");

//---- helpAbout ----

helpAbout.setText("About");

helpAbout.addActionListener(e -> helpAboutClick(e));

help.add(helpAbout);

}

menuBar.add(help);

}

setJMenuBar(menuBar);

//---- redoButton ----

redoButton.setText("Redo");

redoButton.setFocusable(false);

redoButton.addActionListener(e -> redoButtonActionPerformed(e));

//---- undoButton ----

undoButton.setText("Undo");

undoButton.setFocusable(false);

undoButton.setPreferredSize(new Dimension(58, 23));

undoButton.addActionListener(e -> undoButtonActionPerformed(e));

//---- deleteButton ----

deleteButton.setText("Delete");

deleteButton.setFocusable(false);

deleteButton.addActionListener(e -> deleteButtonActionPerformed(e));

//======== figuresScroll ========

{

figuresScroll.setViewportBorder(null);

//---- figuresTree ----

figuresTree.setFocusable(false);

figuresScroll.setViewportView(figuresTree);

}

GroupLayout contentPaneLayout = new GroupLayout(contentPane);

contentPane.setLayout(contentPaneLayout);

contentPaneLayout.setHorizontalGroup(

contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addGroup(contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addGap(35, 35, 35)

.addGroup(contentPaneLayout.createParallelGroup(GroupLayout.Alignment.TRAILING)

.addComponent(deleteButton, GroupLayout.PREFERRED\_SIZE, 74, GroupLayout.PREFERRED\_SIZE)

.addComponent(redoButton, GroupLayout.PREFERRED\_SIZE, 74, GroupLayout.PREFERRED\_SIZE)

.addComponent(undoButton, GroupLayout.PREFERRED\_SIZE, 74, GroupLayout.PREFERRED\_SIZE)))

.addGroup(contentPaneLayout.createSequentialGroup()

.addContainerGap()

.addComponent(figuresScroll, GroupLayout.PREFERRED\_SIZE, 128, GroupLayout.PREFERRED\_SIZE)))

.addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)

.addComponent(canvas, GroupLayout.DEFAULT\_SIZE, 860, Short.MAX\_VALUE)

.addContainerGap())

);

contentPaneLayout.setVerticalGroup(

contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addGap(10, 10, 10)

.addGroup(contentPaneLayout.createParallelGroup()

.addGroup(contentPaneLayout.createSequentialGroup()

.addGap(0, 15, Short.MAX\_VALUE)

.addComponent(figuresScroll, GroupLayout.PREFERRED\_SIZE, 421, GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(undoButton, GroupLayout.PREFERRED\_SIZE, GroupLayout.DEFAULT\_SIZE, GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)

.addComponent(redoButton)

.addGap(18, 18, 18)

.addComponent(deleteButton))

.addComponent(canvas, GroupLayout.DEFAULT\_SIZE, 0, Short.MAX\_VALUE))

.addContainerGap())

);

pack();

setLocationRelativeTo(getOwner());

// JFormDesigner - End of component initialization //GEN-END:initComponents

figuresScroll.setBorder(BorderFactory.createLineBorder(Color.BLACK));

JLabel label = new JLabel("");

label.setHorizontalAlignment(SwingConstants.LEFT);

canvas.add(label);

ScenesManager.instance().setCanvalLabel(label);

}

public Canvas getCanvas() {

return canvas;

}

// JFormDesigner - Variables declaration - DO NOT MODIFY //GEN-BEGIN:variables

// Generated using JFormDesigner Evaluation license - D Pupkin

private JMenuBar menuBar;

private JMenu file;

private JMenuItem newFile;

private JMenuItem fileOpen;

private JMenuItem fileSave;

private JMenuItem loadPackage;

private JMenuItem fileExit;

private JMenu help;

private JMenuItem helpAbout;

private Canvas canvas;

private JButton redoButton;

private JButton undoButton;

private JButton deleteButton;

private JScrollPane figuresScroll;

private JTree figuresTree;

// JFormDesigner - End of variables declaration //GEN-END:variables

private JMenu debug;

private JMenuItem debugShow;

//private JList<String> figuresList;

}