Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

КУРСОВА РОБОТА

з дисципліни: «Інженерія програмного забезпечення»

на тему: програмний додаток «Модуль обробки растрових зображень»

Студента 2-го курсу групи ІО-12

Напряму підготовки: 6.050102 Комп’ютерна інженерія

Спеціальності: 7.8.05010201 Комп’ютерні системи та мережі

Нестерука Юрія Олександровича

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

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

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

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

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

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

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

Київ 2013

ЗМІСТ

[ВСТУП 3](#_Toc354009437)

[1 ТЕХНІЧНЕ ЗАВДАННЯ 4](#_Toc354009438)

[1.1 Загальне завдання 4](#_Toc354009439)

[1.2 Функціональність 4](#_Toc354009440)

[1.3 Вимоги до реалізації 4](#_Toc354009441)

[2 ПРОЕКТУВАННЯ БІБЛІОТЕКИ КЛАСІВ 5](#_Toc354009443)

[2.1 Пакет ImageUtil 5](#_Toc354009444)

[Опис класів пакету ImageUtil 6](#_Toc354009445)

[2.2 Пакет process.app.image.util 7](#_Toc354009446)

[Опис класів (процесів) пакету process.app.image.util 7](#_Toc354009447)

[2.3 Пакет process.app.image.util 8](#_Toc354009448)

[Опис класів (процесів) пакету process.app.vector 8](#_Toc354009449)

[3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 9](#_Toc354009450)

[3.1 Специфікація класів пакету ImageUtil 9](#_Toc354009451)

[3.2 Згенерована javadoc документація 12](#_Toc354009452)

[ВИСНОВКИ 16](#_Toc354009471)

[СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 17](#_Toc354009472)

[ДОДАТОК А. ПРОГРАМНИЙ КОД ПРОЕКТУ 19](#_Toc354009473)

[А.1 Клас LoadAndSaveTool 19](#_Toc354009474)

[А.2 Клас Клас MatrixRepresentationTool 19](#_Toc354009475)

[А.3 Клас BrightnessTool 19](#_Toc354009476)

[А.4 Клас DisplayerTool 22](#_Toc354009477)

[А.5 Клас ImageUtil 25](#_Toc354009478)

[А.6 Клас (процес) LoadImageProcess 26](#_Toc354009479)

[А.7 Клас (процес) SaveImageProcess 26](#_Toc354009480)

[А.8 Клас (процес) ToGrayScaleImageProcess 29](#_Toc354009481)

[А.9 Клас (процес) ToImageProcess 28](#_Toc354009482)

[ДОДАТОК Б. СТРУКТУРА ПРОЕКТУ 30](#_Toc354009483)

[ДОДАТОК В. АЛГОРИТМ РОБОТИ РАСТЕРИЗАТОРА 31](#_Toc354009484)

ВСТУП

У курсовій роботі необхідно розробити пакет утилітних класів для роботи з растровими зображеннями. Також необхідно реалізувати процеси обробки зображень для їх подальшого використання інтерпритатором.

У роботі розглядаються основні принципи роботи з зображеннями як з масивами чисел, та їх растеризація.

Розробка програмного забезпечення виконується на мові програмування Java. Робота містить повну документацію, а так же програмний код проекту та UML діаграму класів.

# Технічне завдання

## Загальне завдання

Необхідно побудувати пакет утилітних класів для роботи з растровими зображеннями кольорової моделі RGB. Повинні бути реалізовані методи для завантаження і збереження зображень, представлення їх у матричній формі, знаходження яскравостей їх пікселів, растеризації їх сіткового представлення.

## Функціональність

Бібліотека класів повинна мати наступну функціональність:

* можливість завантаження/збереження зображень за вказаним шляхом;
* представлення зображень у матричному вигляді (двовимірним масивом чисел);
* знаходження матриці яскравостей пікселів зображення, можливість вказувати власні коефіцієнти для RGB каналів;
* растеризація сіткового представлення зображень у різних режимах відображення;
* реалізація процесів, які може виконувати інтерпритатор.

## Вимоги до реалізації

* мова програмування Java з використанням бібліотеки AWT;
* проект має бути повністю задокументований за допомогою JavaDoc;
* проект має повністю відповідати правилам CheckStyle;

# Проектування бібліотеки класів

## 2.1 Пакет ImageUtil

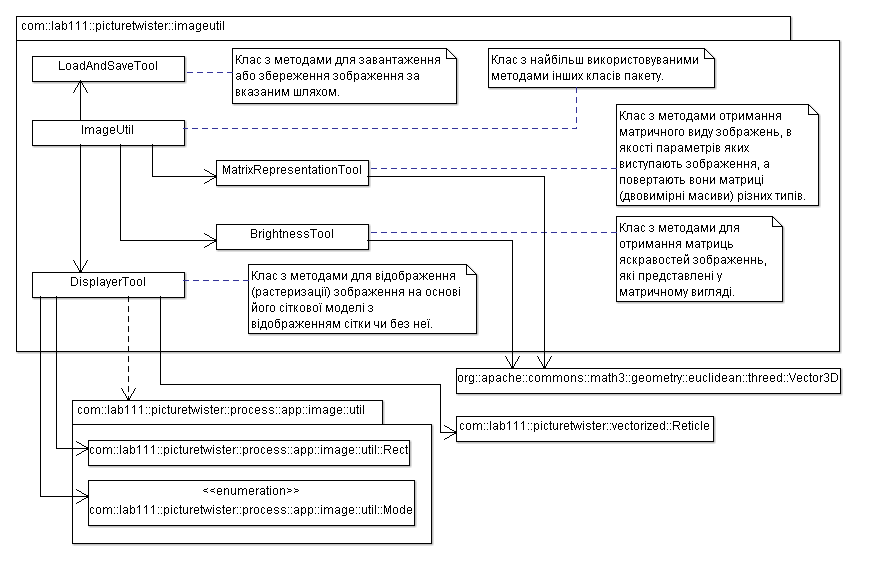


Рисунок 2.1.Структура пакету ImageUtil

Пакет ImageUtil містить наступні класи для роботи з растровими зображеннями:

* **LoadAndSaveTool** – містить методи для завантаження\збереження зображень.
* **MatrixRepresentationTool** – містить методи для отримання матричного представлення зображень.
* **BrightnessTool** – містить методи для отримання матриці яскравостей пікселів зображення на основі його матричного представлення**.**
* **DisplayerTool** – містить методи для растеризації зображення на основі його матричного представлення або сіткової моделі (Reticle).
* **ImageUtil** – містить ті методи вищевказаних класів, які використовуються найчастіше.

## Опис класів пакету ImageUtil

### Клас LoadAndSaveTool

Клас LoadAndSaveTool включає методи для завантаження та збереження зображень. І завантаження (зчитування) і збереження (запис) зображень здійснюється за вказаним шляхом.

### Клас MatrixRepresentationTool

Клас MatrixRepresentationTool містить методи для отримання матричного представлення зображень. Матричним представленням зображення є двовимірний масив чисел, кожне з яких є значенням кольору у RGB палітрі кольорів. Елемент arr[i][j] такого масива відповідає пікселю з координатами x = i, y = j зображення.

### Клас BrightnessTool

Клас BrightnessTool містить методи для отримання матриці яскравостей пікселів зображення на основі його матричного представлення. Елементом цієї матриці є числове значення яскравості пікселя.

Зазвичай значення яскравості пікселя обчислюється за формулою:

де Y – яскравість пікселя, а R, G та B – значення червоної, зеленої та синьої компонент кольору пікселя відповідно. У цій формулі використовується вектор (0.3, 0.59, 0.11) коефіцієнтів для RGB каналів кольору, хоча правильнішим є вектор (0.21, 0.72, 0.07). Проте найбільш коректним є наступне визначення яскравості пікселя:

де позначення аналогічні до попередньої формули.

### Клас DisplayerTool

Клас DisplayerToolмістить методи для растеризації зображення на основі його матричного представлення або сіткової моделі (Reticle). Його методи повертають растеризоване зображення (BufferedImage).

### Клас ImageUtil

Клас містить найчастіше використовувані методи інших класів пакету з параметрами за замовчуванням. Створений для швидкого виконання типових операцій.

## Пакет process.app.image.util

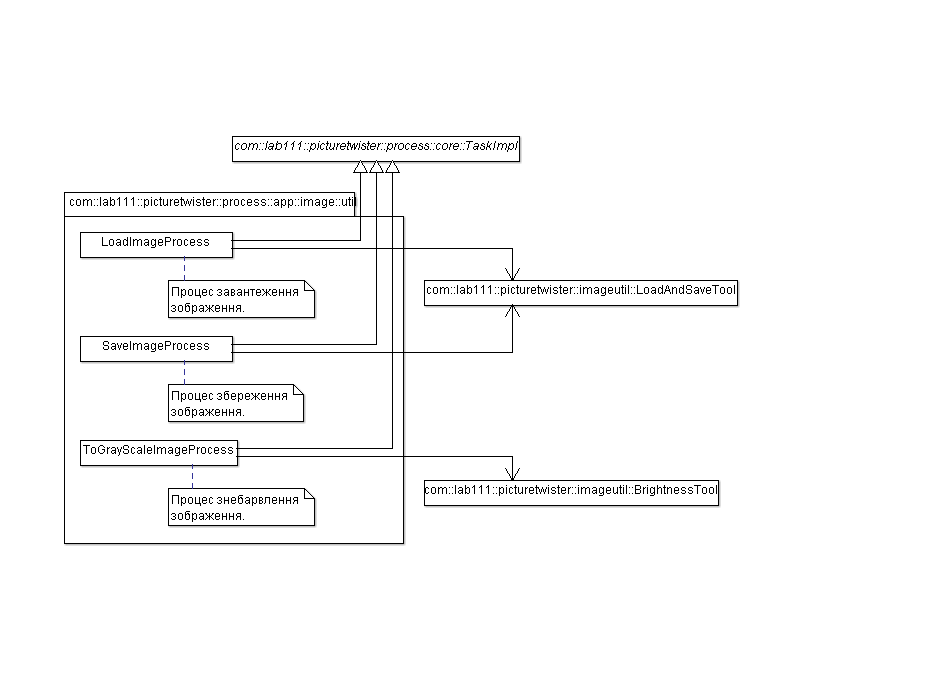


Рисунок 2.2 Структура пакету process.app.image.util

## Опис класів (процесів) пакету process.app.image.util

### Клас (процес) LoadImageProcess

Даний процес призначений для завантаження зображення за вказаною назвою файлу.

Вхід процесу:

* “src” – ім’я файлу, в якому міститься зображення.

Вихід процесу:

* “dest” – завантажене зображення.

### Клас (процес) SaveImageProcess

Даний процес призначений для збереження зображення за вказаним шляхом.

Входи процесу:

* “src” – зображення, яке зберігатиметься.
* “dest” - ім’я файлу, в який зберігається зображення.

Вихід процесу:

* Відсутній.

### Клас (процес) ToGrayScaleImageProcess

Даний процес призначений для знебарвлення зображення.

Вхід процесу:

* “src” – зображення, яке знебарвлюється.

Вихід процесу:

* “dest” – знебарвлене зображення.

## Пакет process.app.image.util

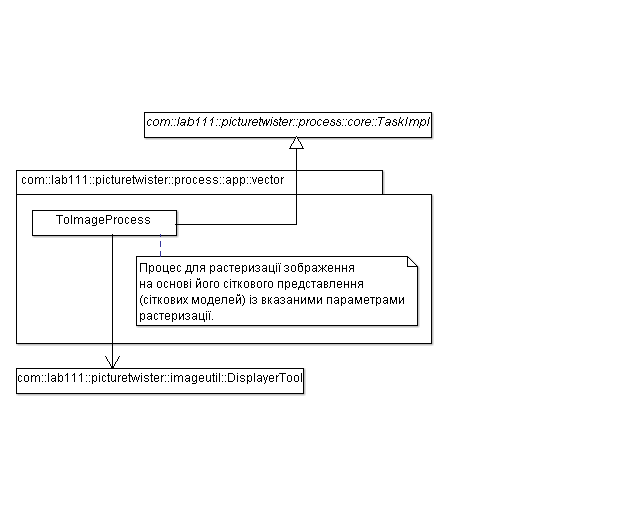


Рисунок 2.3 Структура пакету process.app.vector

## Опис класів (процесів) пакету process.app.vector

### Клас (процес) ToImageProcess

Даний процес призначений для растеризації колекції сіткових моделей що потрапляють у зону відображення у заданих режимах на заданому фоновому зображенні із вказаним збільшенням.

Сіткове представлення (модель) зображення – це масив зв’язаних точок, які утворюють полігони, з кожним з яких зв’язаний певний колір.

Вхід процесу:

* shapes – колекція сіткових моделей, які растеризуватимуться. Можуть бути як з одного, так і з кількох різних зображень.
* background – зображення, на фоні якого відбувається растеризація сіткових моделей. Якщо даний параметр дорівнює null, то растеризація відбувається у нове зображення, розміри якого дорівнюють розмірам viewport’а, збільшених у scale разів.
* penColor – колір, яким відображається сітка і вершини полігонів сіткового представлення (якщо їх відображення увімкнене параметром drawMode).
* bgColor – фоновий колір нового зображення, якщо параметр background дорівнює null.
* mode – режим растеризації, вмикає\вимикає відображення сітки (контурів полігонів), вузлів (вершин полігонів), самих полігонів (їхніх внутрішніх областей).
* viewport – прямокутний «трафарет», який задає позицію і розміри зони відображення. Растеризуюються усі полігони сіткового представлення, які потрапляють у цю зону. Якщо вказане фонове зображення (background), то розміри вихідного зображення будуть такими, як і у viewport’а; якщо параметр background дорівнює null, то розміри створеного зображення дорівнюватимуть розмірам розмірам viewport’а, збільшених у scale разів.
* scale – коефіцієнт збільшення (масштабу) нового зображення відносно розмірів viewport’а. Збільшення відбувається лише у тому випадку, коли background дорівнює null.

Вихід процесу:

* dest – вихідне (пастеризоване) зображення (колекція сіткових представлень).

# Розробка програмного забезпечення

## Специфікація класів пакету ImageUtil

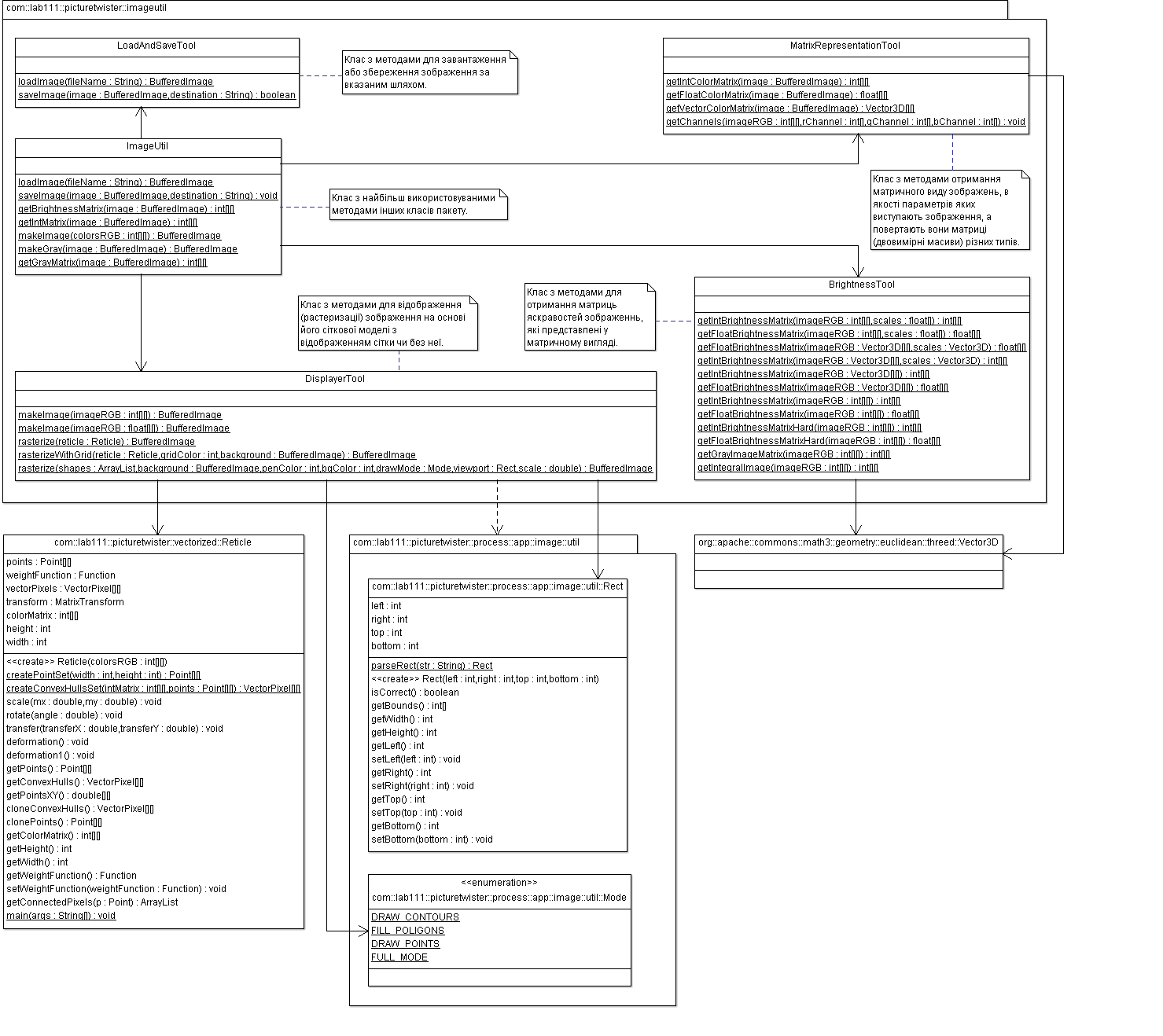


Рисунок 3.1.Діаграма класів пакету ImageUtil

### Клас LoadAndSaveTool

Клас містить наступні методи:

* loadImage(String) - метод для завантаження зображення за вказаним шляхом.
* saveImage(BufferedImage, String) - метод для збереження зображення за вказаним шляхом.

Приклади роботи з методами класу:

//Завантаження зображення

BufferedImage example = LoadAndSaveTool.loadImage("res\\images\\example.png");

//Робота з зображенням

//...

//Збереження зображення

LoadAndSaveTool.saveImage(example, "res\\images\\example \_modified.png");

### Клас MatrixRepresentationTool

Клас містить наступні методи:

* getIntColorMatrix(BufferedImage)– метод для отримання матриці з цілочисельними значеннями кольорів пікселів.
* getFloatColorMatrix(BufferedImage) – метод для отримання матриці зі значеннями кольорів пікселів, які представлені у вигляді чисел з плаваючою комою.
* getVectorColorMatrix(BufferedImage) – метод для отримання матриці зі значеннями кольорів пікселів, які представлені у вигляді тривимірних векторів Vector3D, значення x, y та z координат якого відповідають значенням Red, Green та Blue компонентам кольору пікселя.
* getChannels(int[][], int[], int[], int[]) – метод для отримання лінійних (одновимірних) масивів зі значеннями Red, Green та Blue компонент кольору пікселів зображення в окремих масивах.

Приклад роботи з методами класу:

//Отримання int-матриці зображення

**int**[][] exampleMatrix = MatrixRepresentationTool.*getIntColorMatrix*(example);

//Отримання значення червоної (Red) компоненти кольору піксела з координатами (x, y)

//Молодший байт кольору відповідає синій (Blue) компоненті; слідуючий за ним – зеленій //(Green); останній (старший з трьох) – червоній (Red).

**int** red = (exampleMatrix[x][y] & 0xff0000) >> 16;

//Отримання векторів значення кожного з кналів пікселів зображення

**int** height = exampleMatrix.length;

**int** width = exampleMatrix[0].length;

**int** total = height \* width;

**int**[] rChannel = **new** **int**[total];

**int**[] gChannel = **new** **int**[total];

**int**[] bChannel = **new** **int**[total];

//Отримуємо вектори значень кожного з каналів і сортуємо їх

MatrixRepresentationTool.*getChannels*(imageRGB, rChannel, gChannel, bChannel);

Arrays.*sort*(rChannel);

Arrays.*sort*(gChannel);

Arrays.*sort*(bChannel);

### Клас BrightnessTool

Клас містить наступні методи (перелічені не всі):

* getIntBrightnessMatrix(int[][], float[]) – метод для отримання матриці (двовимірного масиву) з цілочисельними значеннями яскравостей кольорів пікселів. Вектор коефіцієнтів для RGB каналів кольору задається користувачем.
* getFloatBrightnessMatrix(Vector3D[][], Vector3D) – аналогічно до попереднього методу, тільки елементами матриці яскравостей є числа з плаваючою комою. Кольори пікселів представлені у вигляді тривимірних векторів Vector3D; у відповідності до них задаються і коефіцієнти.
* getIntBrightnessMatrix(int[][]) – метод для отримання матриці яскравостей пікселів зображення за замовчуванням. (0.21, 0.72, 0.07) - вектор коефіцієнтів.
* getFloatBrightnessMatrix(int[][]) - аналогічно до попереднього методу, тільки елементами матриці яскравостей є числа з плаваючою комою.
* getIntBrightnessMatrixHard(int[][]) – метод для отримання матриці яскравостей пікселів за формулою (1). Отримані значення яскравостей будуть найточнішими, але потребують більше ресурсів і часу.
* getGrayImageMatrix(int[][]) – метод для отримання зображення у градаціях сірого.

Приклад роботи з методами класу:

//Отримання матриці яскравостей пікселів зображення на основі його матричного

//представлення

**int**[][] exampleBrightness = BrightnessTool.*getIntBrightnessMatrix*(exampleMatrix);

//Отримання значення яскравості піксела з координатами (x, y)

**int** Y = exampleBrightness[x][y];

### Клас DisplayerTool

Клас містить наступні методи:

* makeImage(int[][]) – растеризує зображення на основі його цілочисельного матричного представлення. Кожний піксель утвореного зображення з координатами (x, y), відповідає [x][y] елементу масива (матричного представлення).
* makeImage(float[][]) – аналогічний до попереднього. Єдина відмінність – пастеризація відбувається на основі двовимірного масива (матричного представлення), елементами якого є числа з плаваючою комою.
* rasterize(Reticle) – метод для растеризації зображення на основі його сіткової моделі (сіткового представлення).
* rasterizeWithGrid(Reticle, int, BufferedImage) – метод растеризує зображення з відображенням сітки (Reticle). Колір сітки, як і фонове зображення (поверх якого рисується) задається користувачем.
* rasterize(ArrayList<Reticle>, BufferedImage, int, int, Mode, Rect, double) - метод, що дозволяє растеризовувати колекцію сіткових моделей, що потрапляють у зону відображення (viewport) на заданому фоні у різних режимах і з певним збільшенням. Якщо background != null, то малювання відбувається у нього, інакше - у нове зображення.

*shapes* - колекція сіткових представлень.

*viewport* - зона відображення. Прямокутна рамка, яка відповідає області прорисовки. Якщо background != null, то прорисовується все, що потрапляє у цю область. Інакше створюється нове зображення, розміри якого дорівнюють розмірам viewport'а, збільшеним у scale разів; прорисовується у ньому все, що потрапляє з сіткового представлення у область прорисовки. Іншими словами, viewport - це трафарет. Растеризується все, що потрапляє у його відкриту область.

*drawMode* – задає режим растеризації (відображення сітки, її вузлів, полігонів).

Алгоритм роботи цього методу детальніше описаний у Додатку В.

Приклад роботи з методами класу:

//Отримання сіткового представлення зображення на основі матричного

Reticle ret = **new** Reticle(exampleMatrix);

//Додавання цієї сіткової моделі у колекцію

ArrayList<Reticle> rets = **new** ArrayList<>();

rets.add(ret);

//Растеризація даного стікового представлення

test = DisplayerTool.*rasterize*(rets, **null**, 0x00ff00, 0x000000, Mode.*FULL\_MODE*, **new** Rect(0, 400, 0, 400), 10.0);

## Згенерована javadoc документація

### Клас LoadAndSaveTool

## L

[**loadImage(String)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\LoadAndSaveTool.html#loadImage(java.lang.String)) - Static method in class com.lab111.picturetwister.imageutil.[LoadAndSaveTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\LoadAndSaveTool.html)

Метод для завантаження зображення за вказаним шляхом.

## S

[**saveImage(BufferedImage, String)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\LoadAndSaveTool.html#saveImage(java.awt.image.BufferedImage, java.lang.String)) - Static method in class com.lab111.picturetwister.imageutil.[LoadAndSaveTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\LoadAndSaveTool.html)

Метод для збереження зображення за вказаним шляхом.

### Клас Клас MatrixRepresentationTool

## G

[**getFloatColorMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html#getFloatColorMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[MatrixRepresentationTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html)

Метод для отримання float-матриці кольорів пікселів.

[**getIntColorMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html#getIntColorMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[MatrixRepresentationTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html)

Метод для отримання int-матриці кольорів пікселів.

[**getVectorColorMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html#getVectorColorMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[MatrixRepresentationTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\MatrixRepresentationTool.html)

Метод для отримання векторної (Vector3D) матриці кольорів пікселів.

### Клас BrightnessTool

## G

[**getFloatBrightnessMatrix(int[][], float[])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getFloatBrightnessMatrix(int[][], float[])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання float-матриці яскравостей пікселів зображення із вказаними коефіцієнтами, на які домножуються значення каналів кольрів пікселів.

[**getFloatBrightnessMatrix(Vector3D[][], Vector3D)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getFloatBrightnessMatrix(org.apache.commons.math3.geometry.euclidean.threed.Vector3D[][], org.apache.commons.math3.geometry.euclidean.threed.Vector3D)) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення із вказаними коефіцієнтами, на які домножуються канали кольрів пікселів.

[**getFloatBrightnessMatrix(Vector3D[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getFloatBrightnessMatrix(org.apache.commons.math3.geometry.euclidean.threed.Vector3D[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення із коефіцієнтами за замовчуванням.

[**getFloatBrightnessMatrix(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getFloatBrightnessMatrix(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення за замовчуванням.

[**getFloatBrightnessMatrixHard(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getFloatBrightnessMatrixHard(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення за наступним перетворенням: яскравість пікселя = 255 \* (r\*0.21 + g\*0.72 + b\*0.07) ^(1/2.2), де r = (R/255) ^ 2.2, g = (G/255) ^ 2.2, b = (B/255) ^ 2.2, R, G, B - відповідно червоний, зелений та синій канали кольру пікселя.

[**getGrayImageMatrix(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getGrayImageMatrix(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання зображення у сірих градаціях.

[**getIntBrightnessMatrix(int[][], float[])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntBrightnessMatrix(int[][], float[])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання int-матриці яскравостей пікселів зображення із вказаними коефіцієнтами, на які домножуються значення каналів кольрів пікселів.

[**getIntBrightnessMatrix(Vector3D[][], Vector3D)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntBrightnessMatrix(org.apache.commons.math3.geometry.euclidean.threed.Vector3D[][], org.apache.commons.math3.geometry.euclidean.threed.Vector3D)) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення із вказаними коефіцієнтами, на які домножуються значення каналів кольрів пікселів.

[**getIntBrightnessMatrix(Vector3D[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntBrightnessMatrix(org.apache.commons.math3.geometry.euclidean.threed.Vector3D[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення із коефіцієнтами за замовчуванням.

[**getIntBrightnessMatrix(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntBrightnessMatrix(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення за замовчуванням.

[**getIntBrightnessMatrixHard(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntBrightnessMatrixHard(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для отримання матриці яскравостей пікселів зображення за наступним перетворенням: яскравість піксела = 255 \* (r\*0.21 + g\*0.72 + b\*0.07) ^(1/2.2), де r = (R/255) ^ 2.2 , g = (G/255) ^ 2.2, b = (B/255) ^ 2.2, R, G, B - відповідно значення червоного, зеленого та синього каналів кольру пікселя.

[**getIntegralImage(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html#getIntegralImage(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[BrightnessTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\BrightnessTool.html)

Метод для знаходження інтегральної матриці зображення.

### Клас DisplayerTool

## M

[**makeImage(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html#makeImage(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[DisplayerTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html)

Метод створює зображення на основі int-матриці кольрів зображення.

[**makeImage(float[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html#makeImage(float[][])) - Static method in class com.lab111.picturetwister.imageutil.[DisplayerTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html)

Метод створює зображення на основі float-матриці кольрів зображення

## R

[**rasterize(Reticle)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html#rasterize(com.lab111.picturetwister.vectorized.Reticle)) - Static method in class com.lab111.picturetwister.imageutil.[DisplayerTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html)

Метод створює зображення на основі сіткової моделі (Reticle) зображення.

[**rasterize(ArrayList<Reticle>, BufferedImage, int, int, Mode, Rect, double)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html#rasterize(java.util.ArrayList, java.awt.image.BufferedImage, int, int, com.lab111.picturetwister.process.app.image.util.Mode, com.lab111.picturetwister.process.app.image.util.Rect, double)) - Static method in class com.lab111.picturetwister.imageutil.[DisplayerTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html)

Метод, що дозволяє растеризовувати колекцію сіткових моделей, що потрапляють у зону відображення (viewport) на заданому фоні у різних режимах і з певним збільшенням.

[**rasterizeWithGrid(Reticle, int, BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html#rasterizeWithGrid(com.lab111.picturetwister.vectorized.Reticle, int, java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[DisplayerTool](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\DisplayerTool.html)

Метод створює зображення на основі сіткової моделі (Reticle) зображення з відображенням сітки.

### Клас ImageUtil

## G

[**getBrightnessMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#getBrightnessMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для знаходження матриці яскравостей пікселів зображення.

[**getGrayMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#getGrayMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для отримання int-матриці зображення у градаціях сірого.

[**getIntMatrix(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#getIntMatrix(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для отримання int-матриці кольорів пікселів.

## L

[**loadImage(String)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#loadImage(java.lang.String)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для завантаження зображення за вказаним шляхом.

## M

[**makeGray(BufferedImage)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#makeGray(java.awt.image.BufferedImage)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для отримання зображення у сірих градаціях.

[**makeImage(int[][])**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#makeImage(int[][])) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод створює зображення на основі int-матриці кольрів зображення.

## S

[**saveImage(BufferedImage, String)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html#saveImage(java.awt.image.BufferedImage, java.lang.String)) - Static method in class com.lab111.picturetwister.imageutil.[ImageUtil](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\imageutil\ImageUtil.html)

Метод для збереження зображення за вказаним шляхом.

### Клас (процес) LoadImageProcess

## A

[**allowNestedTask(TaskImpl)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\LoadImageProcess.html#allowNestedTask(com.lab111.picturetwister.process.core.TaskImpl)) - Method in class com.lab111.picturetwister.process.app.image.[LoadImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\LoadImageProcess.html)

## E

[**execute()**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\LoadImageProcess.html#execute()) - Method in class com.lab111.picturetwister.process.app.image.[LoadImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\LoadImageProcess.html)

### Клас (процес) SaveImageProcess

## A

[**allowNestedTask(TaskImpl)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\SaveImageProcess.html#allowNestedTask(com.lab111.picturetwister.process.core.TaskImpl)) - Method in class com.lab111.picturetwister.process.app.image.[SaveImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\SaveImageProcess.html)

## E

[**execute()**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\SaveImageProcess.html#execute()) - Method in class com.lab111.picturetwister.process.app.image.[SaveImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\SaveImageProcess.html)

### Клас (процес) ToGrayScaleImageProcess

## A

[**allowNestedTask(TaskImpl)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\ToGrayScaleImageProcess.html#allowNestedTask(com.lab111.picturetwister.process.core.TaskImpl)) - Method in class com.lab111.picturetwister.process.app.image.[ToGrayScaleImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\ToGrayScaleImageProcess.html)

## E

[**execute()**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\ToGrayScaleImageProcess.html#execute()) - Method in class com.lab111.picturetwister.process.app.image.[ToGrayScaleImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\image\ToGrayScaleImageProcess.html)

### Клас (процес) ToImageProcess

## A

[**allowNestedTask(TaskImpl)**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\vector\ToImageProcess.html#allowNestedTask(com.lab111.picturetwister.process.core.TaskImpl)) - Method in class com.lab111.picturetwister.process.app.vector.[ToImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\vector\ToImageProcess.html)

## E

[**execute()**](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\vector\ToImageProcess.html#execute()) - Method in class com.lab111.picturetwister.process.app.vector.[ToImageProcess](file:///D:\JAVA_workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\process\app\vector\ToImageProcess.html)

ВИСНОВКИ

В ході виконання курсової роботи отримано навички роботи з графічним пакетом AWT. Створено бібліотеку утилітних класів для роботи з растровими зображеннями у кольоровій моделі RGB.

Повністю реалізована вказана у ТЗ функціональність; реалізовано пастеризацію сіткових представлень зображення; створено процеси для подальшого використання утилітних класів інтерпритатором.

Виконані вимоги до реалізації: проект повністю заду коментований за допомогою JavaDoc, використовуються класи і методи графічної бібліотеки AWT.

В результаті розробки всі пункти технічного завдання було виконано.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

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

**ДОДАТКИ**

# ДОДАТОК А. ПРОГРАМНИЙ КОД ПРОЕКТУ

## А.1 Клас LoadAndSaveTool

**package** com.lab111.picturetwister.imageutil;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**import** javax.imageio.ImageIO;

/\*\*

\* Клас з методами для завантеження|збереження зображення за вказаним шляхом.

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 1.0

\* **@since** 0.2

\*/

**public** **class** LoadAndSaveTool {

/\*\*

\* Метод для завантаження зображення за вказаним шляхом.

\* **@param** fileName шлях до зображення.

\* **@return** вказане зображення.

\* **@throws** FileNotFoundException, IOException

\*/

**public** **static** BufferedImage loadImage(String fileName)

**throws** IOException, FileNotFoundException

{

BufferedImage img = **null**;

img = ImageIO.*read*(**new** File(fileName));

**return** img;

}

/\*\*

\* Метод для збереження зображення за вказаним шляхом.

\* **@param** image зображення.

\* **@param** destination шлях (ім'я файлу) запису.

\* **@throws** IOException

\*/

**public** **static** **void** saveImage(BufferedImage image, String destination)

**throws** IOException {

ImageIO.*write*(image, destination.substring(destination.length() - 3), **new** File(destination));

}

}

## А.2 Клас Клас MatrixRepresentationTool

**package** com.lab111.picturetwister.imageutil;

**import** java.awt.image.BufferedImage;

**import** java.util.Arrays;

**import** org.apache.commons.math3.geometry.euclidean.threed.Vector3D;

/\*\*

\* Клас з методами отримання матричного виду зображень

\* (двовимірних масивів зі значеннями кольорів пікселів).

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 0.2

\*/

**public** **class** MatrixRepresentationTool {

/\*\*

\* Метод для отримання int-матриці кольорів пікселів.

\* **@param** image вихідне зображення.

\* **@return** матрицю з кольорами пікселів зображення.

\*/

**public** **static** **int**[][] getIntColorMatrix(BufferedImage image) {

**int** height = image.getHeight();

**int** width = image.getWidth();

**int**[][] result = **new** **int**[width][height];

**for** (**int** i = 0; i < width; i++)

**for** (**int** j = 0; j < height; j++)

result[i][j] = image.getRGB(i, j);

**return** result;

}

/\*\*

\* Метод для отримання float-матриці кольорів пікселів.

\* **@param** image вихідне зображення.

\* **@return** матрицю з кольорами пікселів зображення.

\*/

**public** **static** **float**[][] getFloatColorMatrix(BufferedImage image) {

**int** height = image.getHeight();

**int** width = image.getWidth();

**float**[][] result = **new** **float**[width][height];

**for**(**int** i = 0; i < width; i++)

**for** (**int** j = 0; j < height; j++)

result[i][j] = (**float**) image.getRGB(i, j);

**return** result;

}

/\*\*

\* Метод для отримання векторної (Vector3D) матриці кольорів пікселів.

\* **@param** image вихідне зображення.

\* **@return** матрицю з векторами, координати яких є кольорами пікселів зображення.

\* **@see** org.apache.commons.math3.geometry.euclidean.threed.Vector3D

\*/

**public** **static** Vector3D[][] getVectorColorMatrix(BufferedImage image) {

**int** height = image.getHeight();

**int** width = image.getWidth();

Vector3D[][] result = **new** Vector3D[width][height];

**double** r,g,b;

**for**(**int** i = 0; i < width; i++)

**for**(**int** j = 0; j < height; j++)

{

r = (image.getRGB(j, i) & 0xff0000)>>16;

g = (image.getRGB(j, i) & 0x00ff00)>>8;

b = (image.getRGB(j, i) & 0x0000ff);

result[i][j] = **new** Vector3D(r, g, b);

}

**return** result;

}

**public** **static** **void** getChannels(**int**[][] imageRGB,

**int**[] rChannel, **int**[] gChannel, **int**[] bChannel) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int** total = height \* width;

**for**(**int** i = 0; i < total; i++) {

rChannel[i] = (imageRGB[i % width][i / height] & 0xff0000) >> 16;

gChannel[i] = (imageRGB[i % width][i / height] & 0x00ff00) >> 8;

bChannel[i] = (imageRGB[i % width][i / height] & 0x0000ff);

}

}

}

## А.3 Клас BrightnessTool

**package** com.lab111.picturetwister.imageutil;

**import** org.apache.commons.math3.geometry.euclidean.threed.Vector3D;

/\*\*

\* Клас з методами отримання матриці яскравостей (двовимірних масивів зі значеннями

\* яскравостей кольорів пікселів) зображення, представленого у матричному вигляді.

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 1.5

\* **@since** 0.2

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool

\*/

**public** **class** BrightnessTool {

/\*\*

\* Метод для отримання int-матриці яскравостей пікселів зображення із вказаними коефіцієнтами,

\* на які домножуються значення каналів кольрів пікселів.

\* **@param** imageRGB масив із кольорами пікселів.

\* **@param** scales коефіцієнти, на які домножуються значення каналів кольорів пікселів.

\* Наприклад [0.21, 0.72, 0.07] або [0.3, 0.59, 0.11].

\* **@return** масив з яскравостями пікселів.

\*/

**public** **static** **int**[][] getIntBrightnessMatrix(**int**[][] imageRGB, **float**[] scales) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int**[][] result = **new** **int**[height][width];

**int** r,g,b;

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

r = (imageRGB[i][j] & 0xff0000) >> 16;

g = (imageRGB[i][j] & 0x00ff00) >> 8;

b = imageRGB[i][j] & 0x0000ff;

result[i][j] = Math.*round*(r \* scales[0] + g \* scales[1] + b \* scales[2]);

}

**return** result;

}

/\*\*

\* Метод для отримання float-матриці яскравостей пікселів зображення із вказаними коефіцієнтами,

\* на які домножуються значення каналів кольрів пікселів.

\* **@param** imageRGB масив із кольорами пікселів.

\* **@param** scales коефіцієнти, на які домножуються значення каналів кольорів пікселів.

\* Наприклад [0.21, 0.72, 0.07] або [0.3, 0.59, 0.11].

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **float**[][] getFloatBrightnessMatrix(**int**[][] imageRGB, **float**[] scales) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**float**[][] result = **new** **float**[height][width];

**int** r,g,b;

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

r = (imageRGB[i][j] & 0xff0000) >> 16;

g = (imageRGB[i][j] & 0x00ff00) >> 8;

b = imageRGB[i][j] & 0x0000ff;

result[i][j] = r \* scales[0] + g \* scales[1] + b \* scales[2];

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення із вказаними коефіцієнтами, на які домножуються канали кольрів пікселів.

\* **@param** imageRGB масив векторів, значення координат яких є кольорами пікселів.

\* **@param** scales вектор коефіцієнтів, на який скалярно домножуються вектори кольорів.

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **float**[][] getFloatBrightnessMatrix(Vector3D[][] imageRGB, Vector3D scales) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**float**[][] result = **new** **float**[height][width];

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

result[i][j] = (**float**) Math.*round*(imageRGB[i][j].dotProduct(scales));

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення із вказаними коефіцієнтами,

\* на які домножуються значення каналів кольрів пікселів.

\* **@param** imageRGB масив векторів, значення координат яких є кольорами пікселів.

\* **@param** scales вектор коефіцієнтів, на який скалярно домножуються вектори кольорів.

\* **@return** масив із яскравостями пікселів.

\* **@see** org.apache.commons.math3.geometry.euclidean.threed.Vector3D

\*/

**public** **static** **int**[][] getIntBrightnessMatrix(Vector3D[][] imageRGB, Vector3D scales) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int**[][] result = **new** **int**[height][width];

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

result[i][j] = (**int**) Math.*round*(imageRGB[i][j].dotProduct(scales));

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення із коефіцієнтами за замовчуванням.

\* Коефіцієнти - [0.21, 0.72, 0.07].

\* **@param** imageRGB масив векторів, значення координат яких є кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **int**[][] getIntBrightnessMatrix(Vector3D[][] imageRGB) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int**[][] result = **new** **int**[height][width];

Vector3D scales = **new** Vector3D(0.27, .072, 0.07);

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

result[i][j] = (**int**) Math.*round*(imageRGB[i][j].dotProduct(scales));

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення із коефіцієнтами за замовчуванням.

\* Коефіцієнти - [0.21, 0.72, 0.07].

\* **@param** imageRGB масив векторів, значення координат яких є кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **float**[][] getFloatBrightnessMatrix(Vector3D[][] imageRGB) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**float**[][] result = **new** **float**[height][width];

Vector3D scales = **new** Vector3D(0.27, .072, 0.07);

**for**(**int** i = 0; i < height; i++)

**for**(**int** j = 0; j < width; j++) {

result[i][j] = (**float**) Math.*round*(imageRGB[i][j].dotProduct(scales));

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення за замовчуванням.

\* Коефіцієнти - [0.21, 0.72, 0.07].

\* **@param** imageRGB масив із кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **int**[][] getIntBrightnessMatrix(**int**[][] imageRGB) {

**return** *getIntBrightnessMatrix*(imageRGB, **new** **float**[] {0.21f, 0.72f, 0.07f});

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення за замовчуванням.

\* Коефіцієнти - [0.21, 0.72, 0.07].

\* **@param** imageRGB масив із кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\*/

**public** **static** **float**[][] getFloatBrightnessMatrix(**int**[][] imageRGB) {

**return** *getFloatBrightnessMatrix*(imageRGB, **new** **float**[] {0.21f, 0.72f, 0.07f});

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення за наступним перетворенням:

\* яскравість піксела = 255 \* (r\*0.21 + g\*0.72 + b\*0.07) ^(1/2.2), де

\* r = (R/255) ^ 2.2 ,

\* g = (G/255) ^ 2.2,

\* b = (B/255) ^ 2.2,

\* R, G, B - відповідно значення червоного, зеленого та синього каналів кольру пікселя.

\* **@param** imageRGB масив із кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\* **@see** http://cgm.computergraphics.ru/node/2203

\*/

**public** **static** **int**[][] getIntBrightnessMatrixHard(**int**[][] imageRGB) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int**[][] result = **new** **int**[height][width];

**double** r, g, b;

**for** (**int** i = 0; i < height; i++)

**for** (**int** j = 0; j < width; j++) {

r = (imageRGB[i][j] & 0xff0000) >> 16;

g = (imageRGB[i][j] & 0x00ff00) >> 8;

b = imageRGB[i][j] & 0x0000ff;

r = Math.*pow*(r/255.0, 2.2);

g = Math.*pow*(g/255.0, 2.2);

b = Math.*pow*(b/255.0, 2.2);

result[i][j] = (**int**) Math.*round*(255 \* Math.*pow*(

0.21 \* r + 0.72 \* g + 0.07 \* b, 1/2.2));

}

**return** result;

}

/\*\*

\* Метод для отримання матриці яскравостей пікселів зображення за наступним перетворенням:

\* яскравість пікселя = 255 \* (r\*0.21 + g\*0.72 + b\*0.07) ^(1/2.2), де

\* r = (R/255) ^ 2.2,

\* g = (G/255) ^ 2.2,

\* b = (B/255) ^ 2.2,

\* R, G, B - відповідно червоний, зелений та синій канали кольру пікселя.

\* **@param** imageRGB масив із кольорами пікселів.

\* **@return** масив із яскравостями пікселів.

\* **@see** http://cgm.computergraphics.ru/node/2203

\*/

**public** **static** **float**[][] getFloatBrightnessMatrixHard(**int**[][] imageRGB) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**float**[][] result = **new** **float**[height][width];

**double** r, g, b;

**for** (**int** i = 0; i < height; i++)

**for** (**int** j = 0; j < width; j++) {

r = (imageRGB[i][j] & 0xff0000) >> 16;

g = (imageRGB[i][j] & 0x00ff00) >> 8;

b = imageRGB[i][j] & 0x0000ff;

r = Math.*pow*(r/255.0, 2.2);

g = Math.*pow*(g/255.0, 2.2);

b = Math.*pow*(b/255.0, 2.2);

result[i][j] = 255.0f \* ((**float**) Math.*pow*(0.21 \* r + 0.72 \* g + 0.07 \* b, 1/2.2));

}

**return** result;

}

/\*\*

\* Метод для отримання зображення у сірих градаціях.

\* **@param** imageRGB масив із кольорами пікселів.

\* **@return** масив із кольорами пікселів у сірих тонах.

\*/

**public** **static** **int**[][] getGrayImageMatrix(**int**[][] imageRGB) {

**int**[][] result = **new** **int**[imageRGB.length][imageRGB[0].length];

**for**(**int** i = 0; i < imageRGB.length; i++)

**for**(**int** j = 0; j < imageRGB[0].length; j++)

result[i][j] = imageRGB[i][j]<<16 | imageRGB[i][j]<<8 | imageRGB[i][j];

**return** result;

}

/\*\*

\* Метод для знаходження інтегральної матриці зображення.

\* **@param** image масив із кольорами пікселів.

\* **@return** інтегральну матрицю зображення.

\*/

**public** **static** **int**[][] getIntegralImage(**int**[][] imageRGB) {

**int** height = imageRGB.length;

**int** width = imageRGB[0].length;

**int**[][] result = **new** **int**[height][width];

**int**[][] bright = *getIntBrightnessMatrix*(imageRGB);

result[0][0] = bright[0][0];

**for**(**int** i = 1; i < height; i++)

result[i][0] = bright[i][0] + result[i-1][0];

**for**(**int** j = 1; j < width; j++)

result[0][j] = bright[0][j] + result[0][j-1];

**for**(**int** i = 1; i < height; i++)

**for**(**int** j = 1; j < width; j++)

result[i][j] = bright[i][j] + result[i-1][j] + result[i][j-1] - result[i-1][j-1];

**return** result;

}

}

## А.4 Клас DisplayerTool

**package** com.lab111.picturetwister.imageutil;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.image.BufferedImage;

**import** java.util.ArrayList;

**import** com.lab111.picturetwister.process.app.image.util.Mode;

**import** com.lab111.picturetwister.process.app.image.util.Rect;

**import** com.lab111.picturetwister.vectorized.Reticle;

/\*\*

\* Клас для растеризації зображення на основі

\* матриці кольорів або сіткової моделі Reticle.

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 2.2

\* **@since** 0.3

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool

\* **@see** com.lab111.picturetwister.vectorized.Reticle

\*/

**public** **class** DisplayerTool {

/\*\*

\* Метод створює зображення на основі int-матриці кольрів зображення.

\* **@param** imageRGB матриця кольорів зображення.

\* **@return** зображення.

\*/

**public** **static** BufferedImage makeImage(**int**[][] imageRGB) {

**int** width = imageRGB.length;

**int** height = imageRGB[0].length;

BufferedImage image = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);

//Рисування пікселів

**for**(**int** i = 0; i < width; i++)

**for**(**int** j = 0; j < height; j++)

image.setRGB(i, j, imageRGB[i][j]);

**return** image;

}

/\*\*

\* Метод створює зображення на основі float-матриці кольрів зображення

\* **@param** imageRGB матриця кольорів зображення

\* **@return** зображення

\*/

**public** **static** BufferedImage makeImage(**float**[][] imageRGB) {

**int** width = imageRGB.length;

**int** height = imageRGB[0].length;

BufferedImage image = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);

//Рисування пікселів

**for**(**int** i = 0; i < width; i++)

**for**(**int** j = 0; j < height; j++)

image.setRGB(i, j, (**int**) imageRGB[i][j]);

**return** image;

}

/\*\*

\* Метод створює зображення на основі сіткової моделі (Reticle) зображення.

\* **@param** reticle сіткова модель (Reticle).

\* **@return** зображення.

\*/

**public** **static** BufferedImage rasterize(Reticle reticle) {

**int** width = reticle.getHeight();

**int** height = reticle.getWidth();

BufferedImage result = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);

Graphics graphics = result.getGraphics();

**double**[][] points = reticle.getPointsXY();

width = points.length;

height = points[0].length;

**int**[] xPoints, yPoints;

**int**[][] colors = reticle.getColorMatrix();

//Рисування пікселів (полігонів)

**for**(**int** i = 0; i < width - 2; i += 2)

**for**(**int** j = 0; j < height - 1; j++) {

graphics.setColor(**new** Color(colors[i/2][j]));

xPoints = **new** **int**[] {(**int**)Math.*round*(points[i][j]), (**int**)Math.*round*(points[i][j+1]),

(**int**)Math.*round*(points[i+2][j+1]), (**int**)Math.*round*(points[i+2][j])

};

yPoints = **new** **int**[] {(**int**)Math.*round*(points[i+1][j]), (**int**)Math.*round*(points[i+1][j+1]), (**int**)Math.*round*(points[i+3][j+1]), (**int**)Math.*round*(points[i+3][j])

};

graphics.fillPolygon(xPoints, yPoints, 4);

}

**return** result;

}

/\*\*

\* Метод створює зображення на основі сіткової моделі (Reticle) зображення з відображенням сітки.

\* **@param** reticle сіткова модель (Reticle)

\* **@param** gridColor RGB колір сітки

\* **@param** background фонове зображення

\* **@return** зображення з сіткою пікселів

\* http://wm-help.net/books-online/book/12462/12462-66.html

\*/

**public** **static** BufferedImage rasterizeWithGrid(Reticle reticle, **int** gridColor, BufferedImage background) {

**int** width = reticle.getHeight();

**int** height = reticle.getWidth();

BufferedImage result;

Graphics2D graphics;

**if** (background == **null**) {

result = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);

graphics = result.createGraphics();

}

**else** {

result = background;

graphics = (Graphics2D) result.getGraphics();

}

**double**[][] points = reticle.getPointsXY();

width = points.length;

height = points[0].length;

**int**[] xPoints, yPoints;

**int**[][] colors = reticle.getColorMatrix();

//Рисування пікселів (полігонів)

**for**(**int** i = 0; i < width - 2; i += 2)

**for**(**int** j = 0; j < height - 1; j++) {

graphics.setColor(**new** Color(colors[i/2][j]));

xPoints = **new** **int**[] { (**int**)Math.*round*(points[i][j]), (**int**)Math.*round*(points[i][j+1]),

(**int**)Math.*round*(points[i+2][j+1]), (**int**)Math.*round*(points[i+2][j])

};

yPoints = **new** **int**[] { (**int**)Math.*round*(points[i+1][j]), (**int**)Math.*round*(points[i+1][j+1]),

(**int**)Math.*round*(points[i+3][j+1]), (**int**)Math.*round*(points[i+3][j])

};

graphics.fillPolygon(xPoints, yPoints, 4);

}

//Рисування сітки

graphics.setColor(**new** Color(gridColor));

**for**(**int** i = 0; i < width - 2; i += 2)

**for**(**int** j = 0; j < height - 1; j++) {

xPoints = **new** **int**[] { (**int**)Math.*round*(points[i][j]), (**int**)Math.*round*(points[i][j+1]),

(**int**)Math.*round*(points[i+2][j+1]), (**int**)Math.*round*(points[i+2][j])

};

yPoints = **new** **int**[] { (**int**)Math.*round*(points[i+1][j]), (**int**)Math.*round*(points[i+1][j+1]),

(**int**)Math.*round*(points[i+3][j+1]), (**int**)Math.*round*(points[i+3][j])

};

graphics.drawPolygon(xPoints, yPoints, 4);

}

**return** result;

}

/\*\*

\* Метод, що дозволяє растеризовувати колекцію сіткових моделей, що потрапляють у зону відображення (viewport)

\* на заданому фоні у різних режимах і з певним збільшенням. Якщо background != null, то малювання відбувається

\* у нього, інакше - у нове зображення.

\* **@param** shapes колекція сіткових представлень.

\* **@param** background фонове зображення, на які відрисовуються сіткові моделі.

\* Якщо null, то растеризація відбувається на пусте зображення.

\* **@param** penColor RGB колір сітки\вузлів.

\* **@param** bgColor RGB колір фону, на якому відбуватиметься растеризація, якщо background = null.

\* **@param** drawMode режим растеризації.

\* **@param** viewport зона відображення. Відображаються лише ті сіткові моделі, які потрапляють у неї.

\* Якщо background = null, то нове зображення матиме розміри viewport'а, збільшені у scale разів.

\* **@param** scale коефіцієнт збільшення сіткових моделей при растеризації у нове зображення

\* (збільшення відбувається тоді, коли background = null).

\* **@return** зображення растеризованих сіткових моделей.

\* **@see** com.lab111.picturetwister.process.app.image.util.Mode

\* **@see** com.lab111.picturetwister.process.app.image.util.Rect

\* **@since** 0.4

\*/

**public** **static** BufferedImage rasterize(ArrayList<Reticle> shapes, BufferedImage background, **int** penColor,

**int** bgColor, Mode drawMode, Rect viewport, **double** scale) {

// Початкова ініціалізація зображення - створення BufferedImage, задання його розмірів, тощо

**int** width, height;

**int** left = viewport.getLeft();

**int** right = viewport.getRight();

**int** top = viewport.getTop();

**int** bottom = viewport.getBottom();

BufferedImage result;

Graphics2D graphics;

**if** (background == **null**) {

//У випадку, якщо background відсутній, то малюємо у новому зображенні,

//розміри якого відповідають розмірам viewport, збільшеним у scale разів

width = (**int**) (viewport.getWidth()\*scale);

height = (**int**) (viewport.getHeight()\*scale);

result = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);

graphics = result.createGraphics();

graphics.setBackground(**new** Color(bgColor));

graphics.clearRect(0, 0, width, height);

}

**else** {

//У випадку, якщо параметр background відмінний від null, малюємо у зображення,

//"вирізане" з background'а, відповідно до координат (рамок) viewport

width = viewport.getWidth();

height = viewport.getHeight();

result = background.getSubimage(left, top, width, height);

graphics = (Graphics2D) result.getGraphics();

}

//Підготовка даних до рисування

Reticle shape;

**double**[][] points;

**int**[] xPoints, yPoints;

**double** x, y;

**int**[][] colors;

Color pen = **new** Color(penColor);

**boolean** toDraw = **false**;

//Рисування кожної сіточної моделі shape з колекції shapes

**for**(**int** k = 0; k < shapes.size(); k++) {

shape = shapes.get(k);

points = shape.getPointsXY();

width = points.length;

height = points[0].length;

//Рисування полігонів

**if**(drawMode == Mode.*FULL\_MODE* || drawMode == Mode.*FILL\_POLIGONS*) {

colors = shape.getColorMatrix();

//Рисування полігонів (обхід усіх точок)

**for**(**int** i = 0; i < width - 2; i += 2)

**for**(**int** j = 0; j < height - 1; j++) {

//Формування масивів з координатами чотирьох вузлів сітки

xPoints = **new** **int**[] { (**int**)(points[i][j]), (**int**)(points[i][j+1]),

(**int**)(points[i+2][j+1]), (**int**)(points[i+2][j])

};

yPoints = **new** **int**[] { (**int**)(points[i+1][j]), (**int**)(points[i+1][j+1]),

(**int**)(points[i+3][j+1]), (**int**)(points[i+3][j])

};

toDraw = **false**;

//Перевірка на попадання хача б однієї вершини полігона у область viewport'а

**for**(**int** p = 0; p < 4; p++)

toDraw |= (xPoints[p] > left && xPoints[p] < right &&

yPoints[p] > top && yPoints[p] < bottom);

**if**(toDraw) {

//Перетворення координат вузлів полігонів сіткових моделей у координати viewport'а

**for**(**int** p = 0; p < 4; p++) {

xPoints[p] = (**int**) ((background == **null**) ? (xPoints[p]-left)\*scale : xPoints[p]-left);

yPoints[p] = (**int**) ((background == **null**) ? (yPoints[p]-top)\*scale : yPoints[p]-top);

}

//Рисування

graphics.setColor(**new** Color(colors[i/2][j]));

graphics.fillPolygon(xPoints, yPoints, 4);

}

}

}

//Рисування решітки сіточної моделі

**if**(drawMode == Mode.*FULL\_MODE* || drawMode == Mode.*DRAW\_CONTOURS*) {

graphics.setColor(pen);

//Обхід усіх точок

**for**(**int** i = 0; i < width - 2; i += 2)

**for**(**int** j = 0; j < height - 1; j++) {

//Формування масивів з координатами чотирьох вузлів (полігона) сітки

xPoints = **new** **int**[] { (**int**)(points[i][j]), (**int**)(points[i][j+1]),

(**int**)(points[i+2][j+1]), (**int**)(points[i+2][j])

};

yPoints = **new** **int**[] { (**int**)(points[i+1][j]), (**int**)(points[i+1][j+1]),

(**int**)(points[i+3][j+1]),(**int**)(points[i+3][j])

};

toDraw = **false**;

//Перевірка на попадання хача б однієї вершини полігона у область viewport'а

**for**(**int** p = 0; p < 4; p++)

toDraw |= (xPoints[p] > left && xPoints[p] < right &&

yPoints[p] > top && yPoints[p] < bottom);

**if**(toDraw) {

//Перетворення координат вузлів полігонів сіткових моделей у координати viewport'а

**for**(**int** p = 0; p < 4; p++) {

xPoints[p] = (**int**) ((background == **null**) ? (xPoints[p]-left)\*scale : xPoints[p]-left);

yPoints[p] = (**int**) ((background == **null**) ? (yPoints[p]-top)\*scale : yPoints[p]-top);

}

//Рисування

graphics.drawPolygon(xPoints, yPoints, 4);

}

}

}

//Рисування вузлів сіточної моделі

**if**(drawMode == Mode.*DRAW\_POINTS*) {

graphics.setColor(pen);

//Обхід усіх точок

**for**(**int** i = 0; i < width - 1; i ++)

**for**(**int** j = 0; j < height; j++) {

//Формування координат вузла сітки

x = points[i][j];

y = points[i+1][j];

//Перевірка на попадання хача б однієї вершини полігона у область viewport'а

toDraw = (x >= left && x < right && y >= top && y < bottom);

**if**(toDraw) {

x = (background == **null**) ? (x - left)\*scale : x - left;

y = (background == **null**) ? (y - top)\*scale : y - top;

result.setRGB((**int**) Math.*round*(x), (**int**) Math.*round*(y), penColor);

}

}

}

}

**return** result;

}

}

## А.5 Клас ImageUtil

**package** com.lab111.picturetwister.imageutil;

**import** java.awt.image.BufferedImage;

**import** java.io.IOException;

/\*\*

\* Клас з найбільш використовуваними методами з параметрами за замовчуванням.

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 2.0

\* **@since** 0.1

\* **@see** com.lab111.picturetwister.imageutil

\* http://articles.org.ru/docum/java/gl15/gl15.php

\*/

**public** **class** ImageUtil {

/\*\*

\* Метод для завантаження зображення за вказаним шляхом.

\* **@param** fileName шлях до зображення.

\* **@return** вказане зображення.

\* **@throws** IOException, FileNotFoundException

\* **@see** com.lab111.picturetwister.imageutil.LoadAndSaveTool#loadImage(String)

\*/

**public** **static** BufferedImage loadImage(String fileName)

**throws** IOException {

**return** LoadAndSaveTool.*loadImage*(fileName);

}

/\*\*

\* Метод для збереження зображення за вказаним шляхом.

\* **@param** image зображення.

\* **@param** destination шлях (ім'я файлу) запису.

\* **@throws** IOException

\* **@see** com.lab111.picturetwister.imageutil.LoadAndSaveTool#saveImage(BufferedImage, String)

\*/

**public** **static** **void** saveImage(BufferedImage image, String destination)

**throws** IOException {

LoadAndSaveTool.*saveImage*(image, destination);

}

/\*\*

\* Метод для знаходження матриці яскравостей пікселів зображення.

\* **@param** image зображення.

\* **@return** матрицю яскравості пікселів зображення.

\* **@see** com.lab111.picturetwister.imageutil.BrightnessTool#getIntBrightnessMatrix(int[][], float[])

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool#getIntColorMatrix(BufferedImage)

\*/

**public** **static** **int**[][] getBrightnessMatrix(BufferedImage image) {

**return** BrightnessTool.*getIntBrightnessMatrix*(MatrixRepresentationTool.*getIntColorMatrix*(image));

}

/\*\*

\* Метод для отримання int-матриці кольорів пікселів.

\* **@param** image вихідне зображення.

\* **@return** матриця, елементами якої є RGB значення відповідних кольорів.

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool#getIntColorMatrix(BufferedImage)

\*/

**public** **static** **int**[][] getIntMatrix(BufferedImage image) {

**return** MatrixRepresentationTool.*getIntColorMatrix*(image);

}

/\*\*

\* Метод створює зображення на основі int-матриці кольрів зображення.

\* **@param** imageRGB матриця кольорів зображення.

\* **@return** зображення.

\* **@see** com.lab111.picturetwister.imageutil.DisplayerTool#makeImage(int[][])

\*/

**public** **static** BufferedImage makeImage(**int**[][] colorsRGB) {

**return** DisplayerTool.*makeImage*(colorsRGB);

}

/\*\*

\* Метод для отримання зображення у сірих градаціях.

\* **@param** image вихідне зображення.

\* **@return** зображення у градаціях сірого.

\* **@see** com.lab111.picturetwister.imageutil.DisplayerTool#makeImage(int[][])

\* **@see** com.lab111.picturetwister.imageutil.BrightnessTool#getGrayImageMatrix(int[][])

\* **@see** com.lab111.picturetwister.imageutil.BrightnessTool#getIntBrightnessMatrix(int[][])

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool#getIntColorMatrix(BufferedImage)

\*/

**public** **static** BufferedImage makeGray(BufferedImage image) {

**return** DisplayerTool.*makeImage*(

BrightnessTool.*getGrayImageMatrix*(

BrightnessTool.*getIntBrightnessMatrix*(

MatrixRepresentationTool.*getIntColorMatrix*(image))));

}

/\*\*

\* Метод для отримання int-матриці зображення у градаціях сірого.

\* **@param** image вихідне зображення.

\* **@return** int-матрицю кольорів перетвореного (у градаціях сірого) зображення.

\* **@see** com.lab111.picturetwister.imageutil.DisplayerTool#makeImage(int[][])

\* **@see** com.lab111.picturetwister.imageutil.BrightnessTool#getIntBrightnessMatrix(int[][])

\* **@see** com.lab111.picturetwister.imageutil.MatrixRepresentationTool#getIntColorMatrix(BufferedImage)

\*/

**public** **static** **int**[][] getGrayMatrix(BufferedImage image) {

**return** BrightnessTool.*getGrayImageMatrix*(

BrightnessTool.*getIntBrightnessMatrix*(

MatrixRepresentationTool.*getIntColorMatrix*(image)));

}

}

## А.6 Клас (процес) LoadImageProcess

**package** com.lab111.picturetwister.process.app.image;

**import** java.awt.image.BufferedImage;

**import** java.io.IOException;

**import** com.lab111.picturetwister.exceptions.TaskRuntimeException;

**import** com.lab111.picturetwister.imageutil.ImageUtil;

**import** com.lab111.picturetwister.interpreter.ContextProvider;

**import** com.lab111.picturetwister.process.core.TaskImpl;

**import** com.lab111.picturetwister.process.core.annotation.Description;

**import** com.lab111.picturetwister.process.core.annotation.InputsList;

**import** com.lab111.picturetwister.process.core.annotation.OutputsList;

**import** com.lab111.picturetwister.process.core.annotation.Port;

@Description ("Load image from source file")

@InputsList (

{

//Ім'я файлу, у якому міститься зображення.

@Port ( description = "image filename",

name = "src",

type = String.**class**

)

})

@OutputsList (

{

//Завантажене зображення.

@Port ( description = "image object",

name = "dest",

type = BufferedImage.**class**

)

})

**public** **class** LoadImageProcess **extends** TaskImpl {

**protected** LoadImageProcess(ContextProvider contextProvider) {

**super**(contextProvider);

}

@Override

**public** **void** execute() **throws** TaskRuntimeException {

String imageFile = (String) contextProvider.getInput("src");

**try** {

contextProvider.setOutput("dest", ImageUtil.*loadImage*(imageFile));

} **catch** (IOException e) {

**throw** **new** TaskRuntimeException("Failed to load image",

**this**.getClass(), e);

}

}

@Override

**public** **boolean** allowNestedTask(TaskImpl task) {

**return** **false**;

}

@Override

**public** **boolean** isDataValid() {

**return** **true**;

}

}

## А.7 Клас (процес) SaveImageProcess

**package** com.lab111.picturetwister.process.app.image;

**import** java.awt.image.BufferedImage;

**import** java.io.IOException;

**import** com.lab111.picturetwister.exceptions.TaskRuntimeException;

**import** com.lab111.picturetwister.imageutil.ImageUtil;

**import** com.lab111.picturetwister.interpreter.ContextProvider;

**import** com.lab111.picturetwister.process.core.TaskImpl;

**import** com.lab111.picturetwister.process.core.annotation.Description;

**import** com.lab111.picturetwister.process.core.annotation.Port;

**import** com.lab111.picturetwister.process.core.annotation.InputsList;

@Description ("Save image to file")

@InputsList({

//Зображення, яке зберігатиметься.

@Port ( description = "image object",

optional = **false**,

name = "src",

type = BufferedImage.**class**

),

//Ім'я файлу, за яким зберігатиметься зображення.

@Port ( description = "image filename",

optional = **false**,

name = "dest",

type = String.**class**

)

})

**public** **class** SaveImageProcess **extends** TaskImpl {

**protected** SaveImageProcess(ContextProvider contextProvider) {

**super**(contextProvider);

}

@Override

**public** **void** execute() **throws** TaskRuntimeException {

BufferedImage imageSource = (BufferedImage) contextProvider.getInput("src");

String imageDest = (String) contextProvider.getInput("dest");

**try** {

ImageUtil.*saveImage*(imageSource, imageDest);

} **catch** (IOException e) {

**throw** **new** TaskRuntimeException("Failed to save image",

**this**.getClass(), e);

}

}

@Override

**public** **boolean** allowNestedTask(TaskImpl task) {

**return** **true**;

}

@Override

**public** **boolean** isDataValid() {

**return** **true**;

}

}

## А.8 Клас (процес) ToGrayScaleImageProcess

**package** com.lab111.picturetwister.process.app.image;

**import** java.awt.image.BufferedImage;

**import** com.lab111.picturetwister.exceptions.TaskRuntimeException;

**import** com.lab111.picturetwister.imageutil.ImageUtil;

**import** com.lab111.picturetwister.interpreter.ContextProvider;

**import** com.lab111.picturetwister.process.core.TaskImpl;

**import** com.lab111.picturetwister.process.core.annotation.Description;

**import** com.lab111.picturetwister.process.core.annotation.InputsList;

**import** com.lab111.picturetwister.process.core.annotation.OutputsList;

**import** com.lab111.picturetwister.process.core.annotation.Port;

/\*\*

\* Процес для отримання зображення у градаціях сірого.

\* Вхідні дані: "src" - зображення (BufferedImage);

\* Вихідні дані: "dest" - зображення у градація сірого (BufferedImage).

\* **@author** Нестерук Юрій (yuriy.nesteruk@gmail.com)

\* **@version** 1.0

\* **@since** 2.5

\* **@see** com.lab111.picturetwister.imageutil.ImageUtil#makeGray(BufferedImage)

\*/

@Description("Convert color image to grayscale")

@InputsList (

{

//Зображення, яке знебервлюється

@Port( description = "color image source",

optional = **false**,

name = "src",

type = BufferedImage.**class**

)

})

@OutputsList (

{

//Результуюче зображення

@Port ( description = "grayscale image destination",

name = "dest",

type = BufferedImage.**class**

)

})

**public** **class** ToGrayScaleImageProcess **extends** TaskImpl {

**protected** ToGrayScaleImageProcess(ContextProvider contextProvider) {

**super**(contextProvider);

}

@Override

**public** **void** execute() **throws** TaskRuntimeException {

BufferedImage image = (BufferedImage) contextProvider.getInput("src");

image = ImageUtil.*makeGray*(image);

contextProvider.setOutput("dest", image);

}

@Override

**public** **boolean** allowNestedTask(TaskImpl task) {

**return** **true**;

}

@Override

**public** **boolean** isDataValid() {

**return** **true**;

}

}

## А.9 Клас (процес) ToImageProcess

**package** com.lab111.picturetwister.process.app.vector;

**import** java.awt.image.BufferedImage;

**import** java.util.ArrayList;

**import** com.lab111.picturetwister.exceptions.TaskRuntimeException;

**import** com.lab111.picturetwister.imageutil.DisplayerTool;

**import** com.lab111.picturetwister.interpreter.ContextProvider;

**import** com.lab111.picturetwister.process.app.image.util.Mode;

**import** com.lab111.picturetwister.process.app.image.util.Rect;

**import** com.lab111.picturetwister.process.core.TaskImpl;

**import** com.lab111.picturetwister.process.core.annotation.Description;

**import** com.lab111.picturetwister.process.core.annotation.InputsList;

**import** com.lab111.picturetwister.process.core.annotation.OutputsList;

**import** com.lab111.picturetwister.process.core.annotation.Port;

**import** com.lab111.picturetwister.vectorized.ConvexHull;

**import** com.lab111.picturetwister.vectorized.Reticle;

@Description ("Rasterize vector model")

@InputsList ({ // optional vs allowNull ?

//Колекція сіткових моделей, які будуть растеризуватися

@Port (description = "Vector Model",

name = "shapes",

type = {ConvexHull.**class**, Reticle.**class**},

collection = ArrayList.**class**),

//Зображення, на фоні якого відбувається растеризація (малюються сіткові моделі).

//Якщо null, то малювання відбувається у нове зображення,

//розміри і позиція на сітковому представленні якого відповідає viewport'у

@Port (description = "Background Image",

name = "background",

allowNull = **true**,

type = BufferedImage.**class**),

//Колір, яким буде зображатися сітка або вузли (в залежності від режиму)

//сіткового представлення

@Port (description = "Pen Color",

name = "penColor",

allowNull = **true**,

type = Integer.**class**),

//Колір фону, на якому відбувається растеризація, якщо background = null.

@Port (description = "Background Color",

name = "bgColor",

allowNull = **true**,

type = Integer.**class**),

//Режим растеризації (повний, тільки полігони, тільки сітка, тільки вузли)

@Port (description = "Draw Mode",

name = "mode",

type = Mode.**class**),

//Прямокутна рамка, яка відповідає області прорисовки. Якщо background != null,

//то прорисовується все, що потрапляє у цю область. Інакше створюється нове

//зображення, розміри якого дорівнюють розмірам viewport'а, збільшеним у

//scale разів; прорисовується у ньому все, що потрапляє з сіткового представлення

//у область прорисовки.

//Іншими словами, viewport - це трафарет. Растеризується все, що потрапляє у його область.

@Port (description = "View Port",

name = "viewport",

type = Rect.**class**),

//Коефіцієн збільшення (масштабу) області viewport'а, якщо background = null.

@Port (description = "Scale",

name = "scale",

allowNull = **true**,

type = Double.**class**)

})

@OutputsList ({

//Вихідне зображення.

@Port (description = "Destination image",

name = "dest",

type = BufferedImage.**class**)

})

**public** **class** ToImageProcess **extends** TaskImpl {

**protected** ToImageProcess(ContextProvider contextProvider) {

**super**(contextProvider);

}

@Override

**public** **void** execute() **throws** TaskRuntimeException {

ArrayList<Reticle> shapes = (ArrayList<Reticle>) contextProvider.getInput("shapes");

BufferedImage background = (BufferedImage) contextProvider.getInput("background");

**int** penColor = (contextProvider.getInput("penColor") != **null**) ?

(**int**) contextProvider.getInput("penColor") : 0xff0000;

**int** bgColor = (contextProvider.getInput("bgColor") != **null**) ?

(**int**) contextProvider.getInput("bgColor") : 0x000000;

Mode drawMode = (Mode) contextProvider.getInput("mode");

Rect viewport = (Rect) contextProvider.getInput("viewport");

**double** scale = (contextProvider.getInput("scale") != **null**) ?

(**double**) contextProvider.getInput("scale") : 1.0;

BufferedImage result = DisplayerTool.*rasterize*(shapes, background, penColor, bgColor, drawMode, viewport, scale);

contextProvider.setOutput("dest", result);

}

@Override

**public** **boolean** allowNestedTask(TaskImpl task) {

// **TODO** Auto-generated method stub

**return** **false**;

}

@Override

**public** **boolean** isDataValid() {

**if**(contextProvider.getInput("background") == **null** &&

contextProvider.getInput("bgColor") == **null**) {

errorMessage = "background і bgColor не можуть одночасно бути null.";

**return** **false**;

}

**if**(contextProvider.getInput("penColor") == **null** &&

((Mode) contextProvider.getInput("mode") != Mode.*FILL\_POLIGONS*)) {

errorMessage = "В даному режимі растеризації необхідно взакати колір сітки (вузлів).";

**return** **false**;

}

**if**(!( (Rect)contextProvider.getInput("viewport") ).isCorrect()) {

errorMessage = "Границі (рамки) viewport'а вказані некоректно.";

**return** **false**;

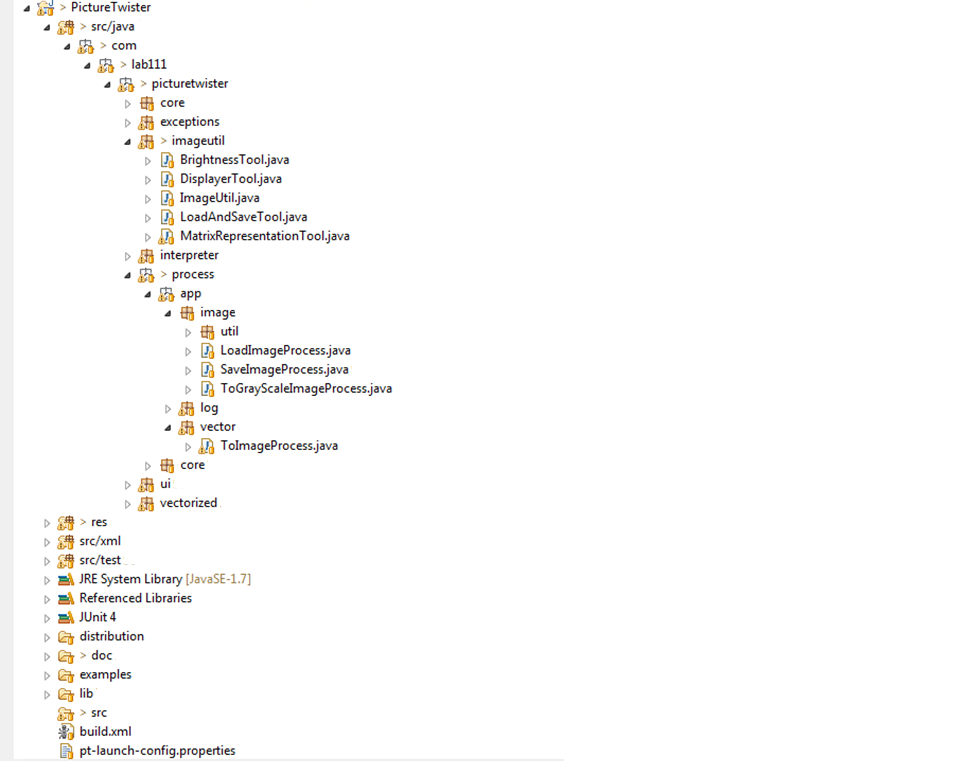
}

**return** **true**;

}

}

ДОДАТОК Б. СТРУКТУРА ПРОЕКТУ



ДОДАТОК В. Алгоритм роботи растеризатора

