Зміст

[Вступ 3](#_Toc354040048)

[1 Технічне завдання 4](#_Toc354040049)

[1.1 Загальне завдання 4](#_Toc354040050)

[1.2 Вимоги до функціональності 4](#_Toc354040051)

[1.3 Вимоги до реалізації 4](#_Toc354040052)

[2 Проектування програмного додатку 5](#_Toc354040053)

[2.1 Векторна графіка 5](#_Toc354040054)

[2.2 Побудова векторної моделі 5](#_Toc354040055)

[2.3 UML діаграма класів 6](#_Toc354040056)

[2.4 Проектування алгоритмів 6](#_Toc354040057)

[3 Розробка редактора векторних зображень 9](#_Toc354040058)

[3.2 Документація проекту 9](#_Toc354040059)

[4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24](#_Toc354040101)

[ВИСНОВКИ 25](#_Toc354040102)

[СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ 26](#_Toc354040103)

[ДОДАТОК А. ПРОГРАМНИЙ КОД ПРОЕКТУ 26](#_Toc354040104)

[ДОДАТОК Б. СТРУКТУРА ПРОЕКТУ 44](#_Toc354040105)

Вступ

У курсовій роботі потрібно розробити програмний додаток для представлення растрової моделі зображення векторною моделлю.

У роботі розглядаються основні принципи побудови графічного інтерфейсу користувача, основні елементи інтерфейсу, організація обробки подій, система відлову помилок і організація багатопоточності. Також

Розробка модулю виконується на мові програмування Java. Робота містить повну документацію, а також програмний код проекту та UML діаграму класів.

# Технічне завдання

## Загальне завдання

Необхідно розробити програмний додаток для побудови векторної моделі зображення та її редагування.

## Вимоги до функціональності

Програма має містити наступну функціональність:

* можливість отримати на вхід растрове зображення і представити його векторною моделлю;
* згладжування контурів зображення;
* пошук контурних ланцюгів;
* можливість збереження результату у вигляді файлу формату JPEG або PNG.

## Вимоги до реалізації

* мова програмування Java;
* інтерфейс користувача має забезпечувати доступ до всієї функціональності програми;
* проект має бути повністю задокументований за допомогою JavaDoc;
* проект має повністю відповідати правилам CheckStyle;

# Проектування програмного додатку

## 2.1 Векторна графіка

Растрова графіка подає зображення як набір пікселів. Піксель є неподільним об’єктом квадратної форми що має певний колір. Векторна ж графіка подає зображення, яке складається з сукупності таких геометричних примітивів, як точки, лінії, полігони.

Перевага векторної графіки перед растрової в тому, що векторне зображення легко модифікується, масштабується і зсувається. Це досягається простим перерахунком координат складових його точок. При цьому зрушувати і масштабувати можна окремі честі векторного зображення, жодним чином не зачіпаючи інші його частини.   
 При цьому якість векторного зображення не залежить від масштабу, у той час як зміна масштабу растрового зображення завжди пов'язане з втратою якості.

## 2.2 Побудова векторної моделі

Задача векторизації полягає у представлення пікселів графічними примітивами. Для реалізації цієї задачі було створено клас Point.java який описує один кут пікселя і місить вказівники на сусідні пікселі зліва, зверху, справа, знизу. Таким чином можна побудувати з таких точок геометричний примітив – полігон. Цю задачу реалізує клас ConvexHull.java який містить список точок, а також може містити різну атрибутику для полігону(наприклад площу, периметр, випуклість і т.д.). Щоб описати один піксель векторною моделлю використовується клас VectorPixel.java. Цей клас є предком класа ConvexHull.java і додатково містить інформацію про колір пікселя. Отже ми маємо модель растрового пікселя у векторній формі, який являє собою сукупність чотирьох точок Point і кольору. Маючи матрицю кольорів ми можемо отримати все зображення у векторній формі яке складається з цих самих векторних пікселів. Це реалізовано в класі Reticle.java.

## 2.3 UML діаграма класів

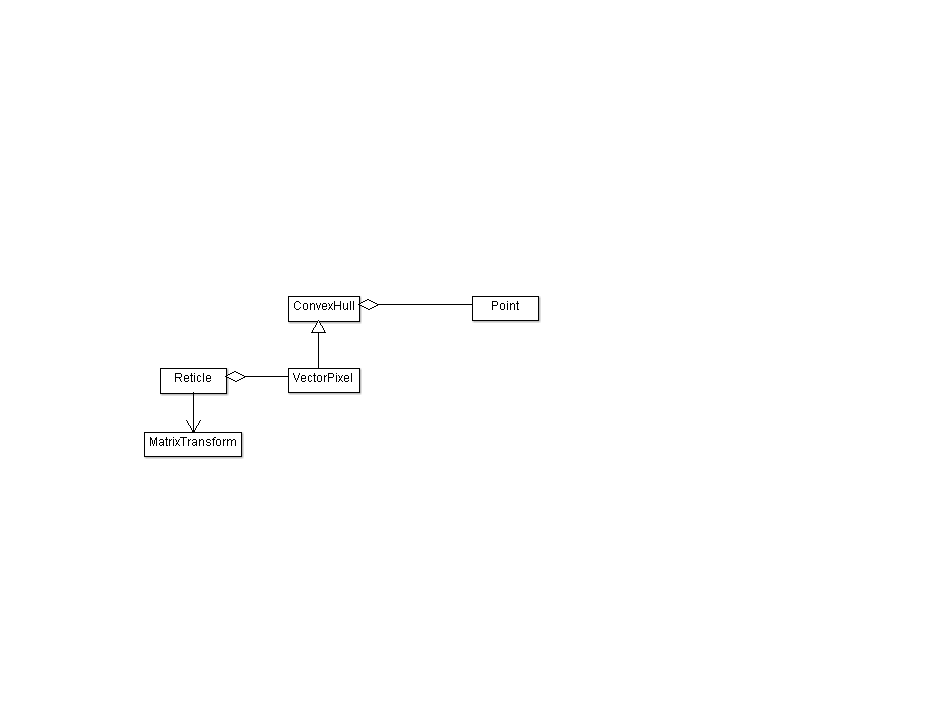


Рисунок 2.1 UML діаграма класів.

## 2.4 Проектування алгоритмів

Для згладжування контурів будемо деформувати решітку таким чином щоб змістити точку в центр мас системи.

Щоб знайти центра мас полігона ми будемо знаходити центр мас для системи трикутників, попередньо отримавши ці трикутники, розбивши наш полігон за допомогою тріангуляції.

Тріангуляція – це розбиття полігона на трикутники. Вона використовується для зведення розв’язання певних задач в області, що має складну конфігурацію, до сукупності задач, які розв’язуються в області трикутника – простої геометричної фігури. Для того, щоб виконати тріангуляцію довільного полігона використовується наступний алгоритм.  
 Опис алгоритму тріангуляції.

1. Проходимо контур по всім точкам і вибираємо трикутники із трьох

сусідніх точок. Нехай це будуть точки Ai-1, A i, Ai+1, тоді точки

Ai-1, Ai+1 з’єднуємо хордою. Таким чином можна отримати N різних

трикутників для контуру що складається з N точок.

1. Для кожного трикутника перевіряємо, чи є ця частина контуру опуклою.

Приклад опуклої частини контуру.



Приклад не опуклої частини контуру.



2)

Якщо хорда Ai-1Ai+1 проходить не всередині контуру то такий трикутник не проходить перевірку і більше не розглядається.

1. З усіх трикутників отриманих при проході вибираємо той, у якого відношення площі до периметру є найбільшим (це дозволяє вибрати трикутник найбільш наближений до рівностороннього). Заносимо цей трикутник до списку.

4) Видаляємо з контуру A i - точку трикутника, який ми вибрали як "найкращий".

5) Повторюємо попередні пункти до тих пір поки в контурі не залишиться три точки.

6) З'єднавши ці три точки отримуємо останній трикутник і заносимо його до списку.



Рисунок 2.1 Приклад роботи алгоритму тріангуляції.

Тепер маючи функціонал для розбиття полігону(в нашому випадку пікселя) ми можемо знайти центр мас системи.

# Розробка редактора векторних зображень

## 3.2 Документація проекту

* Package com.lab111.picturetwister.vectorized

|  |  |
| --- | --- |
| Class Summary | |
| **Class** | **Description** |
| [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) | Клас описує "оболочку", яка складається з точок Point і може бути довільної форми. |
| [MatrixTransform](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html) | Клас що описує матрицю перетоврень |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | Клас описує точку - як кут пікселя. |
| [Reticle](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html) | Клас описує сітку точок, зв'язаних між собою |
| [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html) | Клас, який описує векторний піксель, наслідується від класа ConvexHull |

com.lab111.picturetwister.vectorized

## Class Point

* java.lang.Object
  + com.lab111.picturetwister.vectorized.Point

public class **Point**

extends java.lang.Object

Клас описує точку - як кут пікселя.

**Author:**

Бедь Анатолій Михайлович

### Field Summary

|  |  |
| --- | --- |
| Fields | |
| **Modifier and Type** | **Field and Description** |
| private java.util.HashMap<java.lang.String,[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> | [**neighboringPoints**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#neighboringPoints)  Мапа, яка зберігає "сусідів" даної точки, у вигляді ключ - точка, ключі: left - точка зліва right - точка српава top - точка зверху bottom - точка знизу |
| private double | [**x**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#x)  Координата Х на сітці |
| private double | [**y**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#y)  Координата y на сітці |

### Constructor Summary

|  |
| --- |
| Constructors |
| **Constructor and Description** |
| [**Point**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#Point%28double,%20double%29)(double x, double y)  Конструктор, який створює точку з координатами (x, y) на сітці |

### Method Summary

|  |  |
| --- | --- |
| Methods | |
| **Modifier and Type** | **Method and Description** |
| protected java.lang.Object | [**clone**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#clone%28%29)() |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getPointBottom**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getPointBottom%28%29)()  Метод повертає точку яка є сусідньою знизу для даної точки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getPointLeft**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getPointLeft%28%29)()  Метод повертає точку яка є сусідньою зліва для даної точки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getPointRight**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getPointRight%28%29)()  Метод повертає точку яка є сусідньою справа для даної точки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getPointTop**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getPointTop%28%29)()  Метод повертає точку яка є сусідньою зверху для даної точки |
| double | [**getX**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getX%28%29)()  Метод повертає координату x |
| double | [**getY**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#getY%28%29)() |
| boolean | [**inRect**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#inRect%28com.lab111.picturetwister.vectorized.Point,%20com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) vertex1, [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) vertex2)  Метод перевіряє, чи входить дана точка в прямокутник утворений двома точками, які зєднані діагоналлю цього прямокутника |
| void | [**setPointBottom**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setPointBottom%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointBottom)  Метод встановлює точку яка є сусідньою знизу для даної точки |
| void | [**setPointLeft**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setPointLeft%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointLeft)  Метод встановлює точку яка є сусідньою зліва для даної точки |
| void | [**setPointRight**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setPointRight%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointRight)  Метод встановлює точку яка є сусідньою справа для даної точки |
| void | [**setPointTop**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setPointTop%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointTop)  Метод встановлює точку яка є сусідньою зверху для даної точки |
| void | [**setX**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setX%28double%29)(double x) |
| void | [**setY**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html#setY%28double%29)(double y) |

### Methods inherited from class java.lang.Object

equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

### Field Detail

#### x

private double x

Координата Х на сітці

#### y

private double y

Координата y на сітці

#### neighboringPoints

private java.util.HashMap<java.lang.String,[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> neighboringPoints

Мапа, яка зберігає "сусідів" даної точки, у вигляді ключ - точка, ключі: left - точка зліва right - точка српава top - точка зверху bottom - точка знизу

### Constructor Detail

#### Point

* + - public Point(double x,

double y)

Конструктор, який створює точку з координатами (x, y) на сітці

**Parameters:**

x - абсциса на сітці

y - ордината на сітці

### Method Detail

#### getPointLeft

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getPointLeft()

Метод повертає точку яка є сусідньою зліва для даної точки

**Returns:**

сусідню точку зліва

#### setPointLeft

public void setPointLeft([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointLeft)

Метод встановлює точку яка є сусідньою зліва для даної точки

**Parameters:**

pointLeft - сусіднья зліва точка для даної точки

#### getPointRight

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getPointRight()

Метод повертає точку яка є сусідньою справа для даної точки

**Returns:** сусідню точку справа

#### setPointRight

public void setPointRight([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointRight)

Метод встановлює точку яка є сусідньою справа для даної точки

**Parameters:**

pointLeft - сусіднья справа точка для даної точки

#### getPointTop

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getPointTop()

Метод повертає точку яка є сусідньою зверху для даної точки

**Returns:**

сусідню точку зверху

#### setPointTop

public void setPointTop([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointTop)

Метод встановлює точку яка є сусідньою зверху для даної точки

**Parameters:**

pointLeft - сусіднья зверху точка для даної точки

#### getPointBottom

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getPointBottom()

Метод повертає точку яка є сусідньою знизу для даної точки

**Returns:**

сусідню точку знизу

#### setPointBottom

public void setPointBottom([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) pointBottom)

Метод встановлює точку яка є сусідньою знизу для даної точки

**Parameters:**

pointLeft - сусіднья знизу точка для даної точки

#### getX

public double getX()

Метод повертає координату x

**Returns:**

координату x

#### getY

public double getY()

**Returns:**

the y

#### toString

public java.lang.String toString()

toString, що тут скажеш

**Overrides:**

toString in class java.lang.Object

#### clone

* + - protected java.lang.Object clone()

throws java.lang.CloneNotSupportedException

**Overrides:**

clone in class java.lang.Object

**Throws:**

java.lang.CloneNotSupportedException

#### setX

public void setX(double x)

**Parameters:**

x - the x to set

#### setY

public void setY(double y)

**Parameters:**

y - the y to set

#### inRect

* + - public boolean inRect([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) vertex1,

[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) vertex2)

Метод перевіряє, чи входить дана точка в прямокутник утворений двома точками, які зєднані діагоналлю цього прямокутника

**Parameters:**

vertex1 - перша точка

vertex2 - друга точка

**Returns:**

true, якщо точка лежить всередині прямокутника і false, якщо не лежить

com.lab111.picturetwister.vectorized

## Class ConvexHull

* java.lang.Object
  + com.lab111.picturetwister.vectorized.ConvexHull
* Direct Known Subclasses:

[VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)

public class **ConvexHull**

extends java.lang.Object

Клас описує "оболочку", яка складається з точок Point і може бути довільної форми.

**Author:**

Бедь Анатолій Михайлович

### Field Summary

|  |  |
| --- | --- |
| Fields | |
| **Modifier and Type** | **Field and Description** |
| protected java.util.HashMap<java.lang.String,java.lang.Object> | [**attribute**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#attribute)  Атрибути |
| protected java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> | [**peakPixel**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#peakPixel)  Список кутів векторного пікселя |

### Constructor Summary

|  |
| --- |
| Constructors |
| **Constructor and Description** |
| [**ConvexHull**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#ConvexHull%28java.util.ArrayList%29)(java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> points)  Конструктор |
| [**ConvexHull**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#ConvexHull%28com.lab111.picturetwister.vectorized.Point...%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)... points)  Конструктор |

### Method Summary

|  |  |
| --- | --- |
| Methods | |
| **Modifier and Type** | **Method and Description** |
| private static [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) | [**bestTriangle**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#bestTriangle%28com.lab111.picturetwister.vectorized.ConvexHull,%20com.lab111.picturetwister.vectorized.ConvexHull%29)([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) a, [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) b) |
| double | [**getArea**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getArea%28%29)()  Метод рахує площу оболочки |
| <T> T | [**getAttribute**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getAttribute%28java.lang.String%29)(java.lang.String key)  Метод для отримання атрибутів оболочки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getCenterOfMass**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getCenterOfMass%28%29)()  Метод повертає Point, шо є точкою центр мас. |
| java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> | [**getPeakPixel**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getPeakPixel%28%29)()  Метод для отримання контуру оболочки |
| double | [**getPerimeter**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getPerimeter%28%29)()  Метод рахує периметер оболочки |
| private static [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) | [**getTriangle**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getTriangle%28com.lab111.picturetwister.vectorized.ConvexHull%29)([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) figure) |
| boolean | [**isInnerPoint**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#isInnerPoint%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) p)  Метод перевіряє, чи знаходиться точка всередині оболочки, яка описується ConvexHull |
| static void | [**main**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#main%28java.lang.String[]%29)(java.lang.String[] args) |
| void | [**setAttribute**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#setAttribute%28java.lang.String,%20java.lang.Object%29)(java.lang.String key, java.lang.Object value)  Метод для встановлення атрибутів оболочки |
| java.util.ArrayList<[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)> | [**split**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#split%28%29)()  Метод для розбиття оболочки на трикутники. |
| java.lang.String | [**toString**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#toString%28%29)() |
| private static boolean | [**triangleIsValid**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#triangleIsValid%28com.lab111.picturetwister.vectorized.ConvexHull,%20java.util.ArrayList%29)([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) t, java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> contour) |
| private static boolean | [**triangleIsValid**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#triangleIsValid%28com.lab111.picturetwister.vectorized.ConvexHull,%20com.lab111.picturetwister.vectorized.ConvexHull%29)([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) t, [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) f) |

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

### Field Detail

#### peakPixel

protected java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> peakPixel

Список кутів векторного пікселя

#### attribute

protected java.util.HashMap<java.lang.String,java.lang.Object> attribute

Атрибути

### Constructor Detail

#### ConvexHull

public ConvexHull(java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> points)

Конструктор

**Parameters:**

points - список точок

#### ConvexHull

public ConvexHull([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)... points)

Конструктор

**Parameters:**

points - перелік точок

### Method Detail

#### getPeakPixel

public java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> getPeakPixel()

Метод для отримання контуру оболочки

**Returns:**

список всіх вершин оболочки

#### getAttribute

public <T> T getAttribute(java.lang.String key)

Метод для отримання атрибутів оболочки

**Parameters:**

key - ключ атрибута

value - значення атрибута

#### setAttribute

* + - public void setAttribute(java.lang.String key,

java.lang.Object value)

Метод для встановлення атрибутів оболочки

**Parameters:**

key - ключ атрибута

value - значення атрибута

#### getPerimeter

public double getPerimeter()

Метод рахує периметер оболочки

**Returns:**

периметер оболочки

#### getArea

public double getArea()

Метод рахує площу оболочки

**Returns:**

площу оболочки

#### isInnerPoint

public boolean isInnerPoint([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) p)

Метод перевіряє, чи знаходиться точка всередині оболочки, яка описується ConvexHull

**Parameters:**

p - точка

**Returns:**

true, якщо p - внутрішня точка оболочки, або якщо вона лежить на контурі, false якщо точка лежить поза оболочкою.

#### triangleIsValid

* + - private static boolean triangleIsValid([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) t,

java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> contour)

#### triangleIsValid

* + - private static boolean triangleIsValid([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) t,

[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) f)

#### bestTriangle

* + - private static [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) bestTriangle([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) a,

[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) b)

#### getTriangle

private static [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) getTriangle([ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html) figure)

#### split

public java.util.ArrayList<[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)> split()

Метод для розбиття оболочки на трикутники. Він базується на наступному алгоритмові.  
1) Проходимо контур по всім точкам і вибераємо трикутники із трьох сісідніх точок. Нехай це будуть точки A(i-1), Ai, A(i+1), тоді точки A(i-1) і A(i+1) зєднуємо хордою. Таким чином можна отримати N різних трикутників для контуру що складається з N точок.  
2) Кожен трикутник перевіряємо на правильність. Якщо хорда A(i-1)A(i+1) проходить не всередині контуру то такий трикутник не проходить перевірку на валідність.  
3) Вибираємо той трикутник у якого відношення площі до периметру є найбільшим (це дозволяє вибрати трикутник найбільш наближений до рівностороннього) Заносимо цей трикутник до списку.   
4) Видаляємо з контуру Ai - точку трикутника, який ми вибрали як "найкращий".   
5) Повторюємо попередні пункти до тих пір поки в контурі не залишиться три точки.   
6) З'єднавши ці три точки отримуємо останній трикутник і заносимо його до списку.

**Returns:**

список трикутників, або null якщо оболочка містить тільки дві точки

#### getCenterOfMass

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getCenterOfMass()

Метод повертає Point, шо є точкою центр мас. Можливі чотири випадки:   
1) ConvexHull містить тільки одну точку. В цьому випадку повертаємо її саму   
2) ConvexHull містить дві точки. В цьому випадку повертаємо середину відрізка, що сполучає ці дві точки   
3) ConvexHull містить три точки. В цьому випадку маємо трикутник, тому повертаємо точку перетину медіан, яка і буде точкою центер мас трикутника   
4) ConvexHull містить більше трьох точок. В цьому випадку розбиваємо наш полігон на трикутники. Центер мас вираховується як середнє арифметичне центра мас всіх трикутників, з урахуванням вагових коефіцієнтів, якими в даному випадку є площі трикутників

**Returns:**

Point - точку центер мас

#### toString

public java.lang.String toString()

**Overrides:**

toString in class java.lang.Object

#### main

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

com.lab111.picturetwister.vectorized

## Class VectorPixel

* java.lang.Object
  + [com.lab111.picturetwister.vectorized.ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)
    - com.lab111.picturetwister.vectorized.VectorPixel

public class **VectorPixel**

extends [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)

Клас, який описує векторний піксель, наслідується від класа ConvexHull

**Author:**

Бедь Анатолій Михайлович

### Field Summary

### Fields inherited from class com.lab111.picturetwister.vectorized.[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)

[attribute](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#attribute), [peakPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#peakPixel)

### Constructor Summary

|  |
| --- |
| Constructors |
| **Constructor and Description** |
| [**VectorPixel**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#VectorPixel%28java.util.ArrayList%29)(java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> points) |

### Method Summary

|  |  |
| --- | --- |
| Methods | |
| **Modifier and Type** | **Method and Description** |
| int | [**getColor**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#getColor%28%29)()  Метод повертає колір векторного пікселя |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) | [**getPoint**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#getPoint%28int%29)(int poz)  Метод повертає точку з заданої позиції |
| double[] | [**getWeightPixel**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#getWeightPixel%28%29)()  Метод повертає добуток "ваги" пікселя на його центер мас |
| void | [**setColor**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#setColor%28int%29)(int color)  Метод встановлює колір векторного пікселя |
| java.lang.String | [**toString**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html#toString%28%29)() |

### Methods inherited from class com.lab111.picturetwister.vectorized.[ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)

[getArea](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getArea%28%29), [getAttribute](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getAttribute%28java.lang.String%29), [getCenterOfMass](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getCenterOfMass%28%29), [getPeakPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getPeakPixel%28%29), [getPerimeter](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#getPerimeter%28%29), [isInnerPoint](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#isInnerPoint%28com.lab111.picturetwister.vectorized.Point%29), [main](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#main%28java.lang.String[]%29), [setAttribute](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#setAttribute%28java.lang.String,%20java.lang.Object%29), [split](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#split%28%29)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

### Constructor Detail

#### VectorPixel

public VectorPixel(java.util.ArrayList<[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)> points)

### Method Detail

#### setColor

public void setColor(int color)

Метод встановлює колір векторного пікселя

**Parameters:**

color -

#### getColor

public int getColor()

Метод повертає колір векторного пікселя

**Returns:**

#### getWeightPixel

public double[] getWeightPixel()

Метод повертає добуток "ваги" пікселя на його центер мас

**Returns:**

#### getPoint

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) getPoint(int poz)

Метод повертає точку з заданої позиції

**Parameters:**

poz - позиція точки яку хочемо отримати

**Returns:**

точку в позиції poz

#### toString

public java.lang.String toString()

**Overrides:**

[toString](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html#toString%28%29) in class [ConvexHull](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\ConvexHull.html)

com.lab111.picturetwister.vectorized

## Class Reticle

* java.lang.Object
  + com.lab111.picturetwister.vectorized.Reticle

public class **Reticle**

extends java.lang.Object

Клас описує сітку точок, зв'язаних між собою

**Author:**

Бедь Анатолій Михайлович

### Field Summary

|  |  |
| --- | --- |
| Fields | |
| **Modifier and Type** | **Field and Description** |
| int[][] | [**colorMatrix**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#colorMatrix) |
| private int | [**height**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#height) |
| private [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**points**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#points) |
| private [MatrixTransform](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html) | [**transform**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#transform) |
| private [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] | [**vectorPixels**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#vectorPixels) |
| private [Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) | [**weightFunction**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#weightFunction) |
| private int | [**width**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#width) |

### Constructor Summary

|  |
| --- |
| Constructors |
| **Constructor and Description** |
| [**Reticle**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#Reticle%28int[][]%29)(int[][] colorsRGB)  Конструктор, який створює сітку |

### Method Summary

|  |  |
| --- | --- |
| Methods | |
| **Modifier and Type** | **Method and Description** |
| [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] | [**cloneConvexHulls**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#cloneConvexHulls%28%29)()  метод повертає копію ConvexHull[][] |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**clonePoints**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#clonePoints%28%29)()  метод повертає копію сітки точок |
| static [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] | [**createConvexHullsSet**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#createConvexHullsSet%28int[][],%20com.lab111.picturetwister.vectorized.Point[][]%29)(int[][] intMatrix, [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] points)  Метод повертає матрицю векторних пікселів |
| static [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**createPointSet**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#createPointSet%28int,%20int%29)(int width, int height)  Метод повертає сітку точок, в якій кожна точка знає про свої сусідні точки зверху, зліва, справа, знизу |
| void | [**deformation**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#deformation%28%29)()  Метод, який ламає сітку. |
| void | [**deformation1**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#deformation1%28%29)() |
| int[][] | [**getColorMatrix**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getColorMatrix%28%29)()  метод повертає матрицю кольорів |
| java.util.ArrayList<[VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)> | [**getConnectedPixels**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getConnectedPixels%28com.lab111.picturetwister.vectorized.Point%29)([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) p)  Мето для отримання пікселів до яких входить задана точка p |
| [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] | [**getConvexHulls**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getConvexHulls%28%29)() |
| int | [**getHeight**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getHeight%28%29)() |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**getPoints**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getPoints%28%29)()  метод повертає сітку зв'язаних точок |
| double[][] | [**getPointsXY**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getPointsXY%28%29)()  метод повертає матрицю координат у вигляді Ax Cx ..... |
| [Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) | [**getWeightFunction**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getWeightFunction%28%29)() |
| int | [**getWidth**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#getWidth%28%29)() |
| static void | [**main**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#main%28java.lang.String[]%29)(java.lang.String[] args) |
| void | [**rotate**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#rotate%28double%29)(double angle)  метод для здійснення повороту сітки на кут |
| void | [**scale**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#scale%28double,%20double%29)(double mx, double my)  метод для маштабування сітки |
| void | [**setWeightFunction**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#setWeightFunction%28com.lab111.picturetwister.core.Function%29)([Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) weightFunction) |
| void | [**transfer**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html#transfer%28double,%20double%29)(double transferX, double transferY)  метод, який дозволяє перемістити сітку |

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

#### points

private [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] points

#### weightFunction

private [Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) weightFunction

#### vectorPixels

private [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] vectorPixels

#### transform

private [MatrixTransform](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html) transform

#### colorMatrix

public int[][] colorMatrix

#### height

private int height

#### width

private int width

### Constructor Detail

#### Reticle

public Reticle(int[][] colorsRGB)

Конструктор, який створює сітку

**Parameters:**

colorsRGB - - матриця кольорів

### Method Detail

#### createPointSet

* + - public static [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] createPointSet(int width,

int height)

Метод повертає сітку точок, в якій кожна точка знає про свої сусідні точки зверху, зліва, справа, знизу

**Parameters:**

width - ширина зображення

height - висота зображення

**Returns:**

Points[][] - сітка точок

#### createConvexHullsSet

* + - public static [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] createConvexHullsSet(int[][] intMatrix,

[Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] points)

Метод повертає матрицю векторних пікселів

**Parameters:**

intMatrix - матриця кольорів

points - сітка точок

**Returns:**

#### scale

* + - public void scale(double mx,

double my)

метод для маштабування сітки

**Parameters:**

mx - параметр збільшення по осі Ох

my - параметр збільшення по осі Оу

#### rotate

public void rotate(double angle)

метод для здійснення повороту сітки на кут

**Parameters:**

angle - кут на який потрібно повернути сітку

#### transfer

* + - public void transfer(double transferX,

double transferY)

метод, який дозволяє перемістити сітку

**Parameters:**

transferX - відстань на яку треба перемістити сітку по координаті X

transferY - відстань на яку треба перемістити сітку по координаті Y

#### deformation

public void deformation()

Метод, який ламає сітку.

#### deformation1

public void deformation1()

#### getPoints

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] getPoints()

метод повертає сітку зв'язаних точок

**Returns:**

points

#### getConvexHulls

public [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] getConvexHulls()

**Returns:**

vectorPixels

#### getPointsXY

public double[][] getPointsXY()

метод повертає матрицю координат у вигляді Ax Cx ..... Px Ay Cy ..... Py Bx Dx ..... Lx By Dy ..... Ly Mx Nx ..... Qx My Ny ..... Qy ............... ............... Rx Tx ...... Hx Ry Ty ...... Hy де точки A(Ax,Ay), B(Bx,By), D(Dx,Dy), C(Cx,Cy) є кутами пікселя (і так далі відповідно)

**Returns:**

#### cloneConvexHulls

public [VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)[][] cloneConvexHulls()

метод повертає копію ConvexHull[][]

**Returns:**

#### clonePoints

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] clonePoints()

метод повертає копію сітки точок

**Returns:**

#### getColorMatrix

public int[][] getColorMatrix()

метод повертає матрицю кольорів

**Returns:**

colorMatrix

#### getHeight

public int getHeight()

**Returns:**

the height

#### getWidth

public int getWidth()

**Returns:**

the width

#### getWeightFunction

public [Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) getWeightFunction()

#### setWeightFunction

public void setWeightFunction([Function](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\core\Function.html) weightFunction)

#### getConnectedPixels

public java.util.ArrayList<[VectorPixel](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\VectorPixel.html)> getConnectedPixels([Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html) p)

Мето для отримання пікселів до яких входить задана точка p

**Parameters:**

p - точка

**Returns:**

com.lab111.picturetwister.vectorized

## Class MatrixTransform

* java.lang.Object
  + com.lab111.picturetwister.vectorized.MatrixTransform

public class **MatrixTransform**

extends java.lang.Object

Клас що описує матрицю перетоврень

**Author:**

Бедь Анатолій Михайлович

### Field Summary

|  |  |
| --- | --- |
| Fields | |
| **Modifier and Type** | **Field and Description** |
| private double[][] | [**matrix**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#matrix) |
| private [Reticle](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html) | [**reticle**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#reticle) |

### Constructor Summary

|  |
| --- |
| Constructors |
| **Constructor and Description** |
| [**MatrixTransform**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#MatrixTransform%28com.lab111.picturetwister.vectorized.Reticle%29)([Reticle](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html) reticle) |

### Method Summary

|  |  |
| --- | --- |
| Methods | |
| **Modifier and Type** | **Method and Description** |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**rotate**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#rotate%28double%29)(double angle)  Метод для повороту сітки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**scale**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#scale%28double,%20double%29)(double mx, double my)  Метод для маштабування сітки побудованої з точок Point |
| void | [**setRotate**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#setRotate%28double%29)(double angle)  Метод для встановлення кута повороту сітки |
| void | [**setScale**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#setScale%28double,%20double%29)(double mx, double my)  Метод для встановлення коефіцієнтів маштабування |
| void | [**setTransfer**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#setTransfer%28double,%20double%29)(double transferX, double transferY)  Метод для встановлення значень переміщення сітки |
| [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] | [**transfer**](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\MatrixTransform.html#transfer%28double,%20double%29)(double transferX, double transferY)  Метод для переміщення сітки |

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Field Detail

#### matrix

private double[][] matrix

#### reticle

private [Reticle](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html) reticle

### Constructor Detail

#### MatrixTransform

public MatrixTransform([Reticle](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Reticle.html) reticle)

### Method Detail

#### scale

* + - public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] scale(double mx,

double my)

Метод для маштабування сітки побудованої з точок Point

**Parameters:**

mx - коефіцієнт маштабування по осі X

my - коефіцієнт маштабування по осі Y

**Returns:**

двовимірний масив точок промаштабованої Reticle

#### setScale

* + - public void setScale(double mx,

double my)

Метод для встановлення коефіцієнтів маштабування

**Parameters:**

mx - коефіцієнт маштабування по осі X

my - коефіцієнт маштабування по осі Y

#### setRotate

public void setRotate(double angle)

Метод для встановлення кута повороту сітки

**Parameters:**

angle - кут повороту заданий в градусах.

#### rotate

public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] rotate(double angle)

Метод для повороту сітки

**Parameters:**

angle - кут повороту заданий в градусах.

**Returns:**

двовимірний масив точок повернутої на кут angle Reticle

#### transfer

* + - public [Point](file:///C:\Users\%D0%A2%D0%BE%D0%BB%D1%8F\workspace\PictureTwister\doc\javadoc\com\lab111\picturetwister\vectorized\Point.html)[][] transfer(double transferX,

double transferY)

Метод для переміщення сітки

**Parameters:**

transferX - відстань для переносу по осі X задана в пікселях

transferY - відстань для переносу по осі Y задана в пікселях

**Returns:**

двовимірний масив точок переміщеної Reticle

#### setTransfer

* + - public void setTransfer(double transferX,

double transferY)

Метод для встановлення значень переміщення сітки

**Parameters:**

transferX - відстань для переносу по осі X задана в пікселях

transferY - відстань для переносу по осі Y задана в пікселях

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Було перевірено на скільки функціонал програми відповідає технічному завдання. Далі наведені скріншоти, які показують роботу додатку.



Рисунок 4.1 Вихідне зображення

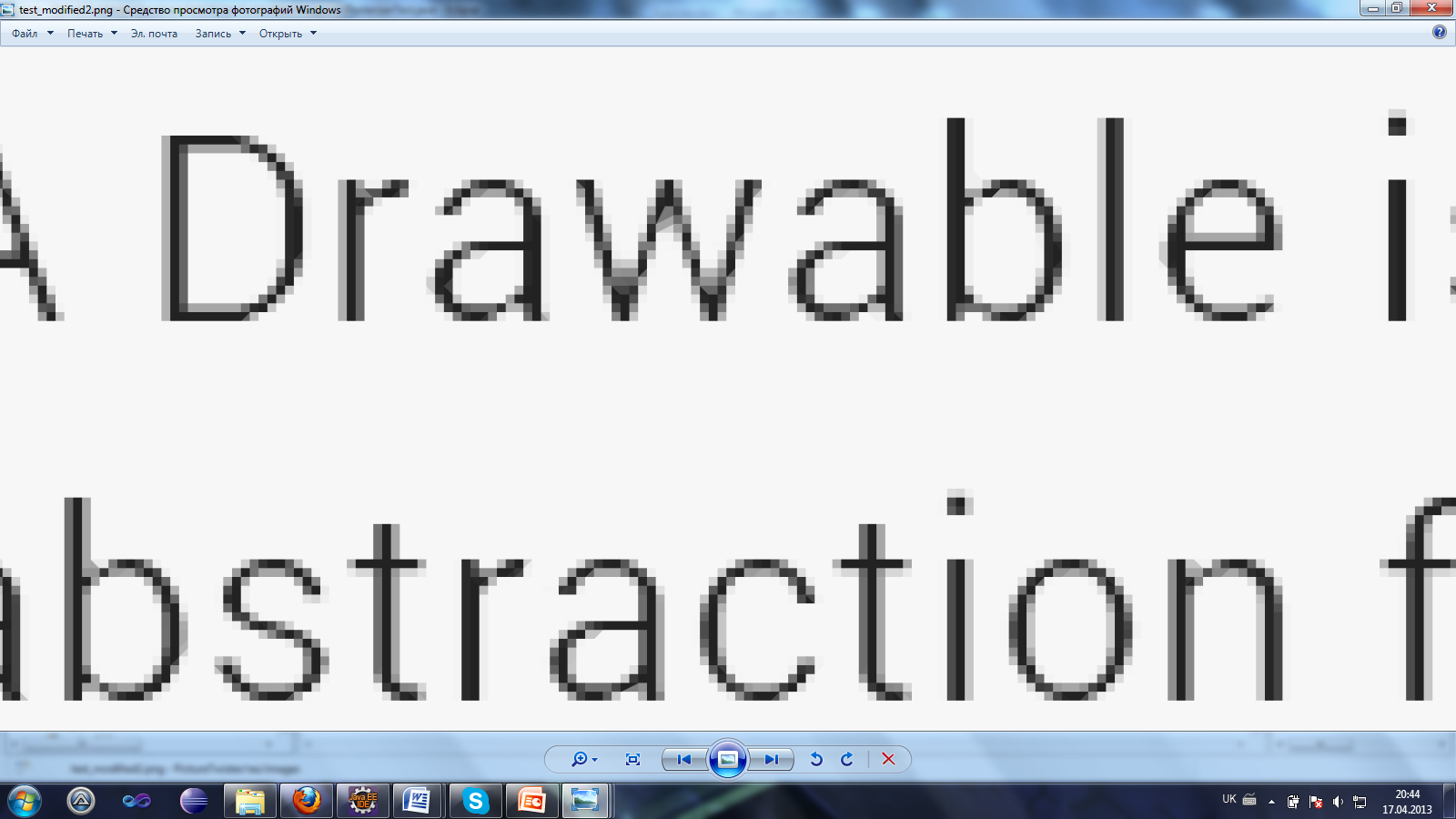


Рисунок 4.2 Зображення після застосування згладжування контурів

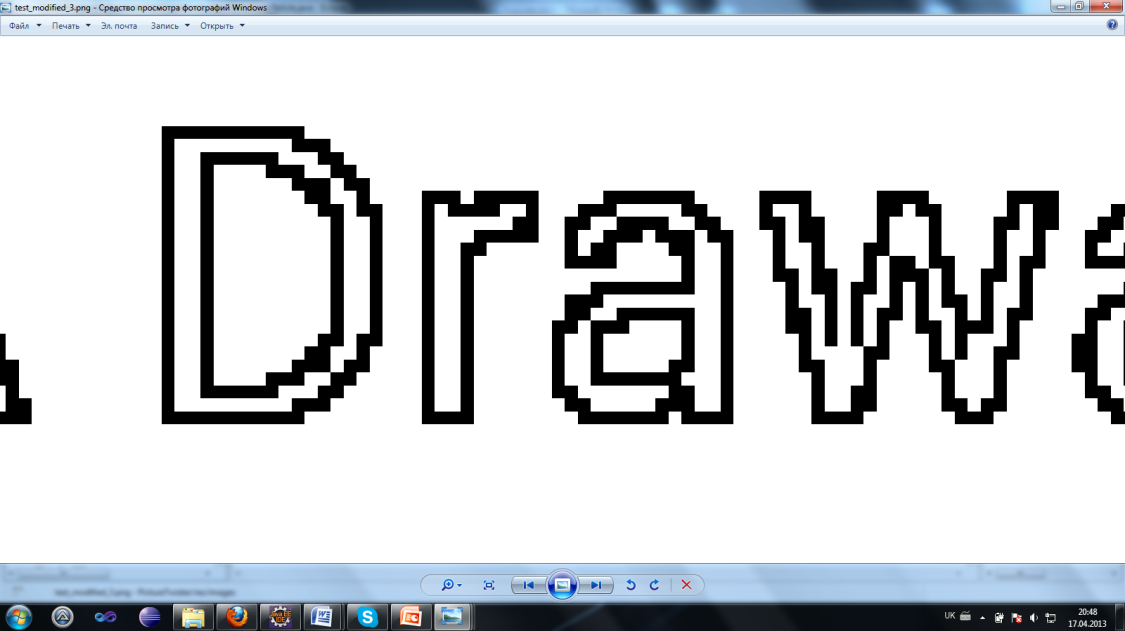


Рисунок 4.3 Результат пошуку контурних ланцюгів

ВИСНОВКИ

Було розроблено програмний додаток для побудови векторної моделі зображення та її редагування. Реалізовано наступну функціональність: можливість отримати на вхід растрове зображення і представити його векторною моделлю; згладжування контурів; пошук контурних ланцюгів; реалізовано збереження результату роботи додатку у вигляді файла формату JPEG або PNG. Дану функціональність було реалізовано на мові програмування Java. Інтерфейс користувача забезпечує доступ до всієї функціональності програми. Проект повністю задокументований за допомогою JavaDoc.

Усі вимоги, описані в технічному завданні були виконані в повній мірі.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Герберт Шилдт SWING: руководство для начинающих – М.: “Вильямс”, 2007. – С. 704. – ISBN 0-07-226314-8.
2. 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.
3. Приемы объектно-ориентированого проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб.: Питер, 2010 – 368 с.: ил. –ISBN 978-5-469-01136-1.
4. Хорстманн Кей С. Java 2. Том 1. Основы / Кей Хорстманн, Гари Корнелл; Пер с англ. – Изд. 8-е. – М.: ООО “И.ДВильямс”, 2011. – 816 c.: ил. – Парал. тит. англ. – (Библиотека профессионала). –ISBN 978-5-8459-1378-4 (рус.).
5. Е. А. Никулин. Компьютерная геометрия и алгоритмы машинной графики; Издательство: БХВ-Петербург Серия: Учебное пособие ISBN 5-94157-264-6; 2003 г.

ДОДАТОК А. ПРОГРАМНИЙ КОД ПРОЕКТУ

**package** com.lab111.picturetwister.vectorized;

**import** java.util.HashMap;

**import** com.lab111.picturetwister.core.Inaccurate;

**import** com.lab111.picturetwister.core.Stat;

/\*\*

\* Клас описує точку - як кут пікселя.

\*

\* **@author** Бедь Анатолій Михайлович

\*

\*/

**public** **class** Point {

/\*\*

\* Координата Х на сітці

\*/

**private** **double** x;

/\*\*

\* Координата y на сітці

\*/

**private** **double** y;

/\*\*

\* Мапа, яка зберігає "сусідів" даної точки, у вигляді ключ - точка, ключі:

\* left - точка зліва right - точка српава top - точка зверху bottom - точка

\* знизу

\*/

**private** HashMap<String, Point> neighboringPoints;

/\*\*

\* Конструктор, який створює точку з координатами (x, y) на сітці

\* **@param** x абсциса на сітці

\* **@param** y ордината на сітці

\*/

**public** Point(**double** x, **double** y) {

**this**.x = x;

**this**.y = y;

neighboringPoints = **new** HashMap<String, Point>(4);

}

/\*\*

\* Метод повертає точку яка є сусідньою зліва для даної точки

\* **@return** сусідню точку зліва

\*/

**public** Point getPointLeft() {

**return** neighboringPoints.get("left");

}

/\*\*

\* Метод встановлює точку яка є сусідньою зліва для даної точки

\* **@param** pointLeft сусіднья зліва точка для даної точки

\*/

**public** **void** setPointLeft(Point pointLeft) {

neighboringPoints.put("left", pointLeft);

}

/\*\*

\* Метод повертає точку яка є сусідньою справа для даної точки

\* **@return** сусідню точку справа

\*/

**public** Point getPointRight() {

**return** neighboringPoints.get("right");

}

/\*\*

\* Метод встановлює точку яка є сусідньою справа для даної точки

\* **@param** pointLeft сусіднья справа точка для даної точки

\*/

**public** **void** setPointRight(Point pointRight) {

neighboringPoints.put("right", pointRight);

}

/\*\*

\* Метод повертає точку яка є сусідньою зверху для даної точки

\* **@return** сусідню точку зверху

\*/

**public** Point getPointTop() {

**return** neighboringPoints.get("top");

}

/\*\*

\* Метод встановлює точку яка є сусідньою зверху для даної точки

\* **@param** pointLeft сусіднья зверху точка для даної точки

\*/

**public** **void** setPointTop(Point pointTop) {

neighboringPoints.put("top", pointTop);

}

/\*\*

\* Метод повертає точку яка є сусідньою знизу для даної точки

\* **@return** сусідню точку знизу

\*/

**public** Point getPointBottom() {

**return** neighboringPoints.get("bottom");

}

/\*\*

\* Метод встановлює точку яка є сусідньою знизу для даної точки

\* **@param** pointLeft сусіднья знизу точка для даної точки

\*/

**public** **void** setPointBottom(Point pointBottom) {

neighboringPoints.put("bottom", pointBottom);

}

/\*\*

\* Метод повертає координату x

\* **@return** координату x

\*/

**public** **double** getX() {

**return** x;

}

/\*\*

\* **@return** the y

\*/

**public** **double** getY() {

**return** y;

}

/\*\*

\* toString, що тут скажеш

\*/

@Override

**public** String toString() {

**return** "( " + x + ", " + y + " )";

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

Point point = **new** Point(**this**.x, **this**.y);

point.setPointBottom(**this**.getPointBottom());

point.setPointLeft(**this**.getPointLeft());

point.setPointRight(**this**.getPointRight());

point.setPointTop(**this**.getPointTop());

**return** point;

}

/\*\*

\* **@param** x

\* the x to set

\*/

**public** **void** setX(**double** x) {

**this**.x = x;

}

/\*\*

\* **@param** y

\* the y to set

\*/

**public** **void** setY(**double** y) {

**this**.y = y;

}

/\*\*

\* Метод перевіряє, чи входить дана точка в прямокутник утворений двома точками,

\* які зєднані діагоналлю цього прямокутника

\* **@param** vertex1 перша точка

\* **@param** vertex2 друга точка

\* **@return** true, якщо точка лежить всередині прямокутника і false, якщо не лежить

\*/

**public** **boolean** inRect(Point vertex1, Point vertex2) {

**if** (vertex1 == **null** || vertex2 == **null**)

**return** **false**;

**double** left = Stat.*min*(vertex1.getX(), vertex2.getX());

**double** right = Stat.*max*(vertex1.getX(), vertex2.getX());

**double** top = Stat.*min*(vertex1.getY(), vertex2.getY());

**double** bottom = Stat.*max*(vertex1.getY(), vertex2.getY());

**double** x = **this**.getX();

**double** y = **this**.getY();

**return** Inaccurate.*moreOrEquals*(x, left)

&& Inaccurate.*lessOrEquals*(x, right)

&& Inaccurate.*moreOrEquals*(y, top)

&& Inaccurate.*lessOrEquals*(y, bottom);

}

}

**package** com.lab111.picturetwister.vectorized;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** org.apache.commons.math3.geometry.euclidean.twod.Vector2D;

**import** com.lab111.picturetwister.core.Inaccurate;

**import** com.lab111.picturetwister.core.Linear;

/\*\*

\* Клас описує "оболочку", яка складається з точок Point і може бути довільної

\* форми.

\* **@author** Бедь Анатолій Михайлович

\*/

**public** **class** ConvexHull {

/\*\*

\* Список кутів векторного пікселя

\*/

**protected** ArrayList<Point> peakPixel;

/\*\*

\* Атрибути

\*/

**protected** HashMap<String, Object> attribute;

**public** ConvexHull() {

attribute = **new** HashMap<String, Object>();

peakPixel = **new** ArrayList<Point>();

}

/\*\*

\* Конструктор

\*

\* **@param** points

\* список точок

\*/

**public** ConvexHull(ArrayList<Point> points) {

**this**.peakPixel = points;

attribute = **new** HashMap<String, Object>();

}

/\*\*

\* Конструктор

\*

\* **@param** points

\* перелік точок

\*/

**public** ConvexHull(Point... points) {

**this**.peakPixel = **new** ArrayList<Point>();

**for** (Point p : points) {

**this**.peakPixel.add(p);

}

attribute = **new** HashMap<String, Object>();

}

/\*\*

\* Метод для отримання контуру оболочки

\*

\* **@return** список всіх вершин оболочки

\*/

**public** ArrayList<Point> getPeakPixel() {

**return** peakPixel;

}

/\*\*

\* Метод для отримання атрибутів оболочки

\*

\* **@param** key ключ атрибута

\* **@param** value значення атрибута

\*/

**public** <T> T getAttribute(String key) {

**return** (T) attribute.get(key);

}

/\*\*

\* Метод для встановлення атрибутів оболочки

\* **@param** key ключ атрибута

\* **@param** value

\* значення атрибута

\*/

**public** **void** setAttribute(String key, Object value) {

attribute.put(key, value);

}

/\*\*

\* Метод рахує периметер оболочки

\* **@return** периметер оболочки

\*/

**public** **double** getPerimeter() {

**double** res = 0;

**for** (**int** i = 0; i < peakPixel.size() - 1; i++) {

res += Linear.*createVector*(peakPixel.get(i), peakPixel.get(i + 1))

.getNorm();

}

res += Linear.*createVector*(peakPixel.get(peakPixel.size() - 1),

peakPixel.get(0)).getNorm();

setAttribute("p", res);

**return** res;

}

/\*\*

\* Метод рахує площу оболочки

\* **@return** площу оболочки

\*/

**public** **double** getArea() {

**if** (peakPixel.size() == 3) {

Vector2D f = Linear

.*createVector*(peakPixel.get(0), peakPixel.get(1));

Vector2D s = Linear

.*createVector*(peakPixel.get(0), peakPixel.get(2));

setAttribute("a", Math.*abs*(Linear.*product*(f, s)) / 2);

**return** Math.*abs*(Linear.*product*(f, s)) / 2;

}

ArrayList<ConvexHull> splitting = split();

**double** res = 0;

**for** (ConvexHull c : splitting)

res += c.getArea();

**return** res;

}

/\*\*

\* Метод перевіряє, чи знаходиться точка всередині оболочки, яка описується

\* ConvexHull

\* **@param** p точка

\* **@return** true, якщо p - внутрішня точка оболочки, або якщо вона лежить на

\* контурі, false якщо точка лежить поза оболочкою.

\*/

**public** **boolean** isInnerPoint(Point p) {

**if** (peakPixel.size() == 3) {

**double** s = **new** ConvexHull(peakPixel.get(0), peakPixel.get(1), p)

.getArea()

+ **new** ConvexHull(peakPixel.get(1), peakPixel.get(2), p)

.getArea()

+ **new** ConvexHull(peakPixel.get(2), peakPixel.get(0), p)

.getArea();

**return** Inaccurate.*equals*(s, getArea());

}

ArrayList<ConvexHull> splitting = split();

**boolean** res = **false**;

**for** (ConvexHull c : splitting)

res |= c.isInnerPoint(p);

**return** res;

}

**private** **static** **boolean** triangleIsValid(ConvexHull t,

ArrayList<Point> contour) {

Point f = t.getPeakPixel().get(0);

Point s = t.getPeakPixel().get(2);

Vector2D fv = Linear.*createVector*(t.getPeakPixel().get(0), t

.getPeakPixel().get(1));

Vector2D sv = Linear.*createVector*(t.getPeakPixel().get(1), t

.getPeakPixel().get(2));

**if** (Linear.*product*(fv, sv) <= 0)

**return** **false**;

**for** (Point p : contour) {

**if** (!t.getPeakPixel().contains(p) && t.isInnerPoint(p))

**return** **false**;

}

**if** (!t.getPeakPixel().contains(contour.get(0))

&& !t.getPeakPixel().contains(contour.get(contour.size() - 1))) {

**if** (Linear.*segmentIntersection*(f, s,

contour.get(contour.size() - 1), contour.get(0)) != **null**)

**return** **false**;

}

**for** (**int** i = 0; i < contour.size() - 1; i++) {

**if** (!t.getPeakPixel().contains(contour.get(i))

&& !t.getPeakPixel().contains(contour.get(i + 1))) {

**if** (Linear.*segmentIntersection*(f, s, contour.get(i),

contour.get(i + 1)) != **null**)

**return** **false**;

}

}

**return** **true**;

}

**private** **static** **boolean** triangleIsValid(ConvexHull t, ConvexHull f) {

**return** *triangleIsValid*(t, f.getPeakPixel());

}

**private** **static** ConvexHull bestTriangle(ConvexHull a, ConvexHull b) {

**if** (a == **null** && b == **null**)

**return** **null**;

**if** (a == **null** && b != **null**)

**return** b;

**if** (a != **null** && b == **null**)

**return** a;

**double** a\_ = a.getAttribute("factor");

**double** b\_ = b.getAttribute("factor");

**return** (a\_ < b\_) ? b : a;

}

**private** **static** ConvexHull getTriangle(ConvexHull figure) {

ConvexHull res = **null**;

ArrayList<Point> contour = figure.getPeakPixel();

**for** (**int** i = 1; i < contour.size() - 1; i++) {

ConvexHull temp = **new** ConvexHull(contour.get(i - 1),

contour.get(i), contour.get(i + 1));

**if** (*triangleIsValid*(temp, figure)) {

temp.setAttribute("factor",

temp.getArea() / temp.getPerimeter());

res = *bestTriangle*(res, temp);

}

}

ConvexHull temp = **new** ConvexHull(contour.get(contour.size() - 1),

contour.get(0), contour.get(1));

**if** (*triangleIsValid*(temp, figure)) {

temp.setAttribute("factor", temp.getArea() / temp.getPerimeter());

res = *bestTriangle*(res, temp);

}

temp = **new** ConvexHull(contour.get(contour.size() - 2),

contour.get(contour.size() - 1), contour.get(0));

**if** (*triangleIsValid*(temp, figure)) {

temp.setAttribute("factor", temp.getArea() / temp.getPerimeter());

res = *bestTriangle*(res, temp);

}

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

System.*out*.println(figure);

}

**return** res;

}

/\*\*

\* Метод для розбиття оболочки на трикутники. Він базується на наступному

\* алгоритмові.<br/>

\* 1) Проходимо контур по всім точкам і вибираємо трикутники із трьох

\* сусідніх точок. Нехай це будуть точки A(i-1), Ai, A(i+1), тоді точки

\* A(i-1) і A(i+1) зєднуємо хордою. Таким чином можна отримати N різних

\* трикутників для контуру що складається з N точок.<br/>

\* 2) Кожен трикутник перевіряємо на правильність. Якщо хорда A(i-1)A(i+1)

\* проходить не всередині контуру то такий трикутник не проходить перевірку

\* на валідність.<br/>

\* 3) Вибираємо той трикутник у якого відношення площі до периметру є

\* найбільшим (це дозволяє вибрати трикутник найбільш наближений до

\* рівностороннього) Заносимо цей трикутник до списку. <br/>

\* 4) Видаляємо з контуру Ai - точку трикутника, який ми вибрали як

\* "найкращий". <br/>

\* 5) Повторюємо попередні пункти до тих пір поки в контурі не залишиться

\* три точки. <br/>

\* 6) З'єднавши ці три точки отримуємо останній трикутник і заносимо його до

\* списку. <br/>

\*

\* **@return** список трикутників, або null якщо оболочка містить тільки дві

\* точки

\*/

**public** ArrayList<ConvexHull> split() {

**if** (**this**.getPeakPixel().size() <= 2)

**return** **null**;

ArrayList<ConvexHull> res = **new** ArrayList<ConvexHull>();

ConvexHull temp = **new** ConvexHull((ArrayList<Point>) **this**.getPeakPixel()

.clone());

**for** (; temp.getPeakPixel().size() > 3;) {

ConvexHull t = *getTriangle*(temp);

res.add(t);

temp.getPeakPixel().remove(t.getPeakPixel().get(1));

}

temp.getArea();

res.add(temp);

**return** res;

}

/\*\*

\* Метод повертає Point, шо є точкою центр мас. Можливі чотири випадки: <br/>

\* 1) ConvexHull містить тільки одну точку. В цьому випадку повертаємо її

\* саму <br/>

\* 2) ConvexHull містить дві точки. В цьому випадку повертаємо середину

\* відрізка, що сполучає ці дві точки <br/>

\* 3) ConvexHull містить три точки. В цьому випадку маємо трикутник, тому

\* повертаємо точку перетину медіан, яка і буде точкою центер мас трикутника <br/>

\* 4) ConvexHull містить більше трьох точок. В цьому випадку розбиваємо наш

\* полігон на трикутники. Центер мас вираховується як середнє арифметичне

\* центра мас всіх трикутників, з урахуванням вагових коефіцієнтів, якими в

\* даному випадку є площі трикутників

\*

\* **@return** Point - точку центер мас

\*/

**public** Point getCenterOfMass() {

**if** (peakPixel.size() == 1)

**return** peakPixel.get(0);

**if** (peakPixel.size() == 2)

**return** Linear.*middlePointOfSegment*(peakPixel.get(0),

peakPixel.get(1));

**if** (peakPixel.size() == 3) {

Point a1 = peakPixel.get(0);

Point b1 = Linear.*middlePointOfSegment*(peakPixel.get(1),

peakPixel.get(2));

Point a2 = peakPixel.get(1);

Point b2 = Linear.*middlePointOfSegment*(peakPixel.get(2),

peakPixel.get(0));

**return** Linear.*segmentIntersection*(a1, b1, a2, b2);

}

ArrayList<ConvexHull> splitting = split();

**double** x\_ = 0;

**double** y\_ = 0;

**for** (ConvexHull c : splitting) {

Point p = c.getCenterOfMass();

**double** a = c.getArea();

x\_ += p.getX() \* a;

y\_ += p.getY() \* a;

}

x\_ /= getArea();

y\_ /= getArea();

**return** **new** Point(x\_, y\_);

}

**public** String toString() {

String res = "[ ";

**for** (Point p : peakPixel) {

res += p.toString() + " ";

}

**return** res + " ]: " + attribute.toString();

}

}

**package** com.lab111.picturetwister.vectorized;

**import** java.util.ArrayList;

/\*\*

\* Клас, який описує векторний піксель, наслідується від класа ConvexHull

\* **@author** Бедь Анатолій Михайлович

\*

\*/

**public** **class** VectorPixel **extends** ConvexHull{

**public** VectorPixel(ArrayList<Point> points) {

**super**(points);

}

/\*\*

\* Метод встановлює колір векторного пікселя

\* **@param** color

\*/

**public** **void** setColor(**int** color){

attribute.put("color", color);

}

/\*\*

\* Метод повертає колір векторного пікселя

\* **@return**

\*/

**public** **int** getColor(){

**return** (**int**) attribute.get("color");

}

/\*\*

\* Метод повертає добуток "ваги" пікселя на його центер мас

\* **@return**

\*/

**public** **double**[] getWeightPixel(){

Point c = getCenterOfMass();

**return** **new** **double**[] {c.getX()\*getColor(), c.getY()\*getColor()};

}

/\*\*

\* Метод повертає точку з заданої позиції

\* **@param** poz позиція точки яку хочемо отримати

\* **@return** точку в позиції poz

\*/

**public** Point getPoint(**int** poz){

**return** peakPixel.get(poz);

}

@Override

**public** String toString() {

**return** "VectorPixel [peakPixel=" + peakPixel.toString() + ", color=" + attribute.get("color") + "]";

}

}

**package** com.lab111.picturetwister.vectorized;

**import** java.awt.Color;

**import** java.awt.Graphics2D;

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

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.lab111.picturetwister.core.Function;

**import** com.lab111.picturetwister.core.Linear;

**import** com.lab111.picturetwister.core.Stat;

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

/\*\*

\* Клас описує сітку точок, зв'язаних між собою

\* **@author** Бедь Анатолій Михайлович

\*/

**public** **class** Reticle {

**private** Point[][] points;

**private** Function weightFunction;

**private** VectorPixel[][] vectorPixels;

**private** MatrixTransform transform;

**public** **int**[][] colorMatrix;

**private** **int** height;

**private** **int** width;

/\*\*

\* Конструктор, який створює сітку

\*

\* **@param** colorsRGB

\* - матриця кольорів

\*/

**public** Reticle(**int**[][] colorsRGB) {

points = *createPointSet*(colorsRGB[0].length, colorsRGB.length);

colorMatrix = colorsRGB;

height = colorsRGB.length;

width = colorsRGB[0].length;

vectorPixels = *createConvexHullsSet*(colorMatrix, points);

transform = **new** MatrixTransform(**this**);

}

/\*\*

\* Метод повертає сітку точок, в якій кожна точка знає про свої сусідні

\* точки зверху, зліва, справа, знизу

\*

\* **@param** width ширина зображення

\* **@param** height висота зображення

\* **@return** Points[][] - сітка точок

\*/

**public** **static** Point[][] createPointSet(**int** width, **int** height) {

Point[][] points = **new** Point[height + 1][width + 1];

/\*

\* Створюємо точку в координатах (0, 0)

\*/

points[0][0] = **new** Point(0, 0);

/\*

\* Заповнення першого рядка матриці зразу вказуємо для даної точки

\* першого рядка її сусідню точку зліва

\*/

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

points[0][j] = **new** Point(0, j);

points[0][j].setPointLeft(points[0][j - 1]);

}

/\*

\* Заповнення першого стовпця матриці зразу вказуємо для даної точки

\* першого стовпця її сусідню точку зверху

\*/

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

points[i][0] = **new** Point(i, 0);

points[i][0].setPointTop(points[i - 1][0]);

}

/\*

\* Заповнення всіх інших елементів матриці Створюємо точку. Для даної

\* точки вказуємо її сусідні точки зверху та зліва для точки яка

\* знаходиться зліва по діагоналі вказуємо сусідні точки знизу і справа.

\* Після виконання циклів, для точок останнього стовпця не буде вказано

\* іхніх сусідніх точок знизу, а для точок останнього рядка - сусідніх

\* точок справа

\*/

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

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

points[i][j] = **new** Point(i, j);

points[i][j].setPointTop(points[i - 1][j]);

points[i][j].setPointLeft(points[i][j - 1]);

points[i - 1][j - 1].setPointRight(points[i - 1][j]);

points[i - 1][j - 1].setPointBottom(points[i][j - 1]);

}

/\*

\* Вказуємо для точок останнього стовпця сусідні їм точок знизу

\*/

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

points[i][width].setPointBottom(points[i + 1][width]);

}

/\*

\* Вказуємо для точок останнього рядка сусідніх їм точки справа

\*/

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

points[height][j].setPointRight(points[height][j + 1]);

}

**return** points;

}

/\*\*

\* Метод повертає матрицю векторних пікселів

\*

\* **@param** intMatrix матриця кольорів

\* **@param** points сітка точок

\* **@return**

\*/

**public** **static** VectorPixel[][] createConvexHullsSet(**int**[][] intMatrix,

Point[][] points) {

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

**int** height = intMatrix.length;

VectorPixel[][] vectorPixels = **new** VectorPixel[height][width];

ArrayList<Point> pointList;

/\*

\* Кладемо в список чотири точки(вершини пікселя) і створюємо векторний

\* піксель, який кладемо в масив.

\*/

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

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

pointList = **new** ArrayList<Point>(4);

pointList.add(points[i][j]);

pointList.add(points[i][j + 1]);

pointList.add(points[i + 1][j + 1]);

pointList.add(points[i + 1][j]);

vectorPixels[i][j] = **new** VectorPixel(pointList);

vectorPixels[i][j].setColor(intMatrix[i][j]);

}

}

**return** vectorPixels;

}

/\*\*

\* метод для маштабування сітки

\*

\* **@param** mx

\* параметр збільшення по осі Ох

\* **@param** my

\* параметр збільшення по осі Оу

\*/

**public** **void** scale(**double** mx, **double** my) {

transform.scale(mx, my);

width = (**int**) (width \* my);

height = (**int**) (height \* mx);

}

/\*\*

\* метод для здійснення повороту сітки на кут

\*

\* **@param** angle

\* кут на який потрібно повернути сітку

\*/

**public** **void** rotate(**double** angle) {

transform.rotate(angle);

}

/\*\*

\* метод, який дозволяє перемістити сітку

\*

\* **@param** transferX

\* відстань на яку треба перемістити сітку по координаті X

\* **@param** transferY

\* відстань на яку треба перемістити сітку по координаті Y

\*/

**public** **void** transfer(**double** transferX, **double** transferY) {

transform.transfer(transferX, transferY);

}

/\*

\* Метод, який ламає сітку.

\*/

**public** **void** deformation() {

VectorPixel[][] hulls = **this**.cloneConvexHulls();

**double** x = 0;

**double** y = 0;

**for** (**int** i = 1; i < vectorPixels.length - 1; i++) {

**for** (**int** j = 1; j < vectorPixels[0].length - 1; j++) {

**if** (Stat.*max*(Stat.*difference*(hulls[i][j].getColor(),

hulls[i - 1][j - 1].getColor()), Stat.*difference*(

hulls[i][j].getColor(), hulls[i - 1][j].getColor()),

Stat.*difference*(hulls[i][j].getColor(),

hulls[i - 1][j + 1].getColor()), Stat

.*difference*(hulls[i][j].getColor(),

hulls[i][j - 1].getColor()), Stat

.*difference*(hulls[i][j].getColor(),

hulls[i][j + 1].getColor()), Stat

.*difference*(hulls[i][j].getColor(),

hulls[i + 1][j - 1].getColor()), Stat

.*difference*(hulls[i][j].getColor(),

hulls[i + 1][j].getColor()), Stat

.*difference*(hulls[i][j].getColor(),

hulls[i + 1][j + 1].getColor())) > 25) {

x = 0;

y = 0;

x = Stat.*sum*(hulls[i][j].getWeightPixel()[0],

hulls[i][j + 1].getWeightPixel()[0],

hulls[i + 1][j].getWeightPixel()[0],

hulls[i + 1][j + 1].getWeightPixel()[0]);

y = Stat.*sum*(hulls[i][j].getWeightPixel()[1],

hulls[i][j + 1].getWeightPixel()[1],

hulls[i + 1][j].getWeightPixel()[1],

hulls[i + 1][j + 1].getWeightPixel()[1]);

**double** mass = Stat.*sum*(hulls[i][j].getColor(),

hulls[i][j + 1].getColor(),

hulls[i + 1][j].getColor(),

hulls[i + 1][j + 1].getColor());

x = x / mass;

y = y / mass;

vectorPixels[i][j].getPoint(2).setX(x);

vectorPixels[i][j].getPoint(2).setY(y);

}

}

}

}

/\*\*

\* Метод для деформації сітки

\*/

**public** **void** deformation1() {

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

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

ArrayList<VectorPixel> connected = getConnectedPixels(points[i][j]);

**double**[] c = **new** **double**[connected.size()];

**for** (**int** k = 0; k < c.length; k++) {

c[k] = connected.get(k).getColor();

}

**if** (Stat.*difference*(Stat.*max*(c), Stat.*min*(c)) > 25) {

**for** (**int** k = 0; k < c.length; k++) {

// connected.get(k).setAttribute("w", 129-Math.abs(new

// Double(connected.get(k).getColor())-128));

**double** w = 129 - Math.*abs*(**new** Double(connected.get(k)

.getColor()) - 128);

w = w \* w;

connected.get(k).setAttribute("w", w);

}

Point res = Linear.*centerOfMassForSys*(connected);

points[i][j].setX(res.getX());

points[i][j].setY(res.getY());

}

}

}

}

/\*\*

\* Метод для отримання сітки зв'язаних точок

\*

\* **@return** повертає сітку зв'язаних точок

\*/

**public** Point[][] getPoints() {

**return** points;

}

/\*\*

\* Метод для отримання векторних пікселів з сітки

\* **@return** vectorPixels

\*/

**public** VectorPixel[][] getConvexHulls() {

**return** vectorPixels;

}

/\*\*

\* метод повертає матрицю координат у вигляді Ax Cx ..... Px Ay Cy ..... Py

\* Bx Dx ..... Lx By Dy ..... Ly Mx Nx ..... Qx My Ny ..... Qy

\* ............... ............... Rx Tx ...... Hx Ry Ty ...... Hy

\*

\* де точки A(Ax,Ay), B(Bx,By), D(Dx,Dy), C(Cx,Cy) є кутами пікселя (і так

\* далі відповідно)

\*

\* **@return**

\*/

**public** **double**[][] getPointsXY() {

**double**[][] vector = **new** **double**[2 \* points.length][points[0].length];

**int** k = 0;

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

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

vector[k][j] = points[i][j].getX();

vector[k + 1][j] = points[i][j].getY();

}

k = k + 2;

}

**return** vector;

}

/\*\*

\* Метод для отримання копії ConvexHull[][]

\* **@return** повертає копію ConvexHull[][], що містить Reticle

\*/

**public** VectorPixel[][] cloneConvexHulls() {

Point[][] points = **this**.clonePoints();

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

**int** height = colorMatrix.length;

VectorPixel[][] vectorPixels = **new** VectorPixel[height][width];

ArrayList<Point> pointList;

/\*

\* Кладемо в список чотири точки(вершини пікселя) і створюємо векторний

\* піксель, який кладемо в масив.

\*/

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

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

pointList = **new** ArrayList<Point>(4);

pointList.add(points[i][j]);

pointList.add(points[i][j + 1]);

pointList.add(points[i + 1][j + 1]);

pointList.add(points[i + 1][j]);

vectorPixels[i][j] = **new** VectorPixel(pointList);

vectorPixels[i][j].setColor(colorMatrix[i][j]);

}

}

**return** vectorPixels;

}

/\*\*

\* Метод для отримання копії сітки зв'язаних між собою точок

\* **@return** повертає копію сітки точок

\*/

**public** Point[][] clonePoints() {

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

**int** height = colorMatrix.length;

Point[][] points = **new** Point[height + 1][width + 1];

/\*

\* Створюємо точку в координатах (0, 0)

\*/

points[0][0] = **new** Point(**this**.points[0][0].getX(),

**this**.points[0][0].getY());

/\*

\* Заповнення першого рядка матриці зразу вказуємо для даної точки

\* першого рядка її сусідню точку зліва

\*/

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

points[0][j] = **new** Point(**this**.points[0][j].getX(),

**this**.points[0][j].getY());

points[0][j].setPointLeft(points[0][j - 1]);

}

/\*

\* Заповнення першого стовпця матриці зразу вказуємо для даної точки

\* першого стовпця її сусідню точку зверху

\*/

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

points[i][0] = **new** Point(**this**.points[i][0].getX(),

**this**.points[i][0].getY());

points[i][0].setPointTop(points[i - 1][0]);

}

/\*

\* Заповнення всіх інших елементів матриці Створюємо точку. Для даної

\* точки вказуємо її сусідні точки зверху та зліва для точки яка

\* знаходиться зліва по діагоналі вказуємо сусідні точки знизу і справа.

\* Після виконання циклів, для точок останнього стовпця не буде вказано

\* іхніх сусідніх точок знизу, а для точок останнього рядка - сусідніх

\* точок справа

\*/

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

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

points[i][j] = **new** Point(**this**.points[i][j].getX(),

**this**.points[i][j].getY());

points[i][j].setPointTop(points[i - 1][j]);

points[i][j].setPointLeft(points[i][j - 1]);

points[i - 1][j - 1].setPointRight(points[i - 1][j]);

points[i - 1][j - 1].setPointBottom(points[i][j - 1]);

}

/\*

\* Вказуємо для точок останнього стовпця сусідні їм точок знизу

\*/

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

points[i][width].setPointBottom(points[i + 1][width]);

}

/\*

\* Вказуємо для точок останнього рядка сусідніх їм точки справа

\*/

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

points[height][j].setPointRight(points[height][j + 1]);

}

**return** points;

}

/\*\*

\* Метод для отримання матриці кольорів

\*

\* **@return** colorповертає матрицю кольорів

\*/

**public** **int**[][] getColorMatrix() {

**return** colorMatrix;

}

/\*\*

\* Мето для отримання висоти сітки

\* **@return** висоту сітки

\*/

**public** **int** getHeight() {

**return** height;

}

/\*\*

\* Мето для отримання ширини сітки

\* **@return** ширину сітки

\*/

**public** **int** getWidth() {

**return** width;

}

/\*\*

\* Метод для отримання вагової фінкції

\* **@return** вагову функцію

\*/

**public** Function getWeightFunction() {

**return** weightFunction;

}

/\*\*

\* Мето для встановлення вагової функції

\* **@param** вагова функція

\*/

**public** **void** setWeightFunction(Function weightFunction) {

**this**.weightFunction = weightFunction;

}

/\*\*

\* Мето для отримання пікселів до яких входить задана точка p

\*

\* **@param** p

\* точка

\* **@return** список пікселів до яких вжодить задана точка

\*/

**public** ArrayList<VectorPixel> getConnectedPixels(Point p) {

ArrayList<VectorPixel> res = **new** ArrayList<VectorPixel>();

**int** i\_ = -1;

**int** j\_ = -1;

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

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

**if** (p == points[i][j]) {

i\_ = i;

j\_ = j;

**break**;

}

}

**if** (i\_ >= 0)

**break**;

}

**if** (i\_ < 0)

**return** res;

**if** (((i\_ - 1) >= 0) && (j\_ - 1) >= 0)

res.add(vectorPixels[i\_ - 1][j\_ - 1]);

**if** (((i\_) < vectorPixels.length) && (j\_ - 1) >= 0)

res.add(vectorPixels[i\_][j\_ - 1]);

**if** (((i\_ - 1) >= 0) && (j\_) < vectorPixels[0].length)

res.add(vectorPixels[i\_ - 1][j\_]);

**if** (((i\_) < vectorPixels.length) && (j\_) < vectorPixels[0].length)

res.add(vectorPixels[i\_][j\_]);

**return** res;

}

/\*\*

\* Метод для виділення контурних ланцюгів

\* **@param** porog параметр

\* **@return** контурний ланцюг

\*/

**public** ConvexHull searchCounture(**int** porog) {

List<ConvexHull> listCounture = **new** ArrayList<ConvexHull>();

ConvexHull hull = **new** ConvexHull();

**int** color1;

**int** color2;

**for** (**int** i = 1; i < points.length - 2; i++) {

**for** (**int** j = 1; j < points[i].length - 2; j++) {

color1 = 0x0000ff & vectorPixels[i][j].getColor();

color2 = vectorPixels[i - 1][j].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(1)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(1));

}

color2 = vectorPixels[i - 1][j - 1].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(0)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(0));

}

color2 = vectorPixels[i][j - 1].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(0)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(0));

}

color2 = vectorPixels[i + 1][j].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(3)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(3));

}

color2 = vectorPixels[i + 1][j].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(3)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(3));

}

color2 = vectorPixels[i + 1][j + 1].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(2)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(2));

}

color2 = vectorPixels[i][j + 1].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(2)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(2));

}

color2 = vectorPixels[i - 1][j + 1].getColor() & 0x0000ff;

**if** (Stat.*difference*(color2, color1) > porog) {

**if** (!hull.peakPixel

.contains(vectorPixels[i][j].getPoint(1)))

hull.peakPixel.add(vectorPixels[i][j].getPoint(1));

}

}

}

listCounture.add(hull);

**return** hull;

}

}

**package** com.lab111.picturetwister.vectorized;

/\*\*

\* Клас що описує матрицю перетоврень

\*

\* **@author** Бедь Анатолій Михайлович

\*

\*/

**public** **class** MatrixTransform {

**private** **double**[][] matrix;

**private** Reticle reticle;

**public** MatrixTransform(Reticle reticle) {

**this**.reticle = reticle;

matrix = **new** **double**[][] { { 1, 0, 0 },

{ 0, 1, 0 },

{ 0, 0, 1 } };

}

/\*\*

\* Метод для маштабування сітки побудованої з точок Point

\* **@param** mx коефіцієнт маштабування по осі X

\* **@param** my коефіцієнт маштабування по осі Y

\* **@return** двовимірний масив точок промаштабованої Reticle

\*/

**public** Point[][] scale(**double** mx, **double** my) {

setScale(mx, my);

Point[][] points = reticle.getPoints();

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

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

points[i][j].setX(points[i][j].getX() \* matrix[0][0]);

points[i][j].setY(points[i][j].getY() \* matrix[1][1]);

}

}

**return** points;

}

/\*\*

\* Метод для встановлення коефіцієнтів маштабування

\* **@param** mx коефіцієнт маштабування по осі X

\* **@param** my коефіцієнт маштабування по осі Y

\*/

**public** **void** setScale(**double** mx, **double** my) {

matrix[0][0] = mx;

matrix[1][1] = my;

}

/\*\*

\* Метод для встановлення кута повороту сітки

\* **@param** angle кут повороту заданий в градусах.

\*/

**public** **void** setRotate(**double** angle) {

angle = angle \* Math.*PI* / 180;

matrix[0][0] = Math.*cos*(angle);

matrix[1][1] = Math.*cos*(angle);

matrix[0][1] = Math.*sin*(angle);

matrix[1][0] = -Math.*sin*(angle);

matrix[2][2] = 1;

}

/\*\*

\* Метод для повороту сітки

\* **@param** angle кут повороту заданий в градусах.

\* **@return** двовимірний масив точок повернутої на кут angle Reticle

\*/

**public** Point[][] rotate(**double** angle) {

setRotate(angle);

Point[][] points = reticle.getPoints();

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

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

points[i][j].setX(points[i][j].getX() \* matrix[0][0]

+ points[i][j].getY() \* matrix[0][1]);

points[i][j].setY(points[i][j].getX() \* matrix[1][0]

+ points[i][j].getY() \* matrix[1][1]);

}

}

**return** points;

}

/\*\*

\* Метод для переміщення сітки

\*

\* **@param** transferX відстань для переносу по осі X задана в пікселях

\* **@param** transferY відстань для переносу по осі Y задана в пікселях

\* **@return** двовимірний масив точок переміщеної Reticle

\*/

**public** Point[][] transfer(**double** transferX, **double** transferY) {

setTransfer(transferX, transferY);

Point[][] points = reticle.getPoints();

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

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

points[i][j].setX(points[i][j].getX() + matrix[0][2]);

points[i][j].setY(points[i][j].getY() + matrix[1][2]);

}

}

**return** points;

}

/\*\*

\* Метод для встановлення значень переміщення сітки

\* **@param** transferX відстань для переносу по осі X задана в пікселях

\* **@param** transferY відстань для переносу по осі Y задана в пікселях

\*/

**public** **void** setTransfer(**double** transferX, **double** transferY) {

matrix[0][2] = transferX;

matrix[1][2] = transferY;

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

matrix[i][i] = 1;

}

}

}

**package** com.lab111.picturetwister.core;

/\*\*

\* Даний інтерфейс описує функцію

\* **@author**

\*

\*/

**public** **interface** Function {

**public** **double** getValue(**double**... args);

}

**package** com.lab111.picturetwister.core;

/\*\*

\* Клас з статичними методами порівняння двох величин

\* **@author**

\*

\*/

**public** **class** Inaccurate {

**public** **static** **double** *ACCURACY* = 0.000001;

**public** Inaccurate() {

// **TODO** Auto-generated constructor stub

}

/\*\*

\* Перевіряє два числа на рівність з точністю 0.000001

\* **@param** a перше число

\* **@param** b друге число

\* **@return** true, якщо a рівне b і false, якщо a не рівне b

\*/

**public** **static** **boolean** equals(**double** a, **double** b) {

**return** Math.*abs*(a - b) <= *ACCURACY*;

}

/\*\*

\* Перевіряє чи a < b з точністю 0.000001

\* **@param** a перше число

\* **@param** b друге число

\* **@return** true, якщо a мешне b і false, якщо a більше або рівне b

\*/

**public** **static** **boolean** less(**double** a, **double** b) {

**return** (b - a) > *ACCURACY*;

}

/\*\*

\* Перевіряє чи a > b з точністю 0.000001

\* **@param** a перше число

\* **@param** b друге число

\* **@return** true, якщо a більше b і false, якщо a менше або рівне b

\*/

**public** **static** **boolean** more(**double** a, **double** b) {

**return** (a - b) > *ACCURACY*;

}

/\*\*

\* Перевіряє чи a <= b з точністю 0.000001

\* **@param** a перше число

\* **@param** b друге число

\* **@return** true, якщо a менше або рівне b і false, якщо a більше b

\*/

**public** **static** **boolean** lessOrEquals(**double** a, **double** b) {

**return** *less*(a, b) || *equals*(a, b);

}

/\*\*

\* Перевіряє чи a >= b з точністю 0.000001

\* **@param** a перше число

\* **@param** b друге число

\* **@return** true, якщо a більше або рівне b і false, якщо a менше b

\*/

**public** **static** **boolean** moreOrEquals(**double** a, **double** b) {

**return** *more*(a, b) || *equals*(a, b);

}

}

**package** com.lab111.picturetwister.core;

**import** java.util.ArrayList;

**import** org.apache.commons.math3.geometry.euclidean.twod.Line;

**import** org.apache.commons.math3.geometry.euclidean.twod.Vector2D;

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

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

/\*\*

\* Клас який містить статичні методи для роботи з геометрією

\* **@author**

\*

\*/

**public** **class** Linear {

/\*\*

\* Метод створю екземпляр класа Vector2D, який описує вектор

\* **@param** a точка початку вектора

\* **@param** b точка кінця вектора

\* **@return** 2D векто (екземпляр класа Vector2D)

\*/

**public** **static** Vector2D createVector(Point a,Point b){

**return** **new** Vector2D(b.getX()-a.getX(), b.getY()-a.getY());

}

/\*\*

\* Метод для аналізу взаємного розташування двох векторів

\* **@param** first перший вектор

\* **@param** second другий вектор

\* **@return** повертає додатнє число, якщо перший вектор випереджає другий,

\* і відємне коли другий випереджає перший,

\* якщо вектора дивляться в одному напрямку то повертає 0

\*

\*/

**public** **static** **double** product(Vector2D first,Vector2D second){

**return** -first.getX()\*second.getY()+second.getX()\*first.getY();

}

/\*\*

\* Метод знахотидь середину відрізка

\* **@param** p1 перша точка

\* **@param** p2 друга точка

\* **@return** Point - точку, яка є серединою відрізка що з'єднує точки p1 і p2

\*/

**public** **static** Point middlePointOfSegment(Point p1, Point p2){

**return** **new** Point(Stat.*average*(p1.getX(),p2.getX()), Stat.*average*(p1.getY(),p2.getY()));

}

/\*\*

\* Метод перетворює вектор в точку

\* **@param** p точка

\* **@return** вектор, початок якого точка (0, 0), а кінець точка p; повертає null, якщо p null

\*

\*/

**public** **static** Vector2D toVector (Point p){

**return** (p == **null**) ? **null** : **new** Vector2D(p.getX(),p.getY());

}

/\*\*

\* Метод перетворє вектор з координатами (x, y) в точку з координатами (x, y)

\* **@param** v вектор з координатами (x, y)

\* **@return** точка з координатами (x, y), де x,y - кординати вектора v

\*/

**public** **static** Point toPoint (Vector2D v){

**return** (v == **null**) ? **null** : **new** Point(v.getX(),v.getY());

}

/\*\*

\* Метод повертає екземпляр класа Line, який описує лінію

\* **@param** p1 перша точка

\* **@param** p2 друга точка

\* **@return** лінію, яка проходить чере точки p1, p2

\*/

**public** **static** Line lineModel(Point p1, Point p2){

**return** **new** Line(*toVector*(p1), *toVector*(p2));

}

/\*\*

\* Метод повертає екземпляр класа Point - точку перетину двох ліній

\* **@param** l1 перша лінія

\* **@param** l2 друга лінія

\* **@return** точка перетину ліній l1 і l2, повертає null якщо лінії не перетинаються

\*/

**public** **static** Point lineIntersection(Line l1, Line l2){

**return** *toPoint*(l1.intersection(l2));

}

/\*\*

\* Метод знаходить точку перетину двох відрізків a1b1 i a2b2

\* **@param** a1 перша точка

\* **@param** b1 друга точка

\* **@param** a2 третя точка

\* **@param** b2 четверта точка

\* **@return** точку перетину двох відрізків, або null якщо відрізки не перетинеаються

\*/

**public** **static** Point segmentIntersection(Point a1, Point b1, Point a2, Point b2){

Line l1 = *lineModel*(a1, b1);

Line l2 = *lineModel*(a2, b2);

Point p = *lineIntersection*(l1, l2);

**if** (p == **null**) **return** **null**;

**if** ( !p.inRect(a1, b1) || !p.inRect(a2, b2)) **return** **null**;

**return** p;

}

**public** **static** Point centerOfMassForPlanarObj(ConvexHull object){

// **TODO** сделать centerOfMassForPlanarObj

**return** **null**;

}

/\*\*

\* Метод знаходить точку центер мас системи з оболочок ConvexHull або його наслідників.<br/>

\* **@param** objects лінійний список з ConvexHull або його наслідників

\* **@return** точку центер мас

\*/

**public** **static** Point centerOfMassForSys(ArrayList<? **extends** ConvexHull> objects){

**double** x\_ = 0;

**double** y\_ = 0;

**double** w\_ = 0;

**for**(ConvexHull c :objects){

Point p = c.getCenterOfMass();

**double** a = c.getAttribute("w");

x\_ += p.getX()\*a;

y\_ += p.getY()\*a;

w\_ += a;

}

x\_ /= w\_;

y\_ /= w\_;

**return** **new** Point(x\_,y\_);

}

**public** **static** Point perpendicularLine(Line line, Point point){

**return** **null**;

}

}

**package** com.lab111.picturetwister.core;

**import** org.apache.commons.math3.stat.StatUtils;

/\*\*

\* Клас з статичними методами математичної статистики

\* **@author**

\*

\*/

**public** **class** Stat {

**public** **static** **double** *ACCURACY* = 0.00000000000001;

**public** Stat() {

}

/\*\*

\* Метод повертає суму елементів

\* **@param** values вхідний масив

\* **@return** суму елементів, або 0 якщо не знадано жодного параметра

\*/

**public** **static** **double** sum(**double**... values) {

**double** result = 0;

**for** (**double** value : values)

result += value;

**return** result;

}

/\*\*

\* Метод шукає максимальний елемент

\* **@param** values вхідний масив

\* **@return** максимальний з елементів, або 0 якщо не знадано жодного параметра

\*/

**public** **static** **double** max(**double**... values) {

**double** result = values[0];

**for** (**double** value : values)

result = (result < value) ? value : result;

**return** result;

}

/\*\*

\* Метод шукає мінімальний елемент

\* **@param** values вхідний масив

\* **@return** мінімальний з елементів, або 0 якщо не знадано жодного параметра

\*/

**public** **static** **double** min(**double**... values) {

**double** result = values[0];

**for** (**double** value : values)

result = (result > value) ? value : result;

**return** result;

}

/\*\*

\* Метод шукає індекс максимального елемента

\* **@param** values вхідний масив

\* **@return** індекс максимального елемента, або 0 якщо не знадано жодного параметра

\*/

**public** **static** **int** maxIndex(**double**... values) {

**double** max = values[0];

**int** res = 0;

**for** (**int** i = 1; i < values.length; i++) {

**if** (max < values[i]) {

max = values[i];

res = i;

}

}

**return** res;

}

/\*\*

\* Метод шукає індекс мінімального елемента

\* **@param** values вхідний масив

\* **@return** індекс мінімального елемента, або 0 якщо не знадано жодного параметра

\*/

**public** **static** **int** minIndex(**double**... values) {

**double** min = values[0];

**int** res = 0;

**for** (**int** i = 1; i < values.length; i++) {

**if** (min > values[i]) {

min = values[i];

res = i;

}

}

**return** res;

}

/\*\*

\* Метод рахує модуль різниці двох чисел

\* **@param** a перше число

\* **@param** b друге число

\* **@return** модуль різниці a і b

\*/

**public** **static** **double** difference(**double** a, **double** b) {

**return** Math.*abs*(a - b);

}

/\*\*

\* Метод шукає середнє арифметичне

\* **@param** values вхідний масив

\* **@return** середнє арифметичне елементів у вхідному масиві, або Double.NaN, якщо масив порожній.

\*/

**public** **static** **double** average(**double**... values) {

**return** StatUtils.*mean*(values);

}

/\*\*

\* Метод шукає середньоквадратичне відхилення

\* **@param** values вхідний масив

\* **@return** середньоквадратичне відхилення

\*/

**public** **static** **double** stdDeviance(**double**... values) {

**return** Math.*sqrt*(StatUtils.*variance*(values));

}}

ДОДАТОК Б. СТРУКТУРА ПРОЕКТУ

