Увод

1. SVG е съкращение от Scalable Vector Graphics. Файловете от този тип са предназначени за обработка на графика с промяна в растерния или векторния режим. Файлът с разширение на файла SVG е най-вероятно файл с мащабируема векторна графика. Файловете в този формат използват текстов формат, базиран на XML, за да опишат как изображението трябва да се появи.
2. Основата цел и задачи са в рамките на този проект да се разработи приложение, което работи със файлове във Scalable Vector Graphics (SVG) формат. Приложението трябва да може да зарежда фигури от файла, да извършва върху тях дадени операции, след което да може да записва промените обратно във файла(като може да се правят промени в реално време в самия файл).
3. Структура на документацията
4. Проектиране
5. Обща архитектура – ООП дизайн (какво представлява)
6. Диаграма (на структура и поведение - по обекти, слоеве с най-важните извадки от кода)
7. Реализация, тестване
8. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода)
9. Как работи програмата (тестови сценарии)
10. Заключение

* Линк към репозиторито в Github:

<https://github.com/ani4ka656/Project-SVG-Files>

1. Проектиране
2. Обща архитектура – ООП дизайн (какво представлява) :

За улеснение, в рамките на проекта ще работим само с основните фигури в SVG - линия, кръг и правоъгълник. За всяка фигура има отделен клас, който съдържа характеристиките им и съответно методите който наследяват от клас Shape. Следователно имаме :

class Line, class Rectangle, class Circle, наследяващи class Shape.

Когато потребителя пусне програмата, той трябва да въведе една от следните команди: open, close, save, save as, help, exit(описани са подробно в частта за работа с командния ред във файла за проектите <https://docs.google.com/document/d/1quesENVOm28Ue37vGhU2oB4d-dsUG0VX1mCELxx6LN4/edit#heading=h.y82tgqy2q5sg>), и тези, които се отнасят за самите SVG файлове

|  |  |
| --- | --- |
| print | Извежда на екрана всички фигури. |
| create | Създава нова фигура. |
| erase <n> | Изтрива фигура с пореден номер <n>. |
| translate [<n>] | Транслира фигурата с пореден номер <n> или всички фигури, ако <n> не е указано. |
| within <option> ... | Извежда на екрана всички фигури, които изцяло се съдържат в даден регион. Потребителят може да укаже чрез <option> какъв да бъде регионът – кръг (circle) или правоъгълник (rectangle) |

За да могат да се осъществят тези операции е създаден class FileМanager, който се грижи за изпълнението им.

Ако файлът въведен от потребителя съществува, фигурите се зареждат от

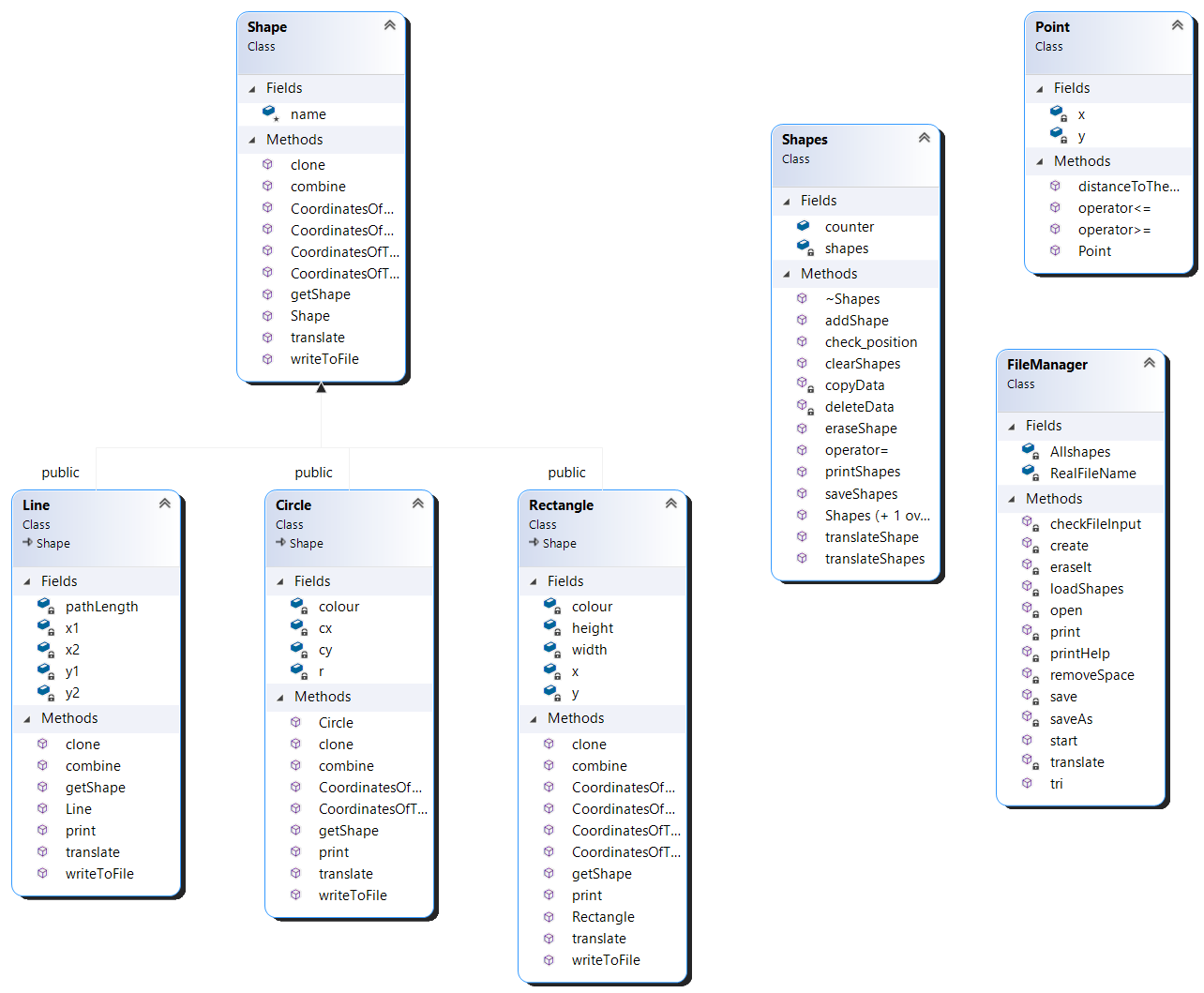
него, а ако не, се създава нов празен файл с това име.

След като се заредят фигурите от въведения от потребителя файл, той трябва да може да изпълнява всички команди, които добавят, изтриват или променят фигурите.

А ако е създаден от потребителя той първо трябва да създаде поне една фигура.

Имаме клас Shapes, който се грижи за създаването на динамичен масив(вектор) от фигурите, който се грижи за изпълнението на командите относно самите фигури.

1. Диаграма



1. Реализация
2. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода)

* class Shape:

Data Members (protected):

std::string name;

Member Functions (public):

Shape(const std::string& name) : name(name){};

virtual std::string getShape() const {return 0;} //virtual function, which when inherited returns the name of the shape

virtual void writeToFile(std::ostream&) = 0; //virtual function, which

when inherited will saves the info of the shape in the file

virtual std::string combine() const {return 0;} //virtual function,

which when inherited returns the info of the shape as a string

virtual void translate(const int, const int) = 0; //virtual function, which when inherited translates the shape

virtual Shape\* clone() const = 0;

virtual Point CoordinatesOfBottomLeftPoint() const { return 0; }

virtual Point CoordinatesOfBottomRightPoint() const { return 0; }

virtual Point CoordinatesOfTopLeftPoint() const { return 0; }

virtual Point CoordinatesOfTopRightPoint() const { return 0; }

* class Line:

Data Members (private):

int x1;

int x2;

int y1;

int y2;

int pathLength;

Member Functions (public):

Line(int = 0, int = 0, int = 0, int = 0, const std::string& = "line");

void print() const;

virtual std::string getShape() const override; //returns the name of

the shape

virtual std::string combine() const override; //returns the info of the

shape as a string

virtual void writeToFile(std::ostream&) override; //saves the info of

the shape in the file

virtual void translate(const int, const int) override; // translates

the shape

virtual Shape\* clone() const override;

* class Rectangle:

Data Members (private):

int x;

int y;

int width;

int height;

std::string colour;

Member Functions (public):

Rectangle(int = 0, int = 0, int = 0, int = 0, std::string = "", const std::string& = "rectangle");

void print() const;

virtual std::string getShape() const override; //returns the name of the shape

virtual std::string combine() const override; //returns the info of the shape as a string

virtual void writeToFile(std::ostream&) override; //saves the info of the shape in the file

virtual void translate(const int, const int) override; // translates the shape

virtual Shape\* clone() const override;

virtual Point CoordinatesOfBottomLeftPoint() const override;

virtual Point CoordinatesOfBottomRightPoint() const override;

virtual Point CoordinatesOfTopLeftPoint() const override;

virtual Point CoordinatesOfTopRightPoint() const override;

* class Circle:

Data Members (private):

int cx;

int cy;

int r;

std::string colour;

Member Functions (public):

Circle(int = 0, int = 0, int = 0, std::string = "", const std::string& = "circle");

void print() const;

virtual std::string getShape() const override; //returns the name of the shape

virtual std::string combine() const override; //returns the info of the shape as a string

virtual void writeToFile(std::ostream&) override; //saves the info of the shape in the file

virtual void translate(const int, const int) override; // translates the shape

virtual Shape\* clone() const override;

virtual Point CoordinatesOfBottomLeftPoint() const override;

virtual Point CoordinatesOfTopRightPoint() const override;

* class Shapes:

Data Members (private):

std::vector<Shape\*> shapes;

Member Functions (private):

void copyData(const Shapes& other); //copies data from other object of type vector<Shape\*>

void deleteData(); //deletes the data in of shapes

Member Functions (public):

Shapes(){};

Shapes(const Shapes& other);

Shapes& operator=(const Shapes& other);

~Shapes();

bool check\_position(const int, const int) const; //returns 1 the position of the element is within the scope of AllShapes

void addShape(Shape\*);

void eraseShape(const int&);

void printShapes() const;

void translateShapes(const int&, const int&) const;

void saveShapes(std::ofstream&) const;

void translateShape(const int&, const int&, const int&) const;

void clearShapes();

* class Point:

Data Members (private):

int x;

int y;

Member Functions (public):

Point(int x = 0, int y = 0)

bool operator<=(const Point& other);

bool operator>=(const Point& other);

double distanceToTheOther(const Point& other);

* class FileManager

Data Members (private):

std::string RealFileName;

Shapes Allshapes;

Member Functions (private):

void loadShapes(std::istream&);

void create(std::string\*); //creates a desired shape

void open(const std::string&); //opens the file that the user has put and if there is such file it opens it, if there is no such file it creates a new file with the name the user wanted

void eraseIt(int); //erases the figure the user wanted to erase

void saveAs(const std::string&); //saves the figures into the new file name

void save(); //saves the figures to the file

void print() cons; //prints all the elements of Allshapes

void translate(std::string\*); //translates the shapes

void printHelp() const; //prints help

void removeSpace(const std::string&, std::string\*); //it transforms the input of the user and transorms it to an array of strings for easier usage in the process

Member Functions (public):

void start(); //while the user writes commands it

1. Как работи програмата (тестови сценарии)

* > open figures.svg Successfully opened figures.svg 1. line 11 7 8 8 3 2. line 7 6 8 5 1 3. line 7 6 8 5 1 4. line 5 8 4 1 7 5. line 5 2 8 7 5 6. line 8 7 4 5 4 7. rectangle 8 7 4 5 blue

> create circle 2 5 8 red 1. line 11 7 8 8 3 2. line 7 6 8 5 1 3. line 7 6 8 5 1 4. line 5 8 4 1 7 5. line 5 2 8 7 5 6. line 8 7 4 5 4 7. rectangle 8 7 4 5 blue 8. circle 2 5 8 red Successfully created circle (8)

> erase 2 Erased a line (2)

> erase 4 Erased a line (4) > print

1. line 11 7 8 8 3

2. line 7 6 8 5 1

3. line 5 8 4 1 7

4. line 8 7 4 5 4 5. rectangle 8 7 4 5 blue 6. circle 2 5 8 red

> translate vertical=10 horizontal=10 6 Translated 6

> print 1. line 11 7 8 8 3 2. line 7 6 8 5 1 3. line 5 8 4 1 7

4. line 8 7 4 5 4 5. rectangle 8 7 4 5 blue 6. circle 12 15 8 red

> translate vertical=10 horizontal=5 Translated all figures

> print 1. line 16 17 13 18 3 2. line 12 16 13 15 1 3. line 10 18 9 11 7 4. line 13 17 9 15 4 5. rectangle 13 17 4 5 blue 6. circle 17 25 8 red

> save Successfully saved figures.svg

> saveas "ffff.svg" Successfully saved ffff.svg

> close Closed current file

> exit Exit

1. Заключение

Проектът е разработен така, че за в бъдеще могат да се добавят нови фигури, с цел по-голяма завършеност на заданието.