



Обектно – ориентирано програмиране
Курсов проект на тема:
„Растерна графика“

Изготвила:	Весела Илиянова Стоянова
Ф.Н.	71949
I курс	Бакалавър
Специалност:	Информационни системи
Ръководител:	Калин Георгиев

07.06.2020 г.

Съдържание:

1. Увод:	4
1.1. Описание и идея на проекта:	4
1.2. Цел и задачи на разработката:	4
1.3. Структура на документацията:	4
2. Преглед на предметната област:	4
2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани:	4
2.2. Дефиниране на проблеми и сложност на поставената задача:	5
2.3. Подходи , методи (евентуално модели и стандарти) за решаване на поставените проблеми:	5
2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми) и качествени (нефункционални) изисквания (скалируемост, поддръжка):	5
3. Проектиране:	6
3.1. Обща архитектура	6
3.2. Диаграми(на структура и поведение – по обекти, слоеве с най-важните извадки от кода):	8
4. Реализация, тестване:	10
4.1. Реализация на класове (включва важни моменти от реализацията на класовете и малки фрагменти от кода):	10
4.2. Планиране, описание и създаване на тестови	15
5. Заключение:	16
5.1. Обобщение на изпълнението на началните цели:	16
5.2. Насоки за бъдещо развитие и усъвършенстване:	16
Използвана литература:	16
Връзка към хранилище в <i>Github</i>:	16

Таблицы и графики:

Фиг.1: Снимка на екрана при стартиране на програмата	5
Фиг.2: Снимка на екрана при въвеждане на различни команди	6
Фиг.3: Общата архитектура на проекта	7
Фиг.4: Основният и производните му класове	7
Фиг.5: Основни трансформации	8
Фиг.6: Функцията <i>grayscale</i> за клас <i>PPMImage</i>	8
Фиг.7: Функцията <i>monochrome</i> за класа <i>PPMImage</i>	9
Фиг.8: Функцията <i>negative</i> за класа <i>PPMImage</i>	9
Фиг.9: Функцията <i>rotate<left></i> за класа <i>Image</i>	10
Фиг.10: <i>Header</i> и <i>Source</i> файловете на проекта	11
Фиг.11: Член-данни и методи на клас <i>Matrix</i>	12
Фиг.12: Член-данни и методи на класа <i>Image</i>	12
Фиг.13: Член-данни и методи на клас <i>PBMImage</i>	13
Фиг.14: Член-данни и методи на класа <i>PGMImage</i>	13
Фиг.15: Член-данни и методи на класа <i>PPMImage</i>	13

Фиг.16: Член-данните и методите на класа <i>Pixel</i>	14
Фиг.17: Член-данните и методите на класа <i>Session</i>	14
Фиг.18: Трансформации върху <i>PBM</i> изображения.....	15
Фиг.19: Трансформации върху <i>PGM</i> изображения.....	15
Фиг.20: Трансформации върху <i>PPM</i> изображения	16

1. Увод:

1.1. Описание и идея на проекта:

В рамките на този проект трябва да се разработи приложение, което представлява конзолен редактор на растерни изображения. Той трябва да поддържа различни файлове, стартиране на сесия, прилагане на различни трансформации върху изображенията и записване на резултатат.

1.2. Цел и задачи на разработката:

Основната цел е програмата да бъде достатъчно лесна и удобна за използване. За целта в проекта е реализиран *CommandsExecutor*, който съдържа метод *showStartMenu()*. В *main()* функцията на програмата ще можем да стартираме програмата само с извикването на този метод. Програмата ще изисква от потребителя да въведе някоя от следните функции: *load*, *help* или *exit*, с помощта на която да определи коя функция да бъде изпълнена. При въвеждането на невалидни данни (данни, различни от *load*, *help*, *exit*) на стандартния изход ще бъдат изведени отново възможните функции. Всички промени, които се правят, трябва да бъдат записвани във файл. Когато бъде записан нов файл, изпълнението на програмата продължава и се извежда съобщение за успешното записване на данните във файл. Програмата завършва, когато бъде въведена функцията *exit*.

1.3. Структура на документацията:

Структурата на документацията се състои в няколко опорни точки. Първоначално се представят належащите проблеми, както и търсеното на достатъчно ефективни решения. След това се преминава към проектиране на проекта и накрая се преминава към реализация и тестване на проекта. Накрая се представят насоки за бъдещо развитие и усъвършенстване.

2. Преглед на предметната област:

2.1. Основни дефиниции, концепции и алгоритми, които ще бъдат използвани:

- Наследяване – начин за създаване на нови класове чрез използване на компоненти и поведения на съществуващи класове.
- Файловите формати *PBM*, *PGM* и *PPM* са производни на класа *Image*.
- Проектът работи с *ASCII* кодовете на различните файлови формати – P1, P2, P3.
- Някои от функциите, дефинирани в *CommandsExecutor* са разделени на две части в зависимост от това кога се извикват. Такива функции са *help()* и *showMenu()*. Функцията *help* е разделена на *showHelp()* и *showAdvancedHelp()* като *showHelp()* се извиква, ако при стартиране на програмата, потребителят иска да види кратка информация за командите, а *showAdvancedHelp()* се извиква, когато вече е зареден един файл и потребителят иска да провери информация за командите. Функцията *showMenu()* е разделена на *showStartMenu()* и *showAdvancedMenu()*, като *showMenu()* се извиква за първоначалното меню, а *showAdvancedMenu()* се извиква, когато вече е зареден файл.

- Форматът PBM представлява изображения с черно-бели пиксели. Форматът PGM представлява изображения с черно-бели и сиви пиксели. Форматът PPM представлява цветни изображения. За повече информация за файловите формати: <https://en.wikipedia.org/wiki/Netpbm>.

2.2. Дефиниране на проблеми и сложност на поставената задача:

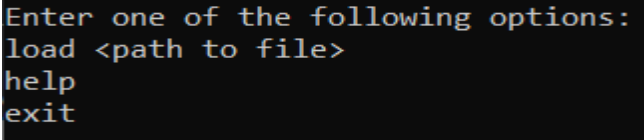
Има множество проблеми, които трябва да бъдат преодоляни с цел успешната реализация на проекта. Те включват гарантирането на удобен и лесен за използване интерфейс. Друг проблем е, че въведените от потребителя данни, трябва да бъдат валидирани. Друг проблем е свързан с функционалността на проекта. Файловият формат на всяко изображение трябва да бъде прочетено и командите да бъдат приложени правилно.

2.3. Подходи , методи (евентуално модели и стандарти) за решаване на поставените проблеми:

Начините за разрешаване на поставените проблеми включват използване на различни методи. Всеки метод ще се грижи за коректността на изпълнението, както и изхода на програмата.

2.4. Потребителски (функционални) изисквания (права, роли, статуси, диаграми) и качествени (нефункционални) изисквания (скалируемост, поддръжка):

Функционалността на проекта започва от *CommandsExecutor*, където се въвежда команда от потребителя, тя се обработва, разпознава коя е командата и я изпълнява. При отваряне на проекта, потребителят има 3 възможности – да отвори съществуващ файл, да види кратка информация за всяка функция и да затвори програмата. Тези функции са показани на фигура 1.



```
Enter one of the following options:
load <path to file>
help
exit
```

Фиг.1: Снимка на екрана при стартиране на програмата

След като потребителят вече е отворил файл, то той има следните възможности:

- да затвори изображението(*close*);
- да запише всички заредени в текущата сесия изображения(*save*);
- да запише под ново име само изображението, което е заредено първо(*save as*);
- да види кратка информация за поддържаните от програмата команди(*help*);
- да излезне от програмата(*exit*);
- да трансформира изображението до черно-бяло с нюанси на сивото(*grayscale*);
- да трансформира изображението до черно-бяло без никакви нюанси на сивото(*monochrome*);

- да трансформира изображението до негативно – цветово обръщане(*negative*);
- да завърти изображението на 90 градуса наляво или надясно(*rotate<direction>*);
- да премахне последно направената трансформация в текущата сесия(*undo*);
- да добави изображение към текущата сесия(*add<image>*);
- да получи подробна информация за текущата потребителска сесия(*session info*);
- да превключи към друга сесия(*switch<session>*).

Тези функции са представени на фигура 2.

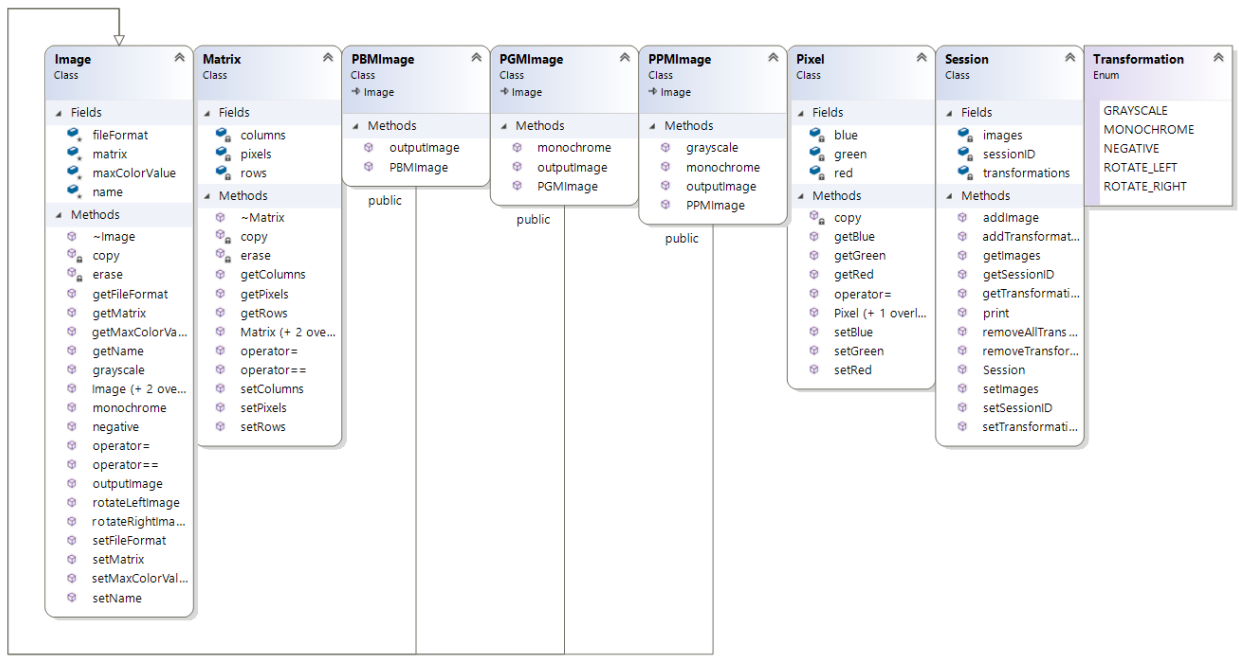
```
Enter one of the following options:
close
save
save as
help
exit
grayscale
monochrome
negative
rotate <direction>
undo
add <image>
session info
switch <session>
collage <direction> <image1> <image2> <outimage>
```

Фиг.2: Снимка на екрана при въвеждане на различни команди

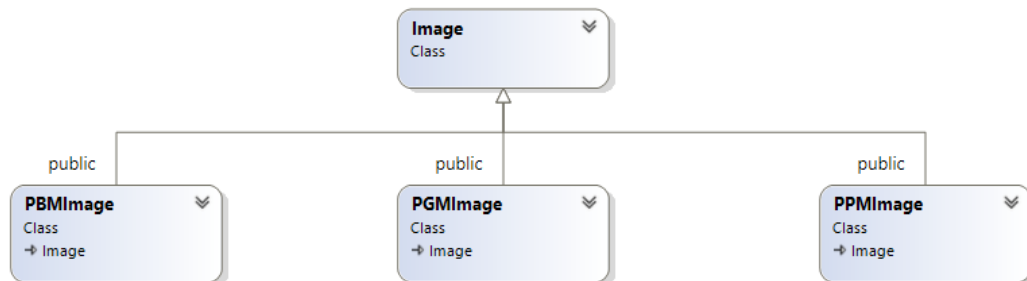
3. Проектиране:

3.1. Обща архитектура

Общата архитектура е изобразена на фигури 3 и 4. Тя е съставена от няколко различни класа.



Фиг.3: Общата архитектура на проекта



Фиг.4: Основният и производните му класове

Обектно – ориентираният дизайн се състои в реализацията на файла *CommandsExecutor*, където се намират основните операции за реализацията на този проект. Това са операциите *grayscale*, *monochrome*, *negative*, *rotate<direction>*, *undo*, *add<image>*, *session info*, *switch<session>*, в допълнение на общите операции *load*, *close*, *save*, *save as*, *help*, *exit*.

3.2. Диаграми(на структура и поведение – по обекти, слоеве с най-важните извадки от кода):

На фигура 5 са показани основните трансформации и файловете, на които те могат да бъдат приложени:

	GRAYSCALE	MONOCHROME	NEGATIVE	ROTATE LEFT	ROTATE RIGHT
PBM file	НЕ	НЕ	ДА	ДА	ДА
PGM file	НЕ	ДА	ДА	ДА	ДА
PPM file	ДА	ДА	ДА	ДА	ДА

Фиг.5: Основни трансформации

Най-важните части от програмата са свързани с различните видове трансформации. Всичките функции са представени за класа *PPMImage*. За останалите файлови формати е аналогично. Реализация на най-важните функции:

- *Grayscale* - фигура 6

```
15 void PPMImage::grayscale()
16 {
17     for (int i = 0; i < matrix->getRows(); i++)
18     {
19         for (int j = 0; j < matrix->getColumns(); j++)
20         {
21             Pixel* pixel = &matrix->getPixels()[i][j];
22             int r = pixel->getRed();
23             int g = pixel->getGreen();
24             int b = pixel->getBlue();
25
26             if (!(r == g && r == b))
27             {
28                 int grayscaleColor = 0.30 * r + 0.59 * g + 0.11 * b;
29                 pixel->setRed(grayscaleColor);
30                 pixel->setGreen(grayscaleColor);
31                 pixel->setBlue(grayscaleColor);
32             }
33         }
34     }
35 }
```

Фиг.6: Функцията *grayscale* за клас *PPMImage*

- *Monochrome* – фигура 7

```
63 void PPMImage::monochrome()
64 {
65     for (int i = 0; i < matrix->getRows(); i++)
66     {
67         for (int j = 0; j < matrix->getColumns(); j++)
68         {
69             Pixel* pixel = &matrix->getPixels()[i][j];
70             int r = pixel->getRed();
71             int g = pixel->getGreen();
72             int b = pixel->getBlue();
73
74             int monochromeColor = (r + g + b) / 3;
75
76             if (monochromeColor > 0 && monochromeColor <= maxColorValue / 2)
77             {
78                 monochromeColor = 0;
79                 pixel->setRed(monochromeColor);
80                 pixel->setGreen(monochromeColor);
81                 pixel->setBlue(monochromeColor);
82             }
83
84             else if (monochromeColor > maxColorValue / 2 && monochromeColor <= maxColorValue)
85             {
86                 monochromeColor = maxColorValue;
87                 pixel->setRed(monochromeColor);
88                 pixel->setGreen(monochromeColor);
89                 pixel->setBlue(monochromeColor);
90             }
91         }
92     }
93 }
94 }
```

Фиг.7: Функцията *monochrome* за класа *PPMImage*

- *Negative* – фигура 8

```
182 void Image::negative()
183 {
184     for (int i = 0; i < matrix->getRows(); i++)
185     {
186         for (int j = 0; j < matrix->getColumns(); j++)
187         {
188             Pixel* pixel = &matrix->getPixels()[i][j];
189             int r = pixel->getRed();
190             int g = pixel->getGreen();
191             int b = pixel->getBlue();
192
193             int negativeRed = maxColorValue - r;
194             int negativeGreen = maxColorValue - g;
195             int negativeBlue = maxColorValue - b;
196
197             pixel->setRed(negativeRed);
198             pixel->setGreen(negativeGreen);
199             pixel->setBlue(negativeBlue);
200         }
201     }
202 }
```

Фиг.8: Функцията *negative* за класа *PPMImage*

- *Rotate<left>* - фигура 9

```

133 void Image::rotateLeftImage()
134 {
135     int rows = matrix->getColumns();
136     int columns = matrix->getRows();
137     Pixel** rotatedMatrixPixels = new Pixel* [rows];
138     for (int i = 0; i < rows; ++i)
139     {
140         rotatedMatrixPixels[i] = new Pixel[columns];
141     }
142
143     for (int i = 0; i < matrix->getRows(); i++)
144     {
145         for (int j = 0; j < matrix->getColumns(); j++)
146         {
147             int newColumns = matrix->getColumns() - j - 1;
148             rotatedMatrixPixels[newColumns][i] = matrix->getPixels()[i][j];
149         }
150     }
151
152     delete matrix;
153     matrix = new Matrix(rows, columns, rotatedMatrixPixels);
154 }

```

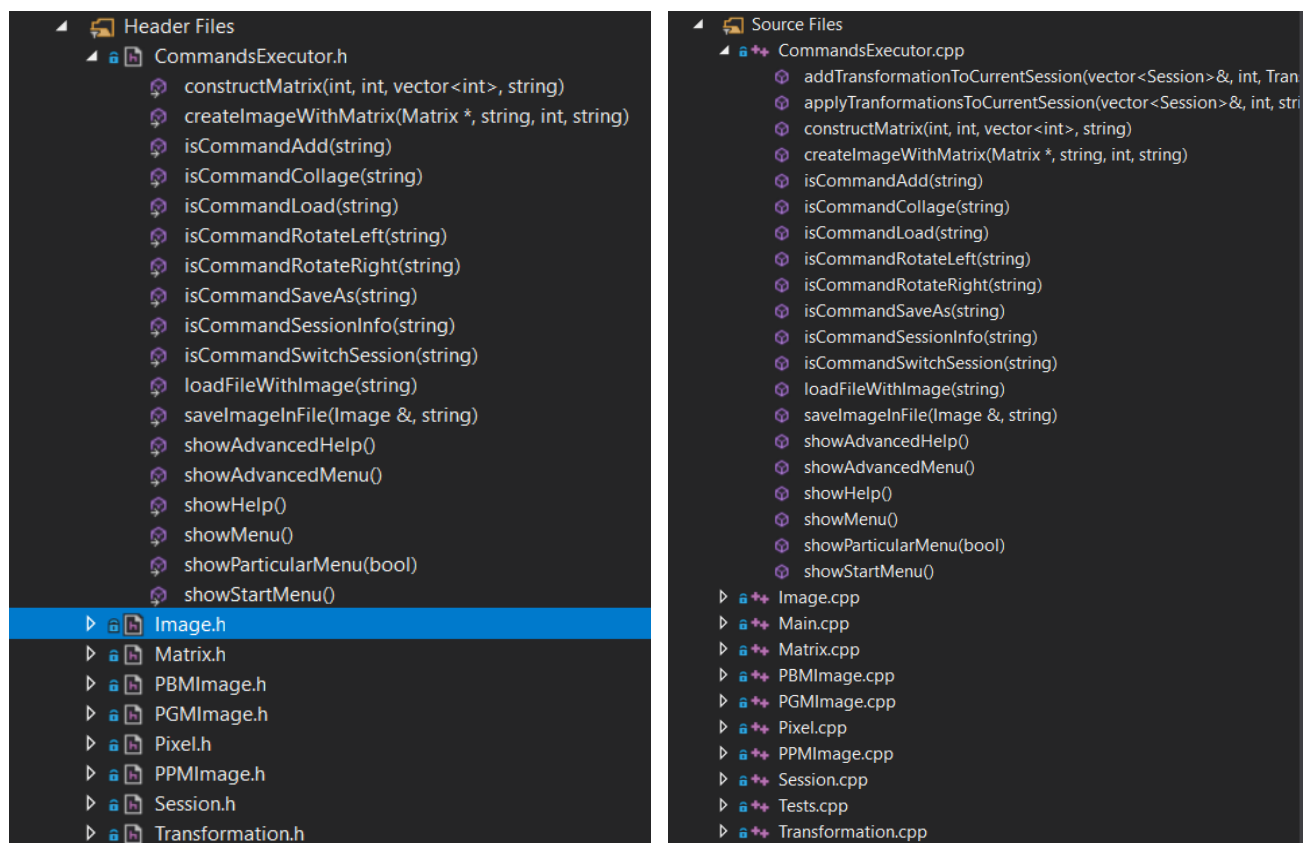
Фиг.9: Функцията *rotate<left>* за класа *Image*

Функцията *rotate<right>* е аналогична.

4. Реализация, тестване:

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

Използвано е разделно компилиране за по – добра организация и четимост – Фиг. 10.



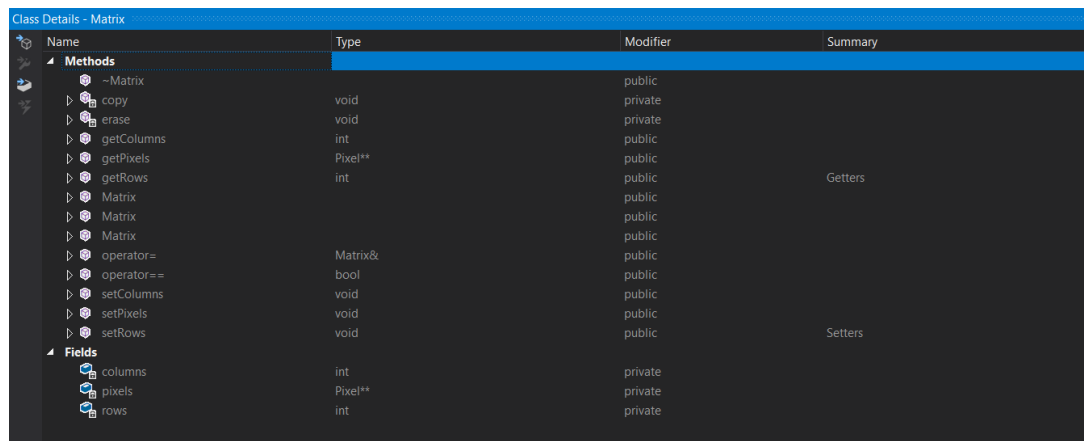
Фиг.10: *Header* и *Source* файловете на проекта

В проекта са реализирани множество от класове, които са представени чрез голямата четворка или в т.н. канонична форма. Също така са реализирани помощни класове и функции за оптимизация и по-добра четимост. Използвани са възможно най-малко глобални променливи и е наблегнато на локалните променливи. Осъществена е връзка между отделните класове.

Командата *load<file>* създава потребителска сесия, която има уникален номер. След тази команда се очаква име на файл, което да се зареди със създаването на сесията. Командата *close* затваря потребителската сесия. Командата *save* записва всички заредени в текущата потребителска сесия изображения след като прилага всички трансформации, а командата *save as* записва под ново име само изображението, което е заредено първо. Командата *help* извежда кратка информация за поддържаните от програмата функции. Командата *exit* излиза от програмата. Командата *grayscale* се прилага само върху изображенията в *PPM* формат. Тя превръща изображенията в черно-бели с нюанси на сивото. Командата *monochrome* преобразува изображенията до черно-бяло без никакви нюанси на сивото. Тази команда се прилага върху изображенията в *PGM* и *PPM* формат. Командата *negative* прави цветово обръщане на изображенията в текущата сесия. Тя се прилага върху всички файлови формати. Командата *rotate<direction>* завърта изображението на 90 градуса в съответната посока. Командата *undo* премахва последно направената трансформация. Ако е стартирана нова сесия и е въведена тази команда, то тя няма никакъв ефект. Командата *add<image>* добавя ново изображение към текущата сесия.

Тази команда може да добавя само по едно изображение към текущата сесия, за да може работи с файлове с интервали в името. Командата *session info* дава възможност на потребителя да получава подробна информация за текущата потребителска сесия – идентификационен номер, участващите изображения и набора от трансформации, които са били приложени върху съответните изображения. Командата *switch<session>* превключва към сесията с идентификационен номер *<session>*. Ако не съществува сесия с такъв идентификационен номер, то на стандартния изход се извежда подходящо съобщение.

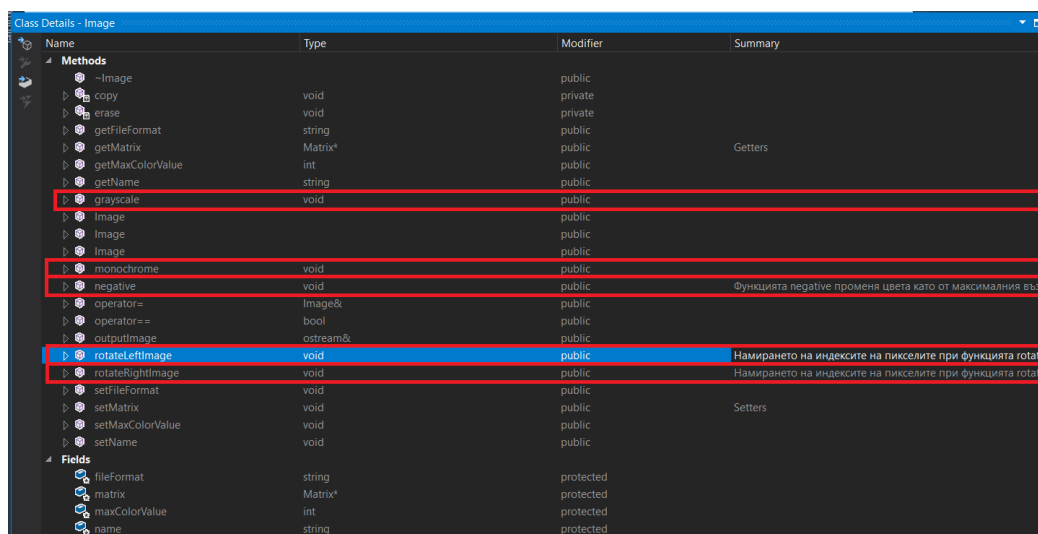
В проекта са реализирани множество от класове. Основният клас в проекта е класът *Matrix*. Чрез него създаваме матрица с определен брой редове и колони. На фигура 11 са представени член-данните и методите на този клас:



Name	Type	Modifier	Summary
Methods			
~Matrix		public	
copy	void	private	
erase	void	private	
getColumns	int	public	
getPixels	Pixel**	public	
getRows	int	public	Getters
Matrix		public	
Matrix		public	
Matrix		public	
operator=	Matrix&	public	
operator==	bool	public	
setColumns	void	public	
setPixels	void	public	
setRows	void	public	Setters
Fields			
columns	int	private	
pixels	Pixel**	private	
rows	int	private	

Фиг.11: Член-данни и методи на клас *Matrix*

Друг основен клас е класът *Image*. В него са дефинирани трансформациите на изображения. Той е базов за класовете *PBMImage*, *PGMImage*, *PPMImage*. На фигура 12 са представени член-данните и методите на този клас:



Name	Type	Modifier	Summary
Methods			
~Image		public	
copy	void	private	
erase	void	private	
getFileFormat	string	public	
getMatrix	Matrix*	public	Getters
getMaxColorValue	int	public	
getName	string	public	
grayscale	void	public	
Image		public	
Image		public	
Image		public	
monochrome	void	public	
negative	void	public	Функцията negative променя цвета като от максималния въз
operator=	Image&	public	
operator==	bool	public	
outputImage	ostream&	public	
rotateLeftImage	void	public	Намирането на индексите на пикселите при функцията rotat
rotateRightImage	void	public	Намирането на индексите на пикселите при функцията rotat
setFileFormat	void	public	
setMatrix	void	public	Setters
setMaxColorValue	void	public	
setName	void	public	
Fields			
fileFormat	string	protected	
matrix	Matrix*	protected	
maxColorValue	int	protected	
name	string	protected	

Фиг.12: Член-данни и методи на класа *Image*

Класовете *PBMImage*, *PGMImage*, *PPMImage* са производни на класа *Image*.

Класът *PBMImage* представя изображения, които са в *PBM* формат. Стрингът *P1* представлява файловия формат. Той използва само наследените от клас *Image* член-данни. Изображенията представляват черни и бели пиксели и нямат максимален цвят. На фигура 13 са представени член-данните и методите на този клас:

Name	Type	Modifier	Summary
outputImage	ostream&	public	
PBMImage		public	За формат PBM няма значение кой цвят от клас Pixel ще вземем

Фиг.13: Член-данни и методи на клас *PBMImage*

Класът *PGMImage* представя изображения, които са в *PGM* формат. Стрингът *P2* представлява файловия формат. Той използва само наследените от клас *Image* член-данни. Изображенията представляват черни и бели пиксели, както и нюанси на сивото, като те се представят с числа в интервала между бялото и черното. Този файлов формат има максимален цвят, който представлява белия цвят. На фигура 14 са представени член-данните и методите на този клас:

Name	Type	Modifier	Summary
monochrome	void	public	Функцията monochrome преобразува изображението до такова, в което има само сиви пиксели.
outputImage	ostream&	public	
PGMImage		public	За формат PBM няма значение кой цвят от клас Pixel ще вземем

Фиг.14: Член-данни и методи на класа *PGMImage*

Класът *PPMImage* представя изображения, които са в *PPM* формат. Стрингът *P3* представлява файловия формат. Той използва само наследените от клас *Image* член-данни. Изображенията представляват цветни пиксели, които се представят чрез 3 числа – червено, зелено и синьо. Тези пиксели са представени в клас *Pixel*. На фигура 15 са представени член-данните и методите на класа *PPMImage*:

Name	Type	Modifier	Summary
grayscale	void	public	Функцията grayscale преобразува цветните изображения в черно-бели.
monochrome	void	public	Функцията monochrome преобразува изображението до такова, в което има само сиви пиксели.
outputImage	ostream&	public	
PPMImage		public	

Фиг.15: Член-данни и методи на класа *PPMImage*

Класът *Pixel* представлява стойностите на червеното, зеленото и синьото. Той е направен, за да по-лесното добавяне на нов файлов формат. На фигура 16 са представени член-данните и методите на класа *Pixel*:

Class Details - Pixel				
	Name	Type	Modifier	Summary
	Methods			
	copy	void	private	
	getBlue	int	public	
	getGreen	int	public	
	getRed	int	public	Getters
	operator=	Pixel&	public	
	Pixel		public	
	Pixel		public	
	setBlue	void	public	
	setGreen	void	public	
	setRed	void	public	Setters
	Fields			
	blue	int	private	
	green	int	private	
	red	int	private	

Фиг.16: Член-данните и методите на класа *Pixel*

Класът *Session* създава потребителска сесия. На фигура 17 са представени член-данните и методите на този клас:

Class Details - Session

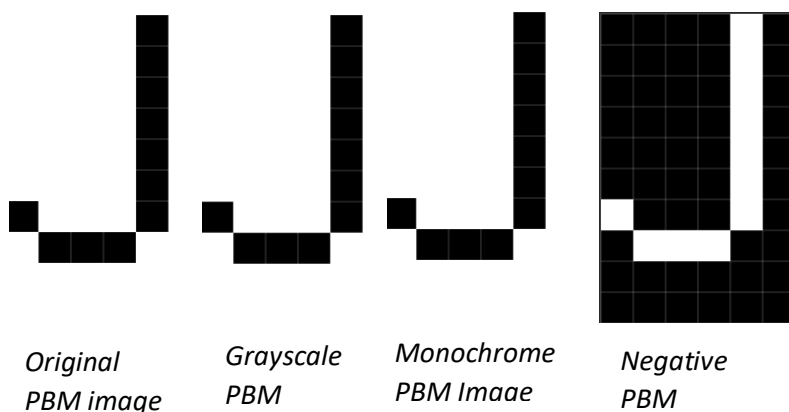
Name	Type	Modifier	Summary
Methods			
addImage	void	public	Getters
addTransformation	void	public	
getImages	vector<Image*>	public	
getSessionID	int	public	
getTransformations	vector<Transformation>	public	
print	void	public	
removeAllTransformations	void	public	
removeTransformation	void	public	
Session		public	Setters
setImages	void	public	
setSessionID	void	public	
setTransformations	void	public	
Fields			
images	vector<Image*>	private	
sessionID	int	private	
transformations	vector<Transformation>	private	

Фиг.17: Член-данните и методите на класа *Session*

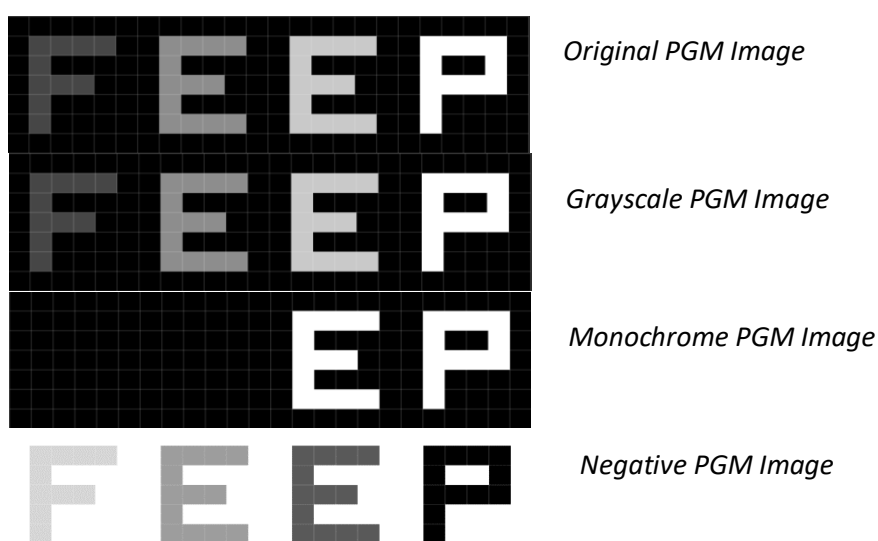
4.2. Планиране, описание и създаване на тестови сценарии(създаване на примери):

Първите стъпки от тестването на проекта включват тестването на конструкторите на всички класове, тъй като ако се получи грешка там, ще доведе до грешки в програмата цялостно и на по-късен етап би било трудно да се намерят тези грешки. След това бяха създадени примерни изображения в *PBM*, *PGM* и *PPM* формат. Като за начало бяха тествани методите *constructMatrix*, *createImageWithMatrix* и *loadFileWithImage*.

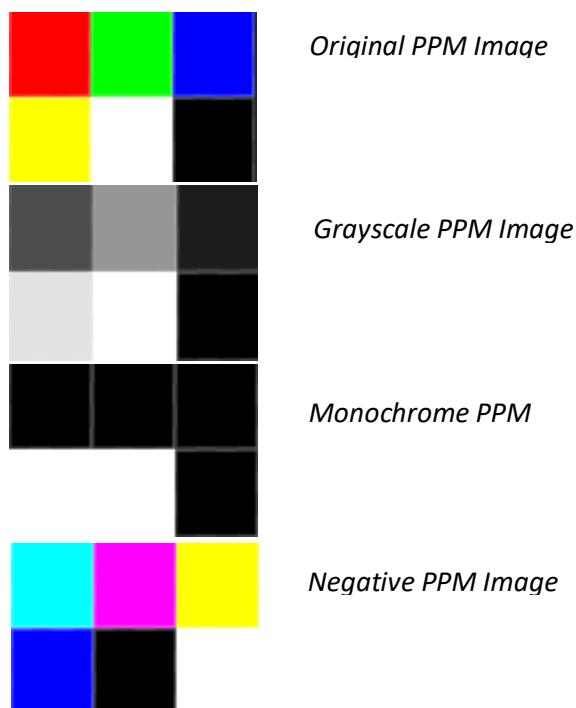
След това върху примерните изображения бяха приложени основните команди – *load*, *close*, *save*, *save as*, *help*, *exit*. Следващите стъпки от тестването на проекта включваха прилагането на различни трансформации върху примерните изображения – *grayscale*, *monochrome*, *negative*, *rotate <direction>*. Всяка от командите *grayscale*, *monochrome* и *negative* беше извикана за всеки един файлов формат, за да се проследи какви трансформации ще направи. Фигури 18,19 и 20 показват това:



Фиг.18: Трансформации върху PBM изображения



Фиг.19: Трансформации върху PGM изображения



Фиг.20: Трансформации върху PPM изображения

Тези трансформации са основни за изображенията, тъй като променят нещо по него – цветове или посока. Другите команди – *undo*, *add <image>*, *session info*, *switch <session>* бяха тествани последни. Командата *undo* беше извиквана, когато е отворено изображение, без да са извършвани трансформации, както и когато са извикани няколко команди. Командата *add<image>* беше извиквана множество пъти, както за изображения, които съществуват, така и за такива, които не съществуват. Командата *session info* беше извиквана, когато са извършвани множество трансформации, както и когато не е извършена нито една. Също така беше извиквана и за едно изображение, и за няколко. Командата *switch <session>* беше извиквана, както при съществуващи, така и при несъществуващи сесии. Бяха създадени сесии, всяка с различен брой изображения. На всяка сесия бяха извиквани по няколко команди, за да се проследи поведението на програмата в различни ситуации.

5. Заключение:

5.1. Обобщение на изпълнението на началните цели:

Проектът е завършен успешно. Интерфейсът е достатъчно функционален и същевременно лесен за използване и удобен.

5.2. Насоки за бъдещо развитие и усъвършенстване:

Част от бъдещите планове за развитие на този проект включват добавяне на различни файлови формати, както и добавяне на нови команди за обработка на растерни изображения.

Използвана литература:

1. <https://en.wikipedia.org/wiki/Netpbm>

Връзка към хранилище в Github:

<https://github.com/VeselaStoyanova>