**Технически университет – Варна**

Факултет по изчислителна техника и автоматизация

Софтуерни и интернет технологии

Курсов проект по Обектно-ориентирано програмиране 1 част

на Даниел Руменов Йовчев

фак. номер: 20621623

1. **Увод**

1.1 Задание на проекта: Т2 Работа със SVG файлове

В рамките на този проект трябва 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 на диска. След като заредите фигурите, потребителят трябва да може да изпълнява дадените в следващия раздел команди, които добавят, изтриват или променят фигурите. След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции:

* извежда на екрана всички фигури
* създава нова фигура
* изтрива фигура с пореден номер
* транслира фигура с пореден номер или всички фигури, ако не е указан пореден номер
* извежда на екрана всички фигури, които изцяло се съдържат в даден регион. Потребителят указва какъв да бъде регионът – правоъгълник или кръг.

1.2 Идея и цел на проекта

Изработеният проект е програма, написана на програмният език JAVA, която дава възможност на потребителя да извършва дадени операции върху фигури от тип SVG (Scalable Vector Graphics). Извличането на фигурите и тяхното съхраняване се извършва, чрез външен svg файл.

Функционалностите на програмата, предоставени на потребителя, включват възможност за създаване на нова фигура от поддържаните типове. Потребителят има възможност да изведе на екрана всички фигури прочетени от файла, включително и тези добавени по време на работата на програмата. Също така е предоставена възможност за транслация, като потребителят има право на избор дали ще транслира всички фигури, или избрана от него, чрез въвеждане на пореден номер. Потребителят има възможност и да провери и изведе на екрана всички фигури, които се намират в зададен регион. Потребителят избира този регион да бъде кръг или правоъгълник.

1.3 Структура на документацията

В документацията ще бъда разгледана по-подробно реализацията на програмата и нейните функционалности, както и проблемите свързани с решението. Структурата на документацията е отделена в смислови глави, чрез които да бъде разгледано решението. Документацията съдържа снимки и диаграми, които да предоставят визуално реализацията на програмата.

1. **Преглед на предметната област**

Предметната област, застъпена в проекта, е свързана с областта на геометрията и графиката. Основна структурна единица е фигурата. Всички функционалности на програмата използват тази структура, като в зависимост от приложението може да извършват и промени върху нея. Алгоритмите са свързани с обработка на колекция от фигури (обхождане, премахване, добавяне).

За създаването на програмата са използвани основните принципи на обектно-ориентираното програмиране, както и методи за обработка на файлове. Потребителят използва програмата, чрез въвеждане в командния ред, като за целта е създаден команден интерпретатор, приемащ командата и извършващ съответното действие.

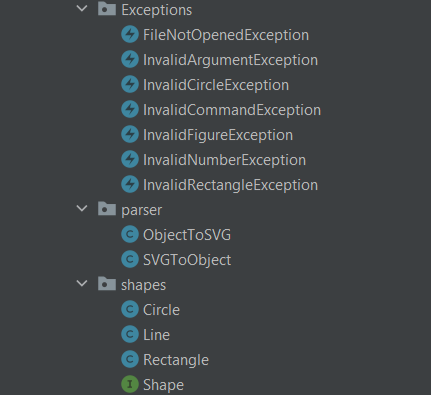
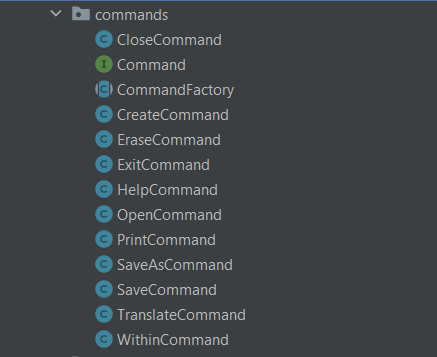
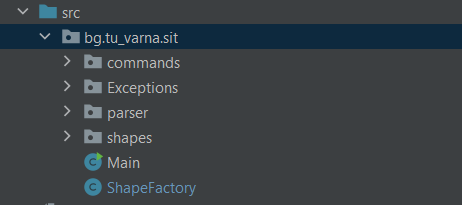
От потребителя се изисква да се запознае с командите, използвани в програмата, като за целта може да въведе в командния ред help.

1. **Проектиране**
   1. **Обща структура**

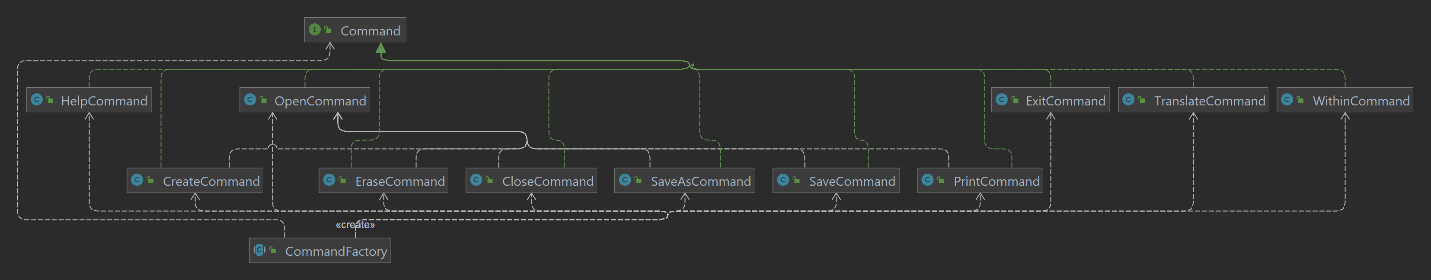
Структурата на проекта е изградена с помощта на пакети. Те са 5 на брой:

* основен пакет (bg.tu\_varna.sit) – в него се намират всички останали пакети, класът съхраняваш колекцията от фигури (ShapeFactory), както и класът изпълняващ програмата Main.
* пакет с команди (commands) – в него се намират класовете с всички команди, които потребителят може да използва. Пакетът съдържа интерфейс Command, обединяващ функционалността на всички класове команди, като също така спомага за реализирането на Command Factory класа – връща инстанция на класа-команда, избрана при въвеждане
* пакет с грешки (Exceptions) – съдържа изключения, които извеждат подходящо съобщение при неправилни входни данни или при изпълнение на дадена функционалност.
* пакет с парсър (parser) – в пакетът се съдържат двата класа отговорни за парсването на фигурите от и във файл. В тях е поместен кода имплементиращ използваният JAXB parser(*Java Architecture for XML Binding*).
* пакет с фигури (shapes) – в него се намират класовете, които отговарят за създаването на съответната фигура. Функционалността им е обединена от интерфейса Shape.

Фигура 1 - пакети



Функционалността на командното меню е изобразена с помощта на UML диаграма на фигура 2. Изобразени са връзките и зависимостите между интерфейса и командите, както и реализирането им в CommandFactory.



Фигура 2 – меню

Основният клас Main служи за взаимодействие на потребителя, като приема въведените от него команди и техните аргументи, ако са нужни за съответната функционалност. При грешна команда се извежда съответната грешка.

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true){

try {

System.out.print(">");

String input=scanner.nextLine();

String[] splittedInput=input.split("[ ]+");

String commandName=splittedInput[0];

Object[] commandArgs=new String[splittedInput.length-1];

System.arraycopy(splittedInput,1,commandArgs,0,commandArgs.length);

Command command= CommandFactory.getCommand(commandName);

if(command!=null){

command.execute(commandArgs);

}

else{

throw new InvalidCommandException();

}

}

catch (Exception e){

System.out.println(e);

}

}

}

}

Graphical user interface

Description automatically generated

Фигура 3 - структура на фигурите

На фигура 3 е изобразена структурата на пакета shapes. Интерфейсът Shape е имплементиран от всички фигури, обединявайки функционалностите им. Това позволява по-лесно добавяне на нова фигура в бъдеще, както и възможност фигурите да бъдат съхранени в обща колекция.

1. **Реализация, тестване**

За представяне реализацията на програмата, ще бъде последователно заредена информацията от файл и след това преминато през създадените функционалности. Примерната работа е представена във фигури 4 - 8.

Стартираме програмата с помощта на команда open:

Фигура 4

Създава се инстанция на клас OpenCommand и съдържанието от файла се зарежда в паметта, чрез JAXB парсър. Булевата променлива openedFile получава истина като стойност, указваща, че файлът е отворен. От тази стойност зависи изпълнението на останалите команди, с изключение на help и exit.



Фигура 5

На фигура 5 се вижда резултатът от изпълнението на команда за принтиране (print). Тя се осъществява от клас PrintCommand. Методът printAll се намира в класа ShapeFactory. За извеждане се обхожда колекцията от фигури.

ShapeFactory.printAll();

//ShapeFactory class

public static void printAll(){

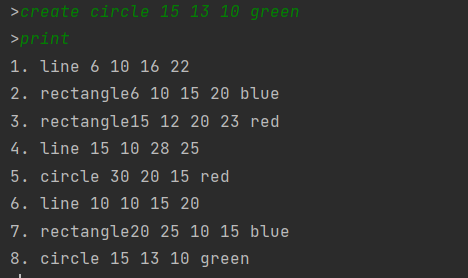
for(Shape shape: shapeList){

int num= shapeList.indexOf(shape)+1;

System.out.println(num+". "+shape);

}

}



Фигура 6

След това създаваме нова фигура, която добавяме към съществуващият списък от фигури. Действието е визуализирано на фиг. 6. В зависимост от въведената от потребителя фигура, се създава нова от съответния тип. В зависимост и от фигурата се прочитат различен брой аргументи, поради специфичните стойности, нужни за създаването на всяка фигура. Кодът, представен долу, показва добавянето на нова фигура от тип кръг. Добавянето на други видове става по аналогичен начин:

String input = args[0].toString();

if(input.equalsIgnoreCase("circle")){

if(args.length<5){

throw new InvalidArgumentException();

}

int x = Integer.parseInt(args[1].toString());

int y = Integer.parseInt(args[2].toString());

int radius = Integer.parseInt(args[3].toString());

String fill = args[4].toString();

if(radius<0){

throw new InvalidCircleException();

}

ShapeFactory.addToList(new Circle(x,y,radius,fill));

}

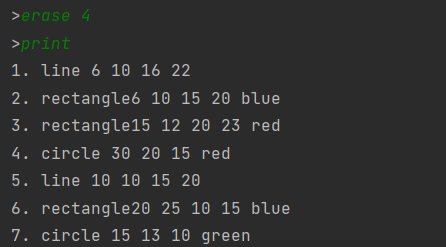
//from ShapeFactory class

public static List<Shape> addToList(Shape shape){

shapeList.add(shape);

return shapeList;

}

Функционалността, предоставяща изтриване на фигура е показана на фиг. 7. От потребителят се изисква да въведе пореден номер на фигура, която да бъде изтрита. Взема се в предвид, че индексът на колекцията започва от 0, затова и това разминаване се коригира в програмния код.

Фигура 7

ShapeFactory.removeFromList(index-1);

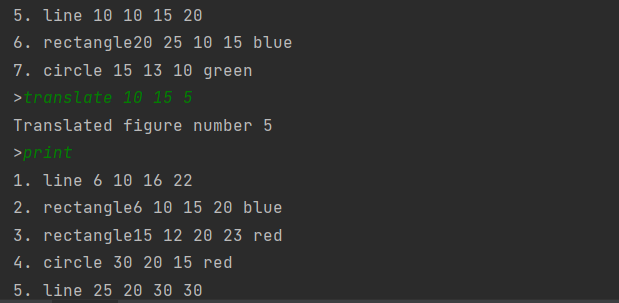
//From ShapeFactory class

public static List<Shape> removeFromList(int index){

shapeList.remove(index);

return shapeList;

}



Фигура 8

В следващата стъпка прилагаме транслация на фигура. Избираме това да бъде фигура номер 5, като потребителят има възможност да транслира и всички налични фигури. Въвежда се стойност по вертикала, хоризонтала и номер на фигура. Резултатът от действието е виден на фиг.8 . В зависимост от фигурата, се прилага транслация на съответната характеристика. В програмният код е представена фигурата отговаряща на номер 5, въведен от нас.

String num = args[2].toString();

int index = Integer.parseInt(num.trim())-1;

Shape translatedShape = ShapeFactory.getShapeList().get(index);

if(translatedShape instanceof Line){

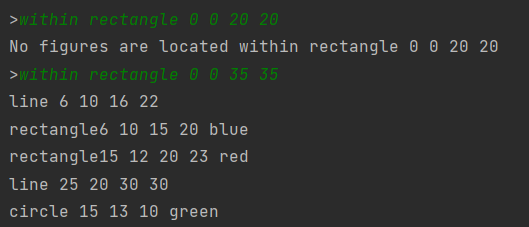
((Line) translatedShape).setY(((Line) translatedShape).getY()+vertical);

((Line) translatedShape).setX(((Line) translatedShape).getX()+horizontal);

((Line) translatedShape).setY2(((Line) translatedShape).getY2()+vertical);

((Line) translatedShape).setX2(((Line) translatedShape).getX2()+horizontal);

}



Фигура 9

Последната функционалност е свързана с проверка за наличие на фигури в зададен от потребителя регион (кръг или правоъгълник). На фиг. 9 е показан резултатът от изпълнението от въведеният регион. За определяне дали фигурата е в даден регион са сравнени координатите на въведеният регион и наличните паметта фигури. Фигурите, които отговарят на условието, се добавят в колекция, която извеждаме на екрана.

List<Shape> withinShapes = new ArrayList<>();

int x = Integer.parseInt(args[1].toString());

int y = Integer.parseInt(args[2].toString());

int width = Integer.parseInt(args[3].toString());

int height = Integer.parseInt(args[4].toString());

search="rectangle "+x+" "+y+" "+width+" "+height;

for(Shape shape: ShapeFactory.getShapeList()){

if(shape instanceof Circle){

if(((Circle) shape).getX()+((Circle) shape).getRadius()<x+width && ((Circle) shape).getX()-((Circle) shape).getRadius() > x && ((Circle) shape).getY()+((Circle) shape).getRadius()<y+height && ((Circle) shape).getY()-((Circle) shape).getRadius() > y){

withinShapes.add(shape);

}

}

else if(shape instanceof Rectangle){

if(((Rectangle) shape).getX()>x && ((Rectangle) shape).getY()>y && (((Rectangle) shape).getX()+((Rectangle) shape).getWidth()<x+width) && (((Rectangle) shape).getY()+((Rectangle) shape).getHeight()<y+height)){

withinShapes.add(shape);

}

}

else if(shape instanceof Line){

if((((Line) shape).getX()>x && ((Line) shape).getY()>y) && (((Line) shape).getX2()<x+width) && (((Line) shape).getY2()<y+height)){

withinShapes.add(shape);

}

}

}

След приключване на работата, можем да затворим файла без да го запазим с команда close или да го запазим, чрез save или saveas, ако искаме да го запазим в друг, определен от нас, файл. За изход от програмата се използва команда exit.

1. **Заключение**

В заключение, можем да кажем, че проектът е изпълнен в съответствие с поставеното условие. Като насока за бъдещо развитие би могло да бъде търсено разширяване на броя поддържани фигури. Възможно е и добавяне и на функционалности за изчертаване на фигури, в съответствие с възможностите, предоставени от SVG формата.

Връзка с github: <https://github.com/danielyovchev/project_OOP1>

Използвана литература и информация:

<https://gist.github.com/ervinsh/9c3be271c8dc62e356ca>

<https://stackoverflow.com/>

<https://gamedev.stackexchange.com/>

<https://www.w3.org/TR/SVG/shapes.html>

<https://www.baeldung.com/jaxb>