Технически университет – Варна

Факултет: ФИТА

Катедра: Софтуерни и интернет технологии (СИТ)

Специалност: СИТ

Тема на проекта:

Работа с SVG файлове

Изготвил: Име: Адам Мохамед Аяш

Факултетен номер: 21621553

Структура на документацията

1. Увод
    a. Задание
2. Преглед на предметната област
    a. Основни дефиниции, концепции и алгоритми
    b. Дефиниране на проблем и сложност на поставената задача
    c. Подходи, методи за решаване на поставените проблеми
3. Проектиране
    a. Обща структура на проекта
    b. Блок схема
4. Реализация и тестване
    a. Реализация на класове
    b. Планиране, описание на тестови сценарии
5. Заключение
    a. Обобщение на изпълнението на началните цели
    b. Насоки за бъдещо развитие и усъвършенстване

Използвана литература

1.Увод

а. Задание:

Проект 2: 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.

За улеснение, в рамките на проекта ще работим само с основните фигури (basic shapes) в SVG. Приложението ви трябва да поддържа поне три от тях. Например можете да изберете да се поддържат линия, кръг и правоъгълник. За повече информация за това кои са базовите фигури, вижте https://www.w3.org/TR/SVG/shapes.html.

Също така, за улеснение считаме, че координатната система, в която работим е тази по подразбиране: положителната полуос X сочи надясно, а положителната полуос Y сочи надолу.

Дизайнът на приложението трябва да е такъв, че да позволява при нужда лесно да можете да добавите поддръжка на нови фигури.

Когато зареждате съдържанието на един SVG файл, трябва да прочетете само фигурите, които приложението ви поддържа и можете да игнорирате всички останали SVG елементи.

След като заредите фигурите, потребителят трябва да може да изпълнява дадените в следващия раздел команди, които добавят, изтриват или променят фигурите.

Когато записвате фигурите във файл, трябва да генерирате валиден SVG файл

Операции

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (`open`, `close`, `save`, `save as`, `help` и `exit`):

`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`)

Работа с командния ред (Command line interface)

Вашата програма трябва да позволява на потребителя да отваря файлове (open), да извършва върху тях някакви операции, след което да записва промените обратно в същия файл (save) или в друг, който потребителят посочи (save as). Трябва да има и опция за затваряне на файла, без записване на промените (close). За целта, когато програмата ви се стартира, тя трябва да позволява на потребителя да въвежда команди и след това да ги изпълнява.

2.Преглед на предметната област

a. Основни дефиниции, концепции и алгоритми

SVG (Scalable Vector Graphic) – Мащабируема векторна графика е тип формат базиран на xml, който има за цел да създава графини изображения с помощта на вектори

b. Дефиниране на проблем и сложност на поставената задача

Проблем 1: Да се реализира програма, която о 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.

Проблем 2: Дизайнът на приложението трябва да е такъв, че да позволява при нужда лесно да може да се добавя поддръжка на нови фигури. Когато се зарежда съдържанието на един SVG файл, трябва да се прочетът само фигурите, които приложението поддържа

Сложност на поставената задача:

Високо ниво на абстрактно и логическо мислене за реализиране на програмата.

c. Подходи, методи за решаване на поставените проблеми

Решаване на проблем 1: При отварянето на даден файл се зареждат всички фигури под главния ***“<svg>”*** таг, след това поддържаните фигури се филтрират посредством функцията

***isShapeSupported*,** която сравнява тагът на текущата фигура с стойностите на енумерацията ***SupportedShapes*.**

Решаване на проблем 2: Добавянето на нова фигура става сравнително лесно, като първо се добавя нова стойност в енумрецията с поддържани фигури **SupportedShapes,** като там ще се съдържа информация за тагът на съответната фигура броят на параметрите и.

Втората стъпка е да се добави нов пасаж за новата фигура във валидиращият клас(**CreateCommandValidator**) който ще направи проверки дали въведените параметри са коректни и дали фигурата се поддържа.

3. Проектиране

a. Обща структура на проекта

Клас **CommandProcessor** съдържа логиката нужна за изпълнение на командите

Клас InputManager служи за четене на входни данни от потребителя

Клас CommandFactory служи за създаване на командите

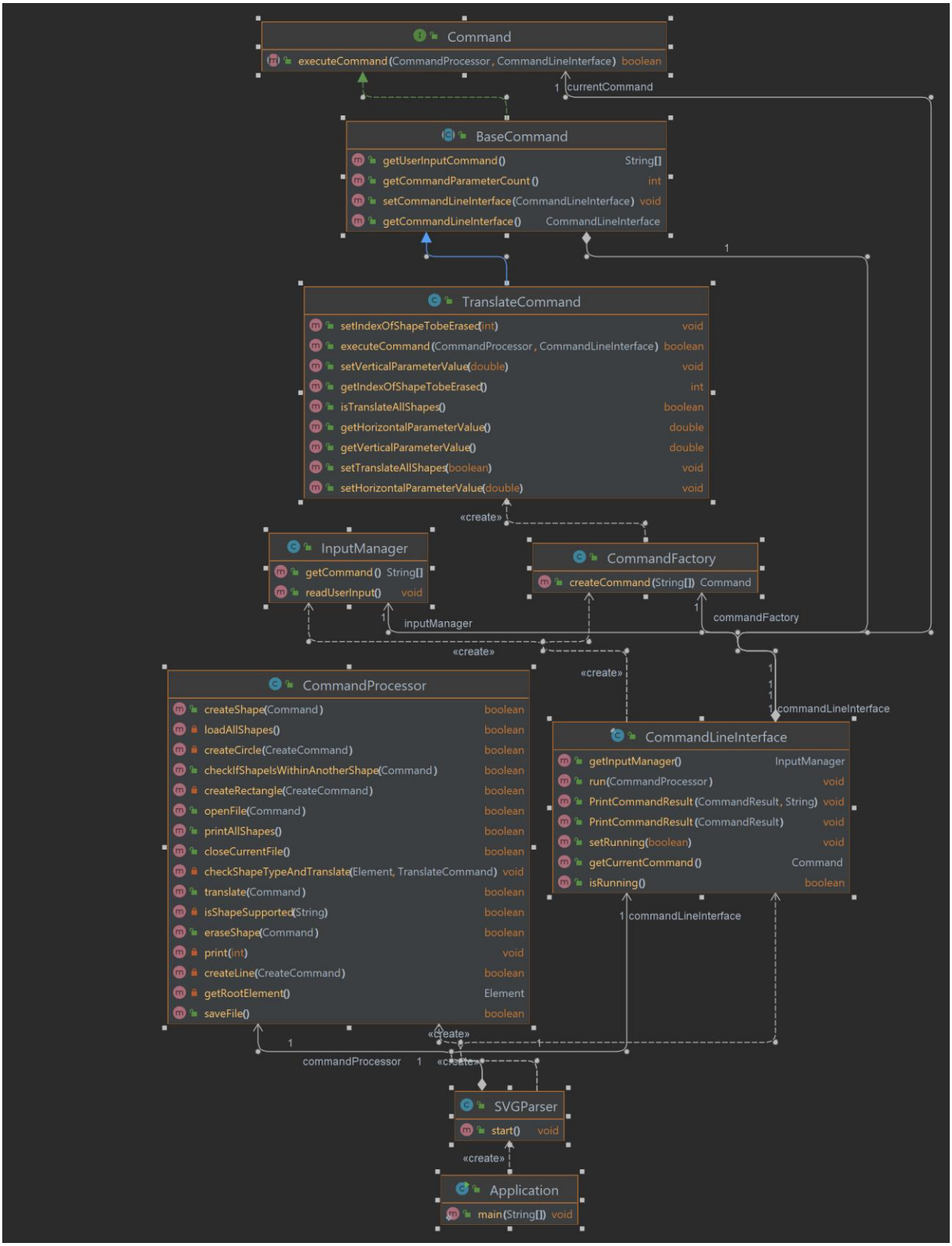
Интерфейс Command дава възможност за създаване на имплементации на различните комадни

Интерфейс CommandValidator дава възможност за създаване на имплементации на различните валидиращи класове

Клас CommandLineInterface – съдържа композиция от класовете InputManager и CommandFactory служи за обработка на входни данни и създаване на командите

Съответно класовете OpenCommand, CloseCommand, SaveCommand, SaveAsCommand и тн. Представляват имплементации на поддържаните команди от приложението, като целта е с помощта на валидиращите класове всяка команда само да се определя и да се валидира.

b. Блок схема


Command
executeCommand(CommandProcessor, CommandLineInterface) boolean
1 currentCommand
BaseCommand
getUserInputCommand() String[]
getCommandParameterCount() int
setCommandLineInterface(CommandLineInterface) void
getCommandLineInterface() CommandLineInterface
1
TranslateCommand
setIndexOfShapeTobeErased(int) void
executeCommand(CommandProcessor, CommandLineInterface) boolean
setVerticalParameterValue(double) void
getIndexOfShapeTobeErased() int
isTranslateAllShapes() boolean
getHorizontalParameterValue() double
getVerticalParameterValue() double
setTranslateAllShapes(boolean) void
setHorizontalParameterValue(double) void
«create»
InputManager
getCommand() String[]
readUserInput() void
CommandFactory
createCommand(String[]) Command
1 inputManager
1 commandFactory
«create»
«create»
1
1
1
1 commandLineInterface
CommandProcessor
createShape(Command) boolean
loadAllShapes() boolean
createCircle(CreateCommand) boolean
checkIfShapeIsWithinAnotherShape(Command) boolean
createRectangle(CreateCommand) boolean
openFile(Command) boolean
printAllShapes() boolean
closeCurrentFile() boolean
checkShapeTypeAndTranslate(Element, TranslateCommand) void
translate(Command) boolean
isShapeSupported(String) boolean
eraseShape(Command) boolean
print(int) void
createLine(CreateCommand) boolean
getRootElement() Element
saveFile() boolean
CommandLineInterface
getInputManager() InputManager
run(CommandProcessor) void
PrintCommandResult(CommandResult, String) void
PrintCommandResult(CommandResult) void
setRunning(boolean) void
getCurrentCommand() Command
isRunning() boolean
1 commandLineInterface
«create»
1
1
commandProcessor 1 «create»
SVGParser
start() void
«create»
Application
main(String[]) void

4. Реализация и тестване

a. Реализация на класове

Четенето на входни данни от потребителя става посредством InputManager класът, който след прочитането на съответната команда и нейните параметри я разделя по интервал и запазва всеки елемент в масив от низове. Фигура 1

Фигура 1

```
public class InputManager {

    //-----Constants-----
    //константа описваща по какъв символ ще се отделят параметрите на въведената команда
    1 usage
    private final String SPLIT_DELIMITER = " ";

    //константа от тип Scanner, служи за четене на вход от потребителя
    2 usages
    private final Scanner USER_INPUT_SCANNER;

    //-----Members-----
    //описва команда и нейните параметри
    2 usages
    private String[] command;

    //-----Constructor-----
    1 usage  Adam Ayash
    public InputManager() {
        USER_INPUT_SCANNER = new Scanner(System.in);
    }

    //-----Methods-----
    Adam Ayash
    public String[] getCommand() { return command; }

    //чете команда от потребителя и разделя съответните параметри, като ги запазва в масив - command
    1 usage  Adam Ayash
    public void readUserInput() {
        command = USER_INPUT_SCANNER.nextLine().split(SPLIT_DELIMITER);
    }

    //-----Overrides-----
}
```

Създаването на командите става посредством класът CommandFactory и методът createCommand фигура 2 , който проверява дали командата е поддържана и ако това условие е изпълнено я създава.

Фигура 2:

```
public Command createCommand(String[] command) throws CommandNotImplementedException, ArrayIndexOutOfBoundsException{
    String currentCommandKeyword = command[0];

    if(currentCommandKeyword.equals(SupportedCommands.EXIT.getCommand()))
        return new ExitCommand(command, SupportedCommands.EXIT.getCommandParameterCount());

    if(currentCommandKeyword.equals(SupportedCommands.OPEN.getCommand()))
        return new OpenCommand(command, SupportedCommands.OPEN.getCommandParameterCount());

    if(currentCommandKeyword.equals(SupportedCommands.SAVE.getCommand()))
        return new SaveCommand(command, SupportedCommands.SAVE.getCommandParameterCount());

    if(currentCommandKeyword.equals(SupportedCommands.CREATE.getCommand()))
        return new CreateCommand(command, SupportedCommands.CREATE.getCommandParameterCount(), isColorless: false);

    if(currentCommandKeyword.equals(SupportedCommands.CLOSE.getCommand()))
        return new CloseCommand(command, SupportedCommands.CLOSE.getCommandParameterCount());

    if(currentCommandKeyword.equals(SupportedCommands.PRINT.getCommand()))
        return new PrintCommand(command, SupportedCommands.PRINT.getCommandParameterCount());
```

Отварянето на файловете и зареждането на фигурите става чрез класа CommandProcessor и методите openFile фигура 3 и методът loadAllShapes фигура 4.

Фигура 3:

```
public boolean openFile(final Command currentCommand){
    OpenCommand openCommand = (OpenCommand) currentCommand;
   if(!isFileOpened) {
       currentFile = new File(((OpenCommand) currentCommand).getFilePath());
       DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
       DocumentBuilder documentBuilder;
       try {
           documentBuilder = documentBuilderFactory.newDocumentBuilder();
           if (currentFile.exists()) {
               currentDocument = documentBuilder.parse(openCommand.getFilePath());
               if(!loadAllShapes())
                   return false;
           } else {
               currentDocument = documentBuilder.newDocument();
               Element svgRootElement = currentDocument.createElement(ROOT_ELEMENT);
               currentDocument.appendChild(svgRootElement);
           }
       } catch (ParserConfigurationException | IOException | SAXException |
                IllegalArgumentException | NullPointerException e) {

           ErrorLogger.logError(e.toString());
           return false;
       }
       isFileOpened = true;
       openCommand.getCommandLineInterface().PrintCommandResult(CommandResult.FILE_SUCCESSFULLY_OPENED, openCommand.getFilePath());
   }else {
       openCommand.getCommandLineInterface().PrintCommandResult(CommandResult.FILE_ALREADY_OPENED);
   }
    return true;
```

Фигура 3:

```
private boolean loadAllShapes(){
    boolean result = true;
    try {
        Element root = getRootElement();
        NodeList shapeList = root.getChildNodes();
        for (int i = 0; i < shapeList.getLength(); i++) {
            Node currentShape = shapeList.item(i);
            if (currentShape.getNodeType() == Node.ELEMENT_NODE) {
                Element shapeElement = (Element) currentShape;
                if (isShapeSupported(shapeElement.getTagName())) {
                    this.shapesList.add(shapeElement);
                }
            }
        }
    }
    catch (ClassCastException | ArrayIndexOutOfBoundsException | NullPointerException e){
        ErrorLogger.logError(e.toString());
        result = false;
    }

    return result;
}
```

Създаването на фигури става посредством класът CreateCommand, който има за цел да създаде транспортен клас за атрибутите на фигурата, като за да се определи дали параметрите на фигурата въведени от потребителя са коректни командата минава през валидиращ клас CreateCommandValidator метод validate фигура 4.

Фигура 4:

```
public boolean validate(Command command) throws ArrayIndexOutOfBoundsException, NullPointerException,
        NumberFormatException{
    try {
        CreateCommand createCommand = (CreateCommand) command;

        SupportedShapes shape = isShapeSupported(createCommand);
        SupportedColors color = isColorSupported(createCommand);

            if (checkShapeParameterCount(createCommand, shape) && shape != null && (color != null || createCommand.isColorless())) {
                createCommand.setShape(checkAndAssignParameters(createCommand, shape));

                if(!createCommand.isColorless())
                createCommand.getShape().setColor(color.getColor(), createCommand.isColorless());
            }
    }
    catch (ArrayIndexOutOfBoundsException| NullPointerException|
            NumberFormatException |  ShapeNotSupportedException e){
        ErrorLogger.logError(e.toString());
        return  false;
    }

    return true;

}
```

d. Планиране, описание на тестови сценарии

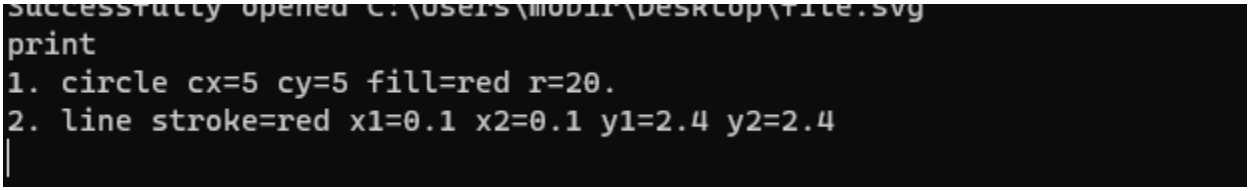
При стартиране на програмата, първоначално се изисква да се отвори xml файл. Фигура 5

Ако се напише друга команда се извежда съобщение, че трябва да се отвори файл.

Фигура 5:

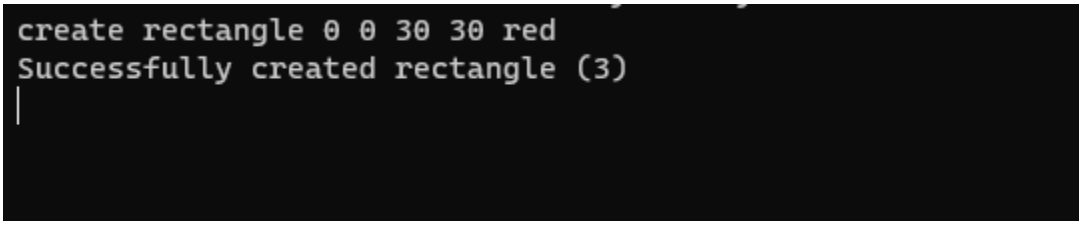
Файлът file.svg съдържа предварително въведени фигури. С команда print можем да ги изведем на екрана . Фигура 6

Фигура 6:



За създаване на нова фигура използваме команда create. Фигура 7

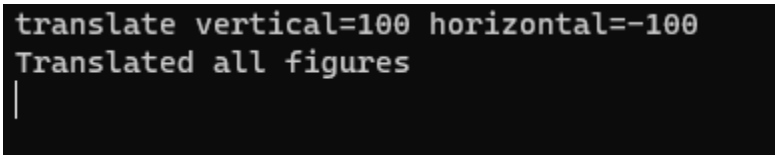
Фигура 7:



За манипулация на координатите на фигурите използваме командата *translate*

Фигура 8

Фигура 8:



За проверка дали наличните фигури се съдържат в зададена от нас фигура използваме командата *within Фигура 9*

Фигура 9: