**Документация**

**на проект SVG Parser**

Изготвена от Тодор Тонев Димитров 81969

1. **Увод**

Гитхъб репо на проекта: <https://github.com/todorakisa/SVG-Parser>

* 1. **Описание и идея на проекта**

Идеята на проекта е да се направи приложение което да обработва *svg* файлове. Да може потребителя да отвори даден файл и чрез някакви команди той да го модифицира. След като са извършени дадени операции по файла, приложението трябва да може да ги запази в оригиналния файл или нов такъв по избор на потребителя.

* 1. **Цел и задачи на разработката**

Целта на приложението е правилно да чете информацията от даден *svg* файл и да я пази в подходяща структора. На тази стуктора трябва да се извършват дадени операции които я променят и след това трябва да може всичката тази инфорамция да се запази в дадения първоначално файл. В това приложение остновните задачи са: Да се измисли поджодящо наследяване на фигурите в *svg* файловете с което да пазим данните(*svg* елементи) от *svg* файла; Да се измисли подходящ начин по който да се изтрива *svg* елемент в структурата; Да се измисли начин по който да се запазва информацията в същия файл. Трябва и да се направят редица команди с които потребителя да модифицира данните, да отваря файлове и да запазва файлове. Ако потребителя не запази модифицирания файл то програмата не трябва да променя оригиналния файл. Също така запазения модифициран файл, трябва да е валиден *svg* файл.

* 1. **Структура на документацията**

В документацията има пет основни глави: 1 Увод 2 Преглед на предметната област 3 Проектиране 4 Реализация и тестване и 5 Заключение. Всяка глава си има различни аспекти за разглеждане на проекта. Във втората глава ще разгледаме проблеми които ще срещнем в разработката на програмата.

1. **Преглед на предметната област**
   1. **Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

За да може да запазваме инфорамцията от *svg* файл трябва да се направи структора за запазване на svg елементитe. Понеже svg има само един parent елемент, а именно <svg> няма нужда да се прави дърво за елементите. Затова ще се ползва масив за всички svg елементи. За да може приложението да поддържа различни фигури ще трябва да се направи наследяване. По този начин опрациите ще се абстрахират от вида на фигурата, а всяка фигура ще си има свои методи за работа с нея. Също така за да може правилно да вземем всичката информация от даден файл трябва да се измисли алгоритъм който да чете от *svg* файл. За да можем правилно да запазваме модифицирания файл ще трябва и всяка фигура да си има свой метод за писане във файл.

* 1. **Дефиниране на проблеми и сложност на поставената задача**

Има няколко основни проблема в задачата и те са:

1. **Намиране на поджодящ подход при наследяването**

За да може приложението да поддържа много фигури ще трябва да се направи наследяване. Ще трябва да може всяка фигура да има свои методи с които да се работи с нея. Така и четенето и писането във файл ще са операции зависещи от дадената фигура.

1. **Алгоритъм за четене от svg файл**

Заради наследяването алгоритъма за четене от файла трябва да използва методите на всяка фигура и по някакъв начин ще трябва да създаваправилната фигура.

1. **Запазване на промените във оригиналния файл**

Тук отново, заради вида на наследяването, записa във файл трабва да е според фигурата.

* 1. **Подходи, методи за решаване на поставените проблемите**

За да се решат проблемите ще трябва да се направи клас фигура който да съдържа всички методи които са едни и същи за всички фигури. Също трябва да съдържа всички член-променливи които всеки елемент има. Този клас трябва да е виртуален и всеки наследник да е задължен да имплементира методите които са само за него. По този начин всеки клас ще имплементира отделно писане във файл. За да оправим четенето от файл трябва да направим клас атрибут и при всяко четене на елемент ще му слагаме всички атрибути дори и основните и после той персонализирано да ги сложи като член променливи. Също така ще трябва клас фактори. Който според елемента който се чете ще създава празен обект от този тип.

* 1. **Потребителски и качествени изисквания**

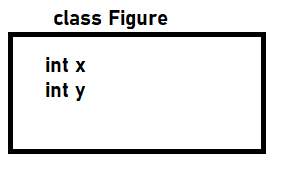
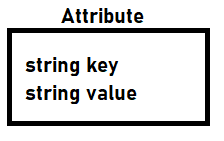
За да може потребителя да работи с програмата трябва да се реализират команди с които дой да управлява какво да се случи с файла. За да може да не се получава объркване с командите, потребителя не трябва да може да ги използва ако не е отворен даден файл и съответно неможе да отваря друг файл преди да е затворил предишния.

1. **Проектиране**
   1. **Обща архитектура – ООП дизайн**

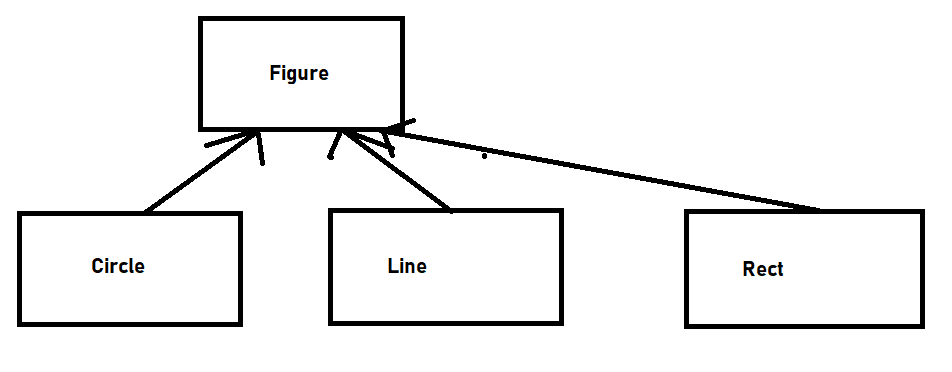
За да може да се запази цялата информация от файла, ще има един основен клас фигура. Тази фигура ще бъда наследена от всеки вид фигура(фиг. 3.2.3), като кръг или правоъгълник. За целта трява да се реализира клас фигура(фиг. 3.2.2). По стандарт *svg* файловете имат един *<svg>* в който са всички фигури за четене. Така в програмата ще имама алгоритъм който да чете всички фигури и според вида на фигурата да използва класа factory с който да създаде съответния обект. После ще трябва всеки обект да си оправи променливите. По този начин в програмата ще имаме масив от пойнтери към главния клас, но всъщност всяка фигура може да бъде различна. Също така за да може да запазваме атрибутите на елементите, всеки елемент ще трябва да има масив от атрибути. За целта тряба да бъде имплементиран клас атрибут който да съдържа в себе си *key* и *value* като стойности(фиг. 3.2.1).

За да може да пише правилно елементите във файл програмата ще използва метод който е различен за всяка фигура. Например метода на класа circle и метода на класа rect(фиг. 3.2.4).

* 1. **Диаграми**

****

Фиг. 3.2.1 Клас атрибут Фиг. 3.2.2 Клас Фигура



Фиг. 3.2.3 Наследяване на класовете



Фиг. 3.2.4 Различни методи за писане във файл на класа кръг и правоъгълник

1. **Реализация и тестване**
   1. **Реализация на класове**

Реализирани са класовете: fileSVG, figure, circle, rectangle, line, attribute, factory. Figure съдържа цялата обща информация за всяка фигура. Той е наследен от circle, rectangle, line. Той съдържа в себе си само координати на една точка (x,y), защото всяка друга фигура притажава такива координати. Figure съдържа и няколко виртуални метода които всяка фигура наследник трябва да имплементира(фиг. 4.1.1). Attribute съдържа цялата информация за един атрибут.



Фиг. 4.1.1 Клас Figure и виртуалните и методи.

Класът Factory(фиг. 4.1.2) съдържа в себе си два статички метода. В единия се прави празна фигура според името й, а в другия се прави фигура според дадени стойности. Втория метод се използва в командата within за да може да се създаде фигура, за която да търсим дали съдържа други фигури.  
   
Фиг. 4.1.2 Клас Factory

Класът fileSVG(фиг. 4.1.3) съдържа в себе си методи с които да работим с инфорацията от файла. Двата основни метода които работят с файлове са fill\_content() и save\_as() ,те използват свойството на полиморфизма и според дадената фигура извикват правилните методи. Метода fill\_content()(фиг. 4.1.4) използва статичния метод за правене на празен обект от класа Factory и после използва виртуалния метод fix за да оправи параметрите на всяка фигура.

   
 Фиг. 4.1.3 Клас FileSVG



Фиг. 4.1.4 Метод fill\_content()

* 1. **Управление на паметта и алгоритми. Оптимизации.**

Всеки от класовете реализирани в проекта има съответен деструктор който да се погриже за заделената памент по време на изпълнение на програмата. По време на изпълнение на алгоритъма за четене от файла, се заделя най-много памет. Една от командите които програмата трябва да поддържа е изтриването на фигура. Понеже се ползва библиотеката вектор, за да се изтрие напълно дадената фигура трябва не само да махне от вектора ами и да се използва ключовата дума *delete*. За да може всяка фигура да се изтрива правилно деструктора на главния родител(класа фигура) е виртуален.

* 1. **Планиране, описание и създаване на тестови сценарии**

Проекта е разработван на Linux и заради това предимство е използвана програма наречена “valgrind” която следи за неосвободена памет в програмата. След тестване на проложението с нея са премахнати всички “memory leaks”.

За да се тества програмата напълно, то трябва да се тества дали за всички имплементирани видове фигури тя работи. В момента програмата има недостатък, който е че атрибутите които се взимат от файла за дадена фигура трябва да са в определен ред за да работи правилно. Този недостатък е включен в главата Заключение

1. **Заключение**
   1. **Обобщение на изпълнението на началните цели**

Изпълнени са основните цели на проекта, както и са разрешени основните проблеми при изпълнавянето на задачите които цели да изпълни това приложение.

* 1. **Насоки за бъдещо развитие и усъвършенстване**

Да се направи така че поредността на елементите на даден svg елемент да няма значение при четенето от фйла. Да се усъвършенства командата translate която да може да приема повече аргументи. Да се добавят нови фигури в програмата за да може да е по практична.