ДОКУМЕНТАЦИЯ НА ПРОЕКТ

ПО

Обектно-ориентирано програмиране

Изготвен от Дара Коева ФН: 45756

Тема №5 Приложение за работа с електронни таблици

Table

10	3.14	"Quoted"	OOP
=R3C4/R8C1			2024
Hello World!			-56.80

Дата: 20.06.2024г.

Съдържание:

- **Увод**

1.1 Описание и идея на проекта

Проекта реализира конзолно приложение, което работи с електронни таблици. Дадената таблица може да съдържа различен тип клетки – цяло число, дробно число, символен низ, формула или празна клетка. Приложението поддържа следната функционалност – зарежда на дадена таблица, извеждане на екрана, редактиране на данни от таблицата и записване на промените в указан файл.

1.2 Цел и задачи на разработката

Нужната функционалност на проекта е да реализира следните команди – open, save, save as, exit, print и edit.

- Open - зарежда таблица от подаден файл, като ако такъв не съществува се създава. След като сме заредили нужната информация файлът се затваря. Ако има проблем при четенето от файла или некоректно въведени данни тогава таблицата не се създава и се извежда съобщение с проблема за потребителя.

- Save - Записва направените промени обратно в същия файл, от който са били прочетени данните. Ако има проблем при записването показва на потребителя, че е било неуспешно.

- Save as - Записва направените промени във файл, като позволява на потребителя да укаже конкретния път. Ако има проблем при записването показва на потребителя, че е било неуспешно.

- Exit - излиза от приложението, като преди това пита потребителя иска ли да запази промените направени по таблицата ако такива са направени.

- Print - извежда заредената таблица на стандартния изход в удобен формат.

- Edit – по дадени координати относно положението на клетката в таблицата и съответна стойност заменя стойността на тази клетка. Може дори да се промени типа на клетката, а ако координатите се намират извън размерите на текущата таблица – тя се разширява. При подадени некоректни данни таблицата остава не променена.

- **Глава 2. Преглед на предметната област**

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

Проектът е реализиран с помощта на тези пет основни части: Cell, Table, Factory,Calculator и CommandLine.

- Cell - абстрактен базов клас с четири основни чисто виртуални метода, които са предефинирани в наследниците.

- Factory – клас (който може да има само един обект), който от подаден низ разпознава какъв трябва да бъде типа на клетката и я създава.

- Calculator – клас (който може да има само един обект), който от подаден низ представляващ формула, която е само от цифри и операция я пресмята и връща стойността на формулата.

- Table - клас с полиморфен контейнер, където записваме указатели от тип Cell. Използваме обект от Factory, за да си създадем необходимите клетки в Таблицата, а с Calculator да сметнем формулите.

- CommandLine – клас (който може да има само един обект), той стартира нашето приложение. Той се занимава с командите, който приложението трябва да поддържа, като взима вход от потребителя и изпълнява тази команда, която е въведена. Ако някоя от командите е сгрешена ще съобщи на потребителя за проблема.

2.2. Дефиниране на проблеми и сложност на поставената задача.

- Изисква се таблицата да може да поддържа едновременно работа с различни типове данни, което е невъзможно със статично свързване.

- Изисква се и различните типове клетки да бъдат колкото е възможно по-уеднаквени с цел по-лесно и аналогично прилагане на желаните команди върху таблицата.

- Проблем при смятането на формулите и то по-точно при формули с координати на клетки. Налага се, ако клетките, които участват в нашата формула не са все още сметнати, то да извършим първо техните изчисления и чак след това да сметнем клетката, от която сме тръгнали.

- Трябва доста да се внимава за некоректни данни и да може програмата да бъде гъвкава, за да реагира на време.

- Необходимо е при създаването на клетките да може да се прецени правилно типа на клетката според подадената информация.

2.3. Подходи, методи за решаване на поставените проблемите.

- Правене на абстрактен базов клас – Cell с няколко основни чисто виртуални функции, които петте наследника (различните типове клетки) да предефинират с цел унифициране. Прави се с динамично свързване.

- За пресмятането на формули, състоящи се от координати на други клетки ще използваме рекурсия. За целта е добре всяка клетка да си има някакъв флаг, чрез който да показва дали дадена клетка трябва да се сметне или вече е била сметната и стойността и може да се използва наготово.

- Създаването на клетките ще се осигурява от клас, които ще е Singleton (клас, който може да има само един обект) и той ще следи спрямо подадените данни какъв трябва да бъде типа. Например, ако започва с “ , то клетката трябва да бъде от тип Символен низ и т.н.

• Глава 3. Проектиране

3.1. Обща архитектура – ООП дизайн.

Абстрактният базов клас Cell има пет наследника EmptyCell, IntCell, DoubleCell, StringCell, Formula. Като Cell има виртуален деструктор и четири чисто виртуални метода:
storeCell() – запазва данните на клетката във файл.
clone() – връща указател, към ново копие на клетката
getData() – връща данните на съответната клетка в символен низ
print() – изкарва на екрана данните на самата клетка.
Всички тези функции са предефинирани подходящо във всеки наследник.
Единствено getData() се отличава в класа StringCell, където там връща не подадените данни за направата на клетката, а низа без кавички и разделителни черти.

Singleton (клас, който може да има само един обект) класа Factory, в който е дефинирана функцията: createCell() – по подаден низ търси съответстващия тип на клетката, след което създава нова клетка от искани тип и връща указател към нея. Може да хвърли изключение ако не може да намери тип, който да отговаря на подадените данни от потребителя.

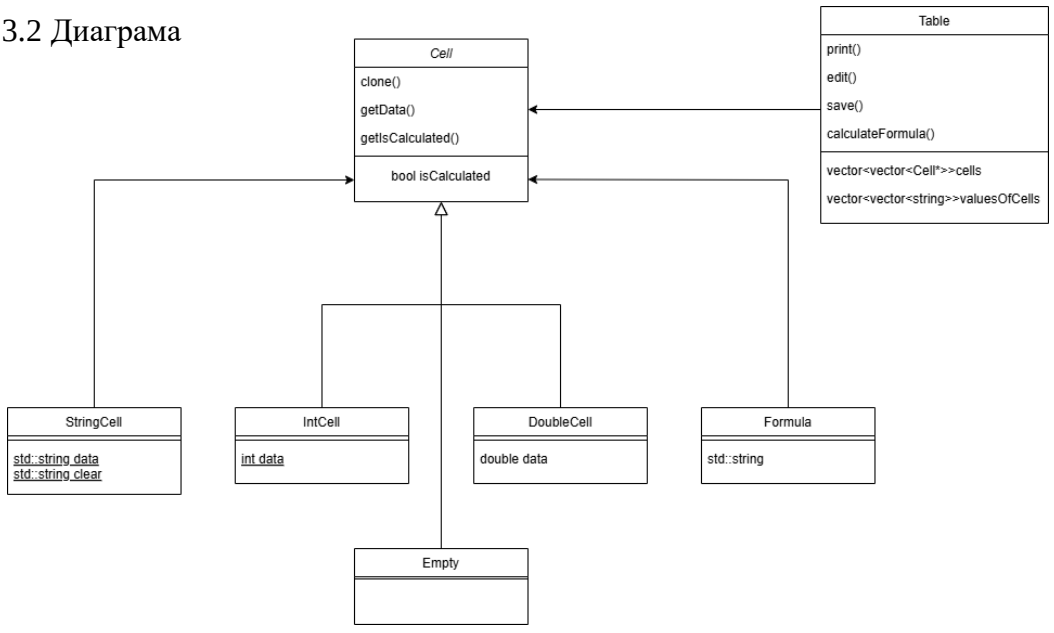
Singleton (клас, който може да има само един обект) класа Calculator. Той смята формули, които се състоят само от числа. В него е дефинирана функцията

CalculateWithOnlyDigits() – която по подаден низ формула го разбива на две числа и съответната операция, пресмята я и връща резултата под формата на символен низ.

Класът Table съдържа полиморфен контейнер, където записваме указатели от тип Cell. Също в класът пазим и вектор от вектор от низове valuesOfCells, където записваме сметнатите стойностите на клетките. Това е с цел по-бързо изпълнение на командата print(), за да не се налага за просто едно принтиране всеки път да се смятат наново формулите. След като се отвори файла посочен от потребителя и се създадат успешно всички клетки с помощта на класа Factory, се извиква функция FullValueOfCells() – чрез нея първо записваме в valuesOfCells всички данни на клетки тип различен от формула. След което всяка формула се пресмята с помощта на рекурсивната функция calculateFormula() и след това се записва в valuesOfCells. При искане на редактиране на клетка се налага не само промяна на матрицата от указатели cells, а и на матрицата от низове valuesOfCells , тъй като нова стойност може да участва, в някоя вече сметната формула или новата клетка да бъде от тип формула, тоест се налага отново да се пресметнат всички клетки. Преди да извикаме отново FullValueOfCells(), преди нея изпълняваме функцията setFormulasCellNotCalculated(), за да може calculateFormula() да ги пресметне успешно. Ползва се структура Coordinates за намиране на клетка, която участва във формула или трябва да бъде променена.

Имаме и клас CommandLine отново Singleton(клас, който може да има само един обект), който има указател към таблица, която се заделя динамично. Пазим и името/пътя на файла, от който е била заредена тази таблица, за да може при извикване на команда save да запазим таблицата обратно в същия файл. В класа имаме и две булеви променливи едната ни казва дали да продължи програмата, а другата дали са направени промени по таблицата след като е била заредена от файл. Както и пазим въведения ред от потребителя в низ. В класа има две основни функции – start() и execute(). start() – е цялата ни програма, приема желаните команди от потребителя опитва се да я изпълни чрез execute() и ако е неуспешно и се хвърли някакво изключение, то се обработва и програмата се справя успешно с него.

3.2 Диаграма



- **Глава 4. Реализация, тестване**

4.1. Реализация на класове

class Cell :

```
class Cell
{
public:
	Cell() { }
	virtual ~Cell () { }
	virtual bool storeCell(std::ofstream&) const = 0;
	virtual Cell* clone() const = 0;
	virtual std::string getData() const = 0;
	virtual void print()const = 0;

	bool getIsCalculated() const;
	void setIsCalculated(bool);

protected:
	bool isCalculated = true;
	};
```

class Factory :

```
enum CellType {

	UNKNOWN = -1,

	INT = 0,
	DOUBLE = 1,
	STRING = 2,
	FORMULA = 3,
	EMPTY = 4

};

class Factory
{
public:
	static Factory& getFactory();
	Cell* creatCell(const std::string) const;

private:
	Factory() = default;
	~Factory() = default;
	Factory(const Factory&) = delete;
	Factory& operator=(const Factory&) = delete;

	bool isIntCell(const std::string&) const;
	bool isDoubleCell(const std::string&) const;
	bool isStringCell(const std::string&) const;
```

```
        bool isFormula(const std::string&) const;
        bool isEmptyCell(const std::string&) const;

        CellType getTypeOfCell(const std::string&) const;

        Cell* createIntCell(const std::string&) const;
        Cell* createDoubleCell(const std::string&) const;
        Cell* createStringCell(const std::string&) const;
        Cell* createFormulaCell(const std::string&) const;
        Cell* createEmptyCell() const;
};
```

class Calculator :

```
class Calculator {
public:
        friend class Table;

        static Calculator& getCalculator();
        std::string CalculateWithOnlyDigits(const std::string&);
private:
        Calculator() = default;
        ~Calculator() = default;
        Calculator(const Calculator&) = delete;
        Calculator& operator= (const Calculator&) = delete;

        void splitFormula(const std::string&, double&, double&, std::string&);
        std::string removeZeros(std::string&);
        bool isOperation(const char c) const;
};
```

class Table :

```
 struct Coordinates {

        size_t row;
        size_t col;

};

class Table
{
public:
        Table(const std::string);
        Table(const Table&);
        Table& operator=(const Table&);
        ~Table();

        void print() const;
```

```
        void edit(const Coordinates&,std::string&);
        bool save(std::ofstream&) const;

private:
        size_t getRows(std::ifstream&) const;
        size_t getColumns(std::ifstream&) const;
        bool isNumberInString(const std::string&) const;
        void removeWhiteSpaces(std::string&);
        size_t lenOfLongestStringInTheCol(const size_t) const;
        void destroy();

        void FullValueOfCells();
        bool isSimpleFormula(const std::string&);
        void splitReferencesFormula(const std::string&, Coordinates&, Coordinates&,
std::string&);
        void calculateFormula(Cell&, Coordinates&);
        void setFormulasCellNotCalculated();
        std::string calculateReferencesFormula(const Coordinates&, const
Coordinates&, const std::string&);

        std::vector<std::vector<Cell*>> cells;
        std::vector<std::vector<std::string>> valuesOfCells;
};
```

class CommandLine :

```
enum  class Commands {

        UNKNOWN = -1,
        OPEN = 0,
        SAVE = 1,
        SAVE_AS = 2,
        PRINT = 3,
        EDIT = 4,
        EXIT = 5

};


class CommandLine {
public:
        static CommandLine& getCommandLine();
        void start(std::istream&);


private:
        CommandLine() = default;
        ~CommandLine() { delete table; }
        CommandLine(const CommandLine&) = delete;
        CommandLine& operator=(const CommandLine&) = delete;
```

```
void open(const std::string&);
void save() const;
void saveAs(const std::string&)const;
void print() const;
void edit(std::string&);
void exit(std::istream&);
Commands findCommand(std::string&);
void execute(std::istream&);
void printInfo()const;

void toLowerCases(std::string&);

Table* table = nullptr;
std::string input;
std::string file;
bool hasBeenChanged = false;
bool mustEnd = false;
};
```

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

Направен е тестови пример – example.txt със следното съдържание :

1.34, ,45, , , , ,

, ,=5>=6, , , , ,
"Hello world!",=R2C3/R4C3,=34*23, , , , ,
, , , , , , ,566.7
=34/0, ,=R6C2-R3C2,"\"Quoted\"", , ,=-20/-5,
"Ivan Petrov",=R1C1+R1C3, , , , , ,

, , , , , , ,
123,=-5/23, , , , , ,

- **Глава 5. Заключение**

5.1. Обобщение на изпълнението на началните цели.

Приложението притежава всички нужни функции: може да зарежда таблица от файл, да извършва промени върху нея, да съхранява направените промени и да я принтира.

5.2. Насоки за бъдещо развитие и усъвършенстване.

Приложението трябва да може да добавя нови типове клетки, които разширяват функционалността на съществуващите. Освен това, може да се добави

възможност за сортиране на данните по определен показател и да се реализира пресмятане на формули, които съдържат повече от една операция.