Курсов Проект ООП – Част 1

ДОКУМЕНТАЦИЯ

Явор Йорданов Чамов | СИТ 1 б) | №21621577

Съдържание

Въведение

Приложената документация има за цел да даде разяснения относно начинът, по който е имплементирано решението на ***Проект 1: Приложение за работа с електронни таблици.***

Разясненията включват описание и идея на проекта, преглед на предметната дейност, етапи на проектиране, обща структура на проекта, класови диаграми, както и входни и изходни данни с цел тестване.

За детайли и обяснения към всеки клас, метод, и член данни погледнете генерираният Javadoc.

Описание и идея на проекта

Проектът може да бъде разделен на две главни части.

Първата част включва изработката на козолен интерфейс, чрез който потребителят взаймодейства с програмата.

Втората част включва разработката на бизнес заданието и логиката на проекта.

CLI (КОНЗОЛЕН ИНТЕРФЕЙС)

Програмата е проектирана да предоставя удобен за потребителя начин за отваряне, редактиране и запазване на файлове с електронни таблици.

Тази програма позволява да отворите файл, да извършите различни операции върху него и след това да запазите промените обратно в същия файл или в нов файл по ваш избор.

Ако решите да затворите файла без запис, тази програма предоставя удобна опция за "затваряне".

Следващите раздели ще предоставят подробни инструкции как да използвате програмата и всички нейни функции.

СЪЗДАВАНЕ И МАНИПУЛИРАНЕ НА ТАБЛИЦА

Всяка електронна таблица се прочита от файл с разширение *.csv* и се зарежда в паметта.

След това посредством CLI команди могат да се извършват различни операции върху таблицата.

Таблицата поддържа всички CRUD операции.

- Създаване / добавяне на нови клетки с данни
- Прочитане на съществуващи клетки с данни
- Редакатиране на съществуващи клетки с данни
- Изтриване на съществуващи клетки с данни

Преглед на предметната дейност

ОСНОВНИ ДЕФИНИЦИИ

В тази част ще разгледаме основните „единици“ (обекти, компоненти) при реализацията на бизнес заданието.

Таблица (TableCell[][])

Таблицата представлява двумерен масив, който може да съдържа различни типове данни.

Той е предназначен да позволи на потребителите да съхраняват и манипулират данни в табличен формат, подобен на електронна таблица.

Тази единициа е централен компонент на приложението, която позволява на потребителите да отварят, редактират и записват таблични данни.

Клетка от таблицата (TableCell)

Класът TableCell представлява една клетка в таблица. Всеки обект TableCell съдържа стойност и тип.

Възможните типове са:

- Цяло число
- Дробно число
- Низ
- Формула
- „Празна“ клетка

Калкулатор за формули (FormulaCalculator)

Класът FormulaCalculator е помощен клас, използван при реализацията на приложението.

Той отговаря за извършването на основни математически изчисления върху стойности, въведени в таблицата.

Хранилище на таблицата (TableRepository)

Класът TableRepository е отговорен за управлението на състоянието на данните в таблицата.

Той съхранява текущото състояние на таблицата в двуизмерен масив от обекти TableCell и предоставя методи за извличане и модифициране на данните.

Всички части на приложението работят с едни и същи таблични данни.

В допълнение към управлението на данните от таблицата, класът TableRepository също предоставя методи за отваряне и затваряне на файлове с таблици и за запазване на промените в данните от таблицата обратно във файла.

Четец на таблица (TableReader)

Класът TableReader отговаря за четенето на таблица от файл и попълването на хранилището с данните.

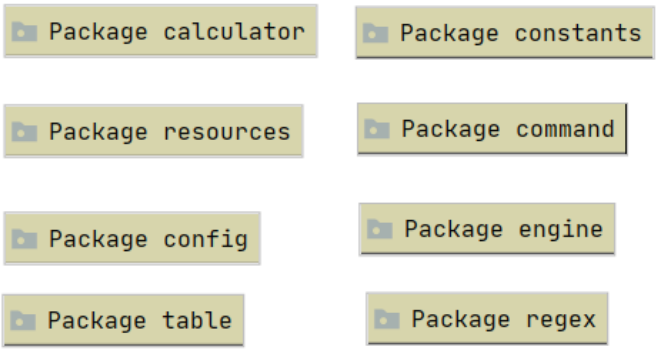
Писач на таблица (TableWriter)

Класът TableWriter отговаря за записването на съдържанието на таблица във файл.

Проектиране

ПАКЕТИ

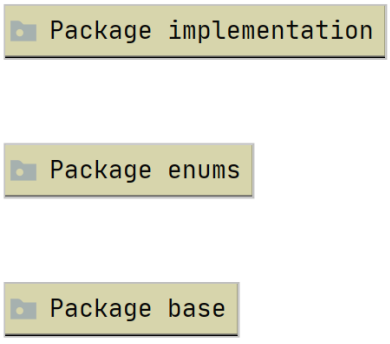
Проектът има следната пакетна структура:



Пакетът "калкулатор" съдържа класа FormulaCalculator, който е отговорен за изчисляването на математически изрази и формули.

Пакетът с ресурси съдържа файловете с данни, използвани от програмата. Тези файлове с данни са под формата на .csv файлове и представляват таблици, които могат да се отварят, модифицират и записват с помощта на програмата. Името на файла по подразбиране е data.csv и се съхранява в пакета с ресурси.

Пакетът „command” е разделен на три подпакета



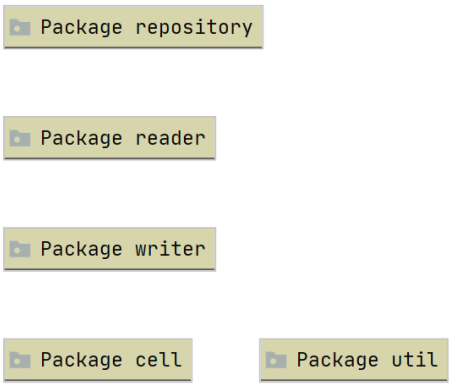
- base – съдржа интерфейси и базови класове за създаване на команди
- implementation съдържа конкретна имплементация на командите
- enums съдържа списък с валидните команди

Пакетът "config" съдържа клас за управление на конфигурацията на приложението.

Пакетът “regex” съдържа Regex шаблоните, използвани в приложението.

Пакетът “constants” съдържа всички изходни съобщения.

Пакетът „table” е разделен на пед подпакета



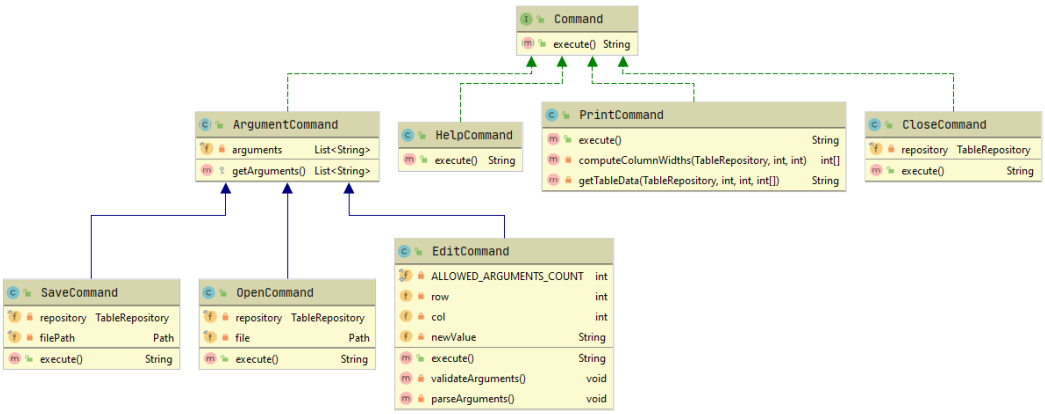
- Пакетът “repository”съдържа класа TableRepository, който отговаря за управлението на състоянието на таблицата и файла, с който е свързана.
- Пакетът “reader” съдържа класа TableReader, който отговаря за четенето на данни от текстов файл и преобразуването му в обекти TableCell.
- Пакетът "writer" съдържа имплементацията на класа TableWriter. Този клас предоставя функционалност за запис на съдържанието на таблица във файл.
- Пакетът "cell" съдържа класа TableCell, който представлява една клетка в таблица на електронна таблица.
- Пакетът “util” съдържа класа CellTypeUtil, който предоставя помощни методи, свързани с изброяването на CellType. Класът CellTypeUtil е отговорен за предоставянето на методи, които помагат при идентифицирането на типа на стойността на клетката.

КЛАСОВЕ

Проектът е разделен на няколко класови йерархии.

- Йерархия на командите
- Йерархия на шаблонът “*Abstract Factory”*
- Йерархия на класовете за четене и запис

Йерархия на командите



Йерархията на показаните класовете е свързана с модела на командите, който е модел на поведенчески дизайн, който позволява да дефинирате семейство от команди, да капсулирате всяка една и да ги направите взаимозаменяеми.

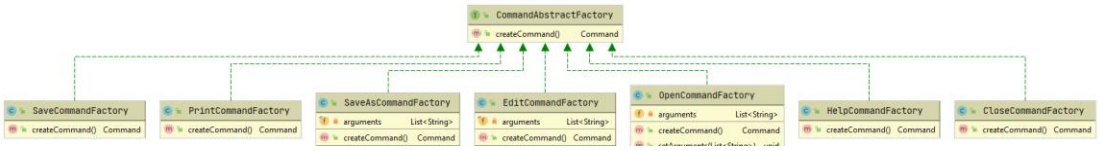
В горната част на йерархията имаме команден интерфейс, който дефинира метода за изпълнение, който трябва да бъде приложен от всяка конкретна команда.

Абстрактният клас ArgumentCommand разширява командния интерфейс и предоставя имплементация за анализиране на аргументите, изисквани от неговите подкласове. Неговите подкласове, EditCommand, SaveCommand и OpenCommand, са конкретни команди, които изискват допълнителни аргументи за изпълнение.

Класовете HelpCommand, PrintCommand и CloseCommand са конкретни команди, които не изискват допълнителни аргументи за изпълнение, следователно не наследяват от ArgumentCommand.

Тази йерархия позволява отделянето на командите от логиката, която ги изпълнява, което улеснява добавянето на нови команди в бъдеще, без да се променя съществуващият код.

Йерархия на шаблонът *“Abstract Factory”*



Тази йерархия на класове имплементира шаблона за проектиране на фабричния метод.

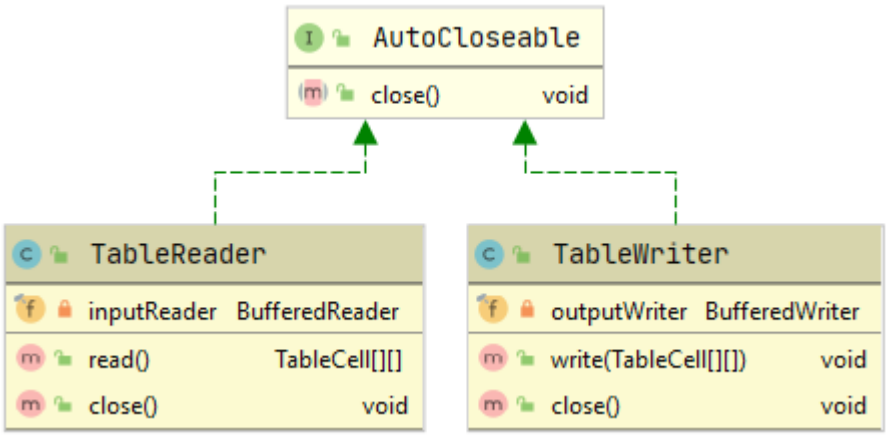
Интерфейсът AbstractCommandFactory е абстрактната фабрика, която дефинира метода createCommand, който връща обект Command.

Конкретните фабрични класове HelpCommandFactory, PrintCommandFactory, CloseCommandFactory, EditCommandFactory, SaveCommandFactory, OpenCommandFactory и SaveAsCommandFactory всички имплементират интерфейса AbstractCommandFactory и всеки замества метода createCommand, за да създаде конкретен команден обект.

Тези конкретни фабрични класове са отговорни за създаването на екземпляри на HelpCommand, PrintCommand, CloseCommand, EditCommand, SaveCommand, OpenCommand и SaveAsCommand, съответно чрез внедряване на метода createCommand на интерфейса AbstractCommandFactory.

Този подход ни позволява да капсулираме създаването на командни обекти и гарантира, че създаването на всяка команда се обработва от съответния фабричен клас.

Йерархия на класовете за четене и запис



Интерфейсът AutoCloseable е част от Java API и предоставя начин за автоматично освобождаване на ресурси, когато вече не са необходими. Интерфейсът декларира един метод close(), който освобождава всички ресурси, използвани от обект, който го прилага.

Класовете TableWriter и TableReader реализират интерфейса AutoCloseable, което означава, че те могат да се използват в *“try-with-resources”* блок, за да се гарантира, че всички ресурси, които използват (напр. файлови потоци), се освобождават автоматично при излизане от блока.

Класът TableWriter отговаря за записването на данни във файл, докато класът TableReader е отговорен за четенето на данни от файл. И двата класа използват съответно класовете BufferedWriter и BufferedReader за четене и запис на данни от/във файл. Тъй като и двамата обработват файлов I/O, важно е да се уверим, че всички ресурси, които използват, са правилно освободени, за което служи интерфейсът AutoCloseable.

Работа с прогарамата. Тестване. Входни и изходни данни.

CSV файл	CLI команда	Изход	Коменар
1, 2, 3, 4, 5 1.1, -2.3, 4.6 -5, -5.4, 13 1	open	Table opened successfully.	Отваряне на таблица с валидни целочислени и дробни данни. Положителни и отрицаелни.
	print	| 1 | 2 | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | | | | -5 | -5.40 | 13 | | | | 1 | | | | |	Отпечатване на таблицата на екрана. Стойностите в първата колона са подравнени в ляво, а в останалите вдясно.
	edit 1 1 5	Cell (1, 1) updated from "1" to "5"	Промяна на елемент в ред 1 и колона 1 с нова стойност 5
	print	| 5 | 2 | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | | | | -5 | -5.40 | 13 | | | | 1 | | | | |	Отпечатване на таблицата след команда edit.
	edit 2 4 10	Cell (2, 4) updated from "" to "10"	Добавяне на елемент на някоя от празните позиции
	print	| 5 | 2 | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | 1 | | | | |	Отпечатване на таблицата след команда edit.

	edit 4 1 “Hello, World!”	Cell (4, 1) updated from "1" to "Hello, World!"	Промяна на типа на данните от целочислен към низ.
	print	| 5 | 2 | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | | | | |	Успешно променихме типа на данните в клетката.
	edit 1 1 ““Tu-Varna”“	Error editing the cell: "Tu-Varna" has unescaped quotes.	Проверка за escape-нати кавички. Вътрешните кавички не са escape-нати и получаваме съобщение за грешка.
	edit 1 1 “\”Tu-Varna\””	Cell (1, 1) updated from "5" to ""Tu-Varna""	Правилно escape-нати кавички.
	print	| "Tu-Varna" | 2 | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | | | | |	Успешно обновихме елемента на ред 1 и колона 1.
	edit 1 2 “\Test\”	Error editing the cell: \Test\ has unescaped backslash.	Проверка за escape-нати наклонени черти \
	edit 1 2 “\\Test\\”	Cell (1, 2) updated from "2" to "\Test\"	Правилно escape-нати наклонени черти.
	edit 4 2 =5+2*3	Cell (4, 2) updated from "" to "=5+2*3"	Прилагане на формула с директно зададени стойности без да се реферират клетки. Операциите се прилагат в правилен ред.

	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | 11.0 | | | |	
	edit 4 2 =42/0	Cell (4, 2) updated from "=5+2*3" to "=42/0"	Добавяне на формула с делене на 0.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | ERROR | | | |	Визуализиране на ERROR заради делението на 0.
	edit 4 3 =R2C2+R3C3	Cell (4, 3) updated from "" to "=R2C2+R3C3"	Прилагане на формула с реферирание на клетки с дробни числа.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | ERROR | 10.7 | | |	Успешно изпълнена формула с реферирани клетки.
	edit 4 4 =R4C3*2	Cell (4, 4) updated from "" to "=R4C3*2"	Формула с реферирание към клетка, която също съдържа формула.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | ERROR | 10.70 | 21.4 | |	Успешно приложена **рекурсия**.
	edit 4 5 =R4C4*R1C1	Cell (4, 5) updated from "" to "=R4C4*R1C1"	Умножаваме числена стойност с низ, който не може да се превърне в число.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | | | -5 | -5.40 | 13 | | | | Hello, World! | ERROR | 10.70 | 21.4 | 0.0 |	Получаваме 0, тъй като низът се представя като 0 по условие.

	edit 3 5 “2”	Cell (3, 5) updated from "" to "2"	Добавяме числова стойност под формата на низ.
	edit 2 5 =R3C1/R3C5	Cell (2, 5) updated from "" to "=R3C1/R3C5"	Умножаваме две числови стойности, като едната от тях е под формата на низ.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | 1.10 | -2.30 | 4.60 | 10 | -2.5 | -5 | -5.40 | 13 | | 2 | Hello, World! | ERROR | 10.70 | 21.4 | 0.0	Получаваме правилен резултат в следствие на cast.
	edit 8 8 “Scale”	Cell (8, 8) updated from "" to "Scale"	Динамично скалиране на таблицата с допълнителни редове и колони.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | | | | 1.10 | -2.30 | 4.60 | 10 | -2.5 | | | | | -5 | -5.40 | 13 | | 2 | | | | | Hello, World! | ERROR | 10.70 | 21.4 | 0.0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Scale |	Таблицата успешно скалира, за да приеме новата стойност.
	edit 8 8 “”	Cell (8, 8) updated from "Scale" to ""	Динамично намляване на таблицата до последният ред и колона, в която има стойност.
	print	| "Tu-Varna" | \Test\ | 3 | 4 | 5 | | 1.10 | -2.30 | 4.60 | 10 | -2.5 | | -5 | -5.40 | 13 | | 2 | | Hello, World! | ERROR | 10.70 | 21.4 | 0.0 |	Таблицата е успешно смалена и паметта освободена
"Hello\, World!", "Numbers	print	| Hello, World! | Numbers from 1 to 5: 1,2,3,4,5 |	Успешно escape-ваме запетаите,

from 1 to 5: 1\,2\,3\,4\,5"			които участват в низовете.
-	help	The following commands are supported: open <file> opens <file> close closes currently opened file save saves the currently open file saveas <file> saves the currently open file in <file> print prints the table formatted to the console edit <row> <col> <new value> edits the value at the given indexes with the new one help prints this information exit exists the program	Output от команда help.
-	print	print No table is currently opened.	Output при липсваща отворена таблица.
-	save	Table saved to file "src\bg\tu_varna\sit\b1\f21621577\resources\data.csv"	Output при успешна save операция.
-	saveas	saveas data2.csv Table saved to file "src\bg\tu_varna\sit\b1\f21621577\resources\data2.csv"	Output при успешна saveas опеарация.