Технически университет – Варна

**Курсов проект**

по ООП – I част

Изготвил: Здравко Иванов Иванов, СИТ, IIк., 4гр., фн: 20621596

Ръководител: доц. Х.Ненов

1.Увод

**-Описание и идея на проекта**

Да се реализира програма, поддържаща операции с прости бази от данни. Базите данни

се състоят от серии от таблици, като всяка таблица е записана в собствен файл. Базата

данни е записана в главен файл (каталог), които съдържа списък от таблиците в базата

данни, като за всяка таблица е зададено име и файл, в който таблицата е записана.

**-Цел и задачи на разработката**

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

|  |
| --- |
| **import <file name>** (Добавя в базата данни нова таблица от файл. Във файла е записана информация за типа на всяка колона. Всяка таблица има име. При опит за зареждане на таблица с име, което съвпада с името на някоя вече заредена таблица, системата да дава грешка. Добавената таблица се записва в каталога на базата от данни.) |
| **showtables** (Показва списък с имената на всички заредени таблици.) |
| **describe <name>** (Показва информация за типовете на колоните на дадена таблица.) |
| **print <name>** (Показва всички редове от дадена таблица. Да се реализира диалогов режим, позволяващ съдържанието на таблицата да се преглежда по страници (такива, че се събират на един екран) със следните команди: следваща страница, предишна  страница, изход.) |
| **export <name> <file name>** (Записва таблица във файл.) |
| **select <column-n> <value> <table name>** (Извежда всички редове от таблицата, които съдържат стойността “value” в клетката с дадения пореден номер. Да се реализира извеждане по страници.) |
| **addcolumn <table name> <column name> <column type>** (Добавя нова колона (с най-голям номер) в дадена таблица. За всички съществуващи редове от таблицата, стойността на тази колона да е празна.) |
| **update <table name> <search column n> <search value> <target column n> <target value>** (За всички редове в таблицата, чиято колона с пореден номер <search column n> съдържа стойността <search column value> се променят така, че колоната им с пореден номер <target column n> да получи стойност <target value>. Да се поддържа стойност NULL.) |
| **delete <table name> <search column n> <search value>** (Изтрива всички редове в таблицата, чиято колона<search column n> съдържа стойността <search column value>.) |
| **insert <table name> <column1> … <column n>** (Вмъква нов ред в таблицата със съответните стойности.) |
| **rename <old name> <new name>** (Преименува таблица. Отпечатва грешка, ако новото име не е уникално.) |
| **count <table name> <search column n> <search value>** (Намира броя на редовете в таблицата, чиито колони съдържат дадената стойност) |

**-Структура на документацията**

1. Увод

2. Преглед на предметната област

3. Проектиране

4. Реализация, тестване

5. Заключение

2.Преглед на предметната област

- **Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

Използвах **Singleton Pattern** при реализацията на всички класове отговарящи за изпълнението на командите в програмата(класовете от пакет “commands”) както и за основния клас “CLI”. Използвах и **Factory Pattern** за връщане на типа на всяка колона от таблиците в класа “StringToType”. Опитах се да приложа и SOLID принципите. За реализацията на четене и запис от/във файл с разширение .xml използвах XMLEncoder и XMLDecoder.

**- Подходи, методи за решаване на поставените проблемите**

При извеждането на всички редове от дадена таблица чрез метода “print <name>” по изискване е зададено информацията да бъде представена на потребителя на отделни страници, но в заданието не е упоменато колко реда от една таблица трябва да съдържа една страница. Заради това, аз реших да задам ограничение от 5 реда на всяка страница с помощта на константата “rowPerPage”, която се намира именно в класа “Print”. Също така съм използвал клас “DataBase” представляващ база от данни съдържаща в себе си всички евентуални таблици с информация, които потребителя би въвел. Таблиците и информацията в тях се съдържат в класа под формата на списък (List<Table> tables). В класа “TableColumn” отговарящ за колоните в таблиците използвах списък List<Object> съдържащ данните от всяка колона. Списъка съдържа тип Object, за да може да поддържа и 3те типа съответстващи по заданието (int,double,string).

3.Проектиране

**-Обща структура на проекта и пакети който ще се реализират**

За да бъде по-добре разбираем кода, разделих проекта в 5 отделни пакета.

**bg.tu\_varna.sit-** Това е основният пакет на програмата. В него се съдържат основните класове както и входната точка на проекта.(CLI, ColumnType, DataBase, DataBaseFileNames, Main, Row, StringToType, Table, TableColumn, Column, DataBaseFile)

**commands**- Това е пакет, който е ключов за целия проект, тъй като съдържа в себе си всички класове необходими за изпълнението на командите, които са планирани в заданието. (AddColumn, Close, Count, Delete, Describe, Export, Help, Import, Insert, Open, Print, Rename, Save, SaveAs, Select, ShowTables, Update)

**commandsinterface-** Този пакет съдържа в себе си само интерфейси с методи, които класовете от предходно споменатия пакет изпълняват, за да се спази DIP принципа. (AddColumn, Close, Count, Delete, Describe, Export, Help, Import, Insert, Open, Print, Rename, Save, SaveAs, Select, ShowTables, Update)

**exceptions-** Това е пакет, който предназначих за съхранение на грешки. В този пакет има само 1 клас “DatabaseException”, който представлява “custom exception”, който се продуцира при възникване на грешка в програмата.

**fileaccess-** Това е пакет, който е ключов за проекта по отношение на работата му с файлове, тъй като съдържа в себе си класът “DatabaseFile”, който е отговорен за четенето и записът от и във файлове с раширение .xml .

4.Реализация и тестване

**Main**-Това е класът представляващ т.нар. входна точка на програмата. Създава се инстанция на класа “CLI” и се извиква метод “start”, след което програмата започва да работи.

CLI cli = CLI.*getInstance*();  
cli.start();

**CLI**-Това е основният клас на програмата обединяващ в себе си всички функционалности и обработващ всички действия на потребителя в хода на изпълнението на програмата. В него приложих Singleton Pattern с което се гарантира една единствена инстанция на класа независимо колко пъти се използва програмата от потребителя. Също така този клас съдържа всички методи за изпълнението на всяка команда като всеки отделен метод използва отделна инстанция на отделен клас съответстващ дадена команда. Два от ключовите методи на класа са методите : “command” и “start”. Методът “start” позволява на потребителя да въведе желаната команда като до приключване на програмата този метод посредством цикъл do-while изпълнява чрез извикване другия метод “command”, който от своя страна се състои в един “switch” оператор,който по зададена команда от потребителя извиква съответния метод съдържащ в себе си инстанция на клас на съответната опция, която потребителя желае програмата да изпълни.

public void **start**(){  
 Scanner scanner = new Scanner(System.*in*);  
 String[] input;  
 do{  
 System.*out*.print('>');  
 input = split(scanner.nextLine().trim());  
 if(input[0].equals("exit"))  
 return;  
  
 try {  
 command(input);  
 }catch (DatabaseException e){System.*out*.println("Error:"+e.getMessage());}  
 }while (true);  
}

**DataBase-** Това е класът наподобяващ база данни, съдържаща в себе си посредством списък всички таблици с информация.

private DatabaseFileNames databaseFileNames;  
private String name;  
private List<Table> tables;  
  
  
public Database(String name, List<Table> tables, DatabaseFileNames databaseFileNames) {  
 this.name = name;  
 this.tables = tables;  
 this.databaseFileNames = databaseFileNames;  
}

**DatabaseFileNames**- Това е класът съдържащ имената на файловете съответстващи на таблиците.

**Row**- Това е класът отговарящ за редовете в таблицата. Данните от този клас не се записват във файл, защото този клас е предназначен само за отпечатване на редовете.

**StringToType**-Този клас приема типът на колоната зададен от потребителя като string и продуцира инстанция от дадения тип чрез “Factory Pattern” в зависимост от това дали е int, double или string.

**Table**- Този клас отговаря за всяка една таблица като съдържа в себе си атрибут за име на таблицата и списък от колони.

**TableColumn**-Този клас отговаря за колоните на една таблица. Съдържа в себе си списък от стойностите на една колона както и име на колоната и тип.

**ColumnType**- Този клас всъщност представлява “enum” съдържащ в себе си типовете колони на една таблица. (INT, DOUBLE, STRING)

**Column**-Това е интерфейс с методи предназначен за класа TableColumn, който всъщност имплементира този интерфейс.

**AddColumn-** Това е класът отговорен за реализацията на команда “addcolumn” в програмата. Именно този клас създава таблица ако не съществува такава и добавя колона към нея заедно с име и тип на самата колона. В този клас приложих “Singleton Pattern”.

**Close**- Това е класът отговорен за реализацията на команда “close” в програмата. Този клас изпълнява реализацията по затварянето на даден файл в програмата. В този клас също приложих “Singleton Pattern”.

**Count**- Това е класът отговорен за реализацията на команда “count” в програмата. Този клас съдържа метод, който намира броя на редовете в таблицата, чиито колони

съдържат дадената стойност. В този клас също приложих “Singleton Pattern”.

**Delete-** Това е класът отговорен за реализацията на команда “Delete”. Този клас съдържа метод, който изтрива всички редове в таблицата, чиято колона

<search column n> съдържа стойността <search column value>. В този клас също приложих “Singleton Pattern”.

**Describe-** Това е класът отговорен за реализацията на команда “Describe”. Този клас съдържа метод, който показва информация за типовете на колоните на дадена таблица. В този клас също приложих “Singleton Pattern”.

**Export-** Това е класът отговорен за реализацията на команда “Export”. Този клас записва таблица във файл. В този клас също приложих “Singleton Pattern”.

**Help-** Това е класът отговорен за реализацията на команда “Help”. Този клас съдържа метод, който извежда на екрана всички налични командни, които програмата поддържа и могат да бъдат зададени от потребителя. В този клас също приложих “Singleton Pattern”.

**Import-** Това е класът отговорен за реализацията на команда “Import”. Този клас съдържа метод, който добавя в базата данни нова таблица от файл. Във

файла е записана информация за типа на всяка колона. Всяка таблица има име. При опит за зареждане на таблица с име, което съвпада с името на някоя вече заредена таблица, системата дава грешка. Добавената таблица се записва в каталога на

базата от данни.. В този клас също приложих “Singleton Pattern”.

**Insert-** Това е класът отговорен за реализацията на команда “Insert”. Този клас съдържа метод, който вмъква нов ред в таблицата със съответните стойности. В този клас също приложих “Singleton Pattern”.

**Open-** Това е класът отговорен за реализацията на команда “Open”. Този клас съдържа метод, който отваря файл с база данни. В този клас също приложих “Singleton Pattern”.

**Print-** Това е класът отговорен за реализацията на команда “Print”. Този клас съдържа метод, който показва всички редове от дадена таблица. Реализира диалогов режим, позволяващ съдържанието на таблицата да се преглежда по страници (такива, че се събират на един екран) със следните команди: следваща страница, предишна страница, изход. В този клас също приложих “Singleton Pattern”.

**Rename-** Това е класът отговорен за реализацията на команда “Rename”. Този клас съдържа метод, който преименува таблица. Отпечатва се грешка, ако новото име не е уникално. В този клас също приложих “Singleton Pattern”.

**Save-** Това е класът отговорен за реализацията на команда “Save”. Този клас съдържа метод, който записва информацията за таблици във файл с .xml разширение наподобяващ база от данни. В този клас също приложих “Singleton Pattern”.

**SaveAs-** Това е класът отговорен за реализацията на команда “SaveAs”. Този клас съдържа метод, който записва информацията за таблици във файл с .xml разширение наподобяващ база от данни. В този клас също приложих “Singleton Pattern”.

**Select-** Това е класът отговорен за реализацията на команда “Select”. Този клас съдържа метод, който извежда всички редове от таблицата, които съдържат стойността “value” в клетката с дадения пореден номер. Реализирах и извеждане по страници. В този клас също приложих “Singleton Pattern”.

**ShowTables-** Това е класът отговорен за реализацията на команда “ShowTables”. Този клас съдържа метод, който показва списък с имената на всички заредени таблици. В този клас също приложих “Singleton Pattern”.

**Update-** Това е класът отговорен за реализацията на команда “Update”. Този клас съдържа метод, който за всички редове в таблицата, чиято колона с пореден номер <search column n> съдържа стойността <search column value> се променят така,

че колоната им с пореден номер <target column n> да получи стойност <target value>. Поддържа и стойност NULL. В този клас също приложих “Singleton Pattern”.

**DatabaseFile-** Това е един от ключовите класове за целия проект. Този клас съдържа в себе си 4 метода отговорни за четенето,записа,експортирането и импортирането на таблици с информация в тях от и във файлове с разширение .xml. За целта използвах “XMLEncoder” и “XMLDecoder”.

Тестване

**addcolumn <table name> <column name> <column type>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**insert <table name> <column 1> … <column n>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**print <table name>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**count <table name> <search column n> <search value>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**rename <old name> <new name>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**describe <table name>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**select <column-n> <value> <table name>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**showtables**

Картина, която съдържа текст

Описанието е генерирано автоматично

**update <table name> <search column n> <search value> <target column n> <target value>**

Картина, която съдържа текст

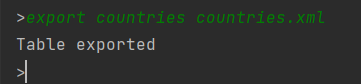
Описанието е генерирано автоматично

**delete <table name> <search column n> <search value>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**export <name> <file name>**



**save / saveas**

Картина, която съдържа текст

Описанието е генерирано автоматично

**close**

Картина, която съдържа текст

Описанието е генерирано автоматично

**open**

Картина, която съдържа текст

Описанието е генерирано автоматично

**import <file name>**

Картина, която съдържа текст

Описанието е генерирано автоматично

**help**

Картина, която съдържа текст

Описанието е генерирано автоматично

**exit**

Картина, която съдържа текст

Описанието е генерирано автоматично

5. Заключение

В проекта съм реализирал Singleton Pattern, Factory Pattern както и SOLID принципите. Използвах и колекции (List, Map) при реализирането на повечето команди. Постарах се да структурирам пакетите заедно с класовете и интерфейсите в тях така, че да може кода да се разбира и чете по-лесно.

**Източници:**

<https://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html>

<https://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html>

<https://www.geeksforgeeks.org/list-interface-java-examples/>

<https://www.geeksforgeeks.org/map-interface-java-examples/>