**Технически университет – Варна**

*Факултет по изчислителна техника и автоматизация*

*Софтуерни и интернет технологии (СИТ)*

**Курсов проект по  
Обектно-ориентирано програмиране II част**

на

**Даниел Йорданов Иванов**

*Фак. Номер: 20621570*

**Задание на проекта:**

**Т10: Студент**

Да се напише програма, реализираща информационна система за обслужване на студенти. Програмата съхранява и обработва необходимите данни във файл.

За всеки студент се съхранява следната информация:

● Име

● Факултетен номер

● Текущо записан курс, специалност, група

● статус (записан, прекъснал, завършил)

● Среден успех от следването до момента

И поддържа следните функционалности :

● добавяне на оценки/изпити (за даден предмет, оценка и студент)

● записване на задължителни/избираеми предмети

● отпечатване на протоколи

● академична справка за оценките на даден студент

**Идея и цел на проекта**

Изработеният проект представлява *Java* програма, реализираща информационна система за обслужване и организация на студенти. Системата съхранява списък със студенти и специалности и техните основни академични характеристики. Потребителят има достъп до системата чрез диалогови команди и има възможност за извършване на множество различни операции с наличните данни. Съхраняването и обработването на тези данни се извършва във външен *xml* документ.

Разработката има за цел да осигури на клиент потребителя една цялостна информационна система за студенти, специалности и дисциплини, симулираща университетска среда. Функционалностите достъпни за потребителя са основните необходими за реализирането на изцяло работеща система, към които спадат добавяне и манипулиране на студенти и техните налични данни, както и извеждане на справки и протоколи от различен характер за вече записаните студенти. В системата се съхранява основна информация за всеки студент (име, факултетен номер, статус, специалност, курс и група), както и по-специфична академична информация като оценки от изпити на записаните дисциплини, среден успех и налични кредити. Всяка специалност се характеризира със свой собствен набор от налични дисциплини (предмети), които могат да бъдат задължителни или избираеми (носещи кредити, нужни за успешно дипломиране). Тези предмети поддържат записване в няколко възможни курса (например 2, 3 и 4) в зависимост от предварително зададените им от администратора характеристики.

Тази документация ще разгледа в дълбоки детайли реализацията и работата на създадената система. Структурно тя е разделена на смислови части или глави. Следващите глави ще разгледат предметната област, проектирането, реализацията и тестването на цялостната задача. В документацията също ще бъдат приложени различни фигури, диаграми и схеми, които да спомогнат за предоставяне на информацията по един прост и леснодостъпен за разбиране начин.

**Преглед на предметната област**

Предметната област, застъпена в проекта е строго академична. Основните концепции, чрез които е осъществена са студент, специалност и дисциплина. Студентът е основната структурна единица, която стои в центъра на реализацията. Всички операции и функции в проекта пряко или косвено извършват манипулации и обработка на структурата. Останалата структурна част на системата е съставена от специалностите и техните дисциплини, които не са конкретно зададени в изискванията на заданието. Поради тази причина те са симулирани и изработени в умален вариант като за целта са използвани вече съществуващи университетски информационни системи. Алгоритмите, които ще бъдат използвани са свързани основно с обработка (сортиране, обхождане) на колекции от данни (студенти и специалности). В множество случаи се използвани различни по вид условни проверки и циклични операции, необходими за тази обработка.

За създаването на системата и справяне на зададените проблеми са използвани фундаменталните концепции на обектно-ориентираното програмиране, спазени са *SOLID* принципите и са използвани няколко шаблона за дизайн (*singleton* и *factory method*). За създаване на диалогова връзка с потребителя се работи с командния ред, като е създаден команден интерпретатор в опростен вид, който приема зададените команди и връща резултат с извършеното действие към конзолата.

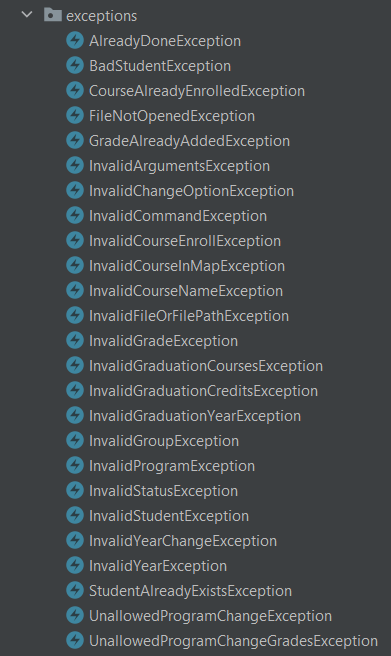
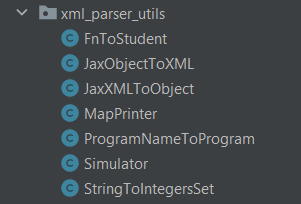
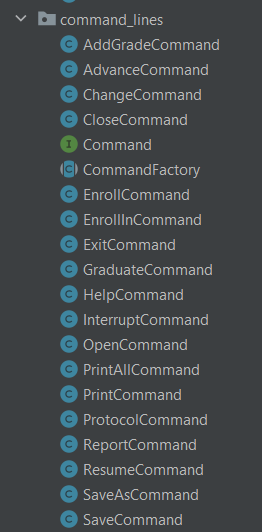
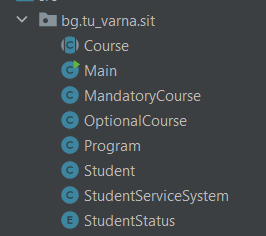
Изискванията към потребителя са да е запознат с командите (като за това може да се използва команда *help*) и със специалностите и дисциплините, с които работи средата (за симулираните такива има създаден текстов файл с направения учебен план на студентите). Необходимата поддръжка е актуализиране на този учебен план при нужда и добавяне на нови студенти в системата.

**Проектиране**

Тук ще разгледаме общата структура на системата. Проектът е разделен в 4 пакета, чиито класове са изобразени на *Фигура 1*:

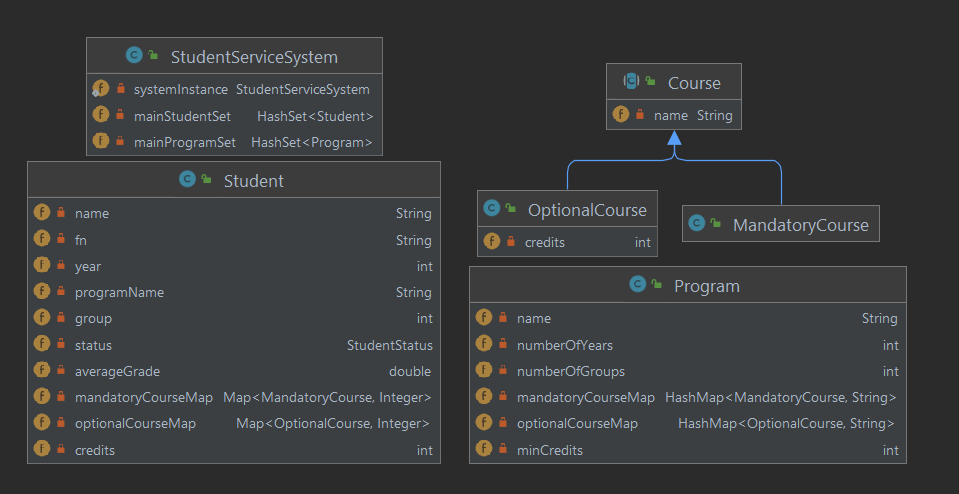
* Основен пакет **(bg.tu\_varna.sit)** с класове на основните структурни единици на системата: Студентска система, Студент, Специалност, Дисциплина (абстрактен), Задължителна дисциплина, Избираема дисциплина, Статус (енумерация) и *Main.*
* Пакет с команди **(command\_lines)**. В него са реализирани класовете с всички команди достъпни на потребителя, както и абстрактен клас ***CommandFactory***, който чрез условна проверка връща нова инстанция на класа от извиканата команда. Също така пакета съдържа и интерфейс ***Command***, който е нужен за работата на *Factory Method* и обединява функциолността на всички команди.
* Пакет с *xml parser* и помощни класове **(xml\_parser\_utils)**. В този пакет са реализирани класовете нужни за работата на *jaxb* *(Java Architecture for XML Binding) parser*, както и множество помощни класове със статични методи с цел да се използват глобално и многократно в кода. Също така в този пакет и поместен клас *Simulator*, чрез който се осъществява симулирането на специалностите при отваряне на нов файл за работа със системата.
* Пакет с грешки – изключения **(exceptions)**. Това е пакет с 25 класа изключения с подходящи съобщения за грешка при неправилни входни данни или други нередности, които могат да се случат при действието на програмата.

Фигура 1



Също така в проекта са прикачени 2 примерни файла с данни **try.xml** и **backup.xml**, както и текстов файл с учебния план на симулираната система с 3 специалности и 27 дисциплини **Curriculum.txt**. В него може да се види, че всички избираеми, както и някои задължителни дисциплини могат да бъдат записани в няколко различни курса.

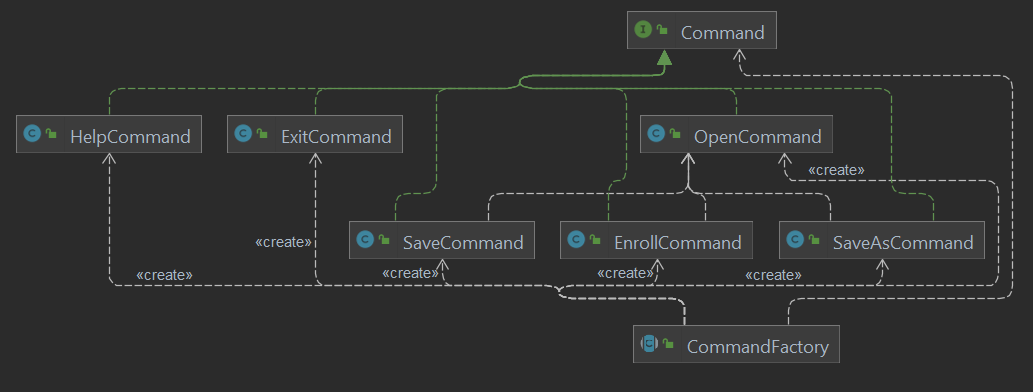
Обобщената UML диаграма на основните структури с техните полета изглежда така.



Фигура 2

На *фигура 2* виждаме, че класът на основната система ***StudentServiceSystem*** използва шаблона за дизайн *Singleton*, като по този начин се позволява единствена инстанция на класа. Това се прилага, така че при отваряне на външен файл с данни, информацията се зарежда в паметта чрез тази инстанция в двете колекции от тип *HashSet* – едната за специалностите *(mainProgramSet)*, а другата за студентите *(mainStudentSet)*. Всяка специалност има брой курсове и брой групи като по подразбиране са зададени на съответно 4 курса и 5 групи.

Що се отнася за функционалността и командите на *фигура 3* виждаме UML схема с връзките и зависимостите между отделните команди и реализирането на *Factory Method*.



Фигура 3 \*За опростяване на схемата са премахнати голяма част от командите.

Основният клас ***Main***служи за връзка с потребителя, като единствената му функция е да осъществи един безкраен цикъл, който приема команди, обработва ги като взима основната команда и нейните аргументи, и извежда съответните грешки при нужда.

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter command:");

while(true){

try {

System.out.print(">");

String userInput = scanner.nextLine();

String[] splittedInput = userInput.split("[ ]+");

String commandName = splittedInput[0];

Object[] commandArgs = new String[splittedInput.length-1];

System.arraycopy(splittedInput,1,commandArgs,0,commandArgs.length);

Command command = CommandFactory.getCommand(commandName);

if(command!=null){

command.execute(commandArgs);

}

else {

throw new InvalidCommandException();

}

}

catch (Exception e) {

System.out.println(e);

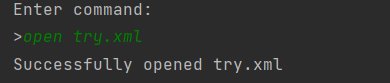
}

}

**Реализация, тестване**

За да представим обобщено работата на всеки един клас и всяка една команда най-добре би било да създаден пример със зареждане на системата, добавяне на един нов студент, преминавайки през всички курсове на специалността си и завършвайки с дипломирането му. Програмата би „хвърлила“ различна грешка при каквито и да е неправилно въведени входни данни (неправилна команда, неправилен брой аргументи на командата, неправилен формат и т.н.) като това се проверява при всяка една от командите. На *фигури от 4 до 9* са представени примери от работата на командите.

Нека стартираме програмата и заредим файл **try.xml** чрез команда **open**.



Фигура 4

Така се създаде нов обект от клас ***OpenCommand*** и съдържанието на файла се зареди в паметта чрез *jaxb parser* и инстанцията на класа ***SystemServiceSystem***. В класа ***OpenCommand*** има поле с булева стойност *openedFile*, коeто приема стойност истина при успешно отваряне на файла. Ако то има стойност лъжа не се позволява използването на други команди освен *open*, *help* и *exit*.

StudentServiceSystem.setSystemInstance(JaxXMLToObject.jaxbXmlFileToObject(filePath));

openedFile=true;

Понеже в заредения файл последният студент е с факултетен номер “015” нека добавим нов студент с факултетен номер “016” в 1 група на специалност *SIT* чрез команда **enroll**.



Фигура 5

Във основната колекция от студенти е добавен нов запис като освен зададените параметри към неговия списък с неположени изпити се добавят тези от първи курс на зададената му специалност **updateMandatoryCourses()** и се изчислява средния му успех **updateAverage()** като неположените изпити според заданието се смятат за 2.

public void updateMandatoryCourses() {

for(Map.Entry<MandatoryCourse, String> current: this.getProgram().getMandatoryCourseMap().entrySet()) {

Set<Integer> yearsSet = StringToIntegersSet.stringToSet(current.getValue());

if(Collections.max(yearsSet) == this.year && !(this.getMandatoryCourseMap().containsKey(current.getKey()))) {

this.getMandatoryCourseMap().put(current.getKey(),0);

}

}

}

public void updateAverageGrade() {

int sum = 0; int br = 0;

for(Map.Entry<MandatoryCourse, Integer> current: this.getMandatoryCourseMap().entrySet()) {

br++;

if(current.getValue() == 0) {

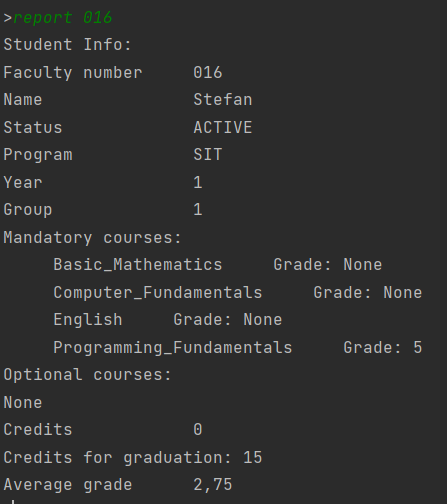
sum+=2;

} else {

sum += current.getValue();

} } this.setAverageGrade((double)sum/br);

След като сме добавили студента, за да може той да премине в следващ курс трябва да му добавим оценки към списъка с изпити (по задание до 2 изпита могат да не бъдат взети) чрез команда **addgrade**. Възможна е промяна на оценката чрез повтаряне на командата, но само при оценка 2. Можем да видим записаните изпити на студента чрез командата **report**.



Фигура 6

Цикъл за добавяне на оценка в списъка с изпити:

for(Map.Entry<MandatoryCourse, Integer> current: student.getMandatoryCourseMap().entrySet()) {

if(current.getKey().getName().equalsIgnoreCase(course) && (current.getValue()==0 || current.getValue()==2)) {

student.getMandatoryCourseMap().put(current.getKey(),grade);

student.updateAverageGrade();

isAdded=true;

System.out.println("Grade " + grade + " successfully added to course " + course + " of student with fn " + fn);

break;

}

}

След като сме добавили оценки по всички нужни изпити, студентът може да премине в следващ втори курс чрез команда **advance.** Ако той е в последния курс на специалността, то командата работи като **graduate** (ако дипломирането е възможно).



Фигура 7

Възможна е редакция на характеристиките (курс, група, специалност) на студента чрез командата **change**. Смяната на курс е възможна само ако той е следващият, а на специалност, ако новата е съвместима (няма дисциплини, чиито липсват в списъка на студента). В симулираната среда с 3 специалности (*SIT*, *CST*, *ICT*) такива в 1-ви курс са и трите, а във 2-ри курс са *SIT* и *CST*. Понеже нашият студент понастоящем е 2-ри курс *SIT* можем да го преместим във 2-ри курс *CST*.



Фигура 8

Друга възможна функционалност на системата е прекъсване/възстановяване на студентските права на студента чрез команди **interrupt/resume**. На студент с прекъснати права не могат да се прилагат команди с изключение на справки (**print, printall, protocol, report**).

В зависимост от учебния план на студента и неговия курс чрез команда **enrollin** той може да бъде записан в задължителни или избираеми дисциплини. Избираемите дисципли са нужни за получаване на кредити, които са необходими за успешно дипломиране като всяка специалност има такъв минимален брой. Списък с това кой предмет колко кредити носи и минималните за специалността могат да се проверят във файл **Curriculum.txt**. Алгоритъмът за записване на дисциплина е следният:

boolean isAdded = false;

for (Program currentProgram: StudentServiceSystem.getInstance().getMainProgramSet()) {

if(currentProgram.equals(student.getProgram())) {

for (Map.Entry<MandatoryCourse, String> currentCourse: currentProgram.getMandatoryCourseMap().entrySet()) {

Set<Integer> currentCourseYearsSet = StringToIntegersSet.stringToSet(currentCourse.getValue());

if(currentCourse.getKey().getName().equalsIgnoreCase(course) && currentCourseYearsSet.contains(student.getYear()) ) {

System.out.println("Student with fn " + fn + " enrolled in " + course + " successfully!");

student.getMandatoryCourseMap().put(currentCourse.getKey(), 0);

student.updateAverageGrade();

isAdded = true;

break;

}

}}}

След като нашият студент е в последният курс на специалност (4-ти), положил е успешно всичките си изпити и има необходимия брой кредити, той може да се дипломира чрез функция **graduate** или **advance**

Фигура 9



Друга команда, която ще разгледаме е **protocol**, коята извежда протоколи за всички студенти, които са записани в дадена дисциплина. Алгоритъмът включва създаване на ArrayList от всички записани студенти сортирани по факултетен номер. Поради нуждата за всяка специалност и курс да се извежда отделен протокол, е реализиран вложенa структура *Map* и е извършена работа с потоци:

Set<Student> studentSet = new HashSet<>();

for(Student current: StudentServiceSystem.getInstance().getMainStudentSet()) {

if(current.getMandatoryCourseMap().containsKey(mandatoryCourse)) {

studentSet.add(current);

}

}

List<Student> studentList = new ArrayList<>(studentSet);

studentList.sort(Comparator.comparing(Student::getFn));

Map<String, Map<Integer, List<Student>>> studentsByProgramYear;

studentsByProgramYear = studentList

.stream().collect(Collectors.groupingBy(Student::getProgramName, Collectors.groupingBy(Student::getYear)));

След като искаме да приключим работа със системата можем да затворим файла без запазване чрез команда **close** или да го запазим чрез команди **save/saveas**. След това се използва команда **exit**.

Като насоки за бъдещо развитие и усъвършенстване бих предложил използване на динамична база данни вместо зареждане на данните от файл. Също така е възможно разширяване на функционалността чрез създаване на команди за обработка и промяна на учебния план, както и за динамично редактиране на основните характеристики на студентите като факултетен номер и име. За обобщение може да се каже, че е реализирана една опростена, но цялостна студентска информационна система, отговаряща на всички изисквания на заданието и нуждите на клиента.

**Връзка с github -** [**https://github.com/DanielI22/StudentInformationSystem.git**](https://github.com/DanielI22/StudentInformationSystem.git)

**Източници:**

[**https://gist.github.com/ervinsh/9c3be271c8dc62e356ca**](https://gist.github.com/ervinsh/9c3be271c8dc62e356ca)

[**https://www.baeldung.com/java-groupingby-collector**](https://www.baeldung.com/java-groupingby-collector)

[**https://stackoverflow.com/**](https://stackoverflow.com/)