****

**Технически университет – Варна**

*Факултет по изчислителна техника и автоматизация*

*Софтуерни и интернет технологии (СИТ)*

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

на

**Симеон Стефанов Стоянов**

*Факултетен №: 206216735*

**Задание на проекта:**

**Т12: Личен календар**

Да се напише програма, реализираща информационна система, която поддържа личен календар, като го записва във файл.

**Глава 1. Увод**

* 1. **Описание и идея на проекта**

Проектът представлява програма, реализираща персонален календар, носещ в себе си информация за различни събития в графика на конкретния потребител. Информацията за всяко събитие се съхранява в списък с основните критерии, които са важни за характеризирането на зададеното събитие. Потребителят борави с данните чрез диалогови команди, които му позволяват да извършва различни организационни операции със събитията в неговия календар. Календарът се съхранява и обработва от външен ***xml*** файл.

* 1. **Цел и задачи на разработката**

Проектът има за цел да бъде лесен за боравене от потребителя с команди, които се доближават до водене на личен органайзер. Основната му функционалност се съдържа във въвеждането и премахването на събития от календара, като потребителят може да определя техните параметри (име на събитие, дата, начален час, краен час, бележка). Съхранените вече събития трябва да подлежат на различни сортировки, търсене на конкретни събития, коригиране на данни, извеждане на статистики по натовареност. Програмата трябва да съхранява неограничен брой събития, които след това да еднозначно да могат да бъдат намирани спрямо параметрите, които притежават. Това предполага времевия диапазон, който заема едно събитие да не се застъпва с този на друго събитие от същия календар. Също така програмата трябва да бъде способна да чете всякакви файлове, отговарящи на формата за календар, с който тя работи. Данните прочетени от файл трябва мигновено да бъдат готови за обработка от различните методи в програмата, като точно от това се ръководи полезността и лесното й управление.

* 1. **Структура на документацията**

Документацията на проекта ще бъде разделена на няколко части. Първата ще разгледа предметната област. Тя ще включва основните алгоритми и концепции, върху които е изградена програмата, ще бъдат дефинирани евентуалните проблеми, които биха се породили, както и техните решения. Следващата глава ще разгледа основното поведение на програмата и структурата на пакетите, по които са разпределени класовете на програмата. В следващата част на документацията ще бъдат направени реализация и постановки на всички методи в програмата, ще се обърне внимание на алгоритмите, които ръководят нейното действие. В заключение ще се направи оценка на цялостната функционалност на програмата и какво би помогнало за нейното цялостно подобряване

**Глава 2.Преглед на предметната област**

За да бъде реализирана основната концепция на програмата, трябваше да бъде създадена структурна единица, която обгръща цялостно елементите на едно събитие. Всеки основен метод, до който има достъп потребителят борави със събитията в програмата. С цел те да бъдат винаги достъпни, запазването им за по-късна обработка във външен файл беше от изключителна важност за ефективната работа на програмата. Алгоритмите, които бяха нужни за реализация на методите използват основно циклични и условни операции. Те извършват различни сортирания и търсения в колекции от събития, изисквани от условието на задачата.

Проблемите, които можеха да се породят при създаването на програмата бяха свързани със съхраняването на данните в правилен формат. Тъй като задачата предполага въвеждане на множество дати и часове от страна на потребителя, бяха създадени методи, които да съгласуват въведените данни с нужния формат за записване и такива, които след това да позволяват и правилното им прочитане. Също така трябваше да се създадат мерки, против припокриване на две различни събития в календара. Трябваше да бъде създадено и решение, при което потребителят свободно да може да борави командния ред. То беше реализирано с помощта на шаблон ***factory******method***и клас ***Scanner***, които да му позволяват при въвеждане на позволена команда, той да получава обратна връзка във вид на реализираната команда, която се очаква програмата да изпълни.

Изискванията към потребителя са да се стреми да спазва коректно формата на въвеждане на даден параметър, ако такъв е указан, с цел да бъдат избегнати неизчистени грешки. Изписването на командата ***help***в командния ред трябва да даде яснота на потребителя за всички команди, които е способен да извърши.

**Глава 3. Проектиране**

Класовете на проекта са разпределени в четири пакета:

Основен пакет(***bg.tu\_varna.sit)*** – в него са поместени двата основни класа, върху които е изградена реализацията на командите: ***Appointment, AppointmentsCalendar.*** Клас ***DateRangeValidator,*** който се ползва за проверяване дали зададен час е разположен във вече зает времеви диапазон и клас ***Main.***

Пакет с команди(***bg.tu\_varna.commands)*** – включва всички класове, които реализират командите въвеждани от потребителя, интерфейс ***Command,*** както и абстрактен клас ***CommandFactory,*** който връща нова инстанция на всеки клас команда, след сравнение за съвпадение между командата въведена от потребителя и съществуващите в програмата команди. Трябва да се отбележи, че командите ***OpenCommand, SaveCommand, SaveAsCommand*** служат за реализацията на ***XML parser(в случая - Java Architecture for XML Binding),*** като съответно първият осъществява четенето от файл, а другите два се грижат за записването във файл.

Пакет (***bg.tu\_varna.xmlAdapters) –*** тъй като самият ***parser*** бе реализиран в пакет ***bg.tu\_varna.commands,*** този пакет служи единствено за поместването на два класа ***DateAdapter и TimeOfDayAdapter,*** функцията на които е идентична – да вземат променлива от тип ***Date*** и да я форматират в ***String*** и обратното. Различават се по това, че променливата се форматира във формат „дд/ММ/гггг” за първия клас и във формат „ХХ:мм” за втория.

Пакет (***bg.tu\_varna.exceptions)*** – включва използваните изключения, които са били нужни в хода на програмата.

Разглеждането на структурата на програмата трябва да започне от функцията ***main***

на основния клас***:***

Public class Main{

public static void main(String[] args) throws ParseException, FileNotOpenedException {

Scanner scanner = new Scanner(System.in);

while(true){

//getting user input

System.out.print("Enter your command: \n >");

String userInput = scanner.nextLine();

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

//preparing data for creating and executing new command

String commandName = splitedInput[0];

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

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

Command command = CommandFactory.getCommand(commandName);

if(command!=null)

command.execute(commandArgs);

else

System.out.println("Bad command. Try again or use help.");}}}

В случая тя реализира един цикъл, който може да приеме безброй команди, докато не е указано напускането на програмата от потребителя или действието й не е прекъснато от открита грешка в данните.

Следващата важна стъпка в реализацията на програмата е реализирането на командите чрез ***factory method.*** Използва се класа ***CommandFactory,*** чието действие беше дефинирано в началото на тази глава. Кодът който обработва командите е следният.

public abstract class CommandFactory {

public static Command getCommand(String commandName){

if(commandName.equalsIgnoreCase("open")){

return new OpenCommand();

}else if(commandName.equalsIgnoreCase("exit")){

return new ExitCommand();

}else if(commandName.equalsIgnoreCase("help")){

return new HelpCommand();

}else if(commandName.equalsIgnoreCase("save")){

return new SaveCommand();

}else if(commandName.equalsIgnoreCase("saveas")){

return new SaveAsCommand();

} //още команди от същия вид…..

else if(commandName.equalsIgnoreCase("change")){

return new ChangeCommand();

}

else{

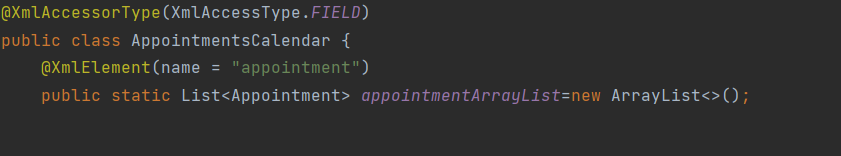
return null;

}

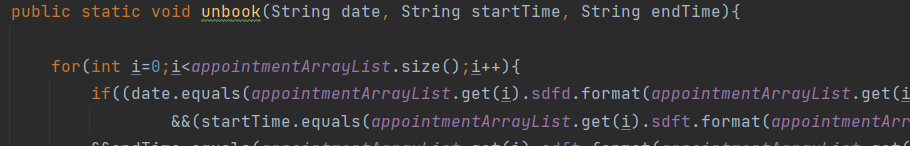
}

}

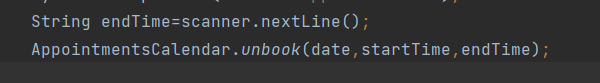
Тъй като всяка зададена от потребителя команда борави с методи обработващи инстанция на събитие, нека обърнем основно внимание на класът който съдържа всички тези методи, както и колекцията от събития, а именно ***AppointmentsCalendar.*** Колекцията от събития представлява ***Arraylist<Appointment>*** и е статична, по тази причина всички методи, боравещи с колекцията, са обявени за статични. Целта е да не се позволи инстанциране на класа Календар и методите, които командите реализират да работят директно с колекцията от срещи дефиниран а в Календара.( фиг. 3.1,фиг. 3.2,фиг. 3.3)



Фиг.3.1 Статична колекция, която съхранява всички срещи



Фиг.3.2 Един от статичните методи, с които командите работят



Фиг.3.3 Директно обръщение към статичния метод

**Глава 4. Реализация, тестване**

Класовете ще бъдат реализирани като за всеки клас, който извършва команда ще бъда илюстрирана тестова постановка и ще бъде показан основният алгоритъм, който реализира методът, заложен в изграждането на конкретната команда.

Клас ***OpenCommand –*** негова инстанция се създава при изписване на ***open*** в командния ред. Зарежда .***xml*** файл или създава нов, ако няма създаден такъв. За изпълнение на останалите команди е нужно да бъде отворен файл, в противен случай ще бъде възпроизведена грешка. Тъй като реализацията на ***xml parser*** не е обособена в отделен клас, той се реализира тук.

public class OpenCommand implements Command {

public static boolean flagOpen = false;

@Override

public void execute(Object[] args){

JAXBContext jaxbContext = null;

try {

jaxbContext = JAXBContext.newInstance(AppointmentsCalendar.class);

File file = new File("file.xml");

Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

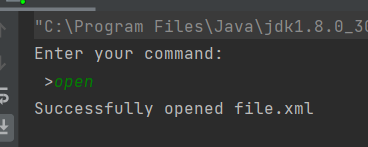
AppointmentsCalendar o = (AppointmentsCalendar) jaxbUnmarshaller.unmarshal(file);

flagOpen=true;

System.out.println(o);

System.out.println("Successfully opened file.xml");

} catch (JAXBException e) {

 e.printStackTrace();

} }

Във файл със вече заредени данни

командата изглежда така: фиг. 4.1->

Класове ***SaveCommand и SaveAsCommand –***техни инстанции се създават при изписване съответно на ***Save и SaveAs*** в командния ред. Различават се по това че първата команда запазва календара в предварително зададения файл, а във втората, чрез Скенер може да бъде въведено името на файла като трябва да се укаже и форматът му (пример – ***example.xml).*** Използването им преди отваряне на нов файл ще доведе до грешка. Тъй като реализацията на ***xml parser*** не е обособена в отделен клас, форматирането на обект в ***xml*** се реализира тук .

public class SaveCommand implements Command {

public void execute(Object[] args) throws ParseException, FileNotOpenedException {

if(!OpenCommand.flagOpen){

throw new FileNotOpenedException();

}

JAXBContext jaxbContext = null;

try {

jaxbContext = JAXBContext.newInstance(AppointmentsCalendar.class);

Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

jaxbMarshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

jaxbMarshaller.marshal(createCompanyObject(), new File("file.xml"));

System.out.println("Successfully saved in file.xml");

jaxbMarshaller.marshal(createCompanyObject(), System.out);

System.out.println("Successfully saved in file.xml");

} catch (JAXBException | ParseException e) {

e.printStackTrace();

}

}

private static AppointmentsCalendar createCompanyObject() throws ParseException {

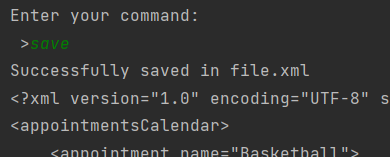
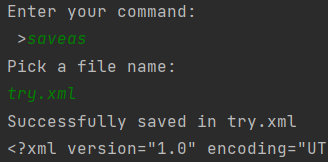
AppointmentsCalendar comp = new AppointmentsCalendar();

comp.setAppointmentArrayList(comp.getAppointmentArrayList());

return comp;

}

}

Команда Запази изглежда така: Команда Запази като изглежда така:

фиг. 4.2-> фиг 4.3->

Клас ***CloseCommand –*** инстанция на класа се създава при изписване на ***close*** в командния ред. Изчиства съдържанието на колекцията от събития и затваря файла без да запази ги запази в него.

class CloseCommand implements Command {

@Override

public void execute(Object[] args) throws FileNotOpenedException {

if(!OpenCommand.flagOpen){

throw new FileNotOpenedException();

}

AppointmentsCalendar.getAppointmentArrayList().clear();

OpenCommand.flagOpen=false;

System.out.println("Progress is cleared");

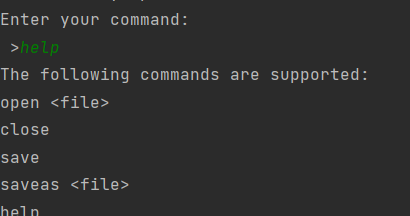
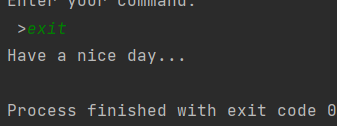
}

Клас ***HelpCommand-*** инстанция на класа се създава при изписване на ***help*** в командния ред.

Извежда низ с всичките команди в програмата и тяхното действие.(Фиг.4.4)

Клас ***ExitCommand-*** инстанция на класа се създава при изписване на ***exit*** в командния ред.

Приключва действието на програмата.(Фиг.4.5)



Фиг. 4.4 Фиг.4.5

Въвеждането на всички команди от следващите класове се осъществява със Скенери по пример на клас ***BookCommand:***

Scanner scanner = new Scanner(System.in);

System.out.println("Pick a date: dd/MM/yyyy");

String date=scanner.nextLine();

System.out.println("Start of appointment:");

String startTime=scanner.nextLine();

System.out.println("End of appointment: ");

String endTime=scanner.nextLine();

System.out.println("Name of appointment: ");

String name = scanner.nextLine();

System.out.println("Note: ");

String note=scanner.nextLine();

Appointment a=new Appointment(date,startTime,endTime,name,note);

Клас ***BookCommand -*** инстанция на класа се създава при изписване на ***book*** в командния ред. След проверка за отворен файл използва Скенер за попълване на ново събитие и след това го добавя в колекцията на Календара. Основен метод: ***addToCollection()***

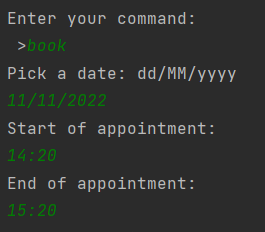
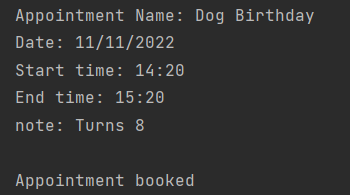
public static List<Appointment> addToCollection(Appointment appointment) throws WrongTimeException {

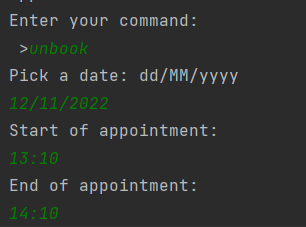
appointmentArrayList.add(appointment);

System.out.println("Booking added successfully");

return appointmentArrayList;

}



Клас ***UnbookCommand -*** инстанция на класа се създава при изписване на ***unbook*** в командния ред. Основен метод: ***unbook(date,startime,endtime)-*** в цикъл, който обхожда колекцията от събития прави съпоставка между въведените от потребителя параметри и тези на всеки елемент от колекцията. Това се случва чрез условен оператор. При срещнато съответствия елементът се премахва.



Клас ***AgendaCommand -*** инстанция на класа се създава при изписване на ***agenda*** в командния ред. Основен метод: ***agenda(date)-*** за реализацията на този метод в клас ***Appointment*** се имплементира интерфейс ***Comparable,*** за да се сравнят началните часове на всяка среща.

В метода декларираме нова колекция от тип ***Appointment.*** В цикъл с размер статичната колекция от събития се търси съвпадение между въведената от потребителя дата и събитие със същата дата. Ако има такива се добавят в новата колекция. Извън цикъла колекцията се сортира, след това се извежда и в края на метода се изтрива. (фиг. 4.10)

public static void agenda(String date){

ArrayList<Appointment>appointmentArrayList1=new ArrayList<>();

for (Appointment a:appointmentArrayList) {

if(date.equals(a.sdfd.format(a.getDate()))){

appointmentArrayList1.add(a); }}

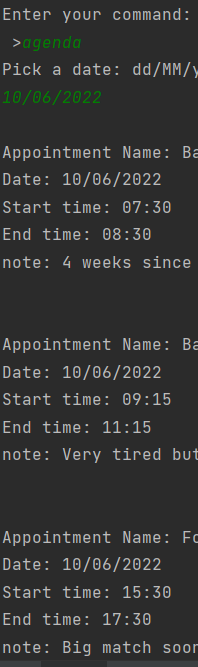
Collections.sort(appointmentArrayList1, new Comparator<Appointment>() {

public int compare(Appointment a1, Appointment a2) {

return a1.getStartTime().compareTo(a2.getStartTime());

}

});

 for (Appointment a1:appointmentArrayList1) {

System.out.println(a1);

}

appointmentArrayList1.clear(); }

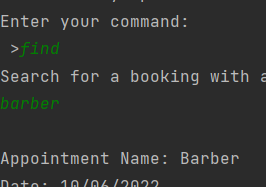
Клас ***FindCommand -*** инстанция на класа се създава

при изписване на ***find*** в командния ред.

Основен метод: ***find(occurrence)-***в цикъл с размер статичната

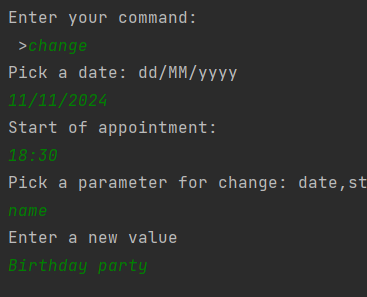
колекция се, чрез условен цикъл се търси съвпадение между

въведения от потребител текст и която и да е бележка или

заглавие на дадено събитие в колекцията (Фиг 4.11)

Фиг. 4.11

Фиг.4.10

Клас ***ChangeCommand -*** инстанция на класа се създава при изписване на ***agenda*** в командния ред. Основен метод: ***change(date,startTime,option,newValue)-*** В цикъл с размер колекцията от събития с условен оператор се проверява дата и начален час на събитие. При открито съвпадение се въвежда параметър, който да бъде променен и нова стойност за него, която да отговаря на правилния формат за съответния параметър.

public static void change(String date,String

startTime,String option,String newValue)

throws ParseException {

for

(int i=0;i<appointmentArrayList.size();i++) {

if(date.equals

(appointmentArrayList.get(i).sdfd.format

(appointmentArrayList.get(i).getDate()))

&&startTime

.equals(appointmentArrayList.get(i).sdft.format

(appointmentArrayList.get(i).getStartTime()))){

if(option

.equalsIgnoreCase("date")){

appointmentArrayList

.get(i).setDate(new SimpleDateFormat("dd/MM/yyyy").parse(newValue));}

if(//по същия начин се въвеждат нови стойности, в случай че параметърът за промяна е различен от date

**Глава 5. Заключение**

Функциите, които проектът изпълнява отговарят на условието. В бъдеще биха могли да бъдат реализирани всички функции които той трябва да извършва, както и да бъдат включени повече изключения, за да бъде сведено до минимум въвеждането на грешни данни. За разширяване на проекта могат да бъдат направени още множество различни извеждания на статистически данни и сортировки на срещите в календара. Може да се направи възможно и боравенето с повече от един календар от един потребител.

Връзка с Github: <https://github.com/SimeonStoyanov01/simeon-st-tu-varna>

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

[**https://stackoverflow.com/questions/5927109/sort-objects-in-arraylist-by-date**](https://stackoverflow.com/questions/5927109/sort-objects-in-arraylist-by-date)

[**https://beginnersbook.com/2013/12/java-arraylist-of-object-sort-example-comparable-and-comparator/**](https://beginnersbook.com/2013/12/java-arraylist-of-object-sort-example-comparable-and-comparator/)

[**https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html**](https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html)

[**https://mkyong.com/java/how-to-compare-dates-in-java/**](https://mkyong.com/java/how-to-compare-dates-in-java/)

[**https://mkyong.com/java/jaxb-hello-world-example/**](https://mkyong.com/java/jaxb-hello-world-example/)