

“Технически университет” – град Варна

Курсов проект
“Софтуерни и Интернет технологии”
Дисциплина: “ООП”
водещ лекции: доц. Хр. Ненов

Билетна каса

Цел на проекта

Да се напише програма, реализираща информационна система, която обслужва билетна каса. Програмата съхранява и обработва необходимите данни във файл.

Представленията се играят в няколко зали, всяка от които има номер, брой редове и брой места на всеки ред. Залите са предварително зададени.

Структура на документацията

- 1.Описание на проекта
- 2.Преглед на предметната област
- 3.Проектиране
- 4.Реализация и тестване
- 5.Заклучение

Преглед на предметната област

Използвани дефиниции:

В проекта са използвани шаблоните за разработка *Singleton* и *DAO*. *Singleton* гарантира, че ще съществува само една инстанция на съответния клас чрез използването на `private` конструктор. Шаблонът *DAO* отделя програмната логика от достъпа до записаните данни във файл.

Проектиране

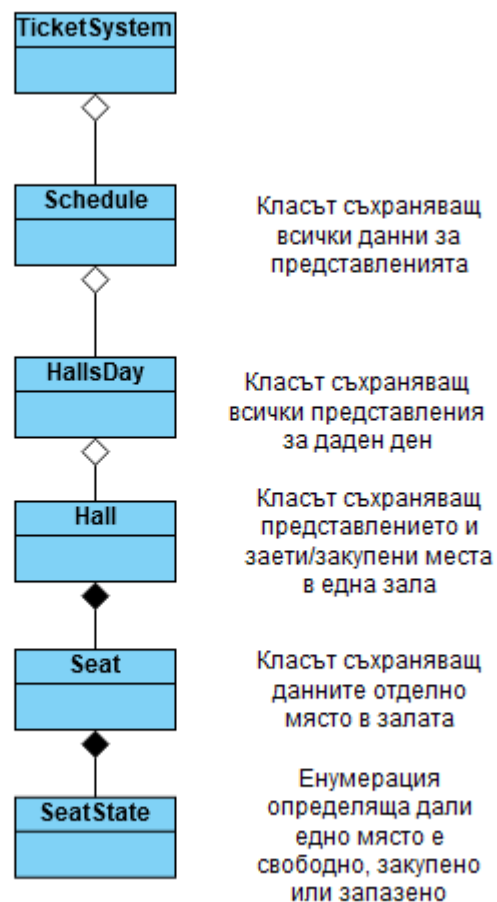
Проекта съдържа следните пакети: `com.company`, `Commands`, `Console`, `XML`, `TicketCodes`.

Като `com.company` съдържа `Main` класа, интерфейсите съответстващи на командите обработвани от програмата, класове отговарящи за съхранението на данните на програмата и класа `TicketSystem` и класове за грешки (`exceptions`). Интерфейсите за командите съдържат един публичен метод, който приема класа за данните `Schedule` заедно с нужните параметри. В `Commands` се съдържат имплементациите на интерфейсите на командите, както и пакета `TicketCodes`, който отговаря за генериране и обработка на кодовете за билетите.

В `Console` се съдържа клас, отговорен за отпечатване на местата в конзолата, подадени като списък.

В `XML` се съдържа класа *`ScheduleDAO`* (`Data Access Object`), който е отговорен за записа на данните на програмата в `XML` файл, както и за четене и десериализация на данните от `XML` файл.

Класовете съхраняващи данните за представленията:



Реализация и тестване

Класът Seat е отговорен за съхранението на състоянието на дадено място в залата. Съдържа енумерация за състоянието (закупено, запазено, свободно), информация за това на кой ред и кое място се намира, както и бележка ако е запазено място. Съдържа методи за достъп и промяна на атрибутите, *toString*, както и параметризиран и дефолтен конструктор.

```
public class Seat {
    private SeatState seatState;
    private int row;
    private int place;
    private String note;
    ...
}
```

Класът Hall отговаря за съхранение на състоянието на една зала за представления за определен ден. Съхранява името на представлението, брой редове, брой места на всеки ред, състоянията на отделните места, както и броя на закупени билети. Съдържа методи за достъп и промяна на атрибутите, параметризиран и дефолтен конструктор.

```
public class Hall {
    private String showName;
    private int maxRows;
    private int maxSeats;
    private Seat[][] seats;
    private int numberBought;
    ...
}
```

Класът HallsDay отговаря за съхранение на състоянието на всички зали за един ден. Съхранява масив от залите, брой редове и места на всяка зала. Съдържа методи за достъп и промяна на атрибутите, параметризиран и дефолтен конструктор, методи за добавяне и премахване на представления.

```
public class HallsDay {
    private Hall[] halls;
    private static int[] hallRows;
    private static int[] hallSeats;
    ...

    public void add(int hallNum, String name) throws EventsException{
        if(hallNum >= hallRows.length || hallNum < 0)
            throw new EventsException("Hall doesn't exist");
        halls[hallNum] = new Hall(name, hallRows[hallNum],
hallSeats[hallNum]);
    }

    public void remove(int hallNum) throws EventsException{
        if(hallNum >= hallRows.length || hallNum < 0)
            throw new EventsException("Hall doesn't exist");
        halls[hallNum] = null;
    }
    ...
}
```

Класът Schedule съхранява представленията в залите за всяка дата. Съдържа масив от броя на редовете и местата на всяка зала, броя на залите и колекция *Map* от залите с ключ датата. Съдържа методи за достъп и промяна на атрибутите, параметризиран и дефолтен конструктор и методи за премахване на запис за дата, ако на нея няма представления.

```
public class Schedule {
    private Map<LocalDate, HallsDay> halls;
    private int numberOfHalls;
    private static int[] hallRows;
    private static int[] hallSeats;
    ...
}
```

```

private boolean allEmpty(HallsDay hall){
    for (Hall h: hall.getHalls()) {
        if(h != null && h.getShowName() != null)
            return false;
    }
    return true;
}

public void removeEmpty(){
    Set<LocalDate> toRemove = new HashSet<>();
    for (Map.Entry<LocalDate, HallsDay> i: halls.entrySet()) {
        if(allEmpty(i.getValue())){
            toRemove.add(i.getKey());
        }
    }
    for (LocalDate i: toRemove) {
        halls.remove(i);
    }
}

```

Класът `TicketSystem` чете входните данни от потребителя, обработва ги с класа `CommandProcessor`, обработва грешките и отпечатва резултата в конзолата. Използван е шаблона *Singleton*.

```

public class TicketSystem {
    private static TicketSystem instance;
    private static final int numberOfHalls = 4;
    private static final int[] rows = {5, 6, 8, 9};
    private static final int[] seats = {12, 10, 14, 19};
    private Schedule schedule;

    private TicketSystem(){
        schedule = new Schedule(TicketSystem.numberOfHalls,
TicketSystem.rows, TicketSystem.seats);
    }

    public static TicketSystem getInstance(){
        if(instance==null)
            instance = new TicketSystem();
        return instance;
    }

    public void start(){
        ProcessCommand processCommand = new CommandProcessor(schedule,
numberOfHalls, rows, seats);
        Scanner scanner = new Scanner(System.in);
        String input;
        String[] command;
        while(true){

```

```

        System.out.print('>');
        input = scanner.nextLine();
        if(input.equals("exit"))
            break;
        command = input.trim().split(" ", 2);

        try {
            processCommand.process(command);
        } catch (CommandException e){
            System.out.println(e.getMessage());
        } catch (DateTimeParseException e){
            System.out.println("Incorrect date formatting!");
        } catch (NumberFormatException e){
            System.out.println("Incorrectly entered number!");
        }
    }
    System.out.println("Exiting the program...");
}
}

```

Класът `CommandProcessor`, чрез метода `process`, приема командата въведена от потребителя, която е разделена на текстов масив от два елемента. Първият елемент е името на командата, а втория (ако има) е аргументи на командата. Чрез структурата `switch` се извиква *private* метод, който обработва командата. В него се създава и използва клас за обработка на съответната команда.

```

        . . .
private void addEvent(String[] params)throws CommandException{
    if(params.length!=2)
        throw new CommandException("Incorrect parameters");
    AddEvent addEvent = new Commands.AddEvent();
    String[] params1 = params[1].split(" ", 3);
    if(params1.length!=3)
        throw new CommandException("Incorrect parameters");
    LocalDate date = LocalDate.parse(params1[0]);
    int hall = Integer.parseInt(params1[1]);
    String name = params1[2].replace("\"", "");
    addEvent.addEvent(schedule, date, hall-1, name);

    System.out.println("Event added");
}
        . . .
@Override
public void process(String[] command)throws CommandException {
    switch (command[0]){
        case "saveas":
            saveAs(command);

```

```

        break;
    case "save":
        save();
        break;
    case "close":
        close();
        break;
    case "open":
        open(command);
        break;

        . . .

    default:
        throw new CommandException("Command not recognised!");
}
}

```

Класът `AddEvent` е пример за клас обработващ команда. Той прибавя ново представление по зададени - дата, номер на зала и име на представление. Ако на същата дата вече го има представлението или ако всички зали са заети, дава грешка.

```

public class AddEvent implements com.company.AddEvent {
    @Override
    public void addEvent(Schedule schedule, LocalDate date, int
hallNumber, String name) throws CommandException {
        if(name == null || name.isBlank())
            throw new CommandException("Show name is not entered!");

        if(schedule.getHallsForDay(date) == null) {
            try {
                schedule.addHallsForDay(date);
            } catch (EventsException e) {
                throw new CommandException("Hall's booked!");
            }
        }
        HallsDay hallsDay = schedule.getHallsForDay(date);

        for (Hall i: hallsDay.getHalls()) {
            if(i != null && i.getShowName() != null &&
i.getShowName().equals(name))
                throw new CommandException("Same show on that date!");
        }

        Hall hall = hallsDay.getHall(hallNumber);

        if(hall != null)
            throw new CommandException("Hall's booked!");
    }
}

```

```

        try {
            hallsDay.add(hallNumber, name);
        } catch (EventsException e) {
            e.printStackTrace();
        }
    }
}

```

Класът ScheduleDAO отговаря за запис в и четене от XML файл. Използван е парсер от пакета java.beans.

Използвани източници:

<https://www.edureka.co/blog/serialization-of-java-objects-to-xml-using-xmlencoder-decoder/>

<https://stackoverflow.com/questions/41373566/localdate-serialization-error>

```

package XML;

import com.company.Schedule;

import java.beans.*;
import java.io.*;
import java.time.LocalDate;

fbclid=IwAR2tzP9l5_o53ipMA7VcxHy5wFAiZpRADrZLhB9DyH1ASIIx9jynis7VbMk
public class ScheduleDAO implements com.company.ScheduleDAO {

    @Override
    public void saveToFile(Schedule schedule, String fileName)throws
FileNotFoundException {

        XMLEncoder encoder=new XMLEncoder(new BufferedOutputStream(new
FileOutputStream(fileName)));

        encoder.setPersistenceDelegate(LocalDate.class, new
PersistenceDelegate() {
            @Override

            protected Expression instantiate(Object oldInstance, Encoder
out) {
                LocalDate localDate = (LocalDate) oldInstance;
                return new Expression(localDate, LocalDate.class, "of",
                    new Object[]{localDate.getYear(),
localDate.getMonth(), localDate.getDayOfMonth()});
            }
        });

        encoder.writeObject(schedule);
        encoder.close();

    }
}

```



```

    @Override
    public Schedule loadFromFile(String fileName)throws
FileNotFoundException {
        XMLDecoder decoder=new XMLDecoder(new BufferedInputStream(new
FileInputStream(fileName)));
        Schedule schedule = (Schedule)decoder.readObject();
        decoder.close();
        return schedule;
    }
}

```

Тестване и примерна работа

addevent

```

>addevent 2022-01-15 3 "Concert"
Event added

```

book

```

>book 2 8 2022-01-15 "Concert" My Note
Seat Booked

```

bookings

```

>bookings 2022-01-15 "Concert"
[row:2 seat:8]?My Note

```

buy

```

>buy 1 2 2022-01-15 "Concert"
Seat code: 0-1-2-2022-01-15

```

check

```

>check 0-1-2-2022-01-15
Ticket is valid
Seat:[row:1 seat:2]

```

```

>check 0-1-2
Ticket is NOT valid!

```

```

>check 0-1-2-2022-01-16
Ticket is NOT valid!

```

close

```

>close
File closed

```

exit

```

>exit
Exiting the program...

```

freeseats

```
>freeseats 2022-01-15 "Concert"
```

```
[row:1 seat:1]
[row:1 seat:2]
[row:1 seat:3]
[row:1 seat:4]
[row:1 seat:5]
[row:1 seat:6]
[row:1 seat:7]
[row:1 seat:8]
[row:1 seat:9]
[row:1 seat:10]
[row:1 seat:11]
[row:1 seat:12]
[row:1 seat:13]
[row:1 seat:14]
[row:2 seat:1]
[row:2 seat:2]
[row:2 seat:3]
[row:2 seat:4]
[row:2 seat:5]
[row:2 seat:6]
[row:2 seat:7]
[row:2 seat:8]
[row:2 seat:9]
[row:2 seat:10]
[row:2 seat:11]
[row:2 seat:12]
[row:2 seat:13]
[row:2 seat:14]
[row:3 seat:1]
[row:3 seat:2]
[row:3 seat:3]
[row:3 seat:4]
[row:3 seat:5]
[row:3 seat:6]
[row:3 seat:7]
[row:3 seat:8]
[row:3 seat:9]
[row:3 seat:10]
[row:3 seat:11]
[row:3 seat:12]
[row:3 seat:13]
[row:3 seat:14]
[row:4 seat:1]
```

help

```
>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 opened file in <file>
help Prints this information
exit Exits the program
addevent <date> <hall> <name> Adds a new event on <date> with name <name> in <hall>
freeseats <date> <name> Lists all the free seats(not booked or bought) for the event <name> on <date>
book <row> <seat> <date> <name> <note> Books a ticket for the show <name> on <date> in seat <row> <seat> and adds a note <note>
unbook <row> <seat> <date> <name> Unbooks the seat for the show <name> on <date> in row <row> in seat <seat>
buy <row> <seat> <date> <name> Buys the free or booked seat for the show <name> on <date> in row <row> in seat <seat>
bookings [<date>] [<name>] Shows all the booked days for the show <name> on <date>, if name is skipped then lists for all shows o
n that day, if both are skipped then lists all booked seats
check <code> Checks if thicket code <code> is valid and displays the seat number
report <from> <to> [<hall>] Shows all the bought tickets form date <from> to date <to> in hall <hall>, if <hall> is skipped then t
he sales for all halls are shown
top [<number>] Show the top <number> most watched shows, if <number> is skipped then shows all shows>
worst <from> <to> Shows the events with less than 10% sales in the period between <from> and <to>, then asks if they should be r
emoved
```

open

```
>open "file.xml"
Successfully opened file.xml
```

report

```
>report 2022-01-01 2022-12-31
Show:Bon Jovi Sales:2
Show:Slavi Trifonov Sales:1
>report 2022-01-01 2022-12-31 1
Show:Bon Jovi Sales:2
```

save

```
>save  
No file found!
```

```
>open "file.xml"  
Successfully opened file.xml  
>save  
Successfully saved file.xml
```

saveas

```
>saveas "newfile.xml"  
Successfully saved to newfile.xml
```

top

```
>top  
Top shows:  
"Desi Slava":3 sales  
"Bon Jovi":2 sales  
"Slavi Trifonov":1 sales  
>top 2  
Top 2 shows:  
"Desi Slava":3 sales  
"Bon Jovi":2 sales
```

unbook

```
>book 2 7 2022-01-15 "Concert" Note  
Seat Booked  
>unbook 2 7 2022-01-15 "Concert"  
Seat Unbooked
```

worst

```
>worst 2020-01-01 2022-12-31  
Shows with less than 10.0% of seats bought:  
Bon Jovi:3%  
Slavi Trifonov:2%  
Desi Slava:2%  
Remove(Y/N)?  
n
```

Заклучение

За бъдещо подобрене на програмата може да бъде добавен графичен интерфейс, допълнителни команди за водене на статистика, както и възможност за закупуване и запазване на голям брой места едновременно. Има възможност за подобряване съхранението на данните и поддръжка на различни файлови формати.

Използвани източници:

<https://www.edureka.co/blog/serialization-of-java-objects-to-xml-using-xmlencoder-decoder/>

<https://stackoverflow.com/questions/41373566/localdate-serialization-error>

<http://www.java2s.com/example/java-api/java/beans/xmlencoder/setpersistencedelegate-2-0.html>

<https://www.janbasktraining.com/blog/serialization-of-java-objects/>