Технически университет Варна

Курсов проект

дисциплина: Обектно-ориентирано програмиране – I

специалност: „Софтуерни и интернет технологии“

изготвил: Теодора Миленова Стойчева

факултетен номер: 22621693

Съдържание

Глава 1. Увод

1.1. Описание и идея на проекта

Проектът предоставя набор от основни функционалности, необходими за работата на билетна каса. Данните, с които касата си служи, се съхраняват в текстови файлове, които потребителят може да достъпва чрез набор операции за отваряне и затваряне, добавяне, редактиране или изтриване на информация, както и за нейното записване или презаписване. Извършването на тези операции се осъществява чрез въвеждани от потребителя предварително зададени команди. Направените промени се запазват във файловете само при зададена конкретна команда, за да се позволи гъвкавост на проекта. Освен операции за модификация и извличане на файловото съдържание, проектът предоставя възможност на потребителя да задава команди за справки, конкретно типични за билетна каса. Резултатът от тях се извлича за потребителя, без да се запазва във файл.

1.2. Цел и задачи на разработката

Проектът разработва следните основни функционалности, свързани с обработка на текстови файлове:

- Отваряне и затваряне на файлове;
- Четене на информация;
- Извеждане информация за поддържаните от програмата команди и последващо изпълнение на тяхна база;
- Изход от програмата.

Поддържат се и следните методи, специфични за билетната каса:

- Добавяне на ново представление между вече съществуващите, с проверки за застъпване;
- Справка за свободни места на представление в зададена зала;
- Резервиране на билет за представление, без мястото да се закупува;
- Отмяна резервация на билет за представление;
- Закупуване на билет за представление;

- Справка за резервирани, но незаплатени билети за представление;
- Проверка валидност на билет по уникален код;
- Справка за закупени билети в зададен времеви диапазон.

1.3. Структура на документацията

Документацията запознава с целта на проекта, преглежда предметната област, показва основни моменти в проектирането и реализацията на информационната система. В структурата последователно е представена разработката на проекта. Увода на документацията уточнява идеята и целите на разработката на проекта, както и неговото основно описание. Прегледът на предметната област анализира проблемите приложените в проекта решения и указва конкретно основните дефиниращи структурни единици и алгоритми. Структурата се представя чрез блок-схеми и диаграми в проектирането на системата. След това са представени подробно реализацията на класовете и тестването на зададените функционалности. За заключение са предложени обобщение и насоки за развитие на билетната каса.

Глава 2. Преглед на предметната област

Проектът се занимава със система, предназначена да управлява информация за резервации и продажби на билети. Основните проблеми, които тази система адресира, са свързани с ефективното планиране на събития, резервациите на индивидуални места, закупуването на билети и генерирането на справки, свързани с тези операции.

2.1. Дефиниране на проблеми и сложност на поставената задача

2.1.1. Дефинирани проблеми:

1. Управление на списък със събития

Необходимо е да се осъществява планирането и управление на сисък със събития, включително обработка на множество събития в различни зали с различни дати и часове.

Сложността произлиза от проследяването на наличността на зали; от нуждата да се гарантира, че събития не се припокриват в една и съща зала, и управлението на подробности за събитията, като имена, дата и час. Задължително е да се съобразят всички възможни зависимости.

2. Резервации и закупуване

Необходимо е да се позволи на потреителите да резервират и отменят резервации на места, гарантирайки, че действията се проследяват коректно и правилно се обработват случаи, при които местата са както резервирани, така и закупени.

Сложността е в управлението на състоянието на всички места, като се предотвратява двойното резервиране чрез правилно актуализиране на състоянието на местата.

3. Отчитане и анализ

Необходимо е да се генерират отчети за наличност на свободни места, продажба на билети и други статистически данни за събития, включително идентифициране на най-популярните и най-неуспешните събития.

Сложността е в обобщаването на данни за различни събития и осигуряването на точно отчитане за определени периоди от време. Трябва да се съобразят типовете данни, които системата обработва за всеки сборен елемент.

2.1.2. Сложност на задачата:

1. Цялост на данните

Трябва да е сигурно, че всички промени в поддържаната информация са точно отразени в системата и че възникнали конфликти и грешки се обработват по подходящ начин.

2. Мащабируемост

Системата трябва да може да обработва по-голям обем данни без влошаване на производителността.

3. Потребителско изживяване

Системата трябва да предоставя удобен потребителски интерфейс за управление на данните чрез ефективни операции.

2.2. Подходи, методи за решаване на поставените проблеми

За справяне с очертаните проблеми са приложени следните подходи и методи:

1. Управление на събития

Подход: Приложена е стабилна система за планиране на събития, която позволява създаването, модифицирането и отмяната на събития.

Метод: Използвани са структури от данни като списъци и карти, за да се съхраняват подробности за събитието и информация за залата. Създадени са проверки за валидиране, за да предотвратите конфликти в графика и да се гарантира, че събитията се планират само в наличните зали.

2. Резервации и закупуване:

Подход: Разработена е система за резервация на места, която позволява на потребителите да резервират и отменят резервации с актуализации в реално време за наличността на места. С нея се интегрира логика за закупуване, за да управлява продажбите и да генерира уникални кодове на билети.

Метод: Използван е обектно-ориентиран дизайн, за да може ясно да се представят обектите места, редове и зали. Използват се методи за проверка на наличността на места, обработка на резервации и управление на състояния на резервации. Проверява се дали състоянието на мястото се актуализира точно и конфликтите са разрешени. Включена е логика за генериране на билети, която създава уникален идентификатор за всеки закупен билет. Актуализират се състоянията на местата.

3. Подход за отчитане и анализ:

Подход: Изпозлват се функции за отчитане и анализ, които предоставят информация за наличността на места, продажбата на билети и популярността на събитията.

Метод: Включени са техники за агрегиране и анализ на данни, за да се генерират отчети въз основа на определени периоди от време и номера на

зали. Приложени са методи за изчисляване на показатели като проценти на продажба на билети.

5. Обработка на грешки и валидиране:

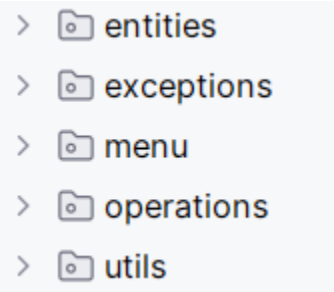
Подход: За да е сигурно, че системата е устойчива на грешки и невалидни входни данни, се прилагат цялостни механизми за обработка на грешки и валидиране.

Метод: Използват се методи за обработка на изключения и управление на грешки по време на резервиране на места, закупуване на билети и планиране на събития. Валидира се въведената от потребителя информация, за да се предотврати неправилно въвеждане на данни.

Следвайки тези подходи и методи, системата има за цел да предостави цялостно решение за управление на събития, резервации на места и закупуване на билети, като същевременно се справя с присъщите сложности.

Глава 3. Проектиране

3.1. Обща структура на проекта и пакети, които ще се реализират.

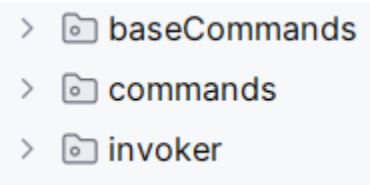


3.1. Базови пакети

Пакетът *entities* съдържа класове-модели, реализации на основните единици на билетната каса.

Пакетът *exceptions* съдържа класовете, които обработват конкретните за проекта изключения.

Пакетът *menu* е разделен на три подпакета:



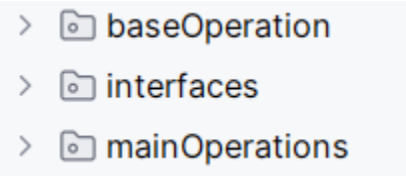
3.2.Подпакети на "menu"

- *baseCommands* - Съдържа командите, които определят основните операции на системата. Тези команди представляват функционалности на ниско ниво, които често се използват в проекта.

- *commands* - Капсулира основните команди, свързани с бизнес логиката на системата. Това са по-специфични команди, които препращат към изпълнение на задачи, свързани с управление на събития, резервации и други.

- *invoker* – Съдържа класа, отговорен за извикване или изпълнение на команди въз основа на въвеждане от потребителя или системни събития, задействайки изпълнението на различни команди.

Пакетът *operations* е разделен на три подпакета:



3.3.Подпакети на "operations"

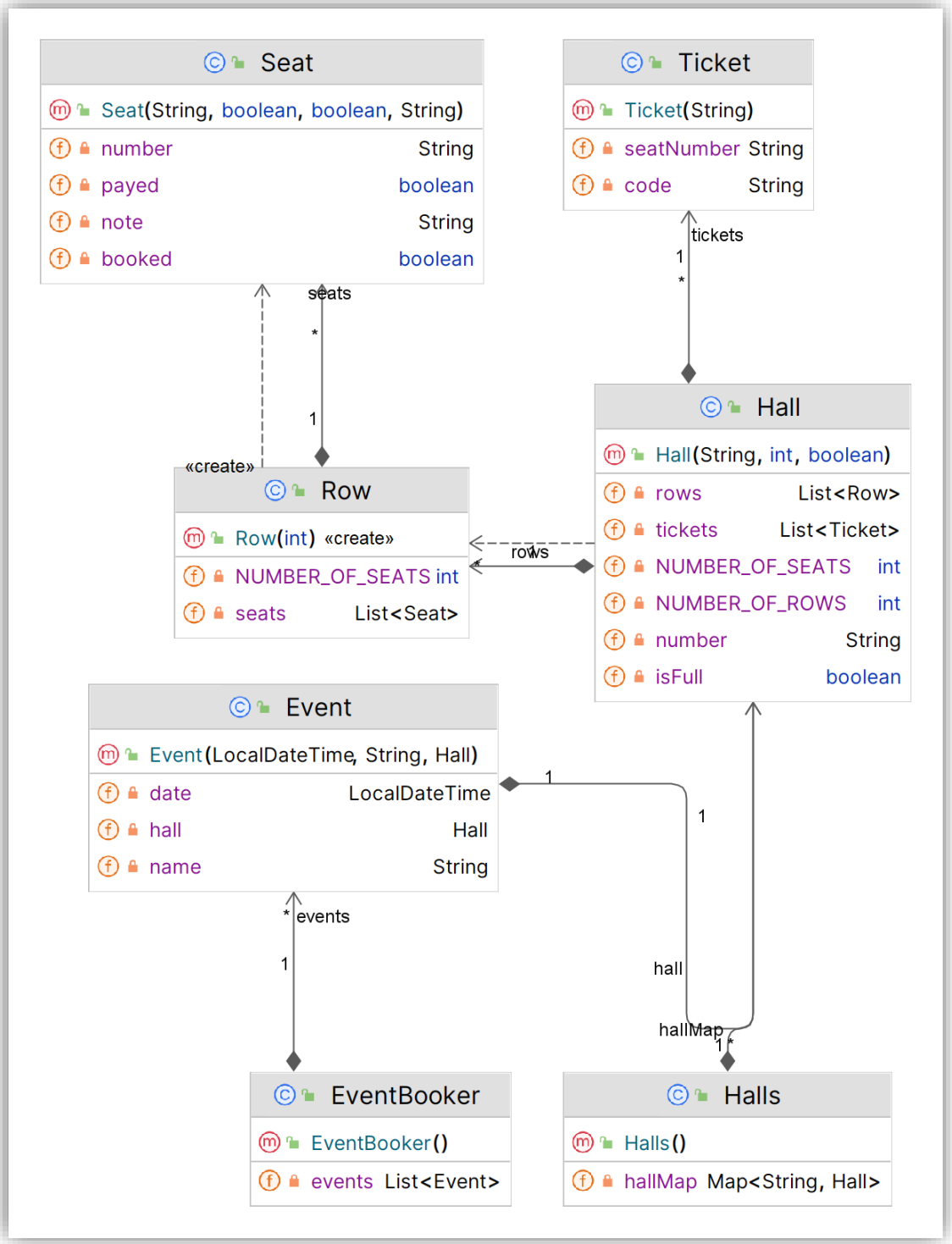
- *baseOperation* – Съдържа основните операции, които определят задачи на системата. Тези операции са на ниско ниво и често предоставят помощни функции или общо поведение, което по-специфичните операции надграждат.

- *interfaces* – Съдържа командния интерфейс, който дефинира договорите или абстракциите, които другите класове и пакети прилагат.

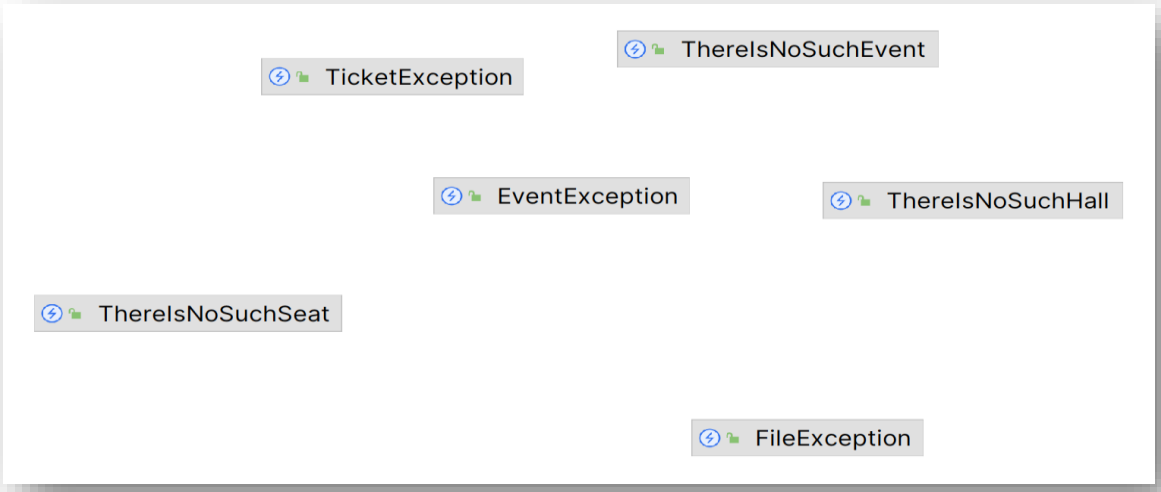
- *mainOperations* – Съдържа основната бизнес логика и специфични за проекта операции. Те изпълняват действителните задачи, които потребителите очакват от системата, като резервиране на места, генериране на отчети или управление на билети.

Пакетът *utils* съдържа енумерация, която поддържа списък с валидните команди, както и генеричен клас, който отразява връзката между броя места и броя продадени билети за представление.

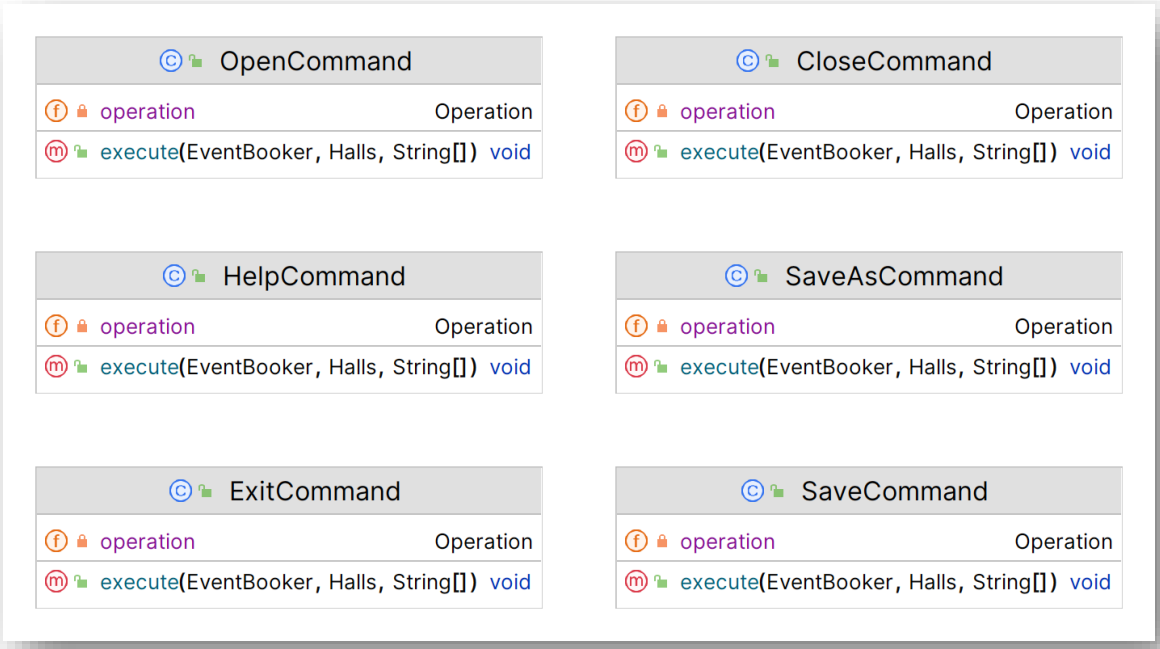
3.2. Диаграми/Блок схеми



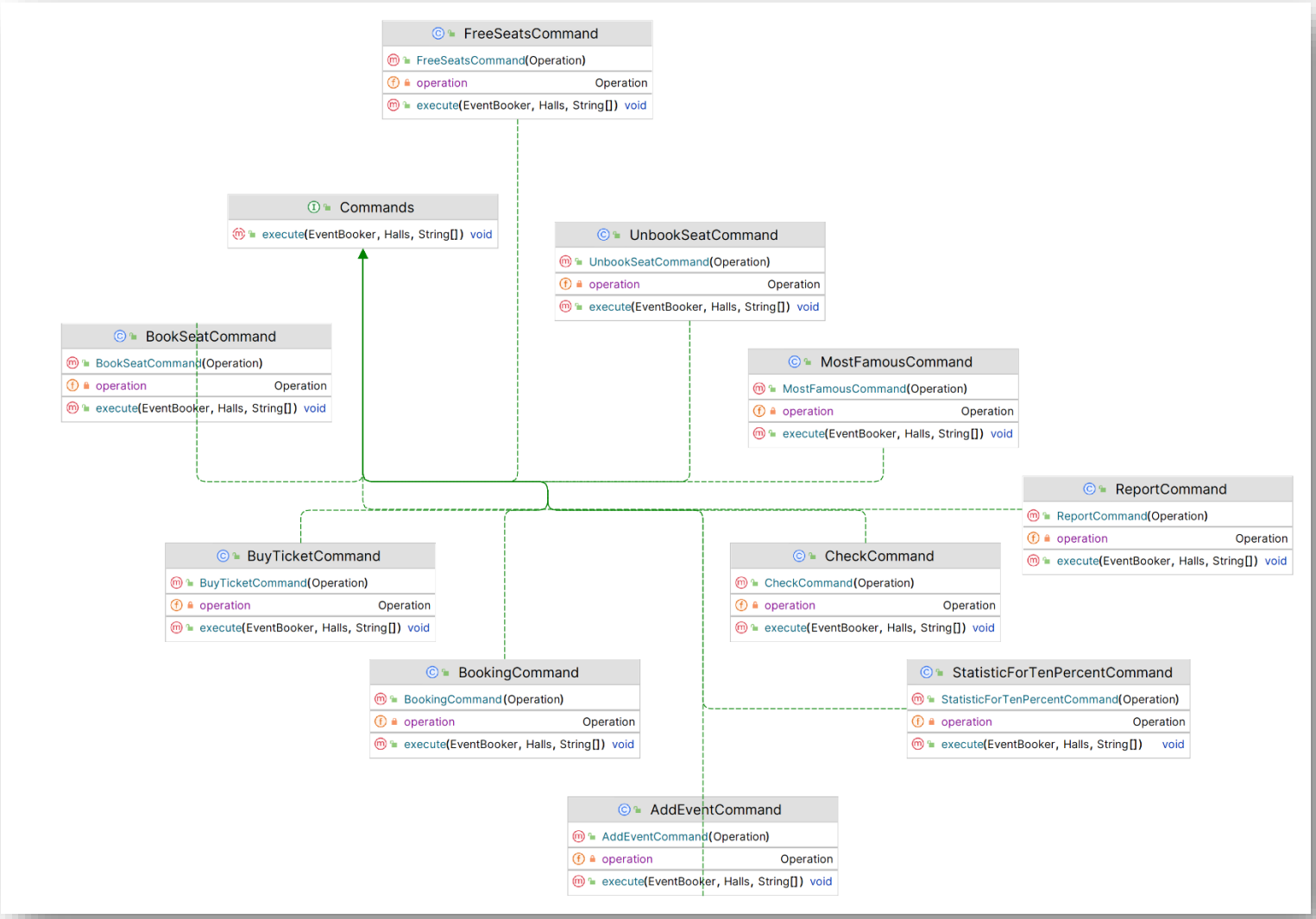
3.3. Класове-модели - entities



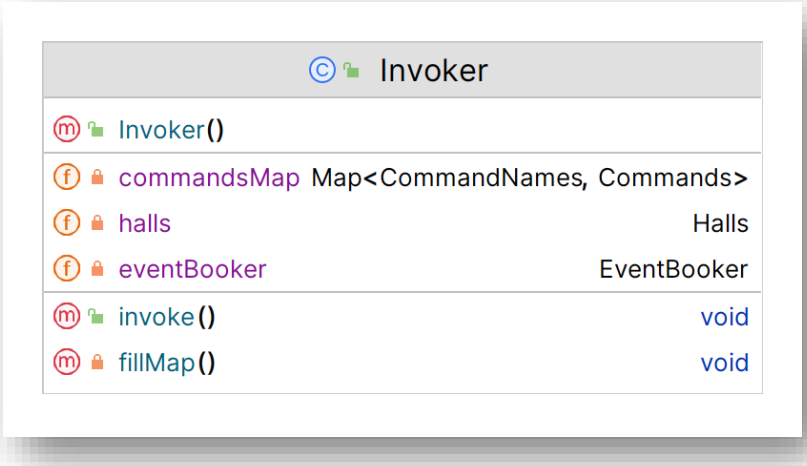
3.4.Класове за обработка на изключения - exceptions



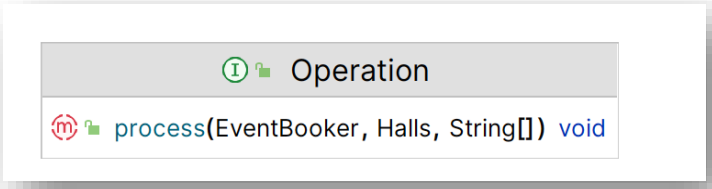
3.5.Класове-команди за базовите функционалности в пакета baseCommands



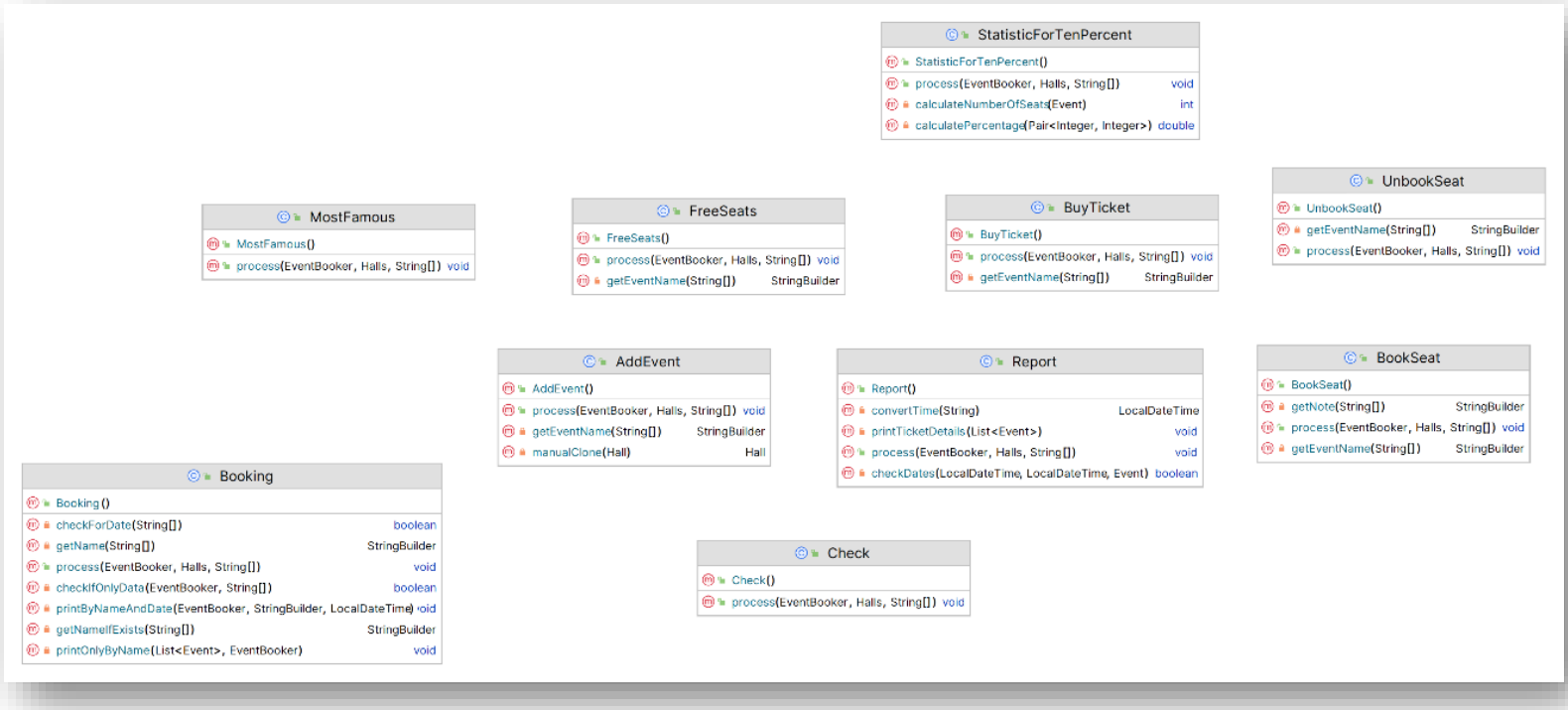
3.6.Класове-команди за бизнес функционалности в пакета baseCommands



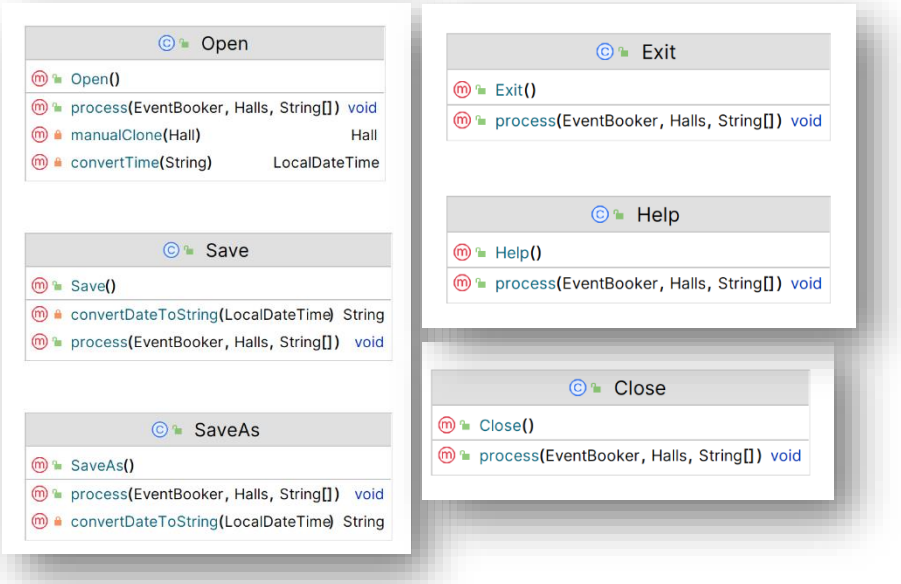
3.7.Клас-централен контролер в пакета invoker



3.8.Капсулиращ интерфейс в пакета interfaces

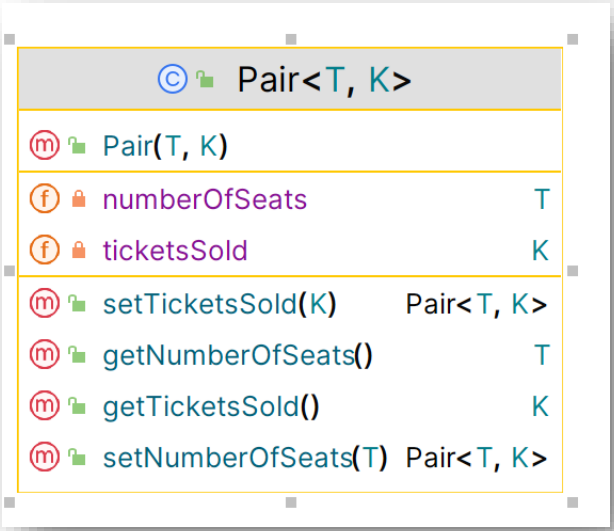


3.9.Класове, извършващи основните операции - baseOperations

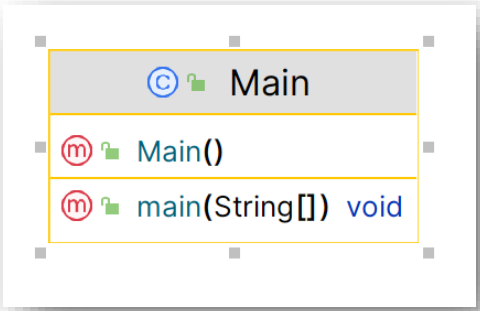


3.10.Класове, извършващи базовите функционалности - mainOperations

3.11.Клас-енумерация за кодиране на възможните команди в пакета util



3.12.Генеричен клас в пакета util



3.13.Стартиране на програмата

Глава 4. Реализация, тестване

4.1. Основни дефиниции

1. Модели

Event:

Определение: Представлява събитие с атрибути име, дата и свързана зала.

Значение: Заема централно място в системата за резервации, управление на подробности и взаимодействия, свързани със събитията.

Seat:

Определение: Представлява отделно място в зала. Има атрибути номер и бележка и бинарни състояния – резервирана и закупена.

Значение: Най-малката обектна единица в проекта, с която се борави при изграждането на зависимости.

Row:

Определение: Представлява ред от седалки в зала, организирайки множество седалки в последователна структура. Съдържа финален брой на седалки и поле-списък от седалки. Логически е междинен слой за седалките и залите.

Значение: Действа като контейнер за обекти Seat, като осигурява достъп до управлението на места в контекста на конкретен ред в зала. Той позволява запитване за отделни места и преминаване през всички места в реда за операции като резервиране и управление на билети.

Hall:

Определение: Представлява сборно пространство, с което са свързани събитията, включва редове и седалки в основата си. Има атрибути за номерата на редове и седалки, както и име, булева променлива за проследяване на капацитета, списъци от редове и от билети.

Значение: Организира и управлява подреждането на местата за събития.

Ticket:

Определение: Представлява билет за събитие с уникален рандомизиран код. Има атрибути за кода и номера на седалката, с която билетът е асоцииран.

Значение: Енкапсулира информацията, свързана със закупените билети и служи за проследяване и управление на продажбите.

EventBooker:

Определение: Управлява събития, резервации и продажба на билети чрез методи за добавяне, премахване и достъпване на събития. Има атрибути списък от събития.

Значение: Действа като централен контролер за управление на събития и операции с билети.

2. Ключови операции

AddEvent:

Определение: Управлява създаването и добавянето на нови събития.

Значение: Позволява регистриране и настройка на нови събития в системата.

BookSeat:

Определение: Управлява резервирането на конкретни места за събития.

Значение: Актуализира състоянието на местата и гарантира, че местата са запазени.

BuyTicket:

Определение: Управлява покупката и издаването на билети.

Значение: Управлява транзакции и генерира кодове на билети.

UnbookSeat:

Определение: Управлява анулирането на резервации за места.

Значение: Освобождава резервирани места и актуализира състоянието им.

3. Помощни класове

Pair:

Определение: Генеричен клас, който съхранява чифт свързани обекти. Има атрибути за брой на седалки и брой продадени билети.

Значение: Улеснява управлението на свързани данни, като позволява манипулацията им в двойка.

CommandNames:

Определение: Енумерация, която дефинира възможните имена на команди, които системата поддържа.

Значение: Осигурява структуриран начин за обработка на различни системни команди.

4. Изпълнение на команди:

Invokator:

Определение: Управлява изпълнението на команди въз основа на въвеждане от потребителя. Има атрибути *map* от команди, поле тип *EventBooker* и *Halls.*

Значение: Действа като централен контролер за анализиране и извикване на команди, координиране на потребителските взаимодействия.

5. Обработка на грешки

TicketException:

Определение: Персонализиран клас-изключение за обработка на грешки, свързани с билети.

Значение: Осигурява начин за управление и докладване на проблеми с операциите с билети.

Тези класове и дефиниции формират гръбнака на системата за резервации, позволявайки създаването, управлението и взаимодействието със събития, места и билети. Те гарантират, че системата работи гладко и предоставя необходимата функционалност за потребителите.

4.2. Реализация на класове

Що се отнася до структурата, потребителската логика в проекта е изпълнена, следвайки командния шаблон за енкапсулиране и подаване на заявки. Методите с бизнес логика за отделните възможни команди са разделени в класове, наследници на интерфейса *Operation* и разширяват неговия метод за изпълнение на команда, но не се достъпват пряко от потребител. Връзката между потребителя и изпълнението на операциите са класовете, които наследяват интерфейса *Command* – те инстанцират в себе си операция и я изпълняват в наследения метод. Карта от всички команди се създава в класа *Invoker*. Той отговаря и за прочитането на командите от конзолата и тяхното инициране, а това позволява в главния клас *Main* единствено да се създава негова инстанция, която да извика главния му метод.

Най-малката градивна единица на системата е обект тип *Seat*. Той присъства като атрибут в класа *Row*, а обекти от тип *Row* и *Ticket* са атрибути в списък в класа *Hall*. Това са основните обектни единици и зависимости. За работа с тях са създадени класовете *Halls* и *EventBooker*. В *Halls* се именуват залите и се подреждат в *hashmap*, който позволява да се достъпват по име, а в *EventBooker* събитията се подреждат в списък с основни методи за манипулация, както и конкретна логика за достъпване на събития. Наследниците на *Command* и *Operation* работят с тези обекти.

Като помощни класове са създадени обособени изключения в пакета *exceptions* и класовете *Pair* и *CommandNames* в пакета *util*. Изключенията са приложени при нужда в методите с бизнес логика. Класът *CommandNames* е енумерация, която кодира възможните команди. В *Pair* се обвързват данни за помощни изчисления.

Нататък се разглеждат всички методи и зависимости.

1. Seat

   В конструктора се дефинират уникалните полета – номер и бележка към седалката и булевите състояния, които указват дали е резервирана или платена.

```
public Seat(String number, boolean booked, boolean payed, String
note) {
        this.number = number;
        this.booked = booked;
        this.payed = payed;
        this.note = note;
    }
```

   Създадени са гетъри и сетъри за всеки атрибут.

2. Row

Състои се от финално поле с броя на седалки и списък от седалките.

```
public Row(int NUMBER_OF_SEATS) {
    this.NUMBER_OF_SEATS = NUMBER_OF_SEATS;
    this.seats = new ArrayList<>();
}
```

Създадени са гетъри и сетъри за всеки атрибут.

```
public void initializeSeats(int number) {
    char start = 'A';
    for (int i = 0; i < NUMBER_OF_SEATS; i++) {
        seats.add(new Seat(String.valueOf(number +
String.valueOf(start++)), false, false, ""));
    }
}
```

Методът `initializeSeats` запълва и именува списъка от седалки във всеки ред, като задава състоянията им.

3. Ticket

Състои се от текстови променливи за кода и за седалката.

```
public Ticket(String seatNumber) {
    this.code = generateCode();
    this.seatNumber = seatNumber;
}

private String generateCode() {
    String alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    StringBuilder code = new StringBuilder();
```

```
    int size = 5;
    Random random = new Random();
    for (int i = 0; i < size; i++) {
        int index = random.nextInt(alphabet.length());
        code.append(alphabet.charAt(index));
    }
    return code.toString();
}
```

Методът `generateCode` генерира случаен буквен код и го обвързва с билета.

4. Hall

Състои се от полета списък с редове, списък с билети, булева променлива за капацитет, номер и променливи за брой на седалки и редове.

```
public Hall(String number, int NUMBER_OF_ROWS, boolean isFull) {
    this.rows = new ArrayList<>();
    this.number = number;
    this.NUMBER_OF_ROWS = NUMBER_OF_ROWS;
    this.isFull = isFull;
    this.tickets = new ArrayList<>();
    this.NUMBER_OF_SEATS = 20;
    initializeRows();
}

public int getMaxAmountOfTickets() {
    return NUMBER_OF_SEATS * NUMBER_OF_ROWS;
}
```

Методът `getMaxAmountOfTickets` връща общия брой на места в зала.

```
public Ticket getTicketBySeatNumber(String number) {
    for (Ticket ticket : tickets) {
        if (ticket.getSeatNumber().equals(number))
            return ticket;
    }
    throw new ThereIsNoSuchSeat("There is no such seat");
}
```

Методът `getTicketBySeatNumber` обхожда списъка с билети и връща билет за конкретна седалка в залата.

5. Halls

Състои се от мап със зали.

```
public Halls() {
    this.hallMap = new HashMap<>();
}
```

Създадени са гетъри и сетъри.

6. EventBooker

Състои се от списък със събития.

```
public EventBooker() {
    this.events = new ArrayList<>();}
```

Създадени са гетъри, сетъри и методи за манипулация на списъка.

```
public Event getEventByNameAndDate(String name, LocalDateTime time) {
    for (Event event : events) {
        if (event.getName().equals(name) &&
event.getDate().isEqual(time)) {
            return event;       } }
    throw new ThereIsNoSuchEvent("There is no such event");}
```

Методът `getEventByNameAndDate` търси и връща от списъка събитие със съвпадащи име и дата.

```
public List<Event> getEventByName(String name) {
    List<Event> result = new ArrayList<>();
    for (Event event : events) {
        if (event.getName().equals(name)) {
            result.add(event);
        }
    }  return result;}
```

Методът `getEventByName` търси и връща от списъка събития със съвпадащо име.

```
public List<Event> getEventByDate(LocalDateTime time) {
    List<Event> result = new ArrayList<>();
    for (Event event : events) {
        if (event.getDate().isEqual(time)) {
            result.add(event);
        }
    }
    return result;
```

Методът `getEventByDate` търси и връща събития със съвпадаща дата.

```
public void printDetailsForBookedSeats(List<Event> eventByName) {
    for (Event event : eventByName) {
        int counter = 0;
        for (Row row : event.getHall().getRows()) {
            counter++;
            for (Seat seat : row.getSeats()) {
                if (seat.isBooked()) {
                    System.out.println("On row number: " + counter
                            + " seat with number " + seat.getNumber()
                            + " is booked with note: " +
seat.getNote());
```

Методът `printDetailsForBookedSeats` итерира списък от обекти Event и отпечатва подробна информация за резервираните места за всяко събитие. Той обработва всяка зала на събитието, преминавайки през нейните редове и

седалки. За всяко резервирано място отпечатва номера на реда, номера на мястото и всяка свързана бележка, като използва брояч за проследяване на номера на реда.

7. Интерфейсът Command и класовете, които го имплементират

Интерфейсът има метод:

```
void execute(EventBooker eventBooker, Halls halls, String[] data);
```

Негови класове-имплементации съдържат поле тип Operation, конструктор с този атрибут и имплементираният метод, в който викат за изпълнение основната функционалност на операцията. Логиката е аналогична.

Пример:

```
public class FreeSeatsCommand implements Commands {
    private final Operation operation;
    public FreeSeatsCommand(Operation operation) {
        this.operation = operation;
    }
    @Override
    public void execute(EventBooker eventBooker, Halls halls, String[]
data) {
        operation.process(eventBooker, halls, data);
    }
```

8. Invoker

Съдържа мап за командите и инстанции от Halls, EventBooker.

```
public Invoker() {
    this.eventBooker = new EventBooker();
    this.halls = new Halls();
    this.commandsMap = new HashMap<>();}

public void invoke() {
    fillMap();
    Scanner scanner = new Scanner(System.in);
    while (true) {
        String command = scanner.nextLine();
        if (command.equals("exit") || command.equals("close")) {
            break;    }
        String[] data = command.split("\\s++");
        if (commandsMap.containsKey(CommandNames.getByCode(data[0]))) {
            try {
commandsMap.get(CommandNames.getByCode(data[0])).execute(eventBooker,
halls, data);}catch (RuntimeException e){
System.out.println(e.getMessage()); }
        } else {
            System.out.println("Invalid command"); }
```

Методът `invoke` управлява изпълнението на команди в приложението. Първо попълва карта на команди и съответните им манипулатори, използвайки метода fillMap. След това влиза в безкраен цикъл, четейки потребителския вход от конзолата. Всяка входна команда се разделя на масив от низове за обработка. Методът проверява дали командата е валидна, като я сравнява с ключовете в commandsMap. Ако е валиден, той се опитва да изпълни съответната команда, използвайки метода за изпълнение на манипулатора на командата, като улавя и отпечатва всички изключения по време на изпълнение, които възникват. Цикълът прекратява, ако потребителят въведе "изход" или "затвори", в противен случай той информира потребителя за всички невалидни команди.

9. Интерфейсът Operation и класовете, които го имплементират

Интерфейсът има метод, който се имплементира във всички класове-операции:

```
void process(EventBooker eventBooker, Halls halls, String[] data);
```

9.1. Close

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    try {
        fileReader = new FileReader(data[1]);
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
    System.out.println("Closing " + data[1]);
    try {
        fileReader.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

}
```

Методът отговаря за затварянето на отворен файл, зададен от масива от данни. Започва с опит за отваряне на файл с помощта на FileReader, инициализиран с пътя на файла от data[1], и ако файлът не бъде намерен, той хвърля RuntimeException. След регистриране, че файлът се затваря, той се опитва да затвори FileReader и ако тази операция е неуспешна, той хвърля RuntimeException за управление на всички грешки, възникнали по време на затварянето на файла.

9.2. Open

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    // data[1]= filePath
    try (FileReader fileReader = new FileReader(data[1])) {
        Scanner scanner = new Scanner(fileReader);
        while (scanner.hasNextLine()) {
            String eventName = scanner.nextLine();
            if (eventName.isEmpty() || eventName.isBlank()) {
                continue;
            }
            String date = scanner.nextLine();
            LocalDateTime localDateTime = convertTime(date);
            //hall info
            String hallNumber = scanner.nextLine();
            Hall hallByName = halls.getHallByName(hallNumber);
            Hall newHall = manualClone(hallByName);
            Event event = new Event(localDateTime, eventName, newHall);
            eventBooker.addEvent(event);
        }
    } catch (IOException e) {
        throw new FileException("Something went wrong with opening the
file");
    }
}

private Hall manualClone(Hall hall) {
    Hall hallCloned = new Hall(hall.getNumber(),
hall.getNUMBER_OF_ROWS(), hall.isFull());
    hallCloned.setTickets(new ArrayList<>());
    return hallCloned;
}
```

Методът manualClone създава копие на даден обект на Hall. Той инициализира нов екземпляр на Hall със същия номер, брой редове и пълен статус като оригиналната зала и след това задава своя списък с билети на нов, празен ArrayList. Това гарантира, че клонираната зала е отделен обект без споделено променливо състояние с оригиналната зала.

Имплементацията обработва четенето и обработката на данни за събитие от файл, определен от data[1]. Той отваря файла с помощта на FileReader и Scanner, за да премине през всеки ред. За всяко събитие той чете името на събитието, датата и номера на зала, преобразува низа за дата в LocalDateTime, клонира съответната зала и след това създава нов обект за събитие с тези подробности. Новото събитие се добавя към EventBooker.

9.3. Exit

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    System.out.println(
            "\nBye...");
    System.exit(0);    }
```

Методът е отговорен за прекратяването на приложението. Той отпечатва съобщение и след това извиква System.exit(0).

9.4. Save

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    try(FileWriter fileWriter=new FileWriter(data[1],false)){
        List<Event> events = eventBooker.getEvents();
        for (int i = 0; i < events.size(); i++) {
            Event event = events.get(i);
            String name = event.getName();
            LocalDateTime date = event.getDate();
            String time=convertDateToString(date);
            Hall hall = event.getHall();
            String number = "Hall-"+hall.getNumber();
            fileWriter.write(name);
            fileWriter.write("\n");
            fileWriter.write(time);
            fileWriter.write("\n");
            fileWriter.write(number);
            if(i<events.size()-1){
                fileWriter.write("\n");
                fileWriter.write(System.lineSeparator());
            }
        }
    } catch (IOException e) {
        throw new FileException("An error occurred while writing
file");
    }
}
```

Методът отговаря за записване на събития в текстов файл. Първо, методът отваря файловия писач за запис в посочения от `data[1]` файл, като задава режим на презаписване. След това извлича списък от събития от `EventBooker` и за всяко събитие записва името, датата и номера на залата в файла. Датата се преобразува в низ чрез метода `ConvertDateToString`. Между събитията се добавят нови редове и разделители, освен след последното събитие.

9.5. SaveAs

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    File file = new File(data[1]);
    try (FileWriter fileWriter = new FileWriter(file)) {
        List<Event> events = eventBooker.getEvents();
        for (int i = 0; i < events.size(); i++) {
            Event event = events.get(i);
            String name = event.getName();
```

```
            LocalDateTime date = event.getDate();
            String time=convertDateToString(date);
            Hall hall = event.getHall();
            String number = "Hall-"+hall.getNumber();
            fileWriter.write(name);
            fileWriter.write("\n");
            fileWriter.write(time);
            fileWriter.write("\n");
            fileWriter.write(number);
            if(i<events.size()-1){
                fileWriter.write("\n");
                fileWriter.write(System.lineSeparator());
            }
        }
    } catch (IOException e) {
        throw new FileException("File not saved");
    }
}
```

Методът е аналогичен на предишния, като този път се отваря `FileWriter` за запис.

9.6. Help

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    String builder =
            "\nhelp\t>\tsee available commands\n" +
...

                    "exit\t\t>\texit the program";
    System.out.println(builder);
}
```

Методът предоставя помощно ръководство за наличните команди в приложението. Той конструира форматиран низ, който изброява всички команди, техния съответен синтаксис и описания на техните функции. След това този низ се отпечатва на конзолата.

9.7. AddEvent

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    String time = data[1] + " " + data[2];
    LocalDateTime date = LocalDateTime.parse(time,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    String eventName = getEventName(data).toString();

    Hall hallByName = halls.getHallByName(data[3]);
    Hall newHall = manualClone(hallByName);
    Event event = new Event(date, eventName, newHall);
```

```
    for (Event eventBookerEvent : eventBooker.getEvents()) {
        if (eventBookerEvent.getName().equals(eventName)) {
            throw new EventException("Event already exists");
        }
    }
    eventBooker.addEvent(event);
}
```

Методът отговаря за добавяне на ново събитие в EventBooker. Първо се събира информацията за датата и времето от `data`, която се обединява и преобразува в `LocalDateTime` обект. След това се извлича името на събитието чрез `getEventName(data)`. Получава се обект на залата от `Halls`, базирана на предоставеното име на залата, и се създава клониран вариант на залата чрез метода `ManualClone`. Създава се ново събитие с помощта на тази зала и получената дата и име. Преди да се добави новото събитие, се проверява дали съществува събитие с такова име в `EventBooker`. Ако такова събитие вече съществува, се хвърля `EventException` със съобщение "Event already exists". Ако събитието не съществува, то се добавя в `EventBooker`.

9.8. Booking

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {


    if (data.length == 1) {
        // only "bookings"
        for (Event event : eventBooker.getEvents()) {
            for (Row row : event.getHall().getRows()) {
                for (Seat seat : row.getSeats()) {
                    if (seat.isBooked()) {
                        System.out.println(event);
                        System.out.println("with note:
"+seat.getNote());
                    }
                }
            }
        }
    } else if (data.length == 2) {
        // has name
        printOnlyByName(eventBooker.getEventByName(data[1]),
eventBooker);
    } else if (data.length == 3) {
        if (checkIfOnlyData(eventBooker, data)) return;
        printOnlyByName(eventBooker.getEventByName(data[1] + " " +
data[2]), eventBooker);
    } else {
        if (checkForDate(data)) {
            String time = data[1] + " " + data[2];
            LocalDateTime dateTime = LocalDateTime.parse(time,
```

```
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
            StringBuilder stringBuilder = getNameIfExists(data);
            if (!stringBuilder.isEmpty()) {
                printByNameAndDate(eventBooker, stringBuilder,
dateTime);
                return;
            }

            printOnlyByName(eventBooker.getEventByDate(dateTime),
eventBooker);

        } else {
            StringBuilder stringBuilder = getName(data);

printOnlyByName(eventBooker.getEventByName(stringBuilder.toString()),
eventBooker);
        }
    }
}
```

Методът в класа Booking обработва командата за извеждане на информация за резервации на събития в различни формати.

1. Когато `data` съдържа само един елемент (например `"bookings"`), методът обхожда всички събития в `EventBooker`. За всяко събитие преглежда редовете и местата в залата, като ако някое място е резервирано, извежда информация за събитието и бележката към резервацията.

2.Когато `data` съдържа два елемента, методът извиква `printOnlyByName`, за да извлече и покаже информация само за събитията с посоченото име.

3. Когато `data` съдържа три елемента, методът проверява дали вторият и третият елемент са дата и време. Ако е така, извиква `printByNameAndDate`, за да извлече информация за резервации на база на посоченото име и дата. Ако не е, извиква `printOnlyByName` с комбинираното име от втория и третия елемент.

4. Ако `data` съдържа повече от три елемента, методът проверява дали данните включват дата. Ако е така, извежда информация за събитията по дата и евентуално по име, ако такова е предоставено. В противен случай, използва `printOnlyByName`, за да извлече информация на база на предоставеното име.

Помощни методи:
1. printByNameAndDate:
Методът `printByNameAndDate получава информация за събитие по име и дата и извежда детайли за резервациите на това събитие. Първо, извиква метода `getEventByNameAndDate` на `EventBooker`, за да получи събитието с

конкретното име и дата. След това, методът `printDetailsForBookedSeats` извежда информация за резервациите на това събитие.

2. getNameIfExists:
Методът `getNameIfExists` извлича и събира името на събитието от масива `data`, започвайки от четвъртия елемент до предпоследния. Това е полезно, когато името на събитието е предоставено след дата и час и може да съдържа интервали. Методът проверява дали последният елемент в `data` е различен от третия, за да включи или не последния елемент в името на събитието.

3. printOnlyByName:
Методът `printOnlyByName` извежда информация за всички събития, които съвпадат с предоставеното име. Извиква метода `printDetailsForBookedSeats` на `EventBooker`, който отпечатва детайлите за резервираните места на тези събития.

4. checkIfOnlyData:
Методът `checkIfOnlyData` проверява дали предоставените данни съдържат само дата (без име на събитие). Ако е така, извиква `printOnlyByName` с информация за събития на определена дата. В противен случай връща `false`, за да сигнализира, че данните съдържат допълнителна информация.

5. checkForDate:
Методът `checkForDate` проверява дали предоставените данни съдържат валидна дата и час. Опитва се да анализира дата и час в определен формат. Ако успешното анализиране на датата е възможно, методът връща `true`; в противен случай, връща `false`. Това помага да се определи дали входните данни включват дата.

9.9. BookSeat

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    String time = data[3] + " " + data[4];
    String eventName = getEventName(data).toString();
    LocalDateTime date = LocalDateTime.parse(time,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    StringBuilder note = getNote(data);

    if (!eventBooker
            .getEventByNameAndDate(eventName, date)
            .getHall()
            .getRows()
            .get(Integer.parseInt(data[1]) - 1)
            .getSeatByNumber(data[2])
            .isBooked()) {
        eventBooker
```

```
                .getEventByNameAndDate(eventName, date)
                .getHall()
                .getRows()
                .get(Integer.parseInt(data[1]) - 1)
                .getSeatByNumber(data[2])
                .setNote(note.toString())
                .setBooked(true);
    } else {
        System.out.println("The seat is already booked.");
    }
}
```

Методът се използва за резервиране на седалка на събитие. Първо, той извлича дата и час от входния масив `data` и ги комбинира в един стринг, който се анализира в `LocalDateTime` обект. След това методът извлича името на събитието и бележката от входните данни. Използвайки тези данни, методът проверява дали избраната седалка е свободна. Ако седалката не е резервирана, се задава бележка на нея и тя се маркира като резервирана. Ако седалката вече е резервирана, методът извежда съобщение.

9.10. BuyTicket

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    //buy 2 3 2024-02-01 14:10:20 az
    String time = data[3] + " " + data[4];
    StringBuilder stringBuilder = getEventName(data);
    String name = stringBuilder.toString();
    LocalDateTime date = LocalDateTime.parse(time,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

    if (!eventBooker.getEventByNameAndDate(name,
date).getHall().isFull()) {
        // not full, add the ticket to the list of sold tickets
        eventBooker
                .getEventByNameAndDate(name, date)
                .getHall()
                .getRows()
                .get(Integer.parseInt(data[1]) - 1)
                .getSeatByNumber(data[2])
                .setPayed(true);

        Ticket ticket = eventBooker.getEventByNameAndDate(name, date)
                .getHall()
                .addTicket(new Ticket(data[2]));

        System.out.println("This is your code: " + ticket.getCode());
        Event event = eventBooker.getEventByNameAndDate(name, date);
        int size = event.getHall().getTickets().size();
        if (size == event.getHall().getMaxAmountOfTickets()) {
            event.getHall().setFull(true);
        }
    } else {
```

```
            System.out.println("The current event is full");
        }
    }
```

Методът се използва за закупуване на билет за събитие. Първо, той извлича датата и часа от входните данни и ги комбинира в обект. Също така извлича името на събитието. След това методът проверява дали залата на събитието не е пълна. Ако залата не е пълна, методът маркира избраната седалка като платена и добавя нов билет в списъка на продадените билети. След добавяне на билета, се извежда кодът на билета. Накрая, ако броят на продадените билети достигне максималния брой билети за събитието, залата се отбелязва като пълна. Ако залата е пълна, методът извежда съобщение.

9.11. Check

```
    public void process(EventBooker eventBooker, Halls halls, String[]
data) {
        for (Event event : eventBooker.getEvents()) {
            for (Ticket ticket : event.getHall().getTickets()) {
                if (ticket.getCode().equals(data[1])) {
                    System.out.println("Valid code");
                    System.out.println(event);
                    return;
                }
            }
        }
        throw new TicketException("Invalid code");
    }
}
```

Методът проверява валидността на билет по код и извежда информация за събитието, ако кодът е валиден. Той преминава през всички събития и техните билети, за да намери билет с код, съвпадащ с предоставения код в `data[1]`. Ако бъде намерен валиден код, методът извежда съобщение "Valid code" и информация за събитието. Ако кодът не бъде намерен, методът хвърля изключение `TicketException` с съобщение "Invalid code".

9.12. FreeSeats

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    String time = data[1] + " " + data[2];
    String eventName = getEventName(data).toString();
    LocalDateTime date = LocalDateTime.parse(time,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
    Event event = eventBooker.getEventByNameAndDate(eventName, date);
    for (int i = 0; i < event.getHall().getRows().size(); i++) {
        System.out.printf("Row %d has ", i + 1);
        int unbooked = 0, unpurchased = 0;
        for (Seat seat : event.getHall().getRows().get(i).getSeats()) {
```

```
            if (!seat.isBooked()) unbooked++;
            if (!seat.isPayed()) unpurchased++;
        }
        System.out.printf("%d unbooked and %d unpurchased tickets%n",
unbooked, unpurchased);
    }
}
```

Методът трябва да предостави информация за наличните места в залата за конкретно събитие на определена дата и час. Първо, методът извлича датата и името на събитието от входните данни и създава обект. След това намира събитието в `eventBooker` по името и датата. Преминава през всички редове в залата на събитието и изчислява броя на необработените и неплатени места за всеки ред. За всеки ред, методът брои местата, които не са резервирани (`unbooked`) и местата, които не са платени (`unpurchased`).

9.13. MostFamous

```
    public void process(EventBooker eventBooker, Halls halls, String[]
data) {
        Map<String, Integer> soldTicketForShow = new HashMap<>();
        for (Event event : eventBooker.getEvents()) {
            int size = event.getHall().getTickets().size();
            if (!soldTicketForShow.containsKey(event.getName())) {
                soldTicketForShow.put(event.getName(), size);
            } else {
                soldTicketForShow.put(event.getName(),
soldTicketForShow.get(event.getName()) + size);
            }
        }
        String mostWatched = "";
        Integer mostBought = 0;
        for (Map.Entry<String, Integer> stringIntegerEntry :
soldTicketForShow.entrySet()) {
            if (stringIntegerEntry.getValue() > mostBought) {
                mostBought = stringIntegerEntry.getValue();
                mostWatched = stringIntegerEntry.getKey();
            }
        }
        System.out.println("Most watched event is with name: " +
mostWatched + " and with sold ticket: " + mostBought);
    }
}
```

Методът идентифицира най-популярното събитие въз основа на броя на продадените билети. Създава хеш карта `soldTicketForShow`, която ще съхранява броя на продадените билети за всяко събитие. Преминава през всички събития в `eventBooker` и за всяко събитие актуализира броя на продадените билети в хеш картата. Ако събитието вече е в картата, добавя броя на билетите към текущата стойност; ако не е, добавя нов запис с броя на билетите.

След като всички събития са обработени, методът намира събитието с най-много продадени билети.

9.14. Report

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    String from = data[1] + " " + data[2];
    String to = data[3] + " " + data[4];
    LocalDateTime fromTime = convertTime(from);
    LocalDateTime toTime = convertTime(to);
    if(fromTime.isAfter(toTime)) { //SWAP
        //fromTime 2024
        //to time 2022
        LocalDateTime temp = fromTime;
        fromTime = toTime;
        toTime = temp;
    }
    if (data.length == 5) {
        //no hall number, group events by hall and print ticket details
        Map<String, List<Event>> hallsToEvents = new HashMap<>();
        for (Event event : eventBooker.getEvents()) {
            if (checkDates(fromTime, toTime, event)) {
                if
(hallsToEvents.containsKey(event.getHall().getNumber())) {

hallsToEvents.get(event.getHall().getNumber()).add(event);
                } else {
                    hallsToEvents.put(event.getHall().getNumber(), new
ArrayList<>());

hallsToEvents.get(event.getHall().getNumber()).add(event);
                }
            }
        }
        for (Map.Entry<String, List<Event>> stringListEntry :
hallsToEvents.entrySet()) {
            System.out.println("Hall " + stringListEntry.getKey());
            List<Event> value = stringListEntry.getValue();
            printTicketDetails(value);
        }
    } else if (data.length == 6) {
        // hall number, filter events by hall number and print ticket
details
        List<Event> result = new ArrayList<>();
        for (Event event : eventBooker.getEvents()) {
            if (checkDates(fromTime, toTime, event)) {
                if (event.getHall().getNumber().equals(data[5])) {
                    result.add(event);
                }
            }
        }
        printTicketDetails(result);
```

Методът извършва следните действия, за да генерира отчет на билетите за събития в определен период от време:

1. Присвояване на начални и крайни времена: Методът съчетава елементите от `data` за началната и крайната дата на периода и ги преобразува в обекти от тип `LocalDateTime`. След това проверява дали началната дата е след крайната и ако е така, размества стойностите, за да осигури правилния времеви интервал.

2. Обработка на събития без конкретна зала: Ако не е посочен номер на зала (т.е., `data.length` е 5), методът създава хеш карта `hallsToEvents`, която групира събитията по номера на залата. След това преминава през всички събития, проверява дали те са в зададения период чрез метода `checkDates` и ги добавя в съответната категория в картата. Накрая, методът извежда информация за билетите за всяка зала, използвайки метода `printTicketDetails`.

3. Обработка на събития за конкретна зала: Ако е посочен номер на зала (т.е., `data.length` е 6), методът събира събитията, които са в зададения период и са в конкретната зала. След това извежда информация за билетите на тези събития.

9.15. StatisticForTenPercent

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    Scanner scanner = new Scanner(System.in);

    // numberSeats,ticketsSold
    Map<String, Pair<Integer, Integer>> eventTickets = new HashMap<>();
    for (Event event : eventBooker.getEvents()) {
        if (!eventTickets.containsKey(event.getName())) {
            int ticketsSold = event.getHall().getTickets().size();
            int numberOfSeats = calculateNumberOfSeats(event);
            Pair<Integer, Integer> pair = new Pair<>(numberOfSeats,
ticketsSold);
            eventTickets.put(event.getName(), pair);
        } else {
            Pair<Integer, Integer> pair =
eventTickets.get(event.getName());
            int numberOfSeats = pair.getNumberOfSeats() +
calculateNumberOfSeats(event);
            pair.setNumberOfSeats(numberOfSeats);
            pair.setTicketsSold(pair.getTicketsSold() +
event.getHall().getTickets().size());
            eventTickets.put(event.getName(), pair);
        }
    }
    for (Map.Entry<String, Pair<Integer, Integer>> stringPairEntry :
```

```
eventTickets.entrySet()) {
        double percentage =
calculatePercentage(stringPairEntry.getValue());
        if (percentage < 10.00) {
            String eventName = stringPairEntry.getKey();
            System.out.println("Do you wish to remove this event
(yes/no): " + eventName);
            String answer = scanner.nextLine();
            if (answer.equalsIgnoreCase("yes")) {

eventBooker.removeEvents(eventBooker.getEventByName(stringPairEntry.get
Key()));
            }}
        }
    }
```

Методът изпълнява следните стъпки за управление на събитията въз основа на продажбата на билети:

Създаване на хеш карта с данни за събитията: Методът инициализира хеш карта eventTickets, която съхранява информация за броя на местата и продадените билети за всяко събитие. Преминава през всички събития и за всяко ново събитие добавя информация за броя на местата и продадените билети в картата. Ако съществува събитие с даденото име, обновява стойностите в картата.

Изчисляване на стойностите: Методът използва методите calculateNumberOfSeats и calculatePercentage за изчисление на броя на местата и процента на продадените билети. За събитията, при които процентът на продадените билети е по-малък от 10%, събитията се премахват от списъка на събитията чрез метода removeEvents.

Премахване на събития: Чрез проверка на потребителския вход, при желание методът премахва събитията, които не са достигнали минималната прагове за процента на продадените билети.

9.16. UnbookSeat

```
public void process(EventBooker eventBooker, Halls halls, String[]
data) {
    // unbook 3 2 2024-12-30 12:45:12 tedi
    String time = data[3] + " " + data[4];
    String eventName = getEventName(data).toString();
    LocalDateTime date = LocalDateTime.parse(time,
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

    if (eventBooker.getEventByNameAndDate(eventName, date)
            .getHall()
            .getRows()
            .get(Integer.parseInt(data[1]) - 1)
            .getSeatByNumber(data[2])
            .isBooked()) {

        eventBooker.getEventByNameAndDate(eventName, date)
                .getHall()
```

```
                .getRows()
                .get(Integer.parseInt(data[1]) - 1)
                .getSeatByNumber(data[2])
                .setBooked(false);
    } else {
        System.out.println("The seat is not booked");
    }
}
```

Извличане на данни от data: Методът започва с извличане на времето и името на събитието от входните данни (data). Времето се съчетава от две части, а името на събитието се получава чрез метода getEventName(data).
Преобразуване на време: Преобразува времето в обект от тип LocalDateTime, използвайки формата "yyyy-MM-dd HH:mm:ss".
Намиране на събитието: Използва методите getEventByNameAndDate за намиране на конкретното събитие, и след това се стига до съответния ред и място в залата.
Анулиране на резервацията: Проверява дали мястото е резерверирано (т.е., дали е booked). Ако мястото е резерверирано, методът го анулира, като задава setBooked(false). Ако мястото не е резерверирано, извежда съобщение.

4.3. Тестване

4.3.1. Отваряне на файла.

```
open data.txt
Opened.
```

```
close data.txt
Closing data.txt
```

4.3.2. Добавяне на представление

```
addevent 2024-11-01 15:00:00 Hall-244 Test
There is no such hall
addevent 2024-11-01 21:99:99 Hall-1 Test
Text '2024-11-01 21:99:99' could not be parsed: Invalid value for MinuteOfHour
addevent 2024-11-01 15:00:00 Hall-1 Test
addevent 2024-11-01 15:00:00 Hall-1 Test
Event already exists
```

4.3.3. Справка за свободни места

```
freeseats 2024-11-01 15:00:00 Test
Row 1 has 20 unbooked and 20 unpurchased tickets
Row 2 has 20 unbooked and 20 unpurchased tickets
Row 3 has 20 unbooked and 20 unpurchased tickets
Row 4 has 20 unbooked and 20 unpurchased tickets
Row 5 has 20 unbooked and 20 unpurchased tickets
```

```
book 1 1A 2024-11-01 15:00:00 Test "tedi"
```

```
freeseats 2024-11-01 15:00:00 Test
Row 1 has 19 unbooked and 20 unpurchased tickets
Row 2 has 20 unbooked and 20 unpurchased tickets
Row 3 has 20 unbooked and 20 unpurchased tickets
Row 4 has 20 unbooked and 20 unpurchased tickets
Row 5 has 20 unbooked and 20 unpurchased tickets
```

4.3.4. Редактиране статуса на билети

```
unbook 1 1A 2024-11-01 15:00:00 Test
freeseats 2024-11-01 15:00:00 Test
Row 1 has 20 unbooked and 20 unpurchased tickets
Row 2 has 20 unbooked and 20 unpurchased tickets
Row 3 has 20 unbooked and 20 unpurchased tickets
Row 4 has 20 unbooked and 20 unpurchased tickets
Row 5 has 20 unbooked and 20 unpurchased tickets
```

```
buy 1 1A 2024-11-01 15:00:00 Test
This is your code: KTtDG
freeseats 2024-11-01 15:00:00 Test
Row 1 has 20 unbooked and 19 unpurchased tickets
Row 2 has 20 unbooked and 20 unpurchased tickets
Row 3 has 20 unbooked and 20 unpurchased tickets
Row 4 has 20 unbooked and 20 unpurchased tickets
Row 5 has 20 unbooked and 20 unpurchased tickets
```

4.3.5. Справки

```
check KTtDG
Valid code
Event: Test
on: 2024-11-01T15:00
in hall: Hall-1
```

```
mostfamous
Most watched event is with name: Test and with sold ticket: 3
```

```
book 1 1B 2024-11-01 15:00:00 Test
bookings
Event: Test
on: 2024-11-01T15:00
in hall: Hall-1
with note: book 1 1B 2024-11-01 15:00:00 Test
```

```
addevent 2024-11-01 15:00:00 Hall-1 Test
buy 1 1C 2024-11-01 15:00:00 Test
This is your code: BTCHh
buy 1 1B 2024-11-01 15:00:00 Test
This is your code: iTrsv
buy 1 1D 2024-11-01 15:00:00 Test
This is your code: LcPv7
buy 1 1D 2024-11-01 15:00:00 Test
This is your code: 0S1RO
report 2022-11-01 14:00:00 2024-11-15 20:00:00 1
Ticket info: BTCHh
1C
Ticket info: iTrsv
1B
Ticket info: LcPv7
1D
Ticket info: 0S1RO
1D
```

```
statistic
Do you wish to remove this event (yes/no): Comedy Show
no
Do you wish to remove this event (yes/no): Business Meetup
no
Do you wish to remove this event (yes/no): Test
yes
Do you wish to remove this event (yes/no): Pop Concert
no
Do you wish to remove this event (yes/no): Dance Class
no
Do you wish to remove this event (yes/no): Tech Conference
no
```

4.3.6. Помощ

```
help

help    >   see available commands
    ----------------------------------
open    >   open <file.txt>
close   >   close currently open file
save    >   save currently open file as <file.txt>
saveas  >   save currently open file as <file2.txt>
    ----------------------------------
addevent    >   add event on <date> in <hall> with <name>
freeseats   >   check free seats on <date> for <name>
book        >   book <row> <seat> on <date> for <name> with <note>
unbook      >   unbook <row> <seat> on <date> for <name>
buy     >   buy <row> <seat> on <date> for <name>
bookings    >   see bookings for [<name>] [<date>]
check       >   check <code> validity
report      >   see purchases for events <from> <to> in [<hall>]
mostfamous  >   see best selling events
statistic   >   see worst selling events
    ----------------------------------
exit        >   exit the program
```

Глава 5. Заключение

Проектът „Билетна каса“ поддържа основните функционалности за работа с файлове и обработка на данни, както и допълнителни операции, с които потребителят може да извършва конкретни справки.

Подобрение на проекта може да се извърши в следните аспекти:

- Добавяне на графичен потребителски интерфейс за по-голямо удобство;

- Добавяне на функционалности, които да улеснят и подобрят достъпа до информацият, напр. възможности за допълнително филтриране на представления или резервиране, закупуване или отмяна на множество билети наведнъж;

- Усъвършенстване на валидацията на въвежданите команди и данни във файловете, напр. при добавяне на събитие;

- Добавяне на функции за работа с различни файлови формати;