Деница Беличева

6 група 1 курс | ФН: 7Mi0800072

Тема 4: Билети

Документация

**Глава 1. Увод**

* 1. **Описание и идея на проекта**

Проектът представлява информационна система, обслужваща билетна каса и поддържа най-необходимите операции за правилното ѝ функциониране. Чрез системата се дава възможност на потребителя с лекота да създава нови събития, да резервира и закупува билети и представя систематизирано най-необходимата информация за извършените действия под формата на справки.

* 1. **Цел и задачи на разработката**

Целта на разработката е да позволява извършването на зададените в условието изисквания по оптимален начин. Проектът реализира следните операции:

* Създаване на ново събитие по въведени дата, име на представление и зала, като се проверява дали съответната зала е свободна на подадената дата;
* Извеждане на справка за свободните места на представление на дата;
* Резервиране на билет и място на даден ред за представление, като се подава парола, чрез която клиентът в бъдеще да може да закупи своя билет. Позволява се добавянето на бележка към резервацията;
* Отменяне на резервация;
* Закупуване на билет за представление. Ако мястото вече е резервирано, клиентът е необходимо да въвежде и паролата си;
* Извеждане на справка за резервираните билети за представление, отговарящо на въведени име и дата, като те могат да бъдат и ALL, при което се извежда информация за всички представления на дадената дата или данни за всички дати. Справката се записва в текстов файл с име от типа „report-eventName-eventDate.txt“;
* Извеждане на справка за закупените билети за представление в даден период от време във въведена зала, като вместо номер, може да се въведе ALL.

Въведената информация и направените промени по време на действието на програмата се записват в текстови файлове, данните от които се прочитат и обработват в системата при повторно стартиране и така могат да се достъпват и при следващите изпълнения.

**Глава 2. Преглед на предметната област**

Един от най-големите проблеми, с който се сблъсках при работата по проекта, беше как най-правилно да организирам данните, който трябва да се съхраняват, така че смислово да се спазват изискванията, а също да улесня целият процес на разбработка на всяка от функционалностите.

По условие, всички данни за залите са предварително зададени и още в началото се прочитат от файл. Това за мен ознаваше, че трябва да бъдат записани в масив, който няма да бъде променян по време на изпълнението на програмата. В него единствено щеше да се търси дали подадена от потребителя зала наистина съществува, за да може да се създаде представление, което да се играе в нея. Така че този динамично заделен масив от зали нямаше да има нужда да се оразмерява, тъй като по никакъв начин не би имало нужда да се прибавят или премахват елементи от него.

Също, по някакъв начин трябваше да се съхранява всякаква информация за представленията (Event) и резервациите (Reservation) по време на изпълнение на програмата. Единият вариант, който обмислях, беше всякакви промени да се изпълняват директно във файловете, в които се запазват данните за тях. Но това означаваше във всяка функция да се прочита цялата информация от файла съответно за резервациите или представленията и веднага след това да се записва отново. Това щеше да се повтаря многократно, а работата с файла щеше да забавя значително процеса на програмата. Затова реших, че по-оптимално би било, както със залите, да се поддържат масиви за резервациите и представленията, съдържащи всякаква необходима информация за тях. Те се прочитат в началото на програмата от файлове, където се съхраняват, тази информация се зарежда в паметта и след това се обработва и променя според подадените команди от потребителя. При приключването на програмата, файловете, от които са били прочетени резервациите и представленията, се презаписват с вече променената информация и така това се повтаря при всяко изпълнение.

Един от въпросите, които си задавах, беше относно как да запиша данните, за да имам възможност да проверя по-лесно и бързо дали дадена зала е свободна на подадена дата, което се изисква още в първата функция за създаването на ново представление. Идея беше да се поддържа отделен клас, който да свързва залите и датите, на които са заети. Класът щеше да включва два динамично заделени масива – един от тип зала (Hall), в който да са записани всички данни за залите, и един от тип дата (Date), в който на индекса, съвпадащ с този на залата, да съоветства масив от датите, на които залата е заета. Тоест, това щеше да бъде масив от указатели, които да сочат към динамично заделени масиви от дати, които трябва да могат да променят размера си когато се появи нова дата, на която залата да е заета.

Друг вариант, който също обмислях, беше в класа Зала (Hall) да съществува член-данна, която да бъде динамично заделен масив от датите, на които залата е заета.

И за двете опции също идея беше датите да бъдат сортирани, за да може да се използва алгоритъмът за двоично търсене - Binary Search, за по-бързо търсене дали на съответна дата залата е заета. Но сортирането при всяко добавяне на дата, а след това и всяко прилагане на Binary Search, щеше да отнеме много повече време отколкото просто линейно търсене (Linear Search) и минаване през всички дати и сравняване. Затова тази идея отпадна.

В последствие, двата варианта – този с отделния клас с двата масива – от тип Зала и тип Дата, и другият, при който се поддържа член-данна масив от дати за всяка зала, отпаднаха също, защото не пасваха логически на цялата архитектура. Който и от тези варианти да бях избрала, тези дати трябваше да се записват някъде, от тях би следвало това да е във файла със залите, но, първо, той не би следвало да се променя, след като е и предварително зададен, и, второ, тези дати щяха да направят файла със залите по-трудно разбираем при четене, а ако трябваше да има отделен файл, където да се пазят заетите дати за всяка зала, програмата щеше да генерира прекалено много файлове, а информацията от тях щеше да се дублира на моменти, тъй като във файлът с представленията вече щеше задължително да фигурира в коя зала и на коя дата се играят те. Затова реших да използвам точно това – при създаване на представление да се минава през списъкът от вече създадени представления и да се проверява дали за подадената зала вече съществува представление на съответната дата.

Друг проблем, с който се сблъсках, беше как да се проверява дали места са свободни, резервирани или продадени при правене на нова резервация, закупуване на билети или извеждане на някоя от справките. Ясно ми беше, че всяко представление трябва по някакъв начин да поддържа „схема на залата“, в която се играе, каквато имат разпечатана на хартия продавачките на билети в билетните центрове, където отбелязват статуса на всяко място. И също как тази информация да се съхранява във файл. Ако тази схема на залата се представи чрез матрица, това, записано във файл, би изглеждало много неясно, а и би означавало за всяко представление да има файл с такава схема, защото няколко такива в един файл биха били трудни за възприемане.

Резервираните и запазените места така или иначе щяха да фигурират във файловете с резервациите под формата ред – място, затова реших, че просто ще си се съхраняват така, а за собствено удобство си създадох клас Места (Seats), който представлява матрица от тип enum с размерите на залата. При създаването на представление се създава обект на този клас и всяко място се отбелязва първоначално като свободно (available), а в последствие при правенето на резервация или запазване на място се отбелязва съответно като резервирано (reserved) или продадено (sold). В началото на програмата, при зареждането на данните от файла за представленията, а след това и този за резервациите, записаните резервирани/продадени места се отбелязват в схемите с местата за всяко представление и след това системата е готова да изпълнява командите на потребителите отново.

**Глава 3. Проектиране**

Споменатите по-горе 3 масива – този за залите, за представленията и резервациите, представляват част от член-данните на класа Билетен център (Ticket Center). Той „дърпа конците“ на цялата система, обработва подадените от потребителя команди и препраща към изпълнение на основните функционалности. Те създават обекти от основните класове – представление (Event) и резервация (Reservation), и ги добавят към големите списъци, премахват резервация, поддържат търсене на зали, представления, резервации, проверяват дали зала е заета на дадена дата и т.н. В main функцията е създаден обект на този клас и всичко останало се извършва от функцията run(), която извежда съобщение с поддържаните команди и номерата, на които те отговарят, а след това потребителят въвежда номерът, съответстващ на желаната от него команда.

Масивите, съхраняващи информацията за представленията и резервациите, всъщност представляват обекти на шаблонния клас List<T>, който поддържа динамично заделен масив от данни от тип Т, който има големина и капацитет. Класът се грижи при изчерпване на капацитета масивът от данни да се оразмери – създава нов масив с двойно по-голям капацитет, в който попълва данните на стария масив, който се изтрива и се заменя с новия, вече разширен такъв. Също е налична функция за добавяне на обекти от тип Т и изтриването на такива от масива с данните.

Основните класове, на които се базират тези, описващи представленията и резервациите, са класовете за символните низове (MyString), датите (Date), залите (Hall) и местата (Seats).

Класът MyString спестява нуждата от обработката на символни низове във всеки клас, в който са необходими такива – например член-данната, съхраняваща името на представление. Тук използвам разширена версия на вече разработен от мен клас MyString за предишни задачи по време на семестъра, като добавено е предефинирането на оператора за сравнение ==, който е подобен на вече разработената функция за сравнение compare, но тук връща булева стойност, отговаряща на това дали двата низа са еднакви или не. Използвам направо оператора ==, тъй като е по-удобен от функцията compare в контекста на проекта.

Класът Date съхранява и валидира подадените от потребителя данни за датите. В селекторите (setters) проверява дали подадената година е преди настоящата 2022-ра, дали съответния месец съществува и дали има такава дата от него и при правилно зададени данни създава обект от типа Дата. Иначе хвърля грешки, които се обработват в основните функции в класа TicketCenter и съобщението им се извежда до потребителя.

Класът Hall съдържа данните за залите – техния номер, броят редове и места на всеки ред. Поддържа предефинирани операторите << и >> за вход и изход и операторът == за сравнение между зали.

Класът Seats е гореописаната матрица от тип enum, в която се отбелязва статусът на съответното място – дали то е свободно, резервирано или вече закупено. Позволява се промяна на статус на място, отпечатване на схема на залата, в която с „o“ се отбелязват непродадените и незапазените билети, а с „x“ – продадените. Също, за удобство, присъства член-данната soldSeatsCount, която брои продадените места. В началото, в конструктора, ѝ се задава стойност 0, а след това, при всяка промяна на статус на място на продадено, стойността ѝ се увеличава с 1.

Класът Event се състои от име, дата, зала и схема с места, а класът Reservation – от представление, ред, място, парола, бележка и член-данна от типа Status – същия enum, който означаваше местата в класът Seats, която означава дали с резервацията се резервира или направо закупува съответното място.

**Глава 4. Реализация**

Тук ще разгледам реализацията на по-големите функции в класа TicketCenter, които реализират изискваните от условието функционалности.

Функцията newEvent() създава ново представление. Тя получава данни за неговото име, датата, на която ще се проведе, и номер на залата. След това, чрез функцията getHallIndex, се проверява дали същестува зала с подадения номер. Тази функция връща цяло число, което отговаря на индекса, на който се намира тази зала в масива със зали, ако тя съществува, а иначе връща -1. Ако залата съществува, с функцията isHallBuzy се обхожда масива с представленията и се проверява дали има представление, което да се играе в съответната зала на подадената дата. Ако няма, се създава ново представление и то се добавя в списъка с представления.

Функцията availableSeats() приема име на представление и дата, на която то се играе, намира го чрез findEvent функцията, която, подобно на getHallIndex, връща цяло число, показващо на кой индекс се намира представлението в списъка с представления, ако такова съществува, или -1, иначе. Ако е намерено представление, за него се извиква print() метода на класа Seats, който отпечатва схемата, показваща запазените и свободните места.

Функцията reserveTicket() резервира билет, като получава име и дата на представление, проверява дали такова съществува и ако да, приема номер на място и редът, на който то се намира. Ако зададеното място е свободно, потребителя въвежда парола за резервацията и бележка по желание и се създава нов обект от тип резервация, който се добавя в списъка с резервации, а съответното място се отбелязва като резервирано.

Функцията cancelReservation() отменя резервация. Тя приема данни за резервация, проверява дали такава съществува чрез findReservation функцията, която работи на същия принцип като findEvent и getHallIndex, и, ако е намерена резервация, иска от потребителя да въведе своята парола, за да я отмени. Ако паролата е правилна, резервацията се премахва от списъка, а досега резервираното място се отбелязва като свободно.

Чрез функцията buyTicket() се закупува билет по въведени данни за представление. Ако такова се намери в списъка с представления, се иска въвеждането на мястото и редът, на който то се намира. След това се проверява неговият статус. Ако е свободно, се създава нова резервация и то се отбелязва като заето, а ако е резервирано, се иска въвеждането на парола за направената резервация. При правилна парола се променя статуса на мястото в резервацията и също в схемата с местата за съответното представление то се отбелязва като заето.

Функцията listOfReservations() записва във файл справка за резервираните билети за представление по подадено име и дата, като се допуска въвеждане на ALL и за двете. Ако за име на представление се въведе ALL, първо започва да се оформя символен низ, който да отговаря на формата „report-NAME-DATE.txt“. В случая, за name се записва ALL, а след това се проверява какво е въведено за датата. Ако и тя е ALL, се създава файл, в който се записват данните за всички резервирани места. Иначе, от подадения низ за датата се оформя обект от тип дата чрез функцията stringToDate и се минава през списъка от резервации, за да се запишат във файла срещанията на резервации на тази дата.

Ако името на представление е различно от ALL, се проверява дали датата е ALL. В този случай във файла се записват всички резервации за съответното представление, независимо от датата му. Ако датата не е ALL, се оформя обект от тип дата от подадения символен низ и след това във файла се записват резервациите, които са за даденото представление на подадената дата.

stringToDate копира низа на датата, минава през всеки негов символ и когато той съответства на число, чрез atoi го превръща в число. nonDigitCount брои срещанията на символи, които не са числа, и така знае дали превърнатото от atoi число трябва да се присвои на деня, месеца или годината, защото се предполага, че при правилно въведена дата ще имаме два разделителя – един между деня и месеца и един между месеца и годината, независимо дали са точки, чертички или интервали. Ако датата е невалидна, това ще се прихване в сетърите на Data класа при създаването на обект.

Помощната функция formatDateString форматира подадения за датата низ. Потребителите може да са въвели датата с различен разделител между деня, месеца и годината, но тя не може да фигурира във формата „ден/месец/година“ в името на файла, т.е. с наклонените черти, тъй като те са забранени. Затова, в тази функция, разделителите задължително се задава да са тиренца „-“.

Функцията listOfBoughtTickets() извежда справка за закупените билети в подадена зала за даден период от време. Въвеждат се начална и крайна дата, като ако крайната е по-ранна от началната, просто се разменят чрез swapDates. Тук отново за номер на зала може да се въведе ALL и в този случай се извеждат данни за всички зали и закупените билети за датите в подадения период от време, а иначе се показва информация само за конкретната зала, ако такава, разбира се, съществува. Тук, освен да се изведат данните за съответното представление, се извежда и soldSeatsCount.

**Глава 5. Заключение**

Успешно са реализирани всички зададени в условието задължителни функционалности и направените ръчно тестове показват, че те работят коректно.

Направената от мен разработка по проекта може да бъде подобрена и оптимизирана в бъдеще като например се отдели halls масива в TicketCenter в отделен клас, който самостоятелно да се грижи за заделянето и освобождаването на динамична памет. Така кодът в TicketCenter ще бъде по-изчистен и лесно разбираем също.

Допълнително биха могли да се разработят и нови функционалности, които да разширят системата, като например бонусите от условието, а също и други видове справки, които да показват кои представления се справят най-добре, за да може да се планират и добавят повече такива.

**Изготвила: Деница Беличева**

**6 група, 1 курс**

**ФН: 7MI0800072**