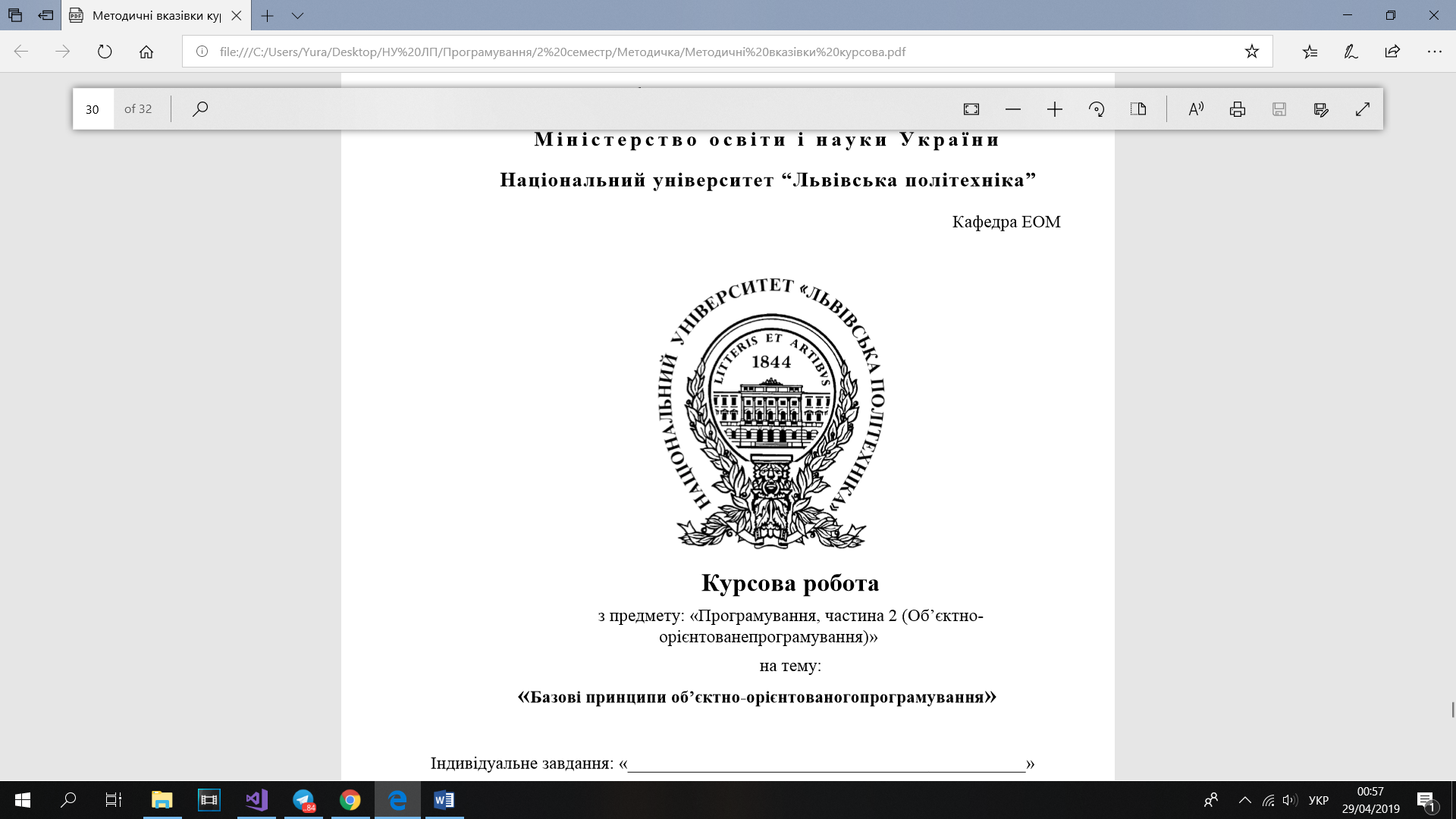
Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Курсова робота**

з предмету: «Програмування, частина 2 (Об’єктно-орієнтоване програмування)» на тему: «Базові принципи об’єктно-орієнтованогопрограмування»

**Індивідуальне завдання: «Театр»**

Виконав:

ст. гр. КІ-15

Леземезюк Р. Т.

Прийняв:

Козак Н.Б.

**Анотація**

Метою курсової роботи є створення спрощеної моделі функціонування театру.Написання програмного забезпечення було виконано мовою програмування С++ із використанням об'єктно-орієнтованого підходу для формування програми у вигляді набору сутностей на об'єктів, що взаємодіють. Програма включає у собі можливість роботи із текстовими файлами, а саме: запис та зчитування даних, пошук необхідної інформації та її видалення, а також сортування за певною характеристикою для зручнішого використання та керування інформацією. Програмне забезпечення передбачає кілька рівнів доступу, що дозволяє встановити роль користувача та надати йому лише ті можливості керування інформацією, що необхідні конкретно даній особі.

**Зміст**

Завдання на курсову роботу………………………….………………..……………4

Вступ……………………………………………….….………………….…….…….5

1Визначення поняття ООП (Об'єктно-орієнтоване програмування ), його основні принципи та обґрунтування вибору його технологій………………........6

1.Огляд та обгрунтування вибору технологій об'єктно-орієнтованого програмування……………………………..…………………………………...……6-7

1.2Фундаментальні поняття…...…………………………………………………….7

1.3Об’єктно-орієнтовані особливості…………………………………………8-14

1.4 Теорія про діаграми……………………………………………………….15-17

2.Проектування……………………………………………………………………..17

2.1.Аналіз об’єктно-орієнтованої моделі об’єкту та алгоритмів функціонування…………………………………………………………………….17

2.1.1. Теорія про діаграми ……………………………………………………17

2.1.1.1 Структурний аналіз предметної області.………………………………….17

2.1.1.2 Аналіз функціонального наповнення проекту……………………………18

2.1.1.3 Вибір інтерфейсу проекту та пунктів меню, які він реалізовує…………19

2.1.2 Аналіз об’єктно-орієнтованої моделі проекту……………………………...20

2.1.3 Аналіз алгоритмів функціонування проекту……………………………….21

2.2 Програмна реалізація проекту…………………………………………………25

2.2.1 Програмна реалізація об’єктно-орієнтованої моделі проекту…………….25

2.2.2 Програмна реалізація алгоритмів функціонування проекту…………27-30

2.2.3 Програмна реалізація інтерфейсу проекту та його пунктів меню……..30-32

3.Відлагодження………………………………………………………………33-35

ДОДАТОК А. Лістинг проекту………………………………………………35-51

**Завдання на курсову роботу**

У даній курсовій роботі повинно бути реалізовано на мові С++ таку сферу розваг як театр. У ній повинні бути використані основні принципи об’єктно-орієнтованого програмування.

**Вступ**

Об'єктно-орієнтоване програмування (ООП) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Попри те, що ця парадигма з'явилась в 1960-тих роках, вона не мала широкого застосування до 1990-тих, коли розвиток комп'ютерів та комп'ютерних мереж дозволив писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Однією з таких мов є С++. C++ (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом (англ. Bjarne Stroustrup) в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) 1979 року та початково отримала назву «Сі з класами». Згодом Страуструп перейменував мову на C++ у 1983 р. Базується на мові С. Вперше описана стандартом ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014. У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java.

***Метою виконання*** цієї курсової роботи є вивчення об’єктно-орієнтованої мови С++ та практичне застосування здобутих моїх знань у цій сфері.

**1.Визначення поняття ООП (Об'єктно-орієнтоване програмування ), його основні принципи та обґрунтування вибору його технологій**

**1.1.Огляд та обгрунтування вибору технологій об'єктно-орієнтованого програмування.**

C++ - універсальна мова програмування, яка дозволяє зробити програмування приємнішим для серйозного програміста. С++ було розроблено на основі мови програмування С. Крім можливостей, які дає С, С++ представляє свої гнучкі і ефективні засоби визначення нових типів, застосування яких значно полегшує та спрощує технологію програмування. Використовуючи визначення нових типів програміст може розділяти свою програму на простіші частини, які легше проконтролювати. Такий метод побудови програм часто називають абстракцією даних. Інформація про типи міститься в деяких обєктах типів, визначених користувачем. Все, чим ми маніпулюємо в програмі, може розглядатися як обєкт. Програмування із застосуванням таких обєктів називають обєктно-орієнтованим. В результаті раціонального використання цього методу одержуємо набагато коротші, доступніші для роззуміння та легше контрольовані прорами.

Ключевим поняттям C++ є клас. Клас - це структури, які містять не тільки обяви даних, а й функцій, що визначаються користувачем. Класи забезпечують приховування даних, гарантовану ініціалізацію даних, неявне перетворення типів для типів, визначених користувачем, динамічне задання типу, контрольоване користувачем управління памяттю і механізмом перегрузки операцій.

Основними поняттями обєктно-орієнтованого програмування є:

інкапсуляція;

спадковість;

абстракція;

поліморфізм.

Інкапсуляція полягає утому, що кожний клас має три рівні доступу до елементів:- рівень, дані на якому можуть бути використані будь-якою функцією;- елементи цього рівня можуть бути використані функціями-членами даного класу або функціями-членами похідних класів;елементи цього рівня можуть бути використані функціями-членами даного класу.

Дані рівні класів називають полями

Спадковість полягає у тому, що деякі класи можуть бути створені на основі інших, шляхом введення додаткових елементів.

При цьому ті класи, на основі яких створені інші, називаються базовими, а ті, які будуються на основі базових - похідними, які створюються шляхом простого копіювання всіх елементів базових класів, тому при описі похідних класів вказуються тільки ті елементи, які є додатковими до базового класу. Похідний клас може мати декілька базових класів, а базовий - декілька похідних.

Абстракція полягає у використанні віртуальних функкцій(функції, які обявляються в базовому класі, але в похідному класі можуть мати зовсім інше значення ). Віртуальна функція в базовому класі може не мати ніякого тіла, але в похідному повинна бути визначеною. Таким чином абстракція дозволяє, звертаючись до однієї і тієї ж функції в різних обєктах, виконувати різні дії.

Поліморфізм - можливість перевизначення різних функцій та операцій. Поліморфізм полягає в тому, що, по-перше, будь-яку операцію або функцію, яка визначена стандартним чином, можна перевизначити всередині класу так, що вона виконуватиме зовсім інші дії, по-друге, може існувати одночасно багато різних функцій у класі з однаковим імям, але з різною кількістю або типами параметрів.

Оголошення класів

С++ підтримує обяву класів, які інкапсулюють члени. Члени класа - члени даних і функцфії-члени. В членах-даних зберігаються значення, які представляють стан екземплярів класу(тобто обєктів які належать класу). Функції - члени встановлюють і перевіряють стан екземплярів класа, керуючи значеннями в членах даних. Загальний синтаксис обяви базового класу в С++ має такий вигляд:

Оголошення класу починається з ключового слова class, після якого слідує імя класу. Декларація членів класа заключається в пару фігурних дужок і закінчуються значком «крапка з комою».

**1.2Фундаментальні поняття**

В результаті дослідження Дебори Дж. Армстронг (англ. Deborah J. Armstrong) комп'ютерної літератури, що була видана протягом останніх 40 років, вдалось відокремити фундаментальні поняття (принципи), використані у переважній більшості визначень об'єктно-орієнтованого програмування. До них належить:

Клас

Клас визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (її атрибути або властивості) та дії, які вона здатна виконувати (її поведінки, методи або можливості). Наприклад, клас Собака може характеризуватись рисами, притаманними всім собакам, зокрема: порода, колір хутра, здатність гавкати. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Як правило, клас має бути зрозумілим для не-програмістів, що знаються на предметній області, що, у свою чергу, значить, що клас повинен мати значення в контексті. Також, код реалізації класу має бути досить самодостатнім. Властивості та методи класу, разом називаються його членами.

***Об'єкт***

Окремий екземпляр класу (створюється після запуску програми і ініціалізації полів класу). Клас Собака відповідає всім собакам шляхом опису їхніх спільних рис; об'єкт Сірко є одним окремим собакою, окремим варіантом значень характеристик. Собака має хутро; Сірко має коричнево-біле хутро. Об'єкт Сірко є екземпляром (примірником) класу Собака. Сукупність значень атрибутів окремого об'єкта називається станом. На основі класу Собака можна, також, створити інший об'єкт Дружок, який відрізнятиметься від об'єкта Сірко своїм станом (наприклад кольором хутра). Обидва об'єкта (Сірко і Дружок) є екземплярами класу Собака.

***Метод***

Можливості об'єкта. Оскільки Сірко — Собака, він може гавкати. Тому гавкати() є одним із методів об'єкта Сірко. Він може мати й інші методи, зокрема: місце(), або їсти(). В межах програми, використання методу має впливати лише на один об'єкт; всі Собаки можуть гавкати, але треба щоб гавкав лише один окремий собака.

***Обмін повідомленнями***

«Передача даних від одного процесу іншому, або надсилання викликів методів.»

***Абстрагування***

Спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. Наприклад Собака Сірко більшу частину часу може розглядатись як Собака, а коли потрібно отримати доступ до інформації специфічної для собак породи коллі — як Коллі і як Тварина (можливо, батьківський клас Собака) при підрахунку тварин Петра.

**1.3Об’єктно-орієнтовані особливості С++**

Сі++ додає до Сі об'єктно-орієнтовані можливості. Він вводить класи, які забезпечують три найважливіші властивості ООП: інкапсуляція, поліморфізм, успадкування.

### Проблеми старого підходу

В мові C основним способом організації даних були структури. Структура складається з набору полів, які ніяк не захищені. Якщо елементи структури мають змінну довжину, їх представляють у вигляді вказівників. Виділення і звільнення пам'яті під ці вказівники робляться вручну. Так, наприклад, одновимірний масив змінної довжини в мові C з перевіркою меж може представлятися таким чином:

**struct** Array {

double\* val;

int len;

};

void FreeArray (**const** **struct** Array\*);

void AllocArray (**const** **struct** Array\*, int len);

double Elem (**const** **struct** Array\*, int i);

void ChangeElem (**const** **struct** Array\*, int i, double x);

Така реалізація небезпечна і неефективна з багатьох причин:

* Необхідно викликати FreeArray і AllocArray. Програміст може забути викликати одну з цих функцій, або викликати її дуже рано/запізно, або двічі, або з вказівником на неправильний масив. Все це приводить до помилок, що важко виявити.
* Функції Elem і ChangeElem повільні.
* Немає ніякого способу перешкодити програмістам створювати і інші функції для роботи із структурою Array. Ці функції можуть робити з полями len і val будь-що.
* Немає ніякого способу перешкодити програмістам безпосередньо міняти поля len і val.
* Присвоєння об'єктів типу struct Array приведе до того, що їхні поля val указуватимуть на одну і ту ж область пам'яті. Немає ніякого способу ні заборонити присвоєння, ні змінити таку поведінку.

Мова Сі++, використовуючи ООП, усуває всі ці проблеми.

**Інкапсуляція**

Основним способом організації інформації в С++ є класи. На відміну від типу, структура (struct) мови С, що складається тільки з полів, клас (class) С++ складається з полів і функцій-членів або методів (англ. member functions). Поля бувають публічними (public), захищеними (protected) і приватними (private). У С++ тип структура аналогічний типу клас, відмінність в тому, що за умовчанням поля і функції-члени у структури публічні, а у класу — приватні.

З публічними полями можна робити зовні класу все, що завгодно. До захищених і приватних полів не можна звертатися ззовні класу, щоб не порушити цілісність даних класу. Спроба такого звернення викличе помилку компіляції. До таких полів можуть звертатися тільки функції-члени класу (а також так звані функції-друзі і функції-члени класів-друзів; про поняття друзів в C++ дивись нижче.) Поза тілом функцій-членів (а також друзів) захищені і власні поля недоступні навіть для читання. Такий захист полів називається інкапсуляциєю.

Використовуючи інкапсуляцію, автор класу може захистити свої дані від некоректного використання. Крім того, вона замислювалася для полегшення сумісної розробки класів. Малося на увазі, що зміна способу зберігання даних, якщо вони оголошені як захищені або приватні, не вимагає відповідних змін в класах, які використовують змінений клас. Наприклад, якщо в старій версії класу дані зберігалися у вигляді лінійного списку, а в новій версії — у вигляді дерева, ті класи, які були написані до зміни формату зберігання даних, переписувати не буде потрібно, якщо дані були приватними або захищеними (у останньому випадку — якщо використовуючі класи не були класами-нащадками), оскільки жоден з них цих класів не міг би безпосередньо звертатися до даних, а тільки через стандартні функції, які в новій версії мають вже коректно працювати з новим форматом даних. Навіть оператор доступу operator [] може бути визначений як така стандартна функція.

Функції-члени, як і поля, можуть бути публічними, захищеними і приватними. Публічні функції може викликати будь-хто, а захищені і власні — тільки функції-члени і друзі.Використовуючи інкапсуляцію, структуру Array з попереднього розділу можна переписати таким чином:

**class** **Array**

{**public**:

void Alloc(int new\_len);

void Free();

**inline** double Elem(int i);

**inline** void ChangeElem(int i, double x);

**protected**:

int len;

double\* val;

};

void Array::Alloc(int new\_len)

{**if** (len>0) Free(); len=new\_len; val=**new** double[new\_len];}

void Array::Free() {**delete** [] val; len=0;}

**inline** double Array::Elem(int i)

{assert(i>=0 && i<len ); **return** val[i];}

**inline** void Array::ChangeElem(int i, double x)

{assert(i>=0 && i<len); val[i]=x;}

Array a;

a.Alloc(10);

a.ChangeElem(3, 2.78);

double b = a.Elem(3);

a.Free();

**Успадкування**

Для створення класів з доданою функціональністю вводять успадкування. Клас-нащадок має поля і функції-члени базового класу, але не має права звертатися до приватних (private) полів і функцій базового класу. У цьому і полягає різниця між приватними і захищеними членами.

Клас-нащадок може додавати свої поля і функції або перевизначати функції базового класу.

За умовчанням, конструктор нащадка без параметрів викликає конструктор базового класу, а потім конструктори доданих елементів. Деструктор працює в зворотному порядку. Інші конструктори доводиться визначати щоразу наново. На щастя, це можна зробити викликом конструктора базового класу.

**class** **ArrayWithAdd** : **public** Array {

ArrayWithAdd(int n) : Array(n) {}

ArrayWithAdd() : Array() {}

ArrayWithAdd(**const** Array& a) : Array(a) {}

void Add(**const** Array& a);

};

Нащадок  — це більш ніж базовий клас, тому він може використовуватися скрізь, де використовується базовий клас, але не навпаки.

Успадкування буває публічним, захищеним і приватним. При публічному успадкуванні, публічні і захищені члени базового класу зберігають свій статус, а до приватних не можуть звертатися навіть функції-члени нащадка. Захищене успадкування відрізняється тим, що при нім публічні члени базового класу є захищеними членами нащадка. При приватному успадкуванні, до жодного члена базового класу навіть функції-члени нащадка права звертатися не мають. Як правило, публічне успадкування зустрічається значно частіше за інші.

Клас може бути нащадком декількох класів. Це називається множинним успадкуванням. Такий клас володіє полями і функціями-членами всіх його предків. Наприклад, клас FlyingCat може бути нащадком класів Cat і FlyingAnimal.

**class** **Cat** {

...

void Purr();

...

};

**class** **FlyingAnimal** {

...

void Fly();

...

};

**class** **FlyingCat** : **public** Cat, **public** FlyingAnimal {

...

PurrAndFly() {Purr(); Fly();}

...

};

**Поліморфізм**

Поліморфізмом в програмуванні називається перевизначення нащадком функцій-членів базового класу, наприклад

**class** **Figure** {

...

void Draw() **const**;

...

};

**class** **Square** : **public** Figure {

...

void Draw() **const**;

...

};

**class** **Circle** : **public** Figure {

...

void Draw() **const**;

...

};

В даному прикладі, яка з функцій буде викликана — Circle::Draw(), Square::Draw() або Figure::Draw(), визначається під час компіляції. Наприклад, якщо написати

Figure\* x = **new** Circle(0,0,5);

x->Draw();

то буде викликане Figure::Draw(), оскільки x — об'єкт класу Figure. Такий поліморфізм називається статичним.

Але в C++ є і динамічний поліморфізм, коли функція, що викликається, визначається під час виконання. Для цього функції-члени повинні бути віртуальними.

**class** **Figure** {

...

**virtual** void Draw() **const**;

...

};

**class** **Square** : **public** Figure {

...

**virtual** void Draw() **const**;

...

};

**class** **Circle** : **public** Figure {

...

**virtual** void Draw() **const**;

...

};

Figure\* figures[10];

figures[0] = **new** Square(1, 2, 10);

figures[1] = **new** Circle(3, 5, 8);

…

**for** (int i = 0; i < 10; i++)

figures[i]->Draw();

У цьому разі для кожного елементу буде викликана Square::Draw() або Circle::Draw() залежно від виду фігури.

Чисто віртуальною функцією називається функція-член, яка не визначена в базовому класі, а тільки в нащадках:

**class** **Figure** {

...

**virtual** void Draw() **const** = 0;

);**Прототипно-орієнтоване програмування**

Не всі з перелічених вище концепцій присутні в усіх об'єктно-орієнтованих мовах програмування. Зокрема, в прототипно-орієнтованому програмуванні не використовуються класи. Як наслідок, зовсім інша, але аналогічна термінологія використовується для визначення об'єкта та екземпляра в цих мовах.

#### **Агрегація**

[KP-UML-Aggregation-20060420.svg](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:KP-UML-Aggregation-20060420.svg)

Агрегація — проста асоціація між двома класами, яка відображає структурне відношення між рівноправними сутностями, коли обидва класи знаходяться на одному концептуальному рівні, і жоден з них не важливіший за решту. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має вищий ранг (ціле) і складається з декількох менших за рангом (частин). Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і її зображуєть як просту асоціацію з незафарбованим ромбом з боку «цілого».

Графічно агрегація представлена порожнім ромбом на блоці класу, і лінією, яка проведена від цього ромба до класу, що міститься в ньому.

#### **Композиція**

Композиція — більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено.

Графічно представляється як і агрегація, але з зафарбованим ромбиком.

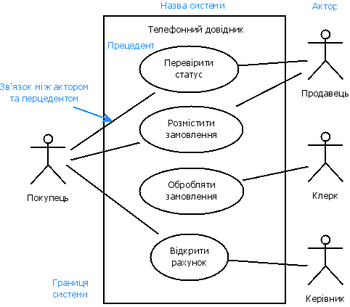
#### **Відмінності між композицією і агрегацією**

Відносини між класом Виш і класами Студент і Факультет злегка відрізняються один від одного, хоча обидва є відносинами агрегування. В виші може бути будь-яка кількість студентів (включаючи нуль), і кожен студент може навчатися в одному або декількох вишах; виш може складатися з одного або декількох факультетів, але кожен факультет належить одному і лише одному вишу. Відношення між класами Виш і Факультет називають композицією, так як при знищенні моделі Виш моделі факультетів, що належать цьому ВНЗ, також повинні бути знищені. Студент і Виш пов'язані агрегацією тому, що Студента не можна видалити при знищенні Вишу.

**1.4 Теорія про діаграми**

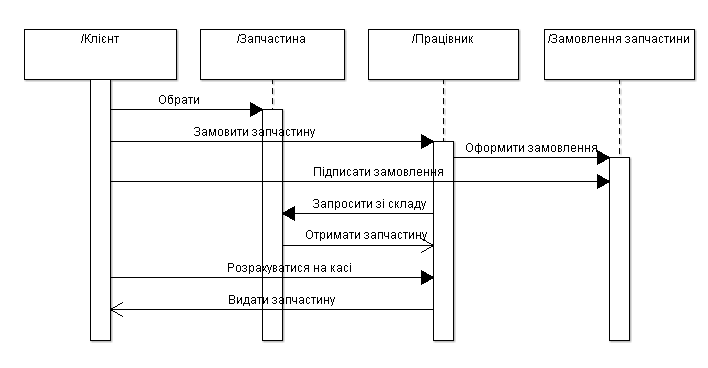
**Діаграма прецедентів**

Діаграма прецедентів — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.



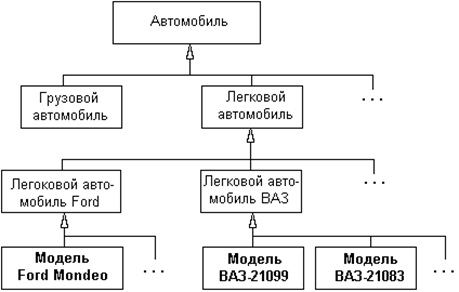
**Діагама послідовності**

**Діаграма послідовності** (англ. *sequence diagram*) — різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень.



**Діаграма класів**

**Діагра́ма кла́сів** (англ. *class diagram*) — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.[[1]](https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%96%D0%B2#cite_note-rumbaught-1) Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.



**Діаграми об’єктів**

**Діаграма об'єктів** — в UML, діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (англ. collaboration diagram).Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.



**2.Проектування**

**2.1.Аналіз об’єктно-орієнтованої моделі об’єкту та алгоритмів його функціонування**

**2.1.1.Аналіз контексту проекту**

**2.1.1.1 Структурний аналіз предметної області**

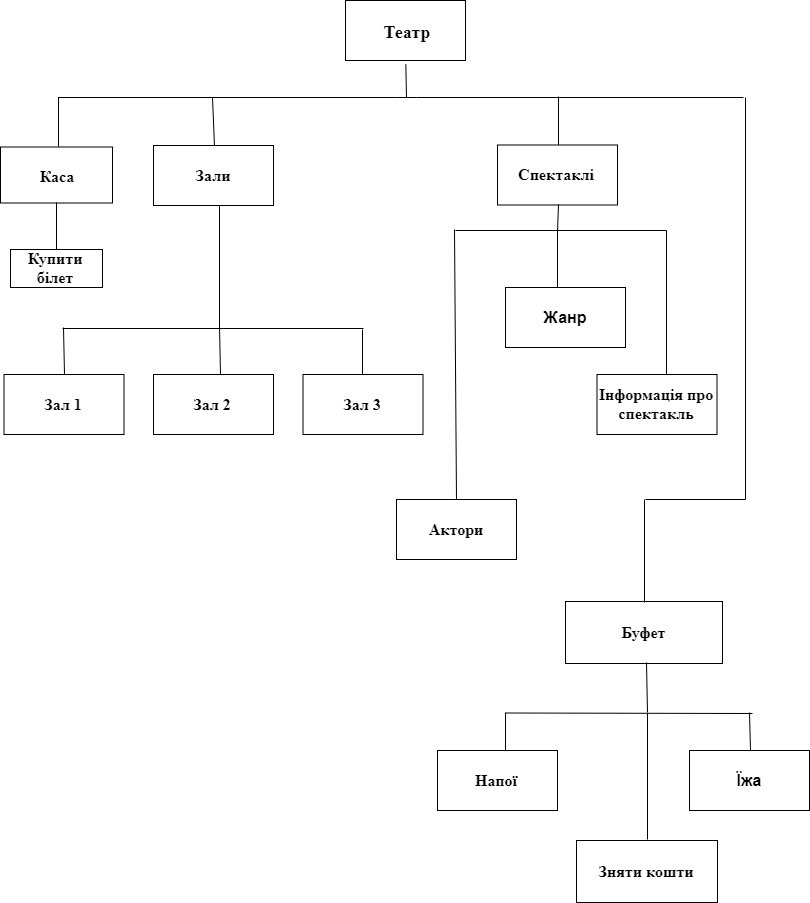
****

Рис.1 Опис предметної області

**2.1.1.2 Аналіз функціонального наповнення проекту**

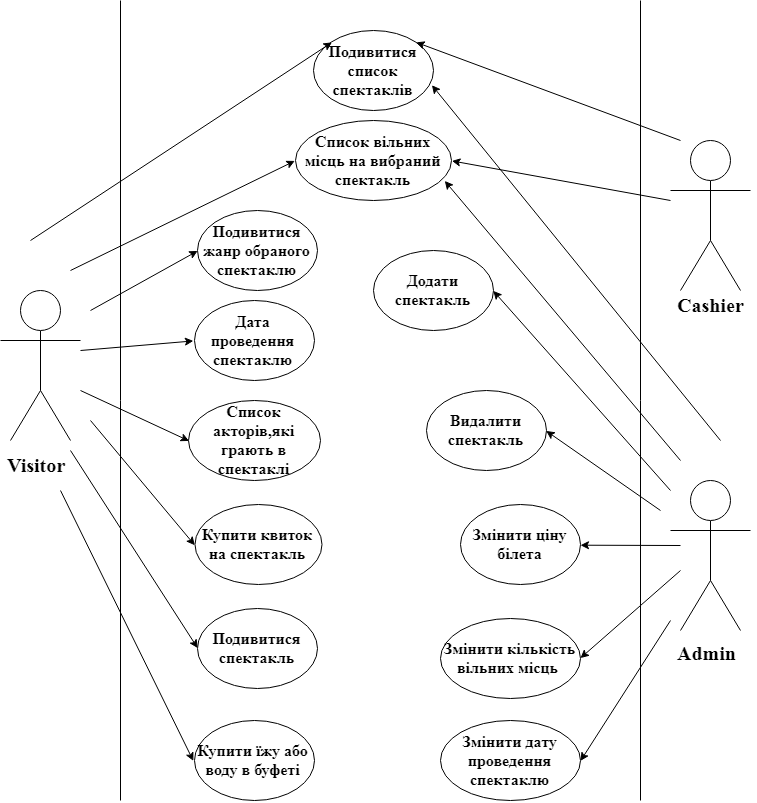
****

Рис.2 Діаграма прецендентів

**2.1.1.3 Вибір інтерфейсу проекту та пунктів меню, які він реалізовує**

Оскільки консольний тип меню є універсальний, я обрав його для реалізації курсової роботи.

Відкривши программу, нам пропонують зробити вибір Відвідувача,Касира або Адміна.

1 - Відвідувач [] – Для вибору режиму відвідувача;

2 – Касир []– Для вибору режиму касира;

3 - Адміністратор []– Для вибору режиму адміністратора;

4 - Вихід [] – Для виходу з програми;

Після вибору режиму роботу з'явиться наступне меню :

При виборі пункту “1”  у нас з'являється меню із такими даними –

* Список спектаклів
* Список вільних місць на вибраний спектакль
* Жанри доступних спектаклів
* Дата проведення спектакля
* Список акторів для вибраного спектаклю
* Купити білет на вибраний спектакль
* Подивитися спектакль
* Повернутися назад у меню

При виборі пункту “2”  у нас з'являється меню із такими даними –

* Подивитися список спектаклів
* Список вільних місць на вибраний спектакль
* Повернутися назад у меню

При виборі пункту “3”   у нас з'являється меню із такими даними –

* Подивитися список спектаклів
* Список вільних місць на вибраний спектакль
* Редагувати спектакль
* Додати спекталь
* Видалити спектакль
* Змінити ціну білета
* Змінити кількість вільних місць на обраний спектакль
* Змінити дату проведення спектаклю
* Повернутися назад у меню

При виборі пункту “4”  ми маємо змогу вийти з програми;

**2.1.2 Аналіз об’єктно-орієнтованої моделі проекту**

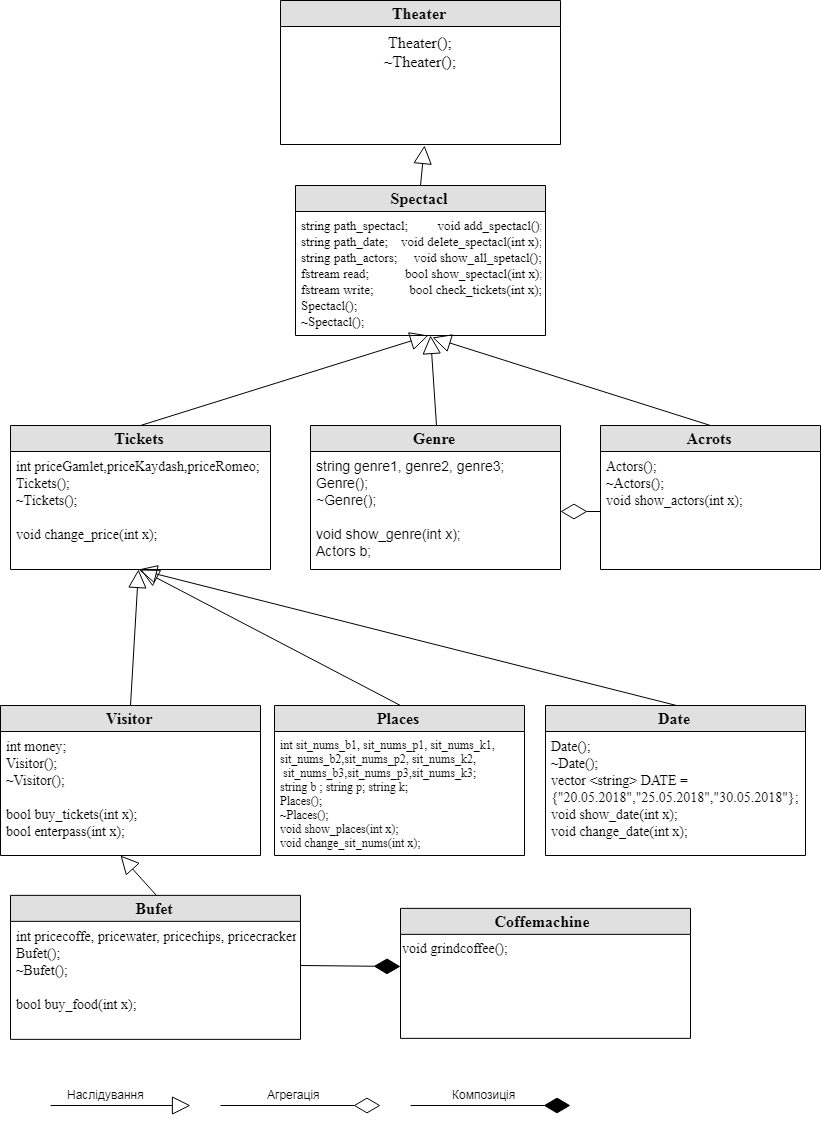
****

Рис.3 Діаграма класів

**2.1.3 Аналіз алгоритмів функціонування проекту**

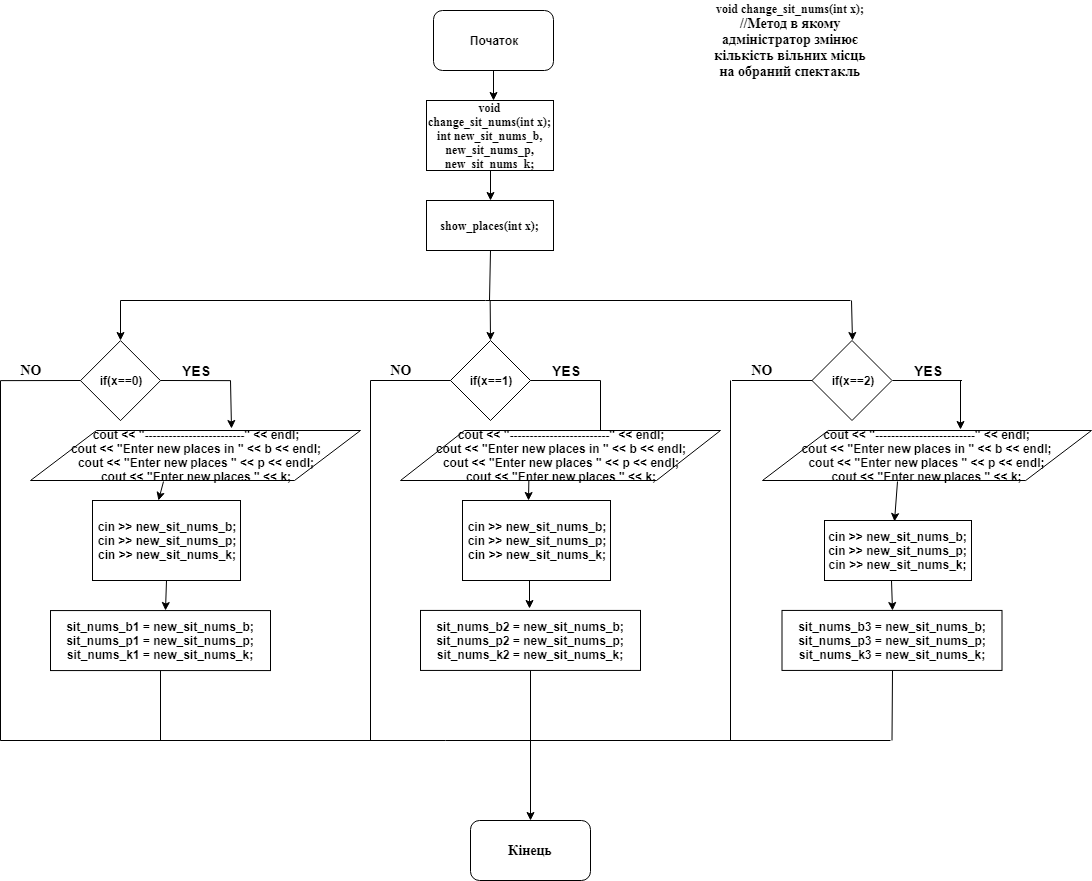
****

Рис.4 Блок-схема алгоритму метода change\_sit\_nums(int x)

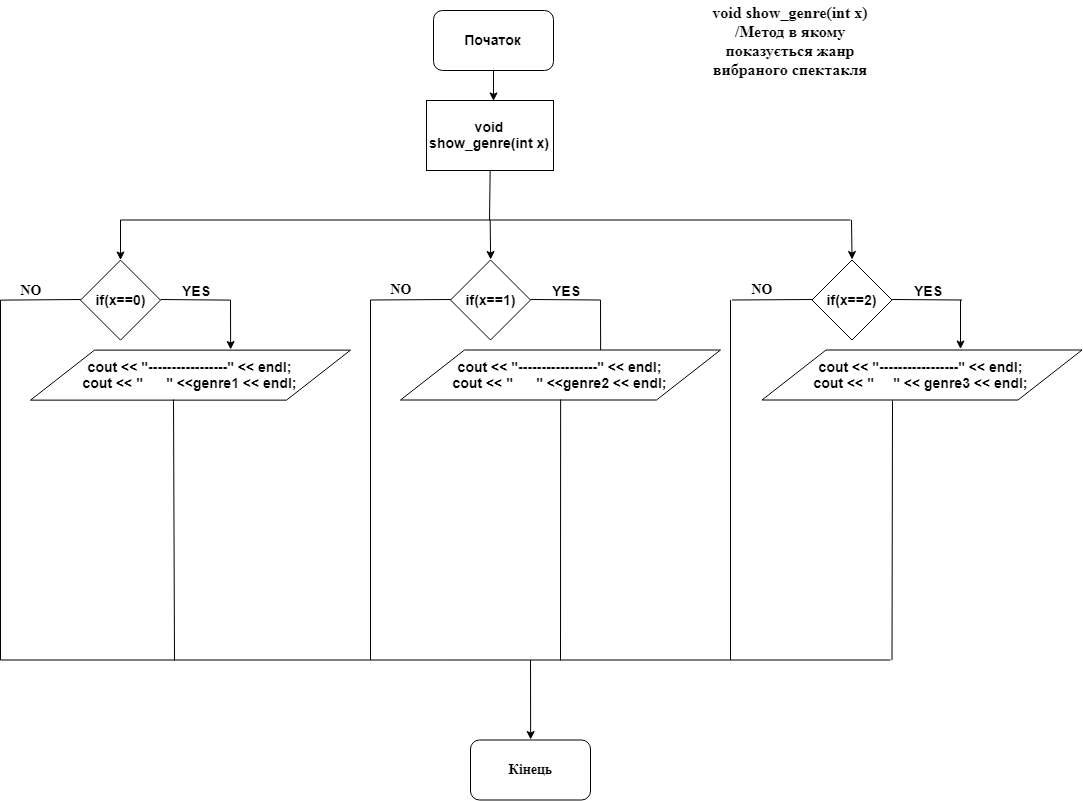
****

Рис.5 Блок-схема алгоритму метода show\_genre(int x)

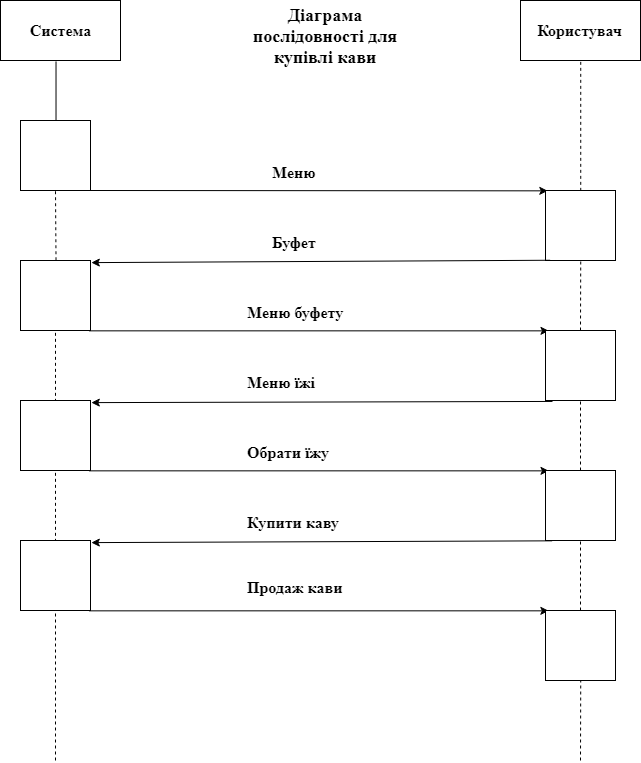
****

Рис.6 Діаграма послідовності

При запуску програми викликається меню, тоді користувач обирає буфет і відправляє повідомлення для виклику буфету. Далі система викликає меню буфету,а користувач обирає меню їжі. Система виводить на екран меню їжі, а користувач обрирає 1 позицію з меню, в даному випадку каву. Далі Система відтворює продаж кави користувачеві і далі повертається в попереднє меню.

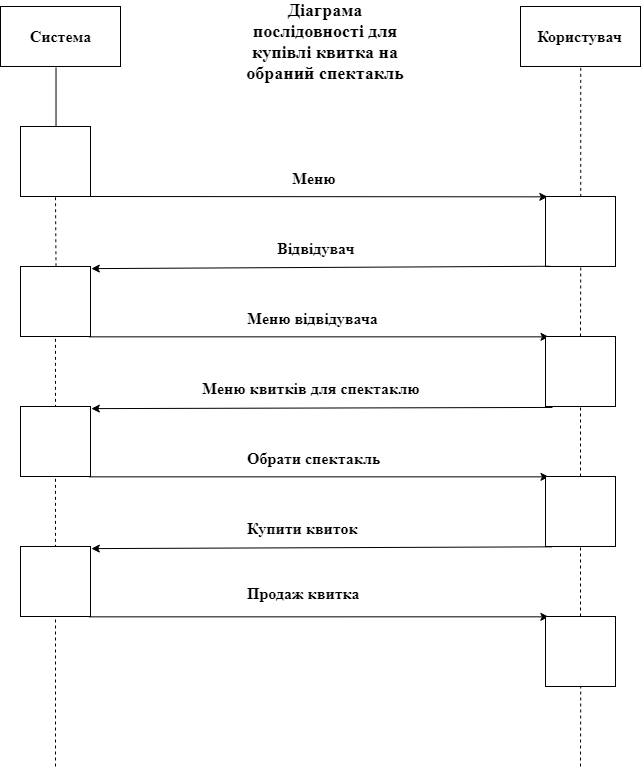


Рис.7 Діаграма послідовності

Спочатку система показує меню. Користувач обрав режим роботи Відвідувача. Далі відкривається Меню відвідувача, в ній користувач обирає меню квитків для спектаклів та обирає квиток на потрібний спектакль. Після цього Система виконує продаж квитка і далі повертається в попереднє меню.

**2.2 Програмна реалізація проекту**

**2.2.1 Програмна реалізація об’єктно-орієнтованої моделі проекту**

Класи Tickets, Genre , Actors наслідують клас Spectacl

Приклад наслідування : Tickets наслідує Spectacl

//Spectacl.h

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include "Theater.h"

#include <vector>

#include <Windows.h>

#include <iomanip>

using namespace std;

class Spectacl :

public Theater

{

protected:

string path\_spectacl;

string path\_date;

string path\_actors;

fstream read;

fstream write;

public:

int ticketgamlet, ticketkaydash, ticketromeo;

Spectacl();

~Spectacl();

vector <string> spectacl = { "Gamlet","Kaydashewa simya","Romeo and Juliet" };

vector <string> actors = { "Andriy Berejnuy" , "Mykola Jadan", "Vasyl Dilovyi" ,"Antonio Banderas","Vladyslav Svychuk","Pavlo Zyma" };

void add\_spectacl();

void delete\_spectacl(int x);

void show\_all\_spetacl();

bool show\_spectacl(int x);

bool check\_tickets(int x);

};

//Tickets.h

#pragma once

#include <iostream>

#include <string>

#include "Spectacl.h"

using namespace std;

class Tickets :

public Spectacl

{

public:

int priceGamlet,priceKaydash,priceRomeo;

Tickets();

~Tickets();

void change\_price(int x);

};

//Tickets.cpp

Tickets::Tickets():Spectacl()

{

priceGamlet = 40;

priceKaydash = 30;

priceRomeo = 50;

}

//Genre.h

#pragma once

#include <iostream>

#include <string>

#include "Spectacl.h"

#include "Actors.h"

using namespace std;

class Genre :

public Spectacl

{

protected:

string genre1, genre2, genre3;

public:

Genre();

~Genre();

Actors b;

void show\_genre(int x);

};

//Actors.h

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include "Spectacl.h"

class Actors :

public Spectacl

{

public:

Actors();

~Actors();

void show\_actors(int x);

};

**2.2.2 Програмна реалізація алгоритмів функціонування проекту**

bool Visitor::buy\_tickets(int x)

{

if (x == 0)

{

if (money < priceGamlet)

{

cout << "You don`t have a money" << endl;

return false;

}

else

cout << "You buy a ticket for spectacl Gamlet" << endl;

money -= priceGamlet;

cout << "Your remainder " << money << endl;

ticketgamlet++;

return true;

}

if (x == 1)

{

if (money < priceKaydash)

{

cout << "You don`t have a money" << endl;

return false;

}

else

cout << "You buy a ticket for spectacl Kaydashewa simya" << endl;

money -= priceKaydash;

cout << "Your remainder " << money << endl;

ticketkaydash++;

return true;

}

if (x == 2)

{

if (money < priceRomeo)

{

cout << "You don`t have a money" << endl;

return false;

}

else

cout << "You buy a ticket for spectacl Romeo and Juliet" << endl;

money -= priceRomeo;

cout << "Your remainder " << money << endl;

ticketromeo++;

return true;

}

}

Метод buy\_tickets Класу Visitor, реалізовує покупку білета. Для початку він перевіряє чи ваших грошей вистачає для покупки. Якщо вистачає, то він переходить до наступного рядку, якщо ні , то вказує на це і виводить на екран, що грошей недосатньо. Далі він віднімає від ваших грошей ціну квитка і виводить на екран вашу решту від покупки білету.

bool Spectacl::check\_tickets(int x)

{

if (x == 0)

{

if (ticketgamlet == 0)

{

cout << "Sorry but you haven`t tickets for this spectacl" << endl;

return false;

}

else

ticketgamlet --;

cout << "The amount of your tickets " << ticketgamlet << endl;

}

if (x == 1)

{

if (ticketkaydash == 0)

{

cout << "Sorry but you haven`t tickets for this spectacl" << endl;

return false;

}

else

ticketkaydash--;

cout << "The amount of your tickets " << ticketkaydash << endl;

}

if (x == 2)

{

if (ticketromeo == 0)

{

cout << Sorry but you haven`t tickets for this spectacl << endl;

return false;

}

else

ticketromeo--;

cout << "The amount of your tickets " << ticketromeo << endl;

}

}

Метод check\_tickets Класу Spectacl , реалізовує перевірку наявності білета.Якщо білет було куплено то метод поверне значення true і відвідувач буде мати змогу подивитися спектакль,після цього в нього відніметься квиток, якщо білет відсутній то метод поверне значення false і відвідувач отримає повідомлення “Sorry but you haven`t tickets for this spectacl”.

bool Bufet::buy\_food(int x)

{

if (x == 0)

{

if (money < pricecoffe)

{

cout << "You haven`t money!" << endl;

return false;

}

else

cout << "You buy a coffe" << endl;

money -= pricecoffe;

}

if (x == 1)

{

if (money < pricewater)

{

cout << "You haven`t money!" << endl;

return false;

}

else

cout << "You buy a water" << endl;

money -= pricewater;

}

if (x == 2)

{

if (money < pricechips)

{

cout << "You haven`t money!" << endl;

return false;

}

else

cout << "You buy a chips" << endl;

money -= pricechips;

}

if (x == 3)

{

if (money < pricecrackers)

{

cout << "You haven`t money!" << endl;

return false;

}

else

cout << "You buy a crackers" << endl;

money -= pricecrackers;

}

}

Метод buy\_food Класу Bufet реалізовує покупку їжі.Після вибору пункту з меню відбувається перевірка чи достатньо у вас грошей,щоб придбати товар з буфету.Якщо достатньо метод поверне значення true і ви успішно придбаете товар,після чого від вашої кількості грошей відніметься вартість товару. Якщо не достатньо метод поверне значення false і виведеться таке повідомлення: "You haven`t money!".

void Spectacl::add\_spectacl()

{

string new\_spectcl;

read.open(path\_spectacl, fstream::out | fstream::app);

if (!read.is\_open())

{

cout << "Eror! File doesn`t open" << endl;

}

else

{

cout << " Enter spectacl what you want to add: " << endl;

cin >> new\_spectcl;

spectacl.push\_back(new\_spectcl);

read << endl;

read << spectacl[3];

cout << "Spectacl added successefully!" << endl;

getchar();

}

read.close();

}

Метод add\_spectacl Класу Spectacl реалізовує додавання нового спектаклю в файл “Spectacl.txt”.

**2.2.3 Програмна реалізація інтерфейсу проекту та його пунктів меню**

extern struct MenuElement menuA[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuB[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuC[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuSpec[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuActors[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuDate[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuDate1[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuGenre[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuPlaces[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuDEL[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuTickets[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menubuyTickets[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menushowSpec[MAX\_MENU\_ITEMS\_COUNT];

extern struct MenuElement menuBufet[MAX\_MENU\_ITEMS\_COUNT];

struct MenuElement

baseMenu[MAX\_MENU\_ITEMS\_COUNT] = {

{ "Visitor", menuA, (void(\*)())NULL },

{ "Cashier",menuB, (void(\*)())NULL},

{ "Admin", menuC, (void(\*)())NULL },

{ "Exit", (struct MenuElement \*)NULL, Exit },

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuA[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Show all spectacl", (struct MenuElement \*)NULL, show\_all\_spetacl },

{ "List of places for the selected spectacl",menuSpec,(void(\*)())NULL },

{ "The genres of available spectacl ", menuGenre, (void(\*)())NULL },

{ "Date of holding",menuDate, (void(\*)())NULL},

{ "List of actors for selected spectacl",menuActors, (void(\*)())NULL},

{ "Buy tickets for spectacl",menubuyTickets, (void(\*)())NULL},

{ "Show spectacl",menushowSpec, (void(\*)())NULL},

{ "Bufet",menuBufet, (void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuSpec[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startplaces },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startplaces1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startplaces2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuBufet[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Coffe | 10 UAH ", (struct MenuElement \*)NULL, startbuyfood },

{"Water | 5 UAH",(struct MenuElement \*)NULL,startbuyfood1},

{"Chips | 20 UAH",(struct MenuElement \*)NULL,startbuyfood2},

{"Crackers | 15 UAH",(struct MenuElement \*)NULL,startbuyfood3},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menushowSpec[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startspectacl },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startspectacl1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startspectacl2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menubuyTickets[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Tickets for spectacl Gamlet", (struct MenuElement \*)NULL, startbuy },

{"Tickets for spectacl Kaydashewa simya",(struct MenuElement \*)NULL,startbuy1},

{"Tickets for spectacl Romeo and Juliet",(struct MenuElement \*)NULL,startbuy2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuActors[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startactors },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startactors1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startactors2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuDate[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startdate },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startdate1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startdate2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuDate1[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startDate },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startDate1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startDate2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuGenre[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuA, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startGenre },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startGenre1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startGenre2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuPlaces[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startPlaces },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startPlaces1},

{"Romeo and Juliet",(struct MenuElement \*)NULL,startPlaces2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuTickets[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Gamlet price", (struct MenuElement \*)NULL, starttickets },

{"Kaydashewa simya price ",(struct MenuElement \*)NULL,starttickets1},

{"Romeo and Juliet price",(struct MenuElement \*)NULL,starttickets2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuDEL[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", menuC, (void(\*)())NULL },

{ "Gamlet", (struct MenuElement \*)NULL, startdel },

{"Kaydashewa simya",(struct MenuElement \*)NULL,startdel1 },

{"Romeo and Juliet",(struct MenuElement \*)NULL,startdel2},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

,

menuB[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Show available spectacl", (struct MenuElement \*)NULL, show\_all\_spetacl },

{ "List of places for the selected spectacl",menuSpec, (void(\*)())NULL},

{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

}

, menuC[MAX\_MENU\_ITEMS\_COUNT] = {

{ "PREV MENU", baseMenu, (void(\*)())NULL },

{ "Show all spectacl", (struct MenuElement \*)NULL, show\_all\_spetacl },

{ "List of places for the selected spectacl",menuSpec, (void(\*)())NULL },

{ "Add spectacl", (struct MenuElement \*)NULL, add\_spectacl },

{ "Delete spectacl", menuDEL,(void(\*)())NULL },

{ "Change ticket price", menuTickets, (void(\*)())NULL},

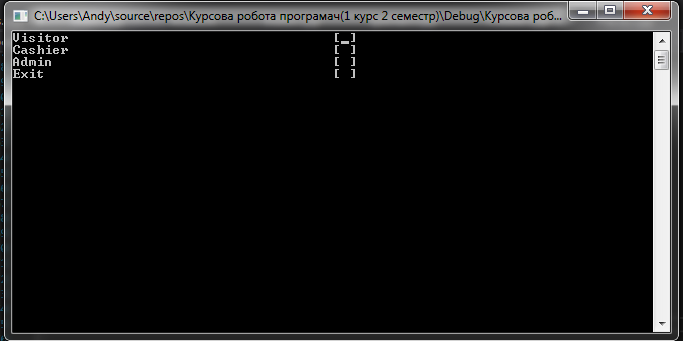
{ "Change the number of places for the selected spectacl",menuPlaces, (void(\*)())NULL },

{ "Change the date of the spectacl", menuDate1,(void(\*)())NULL },

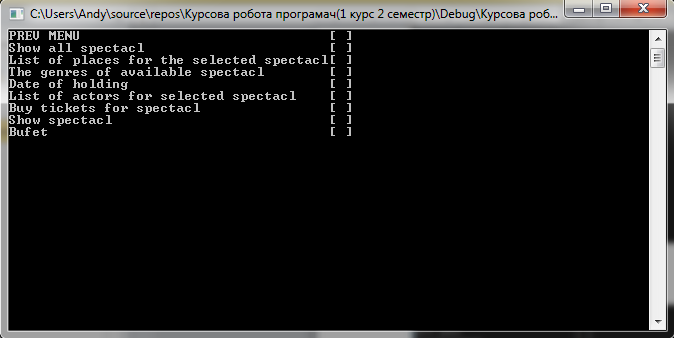
{ "", (struct MenuElement \*)NULL, (void(\*)())NULL }

};

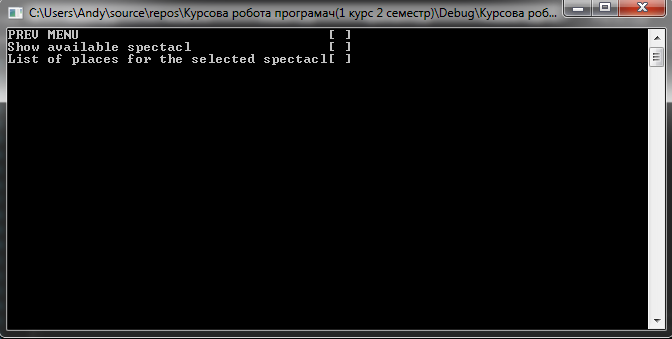
**3.Відлагодження**



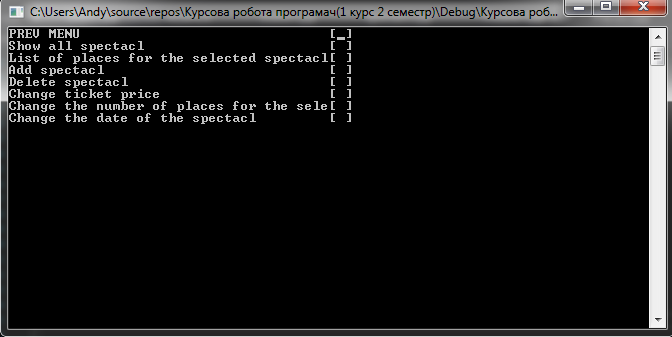
**Рис 3.1 Базове меню**



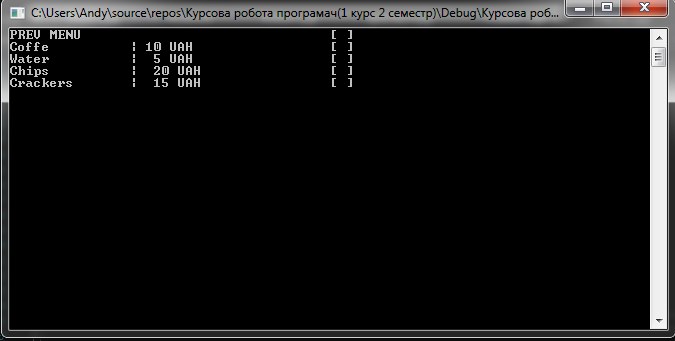
**Рис 3.2 Меню відвідувача**



**Рис 3.3 Меню касира**



**Рис 3.4 Меню адміна**



**Рис 3.5 Меню буфету**

**ДОДАТОК А. Лістинг проекту**

|  |
| --- |
| //Actors.h  #pragma once  #include <iostream>  #include <string>  #include <fstream>  #include <vector>  #include "Spectacl.h"  class Actors :  public Spectacl  {  public:  Actors();  ~Actors();    void show\_actors(int x);  };  //Actors.cpp  #include "Actors.h"  Actors::Actors()  {  path\_actors = "Actors.txt";  }  Actors::~Actors()  {  }  void Actors::show\_actors(int x)  {  string line;  if (x == 0)  {  read.open(path\_actors, fstream::in);  for (int i = 0; i < 2; i++)  {  getline(read, line);  cout << line << endl;  }  read.close();  }  if (x == 1)  {  read.open(path\_actors, fstream::in);  for (int i = 0; i < 2; i++) {  getline(read, line);  }  getline(read, line);  cout << line << endl;  getline(read, line);  cout << line << endl;  read.close();  }  if (x == 2)  {  read.open(path\_actors, fstream::in);  for (int i = 0; i < 4; i++) {  getline(read, line);  }  getline(read, line);  cout << line << endl;  getline(read, line);  cout << line << endl;  read.close();  }  }    //Bufet.h  #pragma once  #include "Theater.h"  #include <iostream>  #include "Visitor.h"  using namespace std;  class Bufet :  public Visitor  {  private:  class Coffemachine  {  public:  void grindcoffee();    };  Coffemachine g;  public:  int pricecoffe, pricewater, pricechips, pricecrackers;  Bufet();  ~Bufet();  void grindcoffee();  bool buy\_food(int x);    };  //Bufet.cpp  #include "Bufet.h"  Bufet::Bufet()  {  pricecoffe = 10;  pricewater = 5;  pricechips = 20;  pricecrackers = 15;  }  Bufet::~Bufet()  {  }  void Bufet::grindcoffee()  {  g.grindcoffee();  }  bool Bufet::buy\_food(int x)  {  if (x == 0)  {  if (money < pricecoffe)  {  cout << "You haven`t money!" << endl;  return false;  }  else  cout << "You buy a coffe" << endl;  money -= pricecoffe;  }  if (x == 1)  {  if (money < pricewater)  {  cout << "You haven`t money!" << endl;  return false;  }  else  cout << "You buy a water" << endl;  money -= pricewater;  }  if (x == 2)  {  if (money < pricechips)  {  cout << "You haven`t money!" << endl;  return false;  }  else  cout << "You buy a chips" << endl;  money -= pricechips;  }  if (x == 3)  {  if (money < pricecrackers)  {  cout << "You haven`t money!" << endl;  return false;  }  else  cout << "You buy a crackers" << endl;  money -= pricecrackers;  }    }  void Bufet::Coffemachine::grindcoffee()  {    cout << "Your coffe are grind!" << endl;    }  //Date.h  #pragma once  #include <iostream>  #include <string>  #include <fstream>  #include "Tickets.h"  class Date :  public Tickets  {  public:  Date();  ~Date();  vector <string> DATE = { "20.05.2018","25.05.2018","30.05.2018" };  void show\_date(int x);  void change\_date(int x);  };  //Date.cpp  #include "Date.h"  Date::Date()  {  path\_date = "Date.txt";  }  Date::~Date()  {  }  void Date::show\_date(int x)  {  string date;  if (x == 0)  {  read.open(path\_date, fstream::in);  read >> date;  cout << date << endl;  read.close();  }  if (x == 1)  {  read.open(path\_date, fstream::in);  getline(read, date);  getline(read, date);  cout << date << endl;  read.close();  }  if (x == 2)  {  read.open(path\_date, fstream::in);  getline(read, date);  getline(read, date);  getline(read, date);  cout << date << endl;  read.close();  }  }  void Date::change\_date(int x)  {  string date;    if (x == 0)  {  cout << "Enter new date: " << endl;  cin >> date;  read.open(path\_date, fstream::out);  DATE.erase(DATE.begin() + 0);  DATE.insert(DATE.begin(), date);  for (int i = 0; i < DATE.size(); i++)  {  read << DATE[i];  read << endl;  }  read.close();  }  if (x == 1)  {  cout << "Enter new date: " << endl;  cin >> date;  read.open(path\_date, fstream::out);  DATE.erase(DATE.begin() + 1);  DATE.insert(DATE.begin()+1, date);  for (int i = 0; i < DATE.size(); i++)  {  read << DATE[i];  read << endl;  }  read.close();  }  if (x == 2)  {  cout << "Enter new date: " << endl;  cin >> date;  read.open(path\_date, fstream::out);  DATE.erase(DATE.begin() + 2);  DATE.insert(DATE.begin()+2, date);  for (int i = 0; i < DATE.size(); i++)  {  read << DATE[i];  read << endl;  }  read.close();  }  }  //Genre.h  #pragma once  #include <iostream>  #include <string>  #include "Spectacl.h"  #include "Actors.h"  using namespace std;  class Genre :  public Spectacl  {  protected:  string genre1, genre2, genre3;  public:  Genre();  ~Genre();  Actors b;  void show\_genre(int x);  };  // Genre.cpp  #include "Genre.h"  Genre::Genre()  {  genre1 = "Drama";  genre2 = "Comedy";  genre3 = "Tragedy";  }  Genre::~Genre()  {  }  void Genre::show\_genre(int x)  {    if (x == 0)  {  cout << "-----------------" << endl;  cout << " " <<genre1 << endl;  }  if (x == 1)  {  cout << "-----------------" << endl;  cout << " " << genre2 << endl;  }  if (x == 2)  {  cout << "-----------------" << endl;  cout << " " << genre3 << endl;  }  }  //Places.h  #pragma once  #include <iostream>  #include <string>  #include "Tickets.h"  using namespace std;  class Places :  public Tickets  {  protected:    int sit\_nums\_b1, sit\_nums\_p1, sit\_nums\_k1, sit\_nums\_b2, sit\_nums\_p2, sit\_nums\_k2, sit\_nums\_b3, sit\_nums\_p3, sit\_nums\_k3;    string b ;  string p;  string k;    public:  Places();    ~Places();  void show\_places(int x);  void change\_sit\_nums(int x);    };  //Places.cpp  #include "Places.h"  Places::Places()  {  b = "balcony: ";  p = "parter: ";  k = "side: ";  sit\_nums\_b1 = 5;  sit\_nums\_k1 = 10;  sit\_nums\_p1 = 17;  sit\_nums\_b2 = 10;  sit\_nums\_k2 = 20;  sit\_nums\_p2 = 30;  sit\_nums\_b3 = 20;  sit\_nums\_k3 = 23;  sit\_nums\_p3 = 21;  }  Places::~Places()  {  }  void Places::show\_places(int x)  {  if (x == 0)  {    cout << b << " |" << sit\_nums\_b1 << endl;  cout << "-------------------" << endl;  cout << p << " |" << sit\_nums\_p1 << endl;  cout << "-------------------" << endl;  cout << k << " |" << sit\_nums\_k1 << endl;  cout << "-------------------" << endl;  }  if (x == 1)  {    cout << b << " |" << sit\_nums\_b2 << endl;  cout << "-------------------" << endl;  cout << p << " |" << sit\_nums\_p2 << endl;  cout << "-------------------" << endl;  cout << k << " |" << sit\_nums\_k2 << endl;  cout << "-------------------" << endl;  }  if(x==2)  {    cout << b << " |" << sit\_nums\_b3 << endl;  cout << "-------------------" << endl;  cout << p << " |" << sit\_nums\_p3 << endl;  cout << "-------------------" << endl;  cout << k << " |" << sit\_nums\_k3 << endl;  cout << "-------------------" << endl;  }    }    void Places::change\_sit\_nums(int x)  {  int new\_sit\_nums\_b, new\_sit\_nums\_p, new\_sit\_nums\_k;    if (x == 0)  {  show\_places(0);  cout << "------------------------------" << endl;  cout << "Enter new places in " << b << endl;  cin >> new\_sit\_nums\_b;  cout << "Enter new places " << p << endl;  cin >> new\_sit\_nums\_p;  cout << "Enter new places " << k << endl;  cin >> new\_sit\_nums\_k;  cout << "New places for spectacl Gamlet: " << b << new\_sit\_nums\_b << " " << p << new\_sit\_nums\_p << " " << k << new\_sit\_nums\_k << endl;  sit\_nums\_b1 = new\_sit\_nums\_b;  sit\_nums\_p1 = new\_sit\_nums\_p;  sit\_nums\_k1 = new\_sit\_nums\_k;    }  if (x == 1)  {  show\_places(1);  cout << "Enter new places in " << b << endl;  cin >> new\_sit\_nums\_b;  cout << "Enter new places " << p << endl;  cin >> new\_sit\_nums\_p;  cout << "Enter new places " << k << endl;  cin >> new\_sit\_nums\_k;  cout << "New places for spectacl Gamlet: " << b << new\_sit\_nums\_b << " " << p << new\_sit\_nums\_p << " " << k << new\_sit\_nums\_k << endl;  sit\_nums\_b2 = new\_sit\_nums\_b;  sit\_nums\_p2 = new\_sit\_nums\_p;  sit\_nums\_k2 = new\_sit\_nums\_k;  }  if (x == 2)  {  show\_places(2);  cout << "Enter new places in " << b << endl;  cin >> new\_sit\_nums\_b;  cout << "Enter new places " << p << endl;  cin >> new\_sit\_nums\_p;  cout << "Enter new places " << k << endl;  cin >> new\_sit\_nums\_k;  cout << "New places for spectacl Romeo and Juliet: " << b << new\_sit\_nums\_b << " " << p << new\_sit\_nums\_p << " " << k << new\_sit\_nums\_k << endl;  sit\_nums\_b3 = new\_sit\_nums\_b;  sit\_nums\_p3 = new\_sit\_nums\_p;  sit\_nums\_k3 = new\_sit\_nums\_k;  }  }  //Spectacl.h  #pragma once  #include <iostream>  #include <string>  #include <fstream>  #include "Theater.h"  #include <vector>  #include <Windows.h>  #include <iomanip>  using namespace std;  class Spectacl :  public Theater    {  protected:  string path\_spectacl;  string path\_date;  string path\_actors;  fstream read;  fstream write;  public:  int ticketgamlet, ticketkaydash, ticketromeo;  Spectacl();  ~Spectacl();  vector <string> spectacl = { "Gamlet","Kaydashewa simya","Romeo and Juliet" };  vector <string> actors = { "Andriy Berejnuy" , "Mykola Jadan", "Vasyl Dilovyi" ,"Antonio Banderas","Vladyslav Svychuk","Pavlo Zyma" };    void add\_spectacl();  void delete\_spectacl(int x);  void show\_all\_spetacl();  bool show\_spectacl(int x);  bool check\_tickets(int x);  };  //Spectacl.cpp  #include "Spectacl.h"  Spectacl::Spectacl()  {  path\_spectacl = "Spectacl.txt";  path\_actors = "Actors.txt";  ticketgamlet = 0;  ticketkaydash = 0;  ticketromeo = 0;  }  Spectacl::~Spectacl()  {  }  void Spectacl::add\_spectacl()  {  string new\_spectcl;  read.open(path\_spectacl, fstream::out | fstream::app);  if (!read.is\_open())  {  cout << "Eror! File doesn`t open" << endl;  }  else  {  cout << " Enter spectacl what you want to add: " << endl;  cin >> new\_spectcl;  spectacl.push\_back(new\_spectcl);  read << endl;  read << spectacl[3];  cout << "Spectacl added successefully!" << endl;  getchar();  }  read.close();  }  void Spectacl::delete\_spectacl(int x)  {  string remainderspectacl;  read.open(path\_spectacl, fstream::out);  if (!read.is\_open())  {  cout << "Eror! File doesn`t open" << endl;  }  else  {  if (x == 0)  {  spectacl.erase(spectacl.begin() + 0);  for (int i = 0; i < spectacl.size(); i++)  {  read << endl;  read << spectacl[i];  }  cout << "Available spectacl: " << endl;  for (int i = 0; i < spectacl.size(); i++)  {  cout << spectacl[i] << endl;  }  read.close();  }  if (x == 1)  {  spectacl.erase(spectacl.begin() + 1);  for (int i = 0; i < spectacl.size(); i++)  {  read << endl;  read << spectacl[i];  }  cout << "Available spectacl: " << endl;  for (int i = 0; i < spectacl.size(); i++)  {  cout << spectacl[i] << endl;  }  read.close();  }  if (x == 2)  {  spectacl.erase(spectacl.begin() + 2);  for (int i = 0; i < spectacl.size(); i++)  {  read << endl;  read << spectacl[i];  }  cout << "Available spectacl: " << endl;  for (int i = 0; i < spectacl.size(); i++)  {  cout << spectacl[i] << endl;  }  read.close();  }  }  }  void Spectacl::show\_all\_spetacl() // robut  {  string line;  read.open(path\_spectacl, fstream::in);  while (!read.eof())  {  getline(read, line);  cout <<"|" <<line << endl;  cout << "|------------------|" << endl;  }  read.close();  }  bool Spectacl::show\_spectacl(int x)  {  if (x == 0)  {  if ((check\_tickets(0) == false))  {  return false;  }  else  cout << "The performance begins" << endl;  cout << "3" << endl;  Sleep(1000);  cout << "2" << endl;  Sleep(1000);  cout << "1" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Andriy Berejnuy: 'Be or not be this is the questions'" << endl;    Sleep(1000);  cout << "Ksenia Voronnaya: 'You turned your eyes to the soul with your pupils, and there are black spots everywhere, and nothing can wash them off!'" << endl;  cout << "------------------------------------------------------" << endl;  Sleep(1000);  cout << "Antrakt! " << endl;  cout << "Waiting ..." << endl;  Sleep(6000);  cout << "------------------------------------------------------" << endl;  cout<< "Andriy Berejnuy: 'Is it more honorable to suffer in your thoughts?'" << endl;  Sleep(1000);  cout << "Ksenia Voronnaya: 'To my sick soul, where sin lives, Everything seems to be the forerunner of evil adversity'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << " Actors go to bow " << endl;  cout << "Actors: 'Thank you for your attention'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Spectacl is over" << endl;  return true;  }  if (x == 1)  {  if ((check\_tickets(1) == false))  {  return false;  }  else  cout << "The performance begins" << endl;  cout << "3" << endl;  Sleep(1000);  cout << "2" << endl;  Sleep(1000);  cout << "1" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Vasyl Dilovyi: 'Drunk to talk - just to waste time'" << endl;  Sleep(1000);  cout << "Antonio Banderas: 'I love that the girl was a bit spittoo to have a heart with pepper'" << endl;  cout << "------------------------------------------------------" << endl;  Sleep(1000);  cout << "Antrakt! " << endl;  cout << "Waiting ..." << endl;  Sleep(6000);  cout << "------------------------------------------------------" << endl;  cout << "Vasyl Dilovyi: 'Take Chemku. This is like a brykne, and you will go over!'" << endl;  Sleep(1000);  cout << "Antonio Banderas: 'They do not say well enough: the girl is like a willow: where will the post, it will be accepted'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << " Actors go to bow " << endl;  cout << "Actors: 'Thank you for your attention'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Spectacl is over" << endl;  return true;  }  if (x == 2)  {  if ((check\_tickets(2) == false))  {  return false;  }  else  cout << "The performance begins" << endl;  cout << "3" << endl;  Sleep(1000);  cout << "2" << endl;  Sleep(1000);  cout << "1" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Vladyslav Svychuk: 'Plague plow on both of your homes I became food for worms because of you ...'" << endl;  Sleep(1000);  cout << "Pavlo Zyma: 'In moments of Despair come down for an eternity hour ...'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Antrakt! " << endl;  cout << "Waiting ..." << endl;  Sleep(6000);  cout << "------------------------------------------------------" << endl;  cout << "Vladyslav Svychuk: 'But what a shine I see on the balcony??'" << endl;  Sleep(1000);  cout << "Pavlo Zyma: 'In my discouragement, the book is the same food'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "'Actors go to bow '" << endl;  cout << "Actors: 'Thank you for your attention'" << endl;  Sleep(1000);  cout << "------------------------------------------------------" << endl;  cout << "Spectacl is over" << endl;  return true;  }  }  bool Spectacl::check\_tickets(int x)  {  if (x == 0)  {  if (ticketgamlet == 0)  {  cout << "Sorry but you haven`t tickets for this spectacl" << endl;  return false;  }  else  ticketgamlet --;  cout << "The amount of your tickets " << ticketgamlet << endl;  }  if (x == 1)  {  if (ticketkaydash == 0)  {  cout << "Sorry but you haven`t tickets for this spectacl" << endl;  return false;  }  else  ticketkaydash--;  cout << "The amount of your tickets " << ticketkaydash << endl;  }  if (x == 2)  {  if (ticketromeo == 0)  {  cout << "Sorry but you haven`t tickets for this spectacl" << endl;  return false;  }  else  ticketromeo--;  cout << "The amount of your tickets " << ticketromeo << endl;  }  }    //Theater.h  #pragma once  #include <iostream>  #include <string>  #include <fstream>  class Theater  {  public:  Theater();  ~Theater();  };  // Theater.cpp  #include "Theater.h"  Theater::Theater()  {  }  Theater::~Theater()  {  }  //Tickets.h  #pragma once  #include <iostream>  #include <string>  #include "Spectacl.h"  using namespace std;  class Tickets :  public Spectacl  {  public:  int priceGamlet,priceKaydash,priceRomeo;  Tickets();  ~Tickets();  void change\_price(int x);    };  //Tickets.cpp  #include "Tickets.h"  Tickets::Tickets():Spectacl()  {  priceGamlet = 40;  priceKaydash = 30;  priceRomeo = 50;    }  Tickets::~Tickets()  {  }  void Tickets::change\_price(int x)  {  int new\_price;  if (x == 0)  {  cout << priceGamlet << endl;  cout << "---------------------------------" << endl;  cout << "Enter new price: " << endl;  cin >> new\_price;  cout << "Spectacl - Gamlet: " << new\_price << endl;  priceGamlet = new\_price;  }  if (x == 1)  {  cout <<priceKaydash << endl;  cout << "---------------------------------" << endl;  cout << "Enter new price: " << endl;  cin >> new\_price;  cout << "New price: " << new\_price << endl;  cout << "Spectacl - Kaydashewa simya: " << new\_price << endl;  priceKaydash = new\_price;  }  if (x == 2)  {  cout <<priceRomeo << endl;  cout << "---------------------------------" << endl;  cout << "Enter new price: " << endl;  cin >> new\_price;  cout << "New price: " << new\_price << endl;  cout << "Spectacl - Romeo and Juliet: " << new\_price << endl;  priceRomeo = new\_price;  }  }  //Visitor.h  #pragma once  #include "Tickets.h"  #include <Windows.h>  #include <iostream>  using namespace std;  class Visitor :  public Tickets  {  public:  int pass;  int money;  Visitor();  ~Visitor();  bool buy\_tickets(int x);  bool enterpass(int x);  };  //Visitor.cpp  #include "Visitor.h"  Visitor::Visitor():Tickets()  {  money = 100;    }  Visitor::~Visitor()  {  }  bool Visitor::buy\_tickets(int x)  {  if (x == 0)  {  if (money < priceGamlet)  {  cout << "You don`t have a money" << endl;  return false;  }  else  cout << "You buy a ticket for spectacl Gamlet" << endl;  money -= priceGamlet;  cout << "Your remainder " << money << endl;  ticketgamlet++;    return true;  }  if (x == 1)  {  if (money < priceKaydash)  {  cout << "You don`t have a money" << endl;  return false;  }  else  cout << "You buy a ticket for spectacl Kaydashewa simya" << endl;  money -= priceKaydash;  cout << "Your remainder " << money << endl;  ticketkaydash++;    return true;    }  if (x == 2)  {  if (money < priceRomeo)  {  cout << "You don`t have a money" << endl;  return false;  }  else  cout << "You buy a ticket for spectacl Romeo and Juliet" << endl;  money -= priceRomeo;  cout << "Your remainder " << money << endl;  ticketromeo++;    return true;  }  }  bool Visitor::enterpass(int x)  {  if (x == 0)  {  cout << "Enter pass: " << endl;  cin >> pass;  if (pass == 123)  {  return true;  }  else  return false;  }  if (x == 1)  {  cout << "Enter pass: " << endl;  cin >> pass;  if (pass == 123)  {  return true;  }  else  return false;  }  } |