**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

**Кафедра ЕОМ**



**Курсова робота**

з предмету: "Програмування, частина 2 (Об'єктно-орієнтовне програмування)

на тему "Базові принципи об'єктно-орієнтованого програмування"

Індивідуальне завдання: "Реалізувати модель роботи телестудії"

Підготував:

студент КІ-17 Зубрицький В.В.

Прийнла:

асистент Шологон Ю.З.

**ЛЬВІВ 2017**

**АНОТАЦІЯ**

Дана курсова робота була підготовлена задля демонстрації знань та навичок, які були отримані під час проходження курсу "Програмування, частина 2 (Об'єктно-орієнтовне програмування)".

У ній мушу виконати завдання згідно з індивідуальним варіантом, що видається викладачем практичних занять. Завдання повинно бути реалізовано за допомогою концепції об’єктно-орієнтованого програмування.

Під час викоанння завдання треба використовувати основні концепції ООП, з якими ознайомився на лекція, практичних заняттях та лабораторних роботах з предмету "Програмування, частина 2 (Об'єктно-орієнтовне програмування)".

**ЗМІСТ**

[**АНОТАЦІЯ** 2](#_Toc481670599)

[**ЗАВДАННЯ НА КУРСОВУ РОБОТУ** 4](#_Toc481670600)

[**ВСТУП** 5](#_Toc481670601)

[**1.ОГЛЯД ТА ОБҐРУНТОВУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ОБЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ** 6](#_Toc481670602)

[**2.АНАЛІЗ ТА РОЗРОБКА АЛГОРИТМУ ЗГІДНО ІНДИВИДУАЛЬНОГО ЗАВДАННЯ** 8](#_Toc481670603)

[**3.ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ ЗГІДНО ІНДИВИДУАЛЬНОГО ЗАВДАННЯ** 1](#_Toc481670604)1

[**Клас**](#_Toc481670606) **Customer** [10](#_Toc481670606)

[**Клас**](#_Toc481670607) **Order** [11](#_Toc481670607)

[**Клас**](#_Toc481670608) **Agreement** [1](#_Toc481670608)3

[**Клас**](#_Toc481670609) **Operator** [1](#_Toc481670609)3

[**Клас**](#_Toc481670609)MakeSomeChanges………………………………………………………………13

[**Клас**](#_Toc481670609) ShowOneLine…………………………………………………………………..….13

[**4.**ЮНІТ ТЕСТУВАННЯ **1**](#_Toc481670612)**5**

**5.** Висновки…………………….……………………………………………………18

**6.Список використаної літератури………………………………………………….19**

**7.Додаток…..……………………………………………………………………………20**

# **ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

Згідно з індивідуальним завдання до курсової роботи реалізовую модель телестудії (10 варіант у журнальному списку). Є оператор і замовник телепрограм які залежать один від одного.

# **ВСТУП**

Мова програмування «С» була розроблена в 1972 р. Деннісом Рiтчi, який є одним з авторів операцiйної системи UNIX. Мова програмування «С» пізніше була використана для програмування цiєi системи, а також багатьох бібліотек обслуговуючих програм, оскільки, являлася універсальною мовою загального призначення. Мова «С» зручна для програмування системних задач. Поява мікрокомп’ютерів закріпила позицiї мови «С». Було створено близько 30 його нових компіляторів, а після проведення Американським національним інститутом стандартів (American National Standards Institute) робiт по стандартизацii в області програмування почали розроблятися компілятори, що вiдповiдали б опублікованому весною 1986 р. проекту стандарту.

Основною перевагою мови «С» вважатьєся висока передача написаних на нiй програм мiж комп'ютерами з різною архітектурою та мiж різними операційними середовищами. Транслятори мови «С» дiють практично на всiх персональних комп'ютерах, які використовуються в наш час. Перечислимо окремі особливості мови «С»:

-В мові С реалізовані окремі операцiї низького рівня (а саме операцiї над бітами). Деякі з таких операцій напряму вiдповiдають машинним командам;

- Мова «С» підтримує механізм вказівників на змiннi i функцii;

- Базовi типи даних мови «С» відображають тi ж об’єкти, з якими потрібно мати справу програмі на мові Асемблера,- байти, машинні слова, символи, строки;

- В мові «С» накладаються лише незначні обмеження на перетворення типів даних, порівняно з іншими мовами програмування високого рівня;

Оскільки програма, яку треба реалізувати, має складну структуру, то в цьому випадку доцільно використовувати об'єктно-орієнтоване програмування (у подальшому ООП). ООП - це одна з парадигм програмування, що розглядає програму як множину, яка складається з об'єктів. Об'єкти мають певні властивості (поля), можуть взаємодіяти один з одним та виконувати певні дії (методи). По суті кожен об'єкт - незалежний автомат, що має певне призначення.

Мета курсової роботи: засвоїти базові принципи ООП, навчитися використовувати ООП для вирішення конкретних задач, набути практичних навичок проектування.

# **1.ОГЛЯД ТА ОБҐРУНТОВУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ОБЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ**

Об'єктно-орiєнтоване програмування(ООП) - результат природної еволюції більш ранніх методологій програмування. Воно виникло з процедурного програмування. В ООП ви відходите від ділення задач на під задачі. Ви намагаєтесь побачити вашу задачу, утворену з взаємодій між абстракціями – ідеалізованими об'єктами реального світу. Таким чином, ціль у тому, щоб використовувати в програмуванні повсякденний досвід і знання про поведінку звичайних об'єктів. Об’єкт – це абстракція, але з чітко визначеними властивостями чи ролями. Об'єктно-орiєнтоване програмування є методом програмування, який імітує виконання людиною якої-небудь роботи. Воно більш структуроване i більш модульне i абстрактне, ніж традиційне програмування.

За винятком другорядних деталей «C++» є надмножиною мови програмування «C». Крім можливостей, що дає «C», «C++» надає гнучкі й ефективні засоби визначення нових типів. Використовуючи визначення нових типів, програміст може розділяти розроблювальну програму на частини, що піддаються легкому контролю. Такий метод побудови програм часто називають абстракцією даних. Інформація про типи утримується в деяких об'єктах типів, визначених користувачем. Такі об'єкти прості й надійні у використанні в тих ситуаціях, коли їхній тип не можна встановити на стадії компіляції. Програмування з застосуванням таких об'єктів часто називають об'єктно-орієнтованим. При правильному використанні цей метод дає більш короткі й зрозуміліші контрольовані програми.

Об’єктно-орієнтовне програмування – методологія, яка концентрується більше на зв'язках між об'єктами, ніж на деталях реалізації. Дані зв'язку зазвичай розвиваються по принципу дерев, при цьому нові типи об’єктів утворюються із вже існуючих.

В «С++» деякі об’єкти є частиною самої мови. Але в цілому задача створення нових об’єктів покладається на користувача. В «Borland C++» є велика кількість типів об’єктів, але реальне використання мови потребує утворення нових типів. Ефективність ООП проявляється тільки при утворенні і застосуванні груп зв'язаних між собою об'єктів. Такі групи зазвичай називають ієрархіями класів. Розвиток цих ієрархій класів є основою в діяльності ООП.

«C++» і його стандартні бібліотеки спроектовані так, щоб забезпечувати мобільність. Наявна на сучасний момент реалізація мови буде йти в більшості систем, що підтримують «C». З «C++» програм можна використовувати «C» бібліотеки, і з «C++» можна використовувати велику частину інструментальних засобів, що підтримують програмування на «C».

Всі мови об’єктно орієнтованого програмування, включаючи «С++», основані на трьох основних концепціях, які називаються: інкапсуляція, поліморфізм і наслідування.

ООП полягає в наступному наборі основних принципів:

* Все є об'єктами.
* Всі дії та розрахунки виконуються шляхом взаємодії (обміну даними) між об'єктами, при якій один об'єкт потребує, щоб інший об'єкт виконав деяку дію. Об'єкти взаємодіють, надсилаючи і отримуючи повідомлення. Повідомлення — це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії.
* Кожен об'єкт має незалежну пам'ять, яка складається з інших об'єктів.
* Кожен об'єкт є представником (екземпляром, примірником) класу, який виражає загальні властивості об'єктів.
* У класі задається поведінка (функціональність) об'єкта. Таким чином усі об'єкти, які є екземплярами одного класу, можуть виконувати одні й ті ж самі дії.
* Класи організовані у єдину деревоподібну структуру з загальним корінням, яка називається ієрархією успадкування. Пам'ять та поведінка, зв'язані з екземплярами деякого класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Для виконання поставленої задачі треба застосувати принцип ООП. Ниже наведено аргументи на користь використання саме цієї концепції програмування.

Модель телестудії доволі складна. У програмі потрібно реалізувати різні можливості оператора і замовника, зробити так, щоб оператор не мав доступу до грошей замовника, а клієнт не має доступу до деяких методів оператора. Щоб реалізувати все це доведеться писати багато коду. Тому розробляти модель телестудії у вигляді послідовно виконуваних інструкцій, застосовуючи процедурне програмування, неефективно. Для реалізації мого завдання цілком доцільно використати ООП та розподілити весь код на окремі класи, кожен з яких буде виконувати певну дію. Це допоможе абстрагуватись від реалізації та зосередитись на вирішені задачі. Застосування ООП-парадигми зробить код більш наглядним та зрозумілим.

Якщо розглядати програму як сукупність об'єктів, які здатні взаємодіяти один з одним, то таким чином можна легко спроектувати навіть найскладнішу систему. В телестудії обов'язково повинен бути оператор який контролює процесом виходу сюжети в прямий ефір. Окрім цього повинен бути клієнт , який може замовляти телесюжет, змінювати його при потребі. Звідси випливає, що нам потрібно два класи - Customer та Operator. Клас Customer має поле Adjustments , за допомогою якого може змінювати дані(збільшити/зменшити час в ефірі, поміняти шов) і Write CustomerInfo для запису клієнта в базу даних. Клас Operator, завдяки поліморфізму має доступ до деякої інформації про користувача. Завдяки такій простій абстракції ми змогли змоделювати телестудію.

ООП включає в себе цілій ряд концепцій, за допомогою яких можна повторно використовувати код. Наслідування дозволяє створювати нові класи на основі вже існуючих. При цьому похідних клас (син) має доступ до всіх властивостей базового класу (батько). Таким чином, якщо треба доповнити вже існуючий і відлагоджений клас новими властивостями або методами, можно застосувати принцип наслідування.[1]

Знаходити помилки у об'єктно-орієнтованому коді значно легше. Це пов'язана з тим, що програма поділена на окремі частини. Якщо виникла помилка або програма працює не так, як треба, то не треба переглядати увесь код. Достатньо відлагодити лише ту частину програми, під час виконання якої виникли проблеми. Окрім цього від помилок уберігає один з принципів ООП - інкапсуляція. Інкапсуляція запобігає некоректному доступу до прихованих даних.

Отже, з вище написаного можна зробити висновок, що для реалізації моделі більярдного клубу найбільше підходить парадигма ООП.

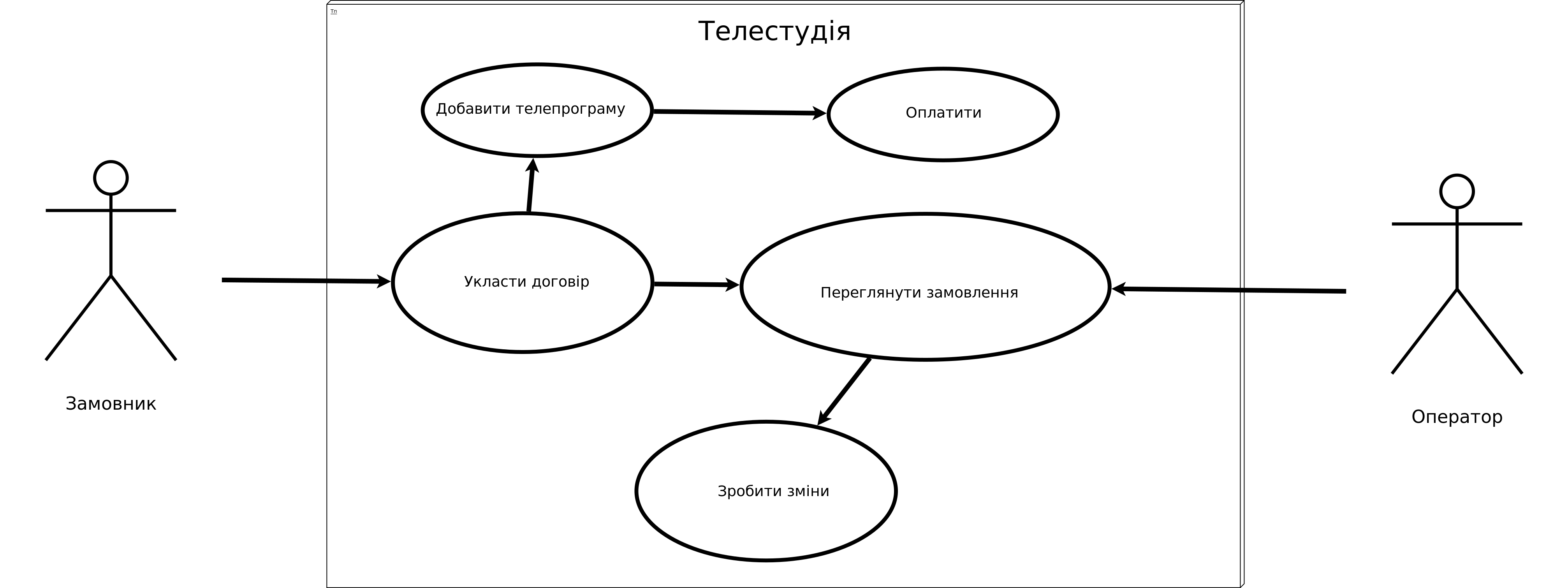
Для написання курсової роботи буду використовувати мову C++ та фреймворк Qt. В якості інструментарію буде використано бібліотека для роботи з файлами та для юніт тестів.

C++ (Сі-плюс-плюс) — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом 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 C.

Qt (варіант вимови від розробників *cute* — *к'ют*) — крос-платформовий інструментарій розробки програмного забезпечення (ПЗ) мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем (ОС), просто компілюючи текст програми для кожної операційної системи без зміни сирцевого коду.

***2. Аналіз та розробка алгоритму згідно індивідуального завдання***

*2.1. Опис алгоритму та оцінка його складності.*



*Рис.2.1. Діаграма прецедентів програми*

Актори:

Покупець

* Укласти договір;
* Добавити телепрограму;
* Виведення на екран своїх замовлень;
* Має можливість змінити деякі параметри.

Оператор

* Може зробити зміни в замовленні;
* Вивести на екран замовлення;
* Обчислює загальну вартість ефірного часу;
* Додати/видалити продукт;

Як видно на діаграмі, оператор та покупець мають кілька спільних можливостей це:

1. Переглядати список замовлень;
2. Зробити поправки;

*2.2 Аналіз особливостей алгоритму.*

Розглянемо детальніше кожну функцію нашої діаграми.

Customer

1. *Укласти договір* – перед замовленням програм потрібно укласти договір в якому буде (ім’я , прізвище , ціна яку заплатив клієнт).
2. *Добавити телепередачу*  –– замовник може добавляти телепрограми. .
3. *Зробити зміни* – має можливість відкорегувати телепрограми.

Operator

1. *Зробити зміни* – оператор теж може зробити зміни в замовленні.
2. *Переглянути всі замовлення* – вивести на екран замовлення.

*2.3 Опис алгоритму у псевдокоді*

У цьому підрозділі я б хотів звернути увагу на те , які саме дані повинні оброблятись, та як вони мають бути упорядковані у текстових документах формату \*.txt .

У моїй програмі використовуватимуться 3 необхідних для повноцінного функціонування програми файлів. Назви цих файлів та короткі дані, вміщені в них, вказані нижче.

1. Orderlist.txt (Ім’я, прізвище, кількість грошей клієнта)
2. Orders.txt (Список замовлених телепрограм)
3. Category.txt (Прайс лист для окремих категорій)

# **3.ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ ЗГІДНО ІНДИВИДУАЛЬНОГО ЗАВДАННЯ**

## 

## **Рис. 3.1**

## Ієрархія класів в програмі організована у вигляді поліморфізму.

## **Клас Сustomer**

СTables - це клас, що представляє собою модель клієнта . Цей клас працює з файлом **agreement.txt** та має наступні протектед поля:

* (\*money) — гроші які має клієнт.
* Name ім’ я замовника.
* Surname — прізвище.

Метод WriteUserInfo(FilenameOrd) потрібен для того, щоб працювати з файлом Agreement.txt. За допомогою цього методу здійснюється запис інформації до бази даних. До файлу записується ім'я клієнта, прізвище, його поточний баланс.

## **Клас Order**

Order потрібен для того, щоб вирахувати ціну за ефірний час. Цей клас працює з файлом Order.txt та має наступні приватні поля:

* ID — кожному замовлені присвоюється номер.
* Price — ціна яка вираховується за формулою.
* Deadline — дата, до якої телепередача буде в ефірі.
* Target — опис телепередачі.
* NameTvProgram — назва телевізійної програми.

Для кожного поля реалізовані get- та set-методи, щоб не порушувати принцип інкапсуляції.

Метод СalculateTotal() потрібен для того, щоб обрахувати загальну вартість замовлення і занести це значення до змінної Price (міститься у класі Order і Operator). Значення Price обраховується згідно з формулою 1.

Price = (ціна для кожної категорії) \* 1000 (1)

Ціна для кожної категорії знаходиться в Category.txt файлі.

Метод ReadList() потрібен для того, щоб працювати з файлом Category.txt. За допомогою цього методу здійснюється читаня інформації з бази даних. До файлу записується ім'я клієнта, кількість замовленого пива, вина, шотів, шампанського та сумарна вартість замовлення.

## **Клас Agreement**

Клас **Agreement** призначений для того, щоб скласти договір. Цей клас працює з файлом agreement.txt та має наступні протектед поля:

* Deadline
* nameTvProgram
* ID
* Agreements

Метод writeAgreements() потрібен для того, щоб працювати з файлом Agreement.txt. За допомогою цього методу здійснюється запис інформації до бази даних. До файлу записується ім'я, прізвище клієнта, ціна яку він заплатив.

## **Клас Operator**

Метод makeAnAmend взаємодіє з класом Order та змінює поле .

Таким чином може змінюватись ціна замовлення.

Метод ShowInfo дозволяє отримати інформацію стосовно всіх клієнтів.

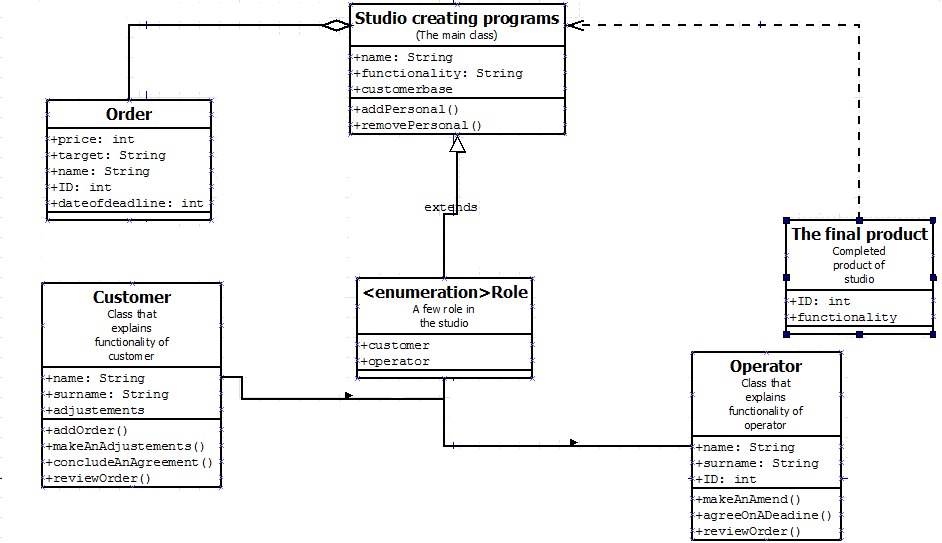
**Клас MakeSomeChanges**

МетодAdjustments взаємодіє з класом Customer.

За допомогою нього можна міняти дані в текстовому файлі.

**Клас ShowOneLine**

Методи LastLineOrderlist і intLastLineOrdList за допомогою бінарного читання файла виводить останні рядки з Orderlist.txt і Orders.txt відповідно.

****

## **Рис. 3.2(UML діаграма)**

## На рисунку 3.2 зображена UML яка в стиснутому вигляди демонструє те, що було описано вище.

# **4.ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ РЕАЛІЗОВАНОЇ ПРОГРАМИ ЗГІДНО ІНДИВІДУАЛЬНОГО ЗАВДАННЯ**

Відлагодження програми відбувалось за допомогою вбудованого в Qt Creator зневаджувача. Зневаджувач (також — зневадник, налагоджувач, програма зневадження, зустр. англіцизм: деба́ґґер) — комп'ютерна програма, яка використовується для тестування і виправлення вад інших програм. Як варіант, код для розгляду може бути запущено на емуляторі інструкцій, що дозволяє більший контроль зупинки процесу при заданих умовах, але, як правило, тоді виконання коду відбувається набагато повільніше, ніж якщо це робиться напряму на процесорі.

Під час написання курсової роботи виникли помилки різного характеру.

Синтаксичні помилки, викликані неправильним записом операторів мови програмування, виявляються на етапі трансляції програми. При цьому виводиться повідомлення, яке вказує оператор, в якому допущена помилка, місце в операторі, де виявлена помилка, і текст, що пояснює причину помилки. Синтаксичні помилки виникають у разі порушення правил запису мовних конструкцій.

За допомогою вбудованого зневаджувача були виправлені синтаксичні помилки. При наявності синтаксичної помилки зневаджувач повідомляв про це, підкреслюючи проблемну частину коду.

Логічні помилки - це помилки в логіці роботи програми. Програмний код виглядає абсолютно правильним, але працює неправильно. Такі помилки виявити і виправити набагато складніше.

Помилки логічного типу були виправлені за допомогою статичного аналізу та покрокового тестування.

Для того щоб впевнитись в тому, що програма працює коректно, було проведено модульне тестування. Модульне тестування - це один з методів тестування програмного забезпечення. Його суть полягає в окремому тестуванні модулів програм. Модулем називають найменшу частинку програми, що може бути протестована. Оскільки використано ООП-парадигму, то в такому випаду найменший модуль програми - метод класу.

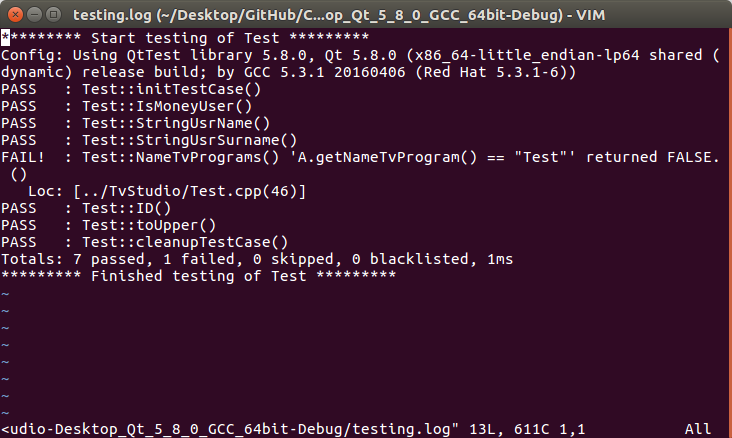
Для тестування створено окремий клас, який називається Test. В ньому тестуються окремі модулі програми за допомогою засобів Qt Creator. Для тестування задіюється макрос QCOMPARE. Він приймає два аргументи: результат виконання функції та очікуване значення. Якщо значення першого аргументу співпадає з другим, то в такому випадку метод працює коректно. У іншому випадку метод потребує доопрацювання, що буде зазначено у результатах модульного тестування. Результати тестування записуються до файлу testing.log.

**Результати тестування програми подаю у вигляді скріншота 4.1.**

****

*Мал. 4.1 Результати тестування програми у*

***вигляді файла***



*рис. 4.1 Результати тестування програми у*

***вигляді файла***

*Методи, перевірені за допомогою модульного тестування, працюють коректно. Сам процес тестування зайняв 1 мілісекунду.*

У випадку, якщо метод працює невірно і очікуваний результат не збігається з реальним, програма повідомить про це.

*У файлі testing.log вказано метод, у якому відбулась помилка. Знаючи у якому методі відбулась помилка, її можно легко виправити. Це і є перевага модульного тестування.*

# ***Висновки:***

# У ході виконання курсової роботи я на практиці застосував свої знання з об’єктно-орієнтованого програмування. Поглибив розуміння основних принципів об’єктно-орієнтованої ідеології програмування. Відпрацював на практиці основні методи та засоби об’єктно-орієнтованого програмування. Навчився розробляти ієрархію класів. Практично застосував такі поняття, як інкапсуляція, наслідування, перевантаження та перевизначення функцій. Набув практичних навиків роботи.

**Список використаної літератури**

1. Прата С. Язык программирования C++. Лекции и упражнения. 6-е издание / Стивен Прата; пер. с англ. – М.: ООО «И.Д. Вильямс», 2012. – 1248 с.: ил. – Парал. тит. Англ.

2. Полный справочник по C++, 4-е издание / Герберт Шилдт — М.: «Вильямс», 2011. - 800 с.

3. Тамре Л. «Введение в тестирование программного обеспечения ».- М.: Издательский дом «Вильямс», 2003.- С. 368.

# 4. Липаев В.В. «Обеспечение качества программных средств. Методы и стандарты». - М.: Синтег, 2001.- С. 246.

# 5. Макгрегор Дж., Сайкс Д. «Тестирование объектно-ориентированного программного обеспечения».- К: Диасофт, 2002.- С. 432.

## Ї

# **ДОДАТОК**

# **Agreement.h**

# #ifndef AGREEMENT\_H

#define AGREEMENT\_H

#include<iostream>

#include <string>

#include "Customer.h"

using namespace std;

class Agreement:virtual public Customer

{

protected:

int ID;

int price;

string nameTvProgram;

public:

Agreement()

{

money = 0; price = 0; ID = 1;

}

void setID(int idOrd)

{

ID = idOrd;

}

int getID()

{

return ID;

}

string getName()

{

return name;

}

string getSurname()

{

return surname;

}

void setPrices(int pr)

{

price = pr;

}

};

#endif

**Customer.h**

**#ifndef CUSTOMER\_H**

#define CUSTOMER\_H

#include<iostream>

#include <string>

using namespace std;

class Customer

{

protected:

int money;

string name;

string surname;

string nameTvProgram;

public:

void Adjustments();

void Agreements();

void WriteOrderList();

void WriteCustomerInfo();

void WriteUserInfo();

void setNameTvProgram(string ntvp)

{

nameTvProgram = ntvp;

}

string getNameTvProgram()

{

return nameTvProgram;

}

void setMoney(int f)

{

money = f;

}

int getMoney()

{

return money;

}

string getName()

{

return name;

}

void setName(string n)

{

name = n;

}

}; #endif

**MakeSomeChange.h**

**#include "Customer.h"**

class MakeSomeChanges : public Customer

{

Customer::Adjustments();

};

**Operator.h**

**#include<iostream>**

#include <string>

#include "Agreement.h"

#include "Order.h"

using namespace std;

class Operator : public Agreement,public Order

{

public:

int ID;

void PriceChange();

void ShowAllFileCateg();

void ShowAllFileOrd();

Operator()

{

money = 0;

ID = 0;

}

string getName()

{

return name;

}

string getSurname()

{

return surname;

}

void setID(int idOrd)

{

ID = idOrd;

}

};

**Order.h**

**#ifndef ORDER\_H**

#define ORDER\_H

#include "Customer.h"

#include<iostream>

#include <string>

using namespace std;

class Order : virtual public Customer

{

protected:

int ID;

int price;

string nameTvProgram;

public:

Order()

{

money = 0; price = 0; ID = 0;

}

void setID(int idOrd)

{

ID = idOrd;

}

string getName()

{

return name;

}

string getSurname()

{

return surname;

}

void setPrices(int pr)

{

price = pr;

}

int getPrices()

{

return price;

}

};

#endif

**ShowOneLine.h**

class ShowOneLine

{

public:

int LastLineOrderlist();

int intLastLineOrdList();

};

# **Test.h**

# **#ifndef TEST\_H**

#define TEST\_H

#include <QObject>

class Test : public QObject

{

Q\_OBJECT

public:

explicit Test(QObject \*parent = 0);

private slots:

void IsMoneyUser();

void StringUsrName();

void StringUsrSurname();

void NameTvPrograms();

void ID();

void toUpper();

};

#endif // TEST\_H

# **Adjustments.cpp**

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#include "Customer.h"

using namespace std;

void Customer::Adjustments(){

string x, y;

cout << "Enter string what you want to change: ";

cin >> x;

cout << "What do you want to burn instead: ";

cin >> y;

string search\_string = x;

string replace\_string = y;

string inbuf;

fstream input\_file("Orderlist.txt", ios::in);

ofstream output\_file("Orderlist1.txt");

while (!input\_file.eof())

{

getline(input\_file, inbuf);

int spot = inbuf.find(search\_string);

if(spot >= 0)

{

string tmpstring = inbuf.substr(0,spot);

tmpstring += replace\_string;

tmpstring += inbuf.substr(spot+search\_string.length(), inbuf.length());

inbuf = tmpstring;

}

output\_file << inbuf << endl;

}

}

# **Agreement.cpp**

#include <QCoreApplication>

#include <QIODevice>

#include <QFile> //Подключаем для работы с классом QFile

#include <QTextStream>

#include <QDebug>

#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include "Customer.h"

using namespace std;

void Read(QString Category)

{

QFile mFile(Category);

if(!mFile.*open*((QFile::ReadOnly | QFile::Text)))

{

qDebug() << "Could not open file for writing";

return;

}

QTextStream in(&mFile);

while (!in.atEnd())

{

QString mText = in.readLine();

qDebug() << mText;

}

mFile.*close*();

}

void Customer::Agreements(){

int categ;

int requestmoney;

do{

cout << "\nSelect a category:";

QString Category = "Category.txt";

Read(Category);

cin >> categ;

if (categ < 1 || categ > 7){

system("clear");

cout << "Entered not offered,try againe!\n";

}

}

while(categ < 1 || categ > 7);

string namePrograms;

ifstream Bank("Bank.txt");

string name;

int a;

while (Bank >> name) {

Bank >> a;

}

requestmoney = categ \* 1000;

if (requestmoney > a)

{

cout << "You don't have money!!!";

}

else

{

string line;

ifstream myfile ("Bank.txt");

if (myfile.is\_open())

{

while ( getline (myfile,line) )

{

cout << line << '\n';

}

myfile.close();

}

else cout << "Unable to open file";

int newBalanse = a - requestmoney;

ofstream Bank;

Bank.open("Bank.txt");

Bank << newBalanse;

Bank.close();

cout << "Enter name you program:";

cin >> namePrograms;

}

ofstream Orders;

Orders.open("Orders.txt", ios::app);

Orders << endl << "Number: " << categ << "\t" <<"Name:"<< namePrograms << endl;

Orders.close();

}

# **Main.cpp**

#include <QApplication>

#include <QTest>

#include <cstdlib>

#include <cstdio>

#include <iostream>

#include "Customer.h"

#include "Order.h"

#include "Agreement.h"

#include "Operator.h"

#include "ShowOneLine.h"

#include "Test.h"

using namespace std;

Order lists;

Operator change;

int main(int argc, char \*argv[])

{

cout <<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Welcome to the TV program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

int r;

while (true) {

do

{

cout << "\nSelect role:\n1.Order\n2.Operator\n3.Exit\n";

cin >> r;

switch (r) {

case 1:

/\*Function for order\*/

system ("clear");

lists.WriteOrderList();

break;

case 2:

/\*Function for operator\*/

system ("clear");

int c;

do{

cout <<"\nSelect action:\n1.Show category || pricelist.\n2.Change price\n3.Show all orders\n4.Exit\n";

cin >> c;

if(c == 1){

system ("clear");

change.ShowAllFileCateg();

}

if(c == 2){

change.PriceChange();

system ("clear");

}

if(c == 3){

system ("clear");

change.ShowAllFileOrd();

}

}

while(c != 4);

system ("clear");

break;

case 3:

break;

default:

system ("clear");

cout <<"Incorrect selection,try again!";

break;

}

}

while(r != 1 && r != 2 && r != 3);

if(r == 3)

break;

}

freopen("testing.log", "w", stdout);

QTest::qExec(new Test, argc, argv);

return 0;

}

# **PriceChange.cpp**

#include <iostream>

#include <fstream>

#include "Operator.h"

using namespace std;

void Operator::PriceChange(){

string x, y;

cout << "Enter price what you want to change: ";

cin >> x;

cout << "What do you want to burn instead: ";

cin >> y;

string search\_string = x;

string replace\_string = y;

string inbuf;

fstream input\_file("OrdersEtalon.txt", ios::in);

ofstream output\_file("Category.txt");

while (!input\_file.eof())

{

getline(input\_file, inbuf);

int spot = inbuf.find(search\_string);

if(spot >= 0)

{

string tmpstring = inbuf.substr(0,spot);

tmpstring += replace\_string;

tmpstring += inbuf.substr(spot+search\_string.length(), inbuf.length());

inbuf = tmpstring;

}

output\_file << inbuf << endl;

}

}

**ShowAllFile.cpp**

#include <QCoreApplication>

#include <QIODevice>

#include <QFile> //Подключаем для работы с классом QFile

#include <QTextStream>

#include <QDebug>

#include <fstream>

#include "Operator.h"

using namespace std;

void ReadCateg(QString Category)

{

QFile mFile(Category);

if(!mFile.*open*((QFile::ReadOnly | QFile::Text)))

{

qDebug() << "Could not open file for writing";

return;

}

QTextStream in(&mFile);

while (!in.atEnd())

{

QString mText = in.readLine();

qDebug() << mText;

}

mFile.*close*();

}

void Operator::ShowAllFileCateg(){

QString Category = "Category.txt";

ReadCateg(Category);

}

void Operator::ShowAllFileOrd(){

QString Category = "Orderlist.txt";

ReadCateg(Category);

}

**ShowOneLine.cpp**

#include "ShowOneLine.h"

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

const int N = 10;

int ShowOneLine::LastLineOrderlist(){

{

fstream fin("Orderlist.txt", ios::in | ios::ate | ios::binary); // открываем файл для бинарного чтения, курсор в конце файла

if (!fin.is\_open()) cout << "Error!" << endl;

else

{

int i;

i = fin.tellg(); // текущая позиция курсора

if (i == 0) cout << "File empty!\n"; // если файл пуст

else

{

char ch;

int flag = 0;

while (true) // находим начало последней строки

{

fin.seekg(-1, ios::cur); // курсор на одну позицию назад от текущей

i = fin.tellg(); // текущая позиция курсора

if (i == 0) break; // если дошли до начала файла, то строка одна

ch = fin.peek(); // читаем символ в позиции курсора

if (ch == N && flag != 0) // если символ новой строки

// и он не последний символ в файле

{

fin.seekg(1, ios::cur); // на начало последней строки

break;

}

flag = 1;

}

string str;

getline(fin, str); // читаем последнюю строку

cout <<" "<< str << endl;

}

}

cin.get();

return 0;

}

}

int ShowOneLine::intLastLineOrdList(){

fstream fin1("Orders.txt", ios::in | ios::ate | ios::binary); // открываем файл для бинарного чтения, курсор в конце файла

if (!fin1.is\_open()) cout << "Error!" << endl;

else

{

int i;

i = fin1.tellg(); // текущая позиция курсора

if (i == 0) cout << "File empty!\n"; // если файл пуст

else

{

char ch;

int flag = 0;

while (true) // находим начало последней строки

{

fin1.seekg(-1, ios::cur); // курсор на одну позицию назад от текущей

i = fin1.tellg(); // текущая позиция курсора

if (i == 0) break; // если дошли до начала файла, то строка одна

ch = fin1.peek(); // читаем символ в позиции курсора

if (ch == N && flag != 0) // если символ новой строки

// и он не последний символ в файле

{

fin1.seekg(1, ios::cur); // на начало последней строки

break;

}

flag = 1;

}

string str;

getline(fin1, str); // читаем последнюю строку

cout <<" "<< str << endl;

}

}

cin.get();

return 0;

}

**Test.cpp**

**#include <QFile>**

#include <QtTest/QTest>

#include "Test.h"

#include "Customer.h"

#include "Order.h"

#include "Operator.h"

#include <ctype.h>

using namespace std;

Test::Test(QObject \*parent) : QObject(parent)

{

}

void Test::IsMoneyUser()

{

Customer A;

A.setMoney(4);

QCOMPARE(A.getMoney(), 4);

}

void Test::ID()

{

Agreement A;

A.setID(4);

QCOMPARE(A.getID(), 4);

}

void Test::StringUsrName()

{

Customer A;

A.setName("Test");

QVERIFY(A.getName() == "Test");

}

void Test::StringUsrSurname()

{

Customer A;

A.setSurame("Test");

QVERIFY(A.getSurame() == "Test");

}

void Test::NameTvPrograms()

{

Customer A;

A.setNameTvProgram("4sdds");

QVERIFY(A.getNameTvProgram() == "Test");

}

void Test::toUpper()

{

QString str = "Hello";

QCOMPARE(str.toUpper(), QString("HELLO"));

}

**WriteCustomerInfo.cpp**

**#include <iostream>**

#include <fstream>

#include <string>

#include <stdlib.h>

#include "Customer.h"

using namespace std;

Customer infoUsr;

void Customer::WriteCustomerInfo()

{

srand(time(0));

int ID = rand()%101;

string myname;

string mysurname;

int balance;

cout << "\nYou name: ";

cin >> myname;

infoUsr.setName(myname);

cout << "\nYou surname: ";

cin >> mysurname;

infoUsr.setSurame(mysurname);

cout << "\nYour Balance($): " ;

cin >> balance;

infoUsr.setMoney(balance);

cout << endl;

ofstream OrderList;

OrderList.open("Orderlist.txt", ios::app);

//OrderList.open("Orderlist1.txt", ios::app);

OrderList << endl << ID << " " << infoUsr.getName() << " " << infoUsr.getSurame()

<< " " << infoUsr.getMoney();

OrderList.close();

ofstream OrderList1;

OrderList1.open("Orderlist1.txt", ios::app);

OrderList1 << endl << ID << " " << infoUsr.getName() << " " << infoUsr.getSurame()

<< " " << infoUsr.getMoney();

OrderList1.close();

ofstream Bank;

Bank.open("Bank.txt");

Bank << infoUsr.getSurame() << " " << infoUsr.getMoney();

Bank.close();

}

**WriteOrderList.cpp**

#include "Customer.h"

#include "ShowOneLine.h"

#include <iostream>

using namespace std;

ShowOneLine line;

Customer agreementwrt;

void Customer::WriteOrderList(){

int ord;

do

{

system ("clear");

cout << "\nSelect action:\n1.Add information about me.\n2.Make agreements.\n"

"3.Do some adjustments.\n4.Exit\n";

cin >> ord;

switch (ord) {

case 1:

system ("clear");

agreementwrt.WriteCustomerInfo();

break;

case 2:

system ("clear");

cout << "\nBy print \'y\', you agree to TvStudio User Agreement, Provacy Policy.\n";

char y;

cin >> y;

if(y == 'y')

{

/\*Agriment function\*/

agreementwrt.Agreements();

}

break;

case 3:

cout << "\nYou information:" ;

line.LastLineOrderlist();

cout << endl << " \t";

line.intLastLineOrdList();

agreementwrt.Adjustments();

break;

case 4:

break;

}