МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

Варіант №7

|  |  |  |
| --- | --- | --- |
| **Керівник :**  доц. каф. БМК,  к.т.н. Алхімова С.М. |  | **Виконав:**  студент гр. БС-03, ФБМІ  Затуловський Г. А.  залікова книжка № БС-0307 |
| Допущено до захисту |  |
| І\_\_\_І \_\_\_\_\_\_\_\_\_ 2022 \_\_\_\_\_\_\_\_\_\_\_  підпис |  |
| Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_  оцінка підпис  І\_\_\_І \_\_\_\_\_\_\_\_\_ 2022 |  |

Київ-2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Факультет | ***біомедичної інженерії*** | | | | |
|  | (назва факультету, інституту) | | | | |
| Кафедра | ***біомедичної кібернетики*** | | | | |
|  | (назва кафедри) | | | | |
| Дисципліна | ***«Об’єктно-орієнтоване програмування»*** | | | | |
|  | (назва) | | | | |
| Курс | ***2*** | Група | ***БС-03*** | Семестр | ***4*** |

**ЗАВДАННЯ**

**на курсовий проект (роботу) студента**

|  |
| --- |
| ***Затуловського Георгія Андрійовича*** |
| (прізвище, ім’я, по батькові) |

.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. Тема роботи: | | ***Розробка програмного забезпечення з використанням об’єктно-*** | | | |
| ***орієнтованого підходу.*** | | | | | |
|  | | | | | |
|  | |  | | | |
| 2. Строк здачі студентом закінченого проекту (роботи) | | | | | ***13.05.2022*** |
|  | |  | | | |
| 3. Вихідні дані до проекту (роботи): | | | | ***Варіант №7*** | |
|  | | | | | |
|  | | | | | |
|  | | | | | |
|  | |  | | | |
| 4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають | | | | | |
| розробці): | ***1. Визначення класів та програмування меню користувача (ієрархія*** | | | | |
| ***класів програми: абстрактний клас – Задача сервісного центру, класи-нащадки –*** | | | | | |
| ***Гарантійний ремонт техніки, регламентне обслуговування техніки). 2.*** | | | | | |
| ***Створення об’єктів та використання контейнерів (тип контейнера – Черга з*** | | | | | |
| ***пріоритетом). 3. Організація роботи з даними через файл. 4. Пошук даних у*** | | | | | |
| ***контейнері (запит для пошуку – Підрахувати загальну кількість гарантійних*** | | | | | |
| ***ремонтів техніки заданої фірми).*** | | | | | |
|  | |  | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): | | | | | |
| ***діаграми класів, послідовності та об’єктів в нотації UML*** | | | | | |
| 6. Дата видачі завдання: | | | ***18.02.2022*** | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №п/п | Назва етапів курсового проекту (роботи) та питань, які мають бути розроблені відповідно до завдання | Термін виконання етапу | Позначки керівника про виконання завдань |
| 1. | Отримання завдання на курсову роботу | 18.02.2022 |  |
| 2. | Огляд технічної літератури за темою роботи | 25.02.2022 |  |
| 3. | Розробка першої частини курсової роботи (визначення класів та програмування меню користувача) | 11.03.2022 |  |
| 4. | Перший контроль за процесом виконання курсової роботи, консультація у викладача | 11.03.2022 |  |
| 5. | Розробка другої частини курсової роботи (створення об’єктів та використання контейнерів) | 25.03.2022 |  |
| 6. | Розробка третьої частини курсової роботи (робота з даними через файл) | 08.04.2022 |  |
| 7. | Другий контроль за процесом виконання курсової роботи, консультація у викладача | 08.04.2022 |  |
| 8. | Розробка четвертої частини курсової роботи (пошук даних у контейнері) | 22.04.2022 |  |
| 9. | Оформлення пояснювальної записки | 13.05.2022 |  |
| 10. | Захист курсової роботи | 26.05.2022 |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис)

**Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Алхімова Світлана Миколаївна

(підпис) (прізвище ім'я, по батькові)

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 р.**

**Анотація**

Затуловський Г.А. Розробка програмного забезпечення з використанням об’єктно-орієнтованого підходу.

Курсова робота з дисципліни «Об’єктно-орієнтоване програмування» присвячена питанню створення ієрархії класів, застосування спадкування та поліморфізму, розробки контейнера для зберігання даних множини об’єктів, які створюються користувачем. У курсовій роботі було виконано визначення класів (ієрархія класів програми: базовий клас – задача сервісного центру, класи-нащадки – Гарантійний ремонт техніки, регламентне обслуговування техніки), розроблено меню користувача, створено множину об’єктів та розроблено контейнер для її зберігання (тип контейнера – черга з пріоритетом), виконано серіалізацію даних елементів контейнера у файл та створення вмісту контейнера через десеріалізацію даних файлу, реалізовано пошук даних у контейнері (запит для пошуку – підрахувати загальну кількість гарантійних ремонтів техніки заданої фірми).

**Структура і обсяг роботи.** Курсова робота складається із вступу, трьох розділів, висновків, списку використаної літератури з 4 джерел і 2 додатків. Загальний обсяг курсової роботи становить 72 сторінок, основного тексту (без додатків) – 50 сторінок, ілюстрацій – 35, таблиць – 24.

**Annotation**

George Zatulovsky Software development using object oriented approach.

Coursework on the Object Oriented Programming course is devoted to the issue of creating a hierarchy of classes, applying inheritance and polymorphism, developing storage container for the collection of objects created by the user. In the coursework class definitions were performed (hierarchy of program classes: base class – the task of the service center, derived classes – warranty repair of equipment, routine maintenance of equipment), user menu was designed, set of objects was created and its storage container was developed (type of container – queue with priority), serialization of container data was done and container elements were created via deserialization of file data, data search in the container was performed (search query – сalculate the total number of warranty repairs of equipment of a given company).

**The structure and the amount of work.** Coursework consists of an introduction, three partitions, conclusions, list of used literature with 4 references, and 2 applications. The total volume of coursework is 72 pages, main text (without applications) – 50 pages, illustrations – 35, tables – 24.

**Аннотация**

Затуловський Г. А. Разработка программного обеспечения с использованием объектно-ориентированного подхода.

Курсовая работа по дисциплине «Объектно-ориентированное программирование» посвящена вопросу создания иерархии классов, применения наследования и полиморфизма, разработки контейнера для хранения данных коллекции объектов, которые создаются пользователем. В курсовой работе было выполнено определение классов (иерархия классов программы: базовый класс – задача сервисного центра, классы-потомки – гарантийный ремонт техники, регламентное обслуживание техники), разработано меню пользователя, создано множество объектов и разработан контейнер для их хранения (тип контейнера – очередь с приоритетом), выполнены сериализация данных элементов контейнера в файл и создание содержимого контейнера путем десериализации данных файла, реализован поиск данных в контейнере (запрос поиска – подсчитать общее количество гарантийных ремонтов техники заданной фирмы).

**Структура и объем работы.** Курсовая работа состоит из введения, трех разделов, выводов, списка использованной литературы из 4 источников и 2 приложения. Общий объем курсовой работы составляет 72 страниц, основного текста (без приложений) – 50 страниц, иллюстраций – 35, таблиц – 24.

**ЗВІТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7](#_Toc104376740)

[ВСТУП 8](#_Toc104376741)

[РОЗДІЛ 1](#_Toc104376742). [СПЕЦИФІКАЦІЯ ВИМОГ ПРОГРАМНОГО ЗАСТОСУНКУ 9](#_Toc104376743)

[1.1 Функціональні вимоги 9](#_Toc104376744)

[1.2 Нефункціональні вимоги 17](#_Toc104376747)

[РОЗДІЛ 2](#_Toc104376748). [СТРУКТУРА ТА ЛОГІКА РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ 18](#_Toc104376749)

[2.1 Логічна структура 18](#_Toc104376750)

[2.2. Фізична структура 27](#_Toc104376751)

[2.3 Логіка роботи програмного застосунку 31](#_Toc104376752)

[РОЗДІЛ 3](#_Toc104376753). [ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ 32](#_Toc104376754)

[3.1 Сценарій тестування програмного застосунку 32](#_Toc104376755)

[3.2 Тестові дані 38](#_Toc104376756)

[3.3 Поетапні результати роботи програмного застосунку 42](#_Toc104376757)

[3.4 Модульне тестування 53](#_Toc104376759)

[ДОДАТОК А](#_Toc104376760). [Код програмного продукту 56](#_Toc104376761)

[ДОДАТОК Б](#_Toc104376762). [Код модульних тестів 71](#_Toc104376763)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Перелік скорочень

Рис. – рисунок

UML – уніфікована мова моделювання

Табл. – таблиця

ОС – операційна система

Перелік термінів

Парадигма – певний набір концепцій або шаблонів

Об'єктно-орієнтоване програмування — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою

Поліморфізм - це об'єктно-орієнтоване поняття програмування, яке позначає здатність змінної, функції чи об'єкта приймати кілька форм. Мова, яка має поліморфізм, дозволяє розробникам програмувати в загальному, а не програмувати на конкретному.

 Успадкування (наслідування) — механізм утворення нових класів на основі використання вже існуючих. При цьому властивості та функціональність батьківського класу переходять до класу нащадка (дочірнього).

# ВСТУП

Об'єкто-орієнтоване програмування — це модель програмування яка базується на стверджені того, що програма це сукупність об’єктів які взаємодіють між собою. Кожен об’єкт в цій моделі є незалежним, і він здатний отримувати, обробляти дані та відправляти ці дані іншим об’єктам. В ООП використано моделі успадкування, модульності, поліморфізму та інкапсуляції.

Завдання курсової робота з дисципліни «Об’єктно-орієнтоване програмування» полягає:

* + 1. Cтворити ієрархію класів
    2. Реалізувати об'єкт так, щоб він використовував незалежну пам'ять, яка складається з інших об'єктів
    3. Застосувати контейнера для зберігання об’єктів класів.
    4. Розробити меню користувача, через який буде відбуватись робота з програмним застосунком.
    5. Застосування спадкування та поліморфізму

Метою курсової роботи є застосування своїх знань, які ми отримали під час проходження курсу дисціпліни «Об'єктно-орієнтоване програмування» для створення програмного застосунку.

# РОЗДІЛ 1

# СПЕЦИФІКАЦІЯ ВИМОГ ПРОГРАМНОГО ЗАСТОСУНКУ

## Функціональні вимоги

Варіанти використання програмного застосунку характеризуються у таблицях 1.1 – 1.9 та рисунках 1.1 – 1.9

* + 1. Опис створення об’єкта типу «регламентоване обслуговування техніки» та записання його у контейнер (чергу з пріоритетом)

*Таблиця 1.1*

**Створення об’єкта типу «регламентоване обслуговування техніки»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 1 | Створення об’єкту «регламен-товане обслугову-вання техніки» | Створити об’єкт типу «регламентоване обслуговування техніки» та додати його в контейнер | Користувач | Основний | Програма створює об’єкт, користувач заповнює поля і об’єкт додається до контейнеру |

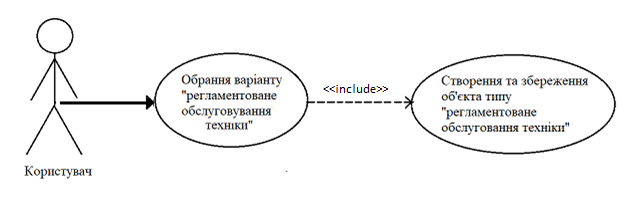


Рис. 1.1 - Створення об’єкту типу «регламентоване обслуговування техніки» та збереження в чергу з пріоритетом

* + 1. Опис створення об’єкта типу «гарантійний ремонт техніки» та записання його у контейнер (чергу з пріоритетом)

*Таблиця 1.2*

**Створення об’єкта типу «гарантійний ремонт техніки»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 2 | Створення об’єкту «гарантійний ремонт техніки» | Створити об’єкт типу «гарантійний ремонт техніки» та додати його в контейнер | Користувач | Основний | Програма створює об’єкт, користувач заповнює поля і об’єкт додається до контейнеру |

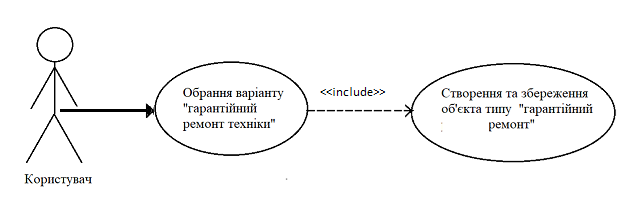


Рис. 1.2 - Створення об’єкту типу «гарантійний ремонт техніки» та збереження в чергу з пріоритетом

* + 1. Опис методу видалення першого об’єкта з контейнера (чергу з пріоритетом)

*Таблиця 1.3*

**Видалення об’єкта з контейнера**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 3 | Видалення об’єкта з черги з пріоритетом | Видалення об’єкта з черги з пріоритетом | Користувач | Основний | У процесі роботи програми надає можливість видалити перший об’єкт з контейнеру |

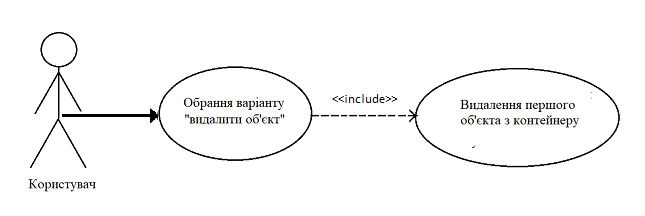


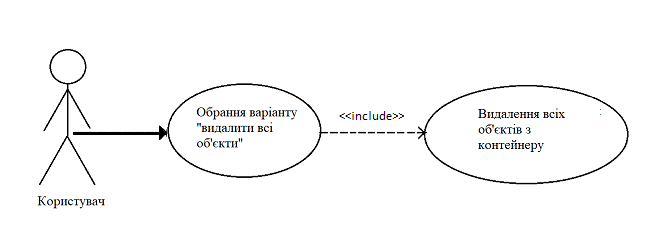
Рис. 1.3 - Видалення першого елементу в черзі з пріоритетом

* + 1. Опис методу видалення всіх об’єктів з контейнера (чергу з пріоритетом)

*Таблиця 1.4*

**Видалення всіх об’єктів з контейнера**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 4 | Видалення всіх об’єкти з черги з пріоритетом | Видалення об’єкти з черги з пріоритетом | Користувач | Основний | У процесі роботи програми надає можливість видалити всіх об’єктів з контейнеру |

 Рис. 1.4 - Видалення всіх елементів в черзі з пріоритетом

* + 1. Опис видаленняоб’єктів з контейнера (чергу з пріоритетом)

*Таблиця 1.5*

**Виведення всіх об’єктів з контейнера**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 5 | Виведення всіх об’єктів черги з пріоритетом у консоль | Вивести у консоль всі елементи черги з пріоритетом | Користувач | Основний | Надає можливість вивести інформацію про об’єкти з контейнеру у консоль |

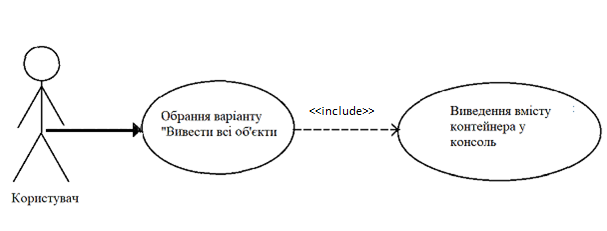


Рис. 1.5 – Виведення всіх об’єктів черги з пріоритетом у консоль

1.1.6. Збереження об'єктів контейнеру (черги з пріоритетом) у файлу

*Таблиця 1.6*

**Збереження об’єктів контейнера з файлу**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 6 | Збереження об’єктів контейнера у файлу | Зберегти  інформацію  про об’єкти у обраний файлу | Користувач | Основний | Надає можливість зберегти інформацію про всі об’єкти контейнера в обраний файл |

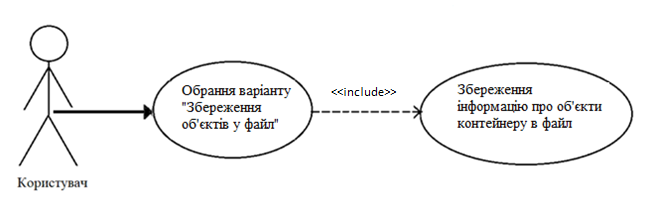
****

Рис. 1.6 - Збереження об’єктів черги з пріоритетом у файл

1.1.7. Завантаження з файлу об'єктів та внесення їх в контейнер (черги з пріоритетом)

*Таблиця 1.7*

**Завантажити об’єктів контейнера з файлу**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 7 | Завантаження об’єктів контейнера з файлу | Завантажити інформацію про об’єкти з файлу | Користувач | Основний | Надає можливість завантажити інформацію про всі об’єкти контейнера з файлу |

****

Рис. 1.7 - Завантаження об’єктів черги з пріоритетом в файл

1.1.8. Запит до об’єктів контейнера (черги з пріоритетом) в файл

*Таблиця 1.8*

**Виконання запиту до контейнера**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 8 | Запит до об’єктів контейнера в файл | Підрахувати кількість техніки у контейнері з заданами параметрами | Користувач | Основний | Надає можливість Підрахувати кількість техніки з одниковим типом роботи та виробником у контейнері |

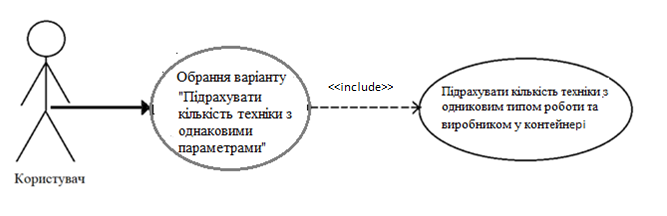
****

Рис. 1.8 - Підрахувати кількість техніки з одниковим типом роботи та виробником в черзі з пріоритетом

1.1.9. Здійснити вихід з програми застосунку

*Таблиця 1.9*

**Вихід з програми**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Індентифікатор | Назва | Мета | Дійові особи | Тип | Опис |
| 9 | Вихід з програми | Закінчити роботу з програмного застосунку | Користувач | Основний | У процесі роботи програми надає можливість виходу з програмного застосунку |

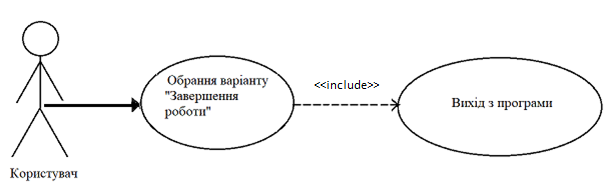
****

Рис. 1.9 – Вихід з програми

## Нефункціональні вимоги

В програмному застосунку реалізовані наступні нефункціональні вимоги:

● ID №10: Можливість на повторне введення номеру вибору пункту в меню при некоректному його введені.

● ID №11: Можливість на повторне введення при некоректному введені даних у заповнені полів об’єкту «регламентоване обслуговування» та «гарантійний ремонт».

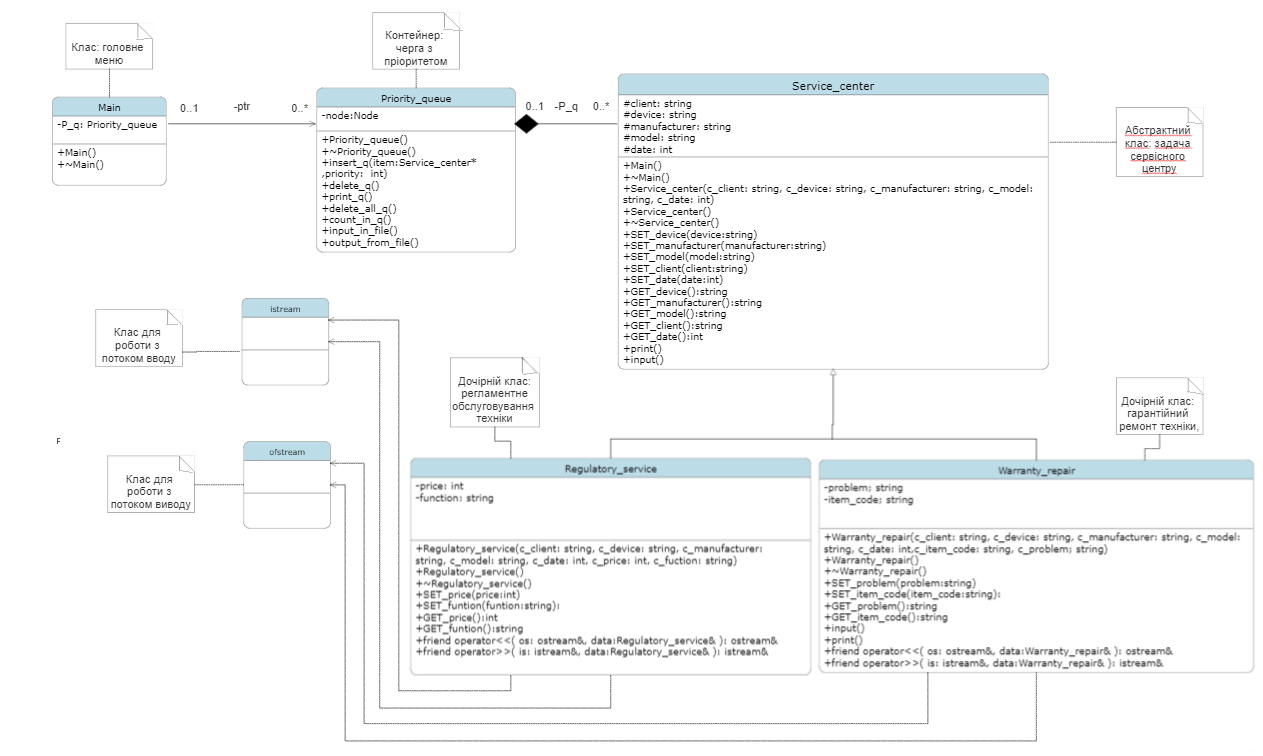
● ID №12: Можливість на повторне обирання файлу при невдалій перевірці на можливість відкриття файлу.

● ID №13: Можливість програмного застосунку тільки на операційній системі Windows.

# РОЗДІЛ 2

# СТРУКТУРА ТА ЛОГІКА РОБОТИ ПРОГРАМНОГО ЗАСТОСУНКУ

## 2.1 Логічна структура



­­Рис. 2.1. - Діаграма класів в нотації UML

Класи програмного застусунку, які були використані у данії роботі опишимо нижче.

2.1.1. Клас “Service\_center”

Клас “Service\_center” використовується для визначення параметрів притаманні “Задачам сервісного центру”, який є абстрактний класом для дочірніх класів. Опис полів та методів класу “Service\_center” розписується в табл. 2.1. та табл 2.2 відповідно.

*Табл 2.1.*

**Поля захищеного інтерфейсу класу “Service\_center”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| 1 | client | string | Ім’я клієнта |
| 2 | device | string | Назва техніки |
| 3 | manufacturer | string | Назва виробника |
| 4 | model | string | Модель техніки |
| 5 | date | int | Рік виготовлення |

*Табл 2.2.*

**Методи публічного класу “Service\_center”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Вхідні дані** | **Повертає значення** | **Призначення** |
| 1 | Service\_center() | - | - | Конструктор за замовчуванням |
| 2 | Service\_center(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date) | client, device, manufacturer, model, date | - | Конструктор за замовчуванням з параметрами |
| 3 | virtual ~Service\_center() | - | - | Деструктор |
| 4 | string GET\_device() | device | - | Отримання  назви техніки |
| 5 | string GET\_manufacturer() | manufacturer |  | Отримання  назви виробника |
| 6 | string GET\_model() | model | - | Отримання  Назви виробника |
| 7 | string GET\_client() | client | - | Отримання  ім’я клієнта |
| 8 | int GET\_date() | date | - | Отримання  рік виготовлення |
| 9 | void SET\_device(string device) | - | device | отримання  назви техніки |
| 10 | void SET\_manufacturer(string manufacturer) | - | manufacturer | Встановлення назви виробника |
| 11 | void SET\_model(string model) | - | model | Встановлення назви виробника |
| 12 | void SET\_client(string client) | - | client | Встановлення ім’я клієнта |
| 13 | void SET\_date(int date) | - | date | Встановлення рік виготовлення |
| 14 | virtual void print() | - | - | Виведення об’єкта консоль |
| 15 | virtual void input() | - | - | Встановлення всіх параметрів об’єкта |

2.1.2. Клас “Main”

Клас “Main” використовується для надання користовичу можливість обирати функції, які реалізовані в програмному застосунку. Опис полів та методів класу “Main” розписується в табл. 2.2. та табл 2.3 відповідно.

*Табл 2.3.*

**Поля приватного інтерфейсу класу “Main”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| P\_q | client | Priority\_queue\* | Черга з пріоритетом |

*Табл 2.4.*

**Методи публічного класу “Main”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Вхідні дані** | **Повертає значення** | **Призначення** |
| 1 | Main() | - | - | Конструктор за замовчуванням |
| 2 | ~ Main | - | - | Деструктор |

2.1.3. Клас “Priority\_queue”

Клас “Priority\_queue” використовується для роботи з об’єктами котрі в ньому знаходяться. Опис полів та методів класу “Priority\_queue” розписується в табл. 2.5. та табл 2.6. відповідно.

*Табл 2.5.*

**Поля приватного інтерфейсу класу “Priority\_queue”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| 1 | node | Node\* | Cтворення об’єкта |

*Табл 2.6.*

**Методи публічного класу “Priority\_queue”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Вхідні дані** | **Повертає значення** | **Призначення** |
| 1 | Priority\_queue(); | - | - | Конструктор за замовчуванням |
| 2 | ~Priority\_queue(); | - | - | Деструктор |
| 3 | Void insert\_q  (Service\_center\* item, int priority); | Item, priority | - | Додати новий об’єкт у чергу з пріоритом |
| 4 | void delete\_q(); | - | - | Видалити об’єкт |
| 5 | void print\_q(); | - | - | Виведення об’єкта консоль |
| 6 | void delete\_all\_q(); | - | - | Видалити всі об’єкти |
| 7 | void count\_in\_q(); | - | - | Підрахувати об’єкти з певними параметрами |
| 8 | void input\_in\_file(); | - | - | Зберегти об’єкти в файл |
| 9 | void output\_from\_file(); | - | - | Загрузити об’єкти з файлу |

2.1.4. Структура “Node”

Структура “Node” створена для роботи з класом “Priority\_queue” як елемент класу. Опис полів структури“Node” розписується в табл. 2.7.

*Табл. 2.7.*

**Структура “Node”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| 1 | data | Service\_center\* | Покажчик на об’єкт |
| 2 | pNext | Node\* | Покажчик на наступний об’єкт |
| 3 | priority | int | Приоритет об’єкту |

2.1.5. Клас “Regulatory\_service”

Клас “Regulatory\_service” використовується, як дочірній клас абстрактного класу “Service\_center”, щоб відрізнити який тип “регламентне обслуговування техніки” від “гарантійний ремонт техніки”. Опис полів та методів класу “Regulatory\_service”розписується в табл. 2.8. та табл 2.9. відповідно.

*Табл. 2.8.*

**Поля приватного інтерфейсу класу “Regulatory\_service”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| 1 | price | int | Ціна обслуговування |
| 2 | funcition | string | Вид обслуговування |

*Табл. 2.9.*

**Методи публічного класу “Regulatory\_service”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Вхідні дані** | **Повертає значення** | **Призначення** |
| 1 | Regulatory\_service() | - | - | Конструктор за замовчуванням |
| 2 | Regulatory\_service(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, int c\_price, string c\_function) | client, device, manufacturer, model, date, price, function | - | Конструктор за замовчуванням з параметрами |
| 3 | ~Regulatory\_service () | - | - | Деструктор |
| 4 | void GET\_price() | - | price | Отримання  ціні обслуговування |
| 5 | void GET\_function() | - | function | Отримання  вид обслуговування |
| 6 | void SET\_ price(int price) | price | - | Встановлення  назви техніки |
| 7 | void SET\_ function (string function) | function | - | Встановлення  назви техніки |
| 8 | void print() | - | - | Виведення об’єкта консоль |
| 9 | void input() | - | - | Встановлення всіх параметрів об’єкта |
| 10 | friend ostream& operator<<(ostream& os, Regulatory\_service& data) | os, data | os | Перевантаження в потік виводу |
| 11 | friend istream& operator>>(istream& is, Regulatory\_service& data) | is, data | is | Перевантаження в потік вводу |

2.1.6. Клас “Warranty\_repair”

Клас “Warranty\_repair” використовується, як дочірній клас абстрактного класу “Service\_center”, щоб відрізнити який тип “гарантійний ремонт техніки” від “регламентне обслуговування техніки”. Опис полів та методів класу “Warranty\_repair” розписується в табл. 2.10. та табл 2.11. відповідно.

*Табл. 2.10.*

**Поля приватного інтерфейсу класу “Warranty\_repair”**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Тип** | **Призначення** |
| 1 | item\_code | string | Серійний номер техніки |
| 2 | problem | string | Вид несправності |

*Табл. 2.11.*

**Методи публічного класу “Warranty\_repair”**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ з/п** | **Назва** | **Вхідні дані** | **Повертає значення** | **Призначення** |
| 1 | Warranty\_repair() | - | - | Конструктор за замовчуванням |
| 2 | Warranty\_repair (string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, int c\_price, string c\_function) | client, device, manufacturer, model, date, item\_code, function | - | Конструктор за замовчуванням з параметрами |
| 3 | ~Warranty\_repair() | - | - | Деструктор |
| 4 | void GET\_ item\_code() | - | item\_code | Встановлення  серійного номеру техніки |
| 5 | void GET\_function | - | function | Встановлення вид несправності |
| 6 | void SET\_ item\_code (string item\_code) | item\_code | - | Отримання серійного номеру техніки |
| 7 | void SET\_ function (string function) | function | - | Отримання серійного номеру техніки |
| 8 | void print() | - | - | Виведення об’єкта консоль |
| 9 | void input() | - | - | Встановлення всіх параметрів об’єкта |
| 10 | friend ostream& operator<<(ostream& os, Regulatory\_service& data) | os, data | os | Перевантаження в потік виводу |
| 11 | friend istream& operator>>(istream& is, Regulatory\_service& data) | is, data | is | Перевантаження в потік вводу |

## 2.2. Фізична структура

Опишимо в табл. 2.12. фізичну структура нашого програмного застосунку, а саме його розподіл компонентів, функцій, змінних та класів.

**Опис фізичної структури програмного застосунку**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Назва файлу** | **Компоненти** | | **Зовнішні компоненти** | |
| **Назва** | **Опис** | **Назва** | **Опис** |
| 1 | Main.h | Main | Визначення класу меню | Priority\_Queue.h | Робота з класом Priority\_Queue |
| Warranty\_repair.h | Робота з класом Warranty\_repair |
| Regulatory\_service.h | Робота з класом Regulatory\_service |
| 2 | Priority\_queue.h | Node | Структура для елементів в контейнері | Service\_center.h | Робота з класом Service\_center.h |
| Priority\_queue | Визначення класу для створення контейнера |
| 3 | Service\_center.h | Service\_center | Визначення абстрактного класу | string | Робота з рядками |
| fstream | Робота з файловим потоком |
| iostream | Робота з екранним потоком |
| 4 | Regulatory\_service.h | Regulatory\_service | Визначення дочірного класу | Service\_center.h | Робота з класом Service\_center.h |
| 5 | Warranty\_repair.h | Warranty\_repair | Визначення дочірного класу | Service\_center.h | Робота з класом Service\_center.h |
| 6 | Main.сpp | Main | Основна функція програмного застосунку | Main.h | Визначення класу меню |
| ~Main |
| 7 | Priority\_queue.cpp | Priority\_queue | Визначення методів класу Priority\_  queue | Priority\_queue.h | Визначення класу меню |
| ~Priority\_queue |
| insert\_q |
| delete\_q |
| print\_q |
| delete\_all\_q |
| count\_in\_q |
| input\_in\_file |
| output\_from\_file |
| 8 | Service\_center.cpp | Service\_center | Визначення методів класу Service\_  center | Service\_center.h | Визначення абстрактного класу Service\_center |
| Service\_center |
| ~Service\_center |
| GET\_device |
| GET\_manufacturer |
| GET\_model |
| GET\_client |
| GET\_date |
| SET\_divece |
| SET\_manufacturer |
| SET\_model |
| SET\_client |
| SET\_date |
| print |
| input |
| 9 | Regulatory\_service.  cpp | Regulatory\_service | Визначення методів дочірного класу Regulatory\_  service | Regulatory\_service.h | Визначення дочірнього класу Regulatory\_service |
| Regulatory\_service |
| ~Regulatory\_service |
| SET\_price |
| SET\_funtion |
| GET\_price |
| GET\_function |
| input |
| print |
| operator<< |
| operator>> |
| 10 | Warranty\_repair.cpp | Warranty\_repair | Визначення методів дочірного класу | Warranty\_repair.h | Визначення дочірнього класу Warranty\_repair |
| Warranty\_repair |
| ~Warranty\_repair |
| SET\_item\_code |
| SET\_problem |
| GET\_problem |
| GET\_item\_code |
| operator<< |
| operator>> |

## 2.3 Логіка роботи програмного застосунку

Логіка роботи подій програмного застосунку часової послідовності зображена послідовності на рис. 2.2 вигляді нотації UML.

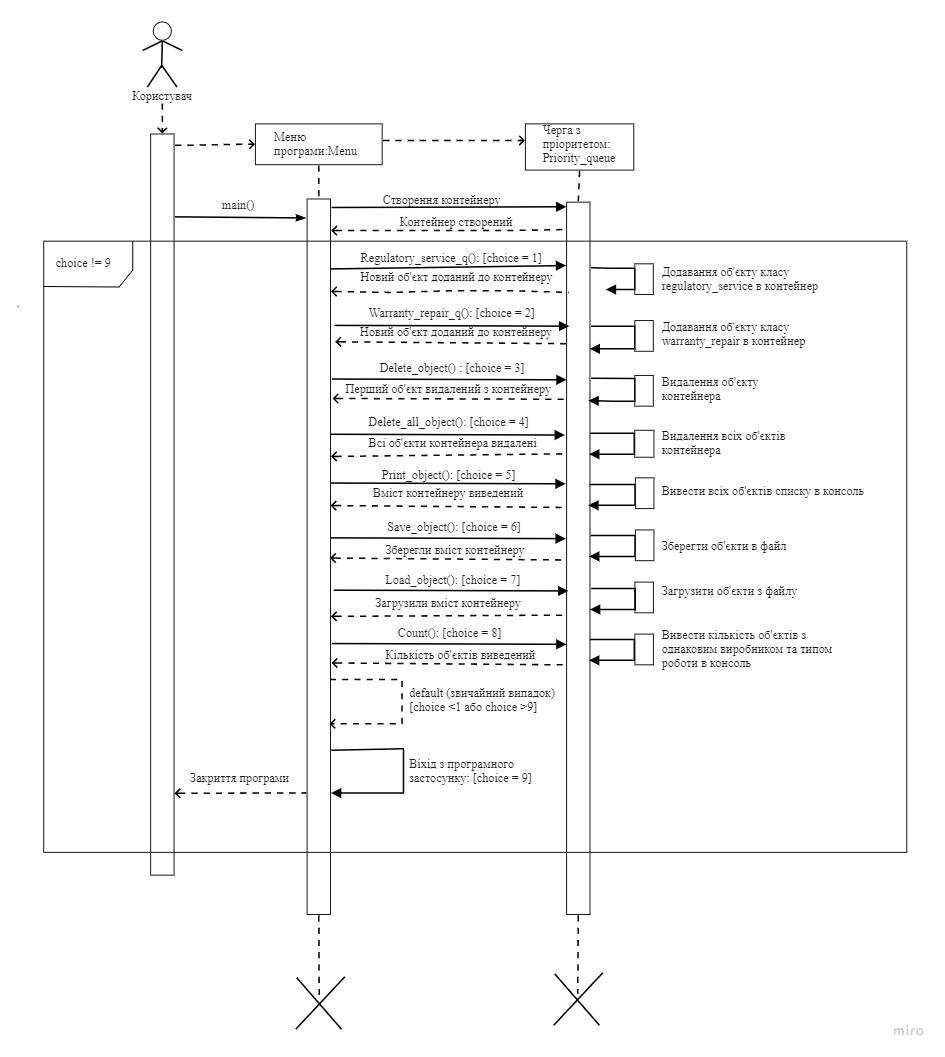


Рис. 2.2. - Часова послідовність подій у вигляді діаграми UML

# РОЗДІЛ 3

# ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

## 3.1 Сценарій тестування програмного застосунку

Сценарій тестування функцій об’єкта та нефункціональних вимог програмного застосунку описується в табл. 3.1 – 3.9 та в 3.10 – 3.13 відповідно.

3.1.1 Створення об’єкта типу «Regulatory\_service»

Сценарій створення і додавання об’єкту типу «регламентоване обслуговування» зображено у таблиці 3.1.

*Таблиця 3.1*

**Сценарій створення об’єкту типу «регламентоване обслуговування»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Назва | ID вимоги | Передумова | Тестові кроки | Очікуваний результат | Післяумова |
| f\_1 | Створення і додавання об’єкту «регламентоване обслуговування» | 1 | Користувач в головному меню пункт обирає пункт №1 | В головному меню вибрати «створити об’єкт регламентоване обслуговування» | Виводить запит на введення даних | Додавння нового об’єкту в контейнер та повернення в головне меню |

3.1.2 Створення об’єкта типу «Warranty\_repair»

Сценарій створення і додавання об’єкту типу «гарантійнийремонт» зображено у таблиці 3.2.

*Таблиця 3.2*

**Сценарій створення об’єкту типу «гарантійний ремонт»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_2 | Створення і додавання об’єкту «гарантійнийремонт» | 2 | Користувач в головному меню пункт обирає пункт №2 | В головному меню вибрати «гарантійнийремонт» | Виводить запит на введення даних | Додавння нового об’єкту в контейнер та повернення в головне меню |

3.1.3 Видалення об’єкту з контейнеру

Сценарій видалення об’єкту з контейнеру зображено у таблиці 3.3.

*Таблиця 3.3*

**Сценарій тестування видалення об’єкту з контейнера**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_3 | Виделання першого об’єкту(голови) контейнеру | 3 | Користувач в головному меню пункт обирає пункт №3 | В головному меню вибрати «Видалення об’єкту» | Виделання першого об’єкту  (голови) контейнеру | Повернення в головне меню |

3.1.4 Видалення всіх об’єктів з контейнеру

Сценарій видалення всіх об’єктів з контейнеру зображено у таблиці 3.4.

*Таблиця 3.4*

**Сценарій тестування видалення всіх об’єктів з контейнера**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_4 | Виделання всіх об’єктів контейнера | 4 | Користувач в головному меню пункт обирає пункт №4 | В головному меню вибрати «Видалення всіх об’єктів» | Виделання всіх об’єктів контейнера | Повернення в головне меню |

3.1.5 Виведення об’єктів з контейнеру в консоль

Сценарій тестування виведення об’єктів з контейнеру в консоль зображено у таблиці 3.5.

*Таблиця 3.5*

**Сценарій тестування виведення об’єктів в консоль**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_5 | Виведення об’єктів в консоль | 5 | Користувач в головному меню пункт обирає пункт №5 | В головному меню вибрати «Виведення об’єктів в консоль» | Виведення об’єктів в консоль | Повернення в головне меню |

3.1.6 Збереження об’єктів з контейнеру в файл

Сценарій тестування збереження об’єктів з контейнеру в файл зображено у таблиці 3.6.

*Таблиця 3.6*

**Сценарій тестування збереження об’єктів з контейнеру в файл**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_6 | Збереження об’єктів з контейнеру в файл | 6 | Користувач в головному меню пункт обирає пункт №6 | В головному меню вибрати «збереження об’єктів з контейнеру в файл» та введення назви файлу | Виводить запит на введення назви файлу та збереження об’єктів з контейнеру в файл | Повернення в головне меню |

3.1.7 Загрузка об’єктів з файл в контейнер

Сценарій тестування загрузка об’єктів з файл в контейнер зображено у таблиці 3.7.

*Таблиця 3.7*

**Сценарій тестування збереження об’єктів з контейнеру в файл**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_7 | Загрузка об’єктів з файл в контейнер | 7 | Користувач в головному меню пункт обирає пункт №7 | В головному меню вибрати «Загрузка об’єктів з файл» та введення назви файлу | Виводить запит на введення назви файлу та загрузка об’єктів з файл в контейнер | Повернення в головне меню |

3.1.8 Розробка запиту: загальна кількість гарантійних ремонтів техніки заданої фірми

Сценарій тестування підрахування загальну кількість гарантійних ремонтів техніки заданої фірми зображено у таблиці 3.8.

*Таблиця 3.8*

**Сценарій тестування збереження об’єктів з контейнеру в файл**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_8 | Підрахування загальну кількість гарантійних ремонтів техніки заданої фірми | 8 | Користувач в головному меню пункт обирає пункт №8 | В головному меню вибрати «Вивести підрахування кількість гарантійних ремонтів техніки заданої фірми» | Підрахування загальну кількість гарантійних ремонтів техніки заданої фірми | Повернення в головне меню |

3.1.9 Вихід з програмного застосунку

Сценарій тестування вихіду з програмного застосунку зображено у таблиці 3.9.

*Таблиця 3.9*

**Сценарій тестування вихіду з програмного застосунку**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_9 | Вихід | 9 | Користувач в головному меню пункт обирає пункт №9 | В головному меню вибрати «Вихід» | Очищення контейнеру від об’єктів та ініціалізувати вихід з програмного застосунку | Завершення програми |

3.1.10. Некоректне введення даних

Сценарій тестування нефункціональних вимог: некоректного введення даних показаний в табл. 3.10.

*Таблиця 3.10*

**Сценарій тестування некоректне введення даних в головному меню**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_11 | Некоректне введення даних в меню | 11 | Ініціалізувати запуск програмного застосунку | Введення некоректних даних | Повідомлення про некоректне введення даних та надання можливість ввести повторно дані | Надання можливості повторного введення даних |

3.1.11. Некоректне введення даних в об’єктах

Сценарій тестування нефункціональних вимог: некоректного введення даних в об’єктах «регламентоване обслуговування» та «гарантійний ремонт» показаний в табл. 3.11.

*Таблиця 3.11*

**Сценарій тестування некоректного введення даних в об’єктах «регламентоване обслуговування» та «гарантійний ремонт»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_11 | Некоректне введення даних | 11 | Обрати в головному меню пункт №1 або №2 | Введення некоректних даних | Надання можливість ввести повторно дані | Надання можливості повторного введення даних |

3.1.12. Некоректне введення назви файла

Сценарій тестування нефункціональних вимог: некоректного введення введення назви файла показаний в табл. 3.12.

*Таблиця 3.12*

**Сценарій тестування некоректного введення назви файла**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_12 | Некоректного введення введення назви файла | 12 | Обрати в головному меню пункт №6 або №7 | Введення некоректного назви файла | Повідомлення про помилкове введення даних та ініціалізувати повернення в головне меню | Повернення в головне меню |

3.1.13. Запуск програмного застосунку з іншої операційної системи

Сценарій тестування нефункціональних вимог: запуск програмного застосунку не з операційної системи Windows показаний в табл. 3.13.

*Таблиця 3.13*

**Сценарій тестування програмного застосунку з іншої операційної системи**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Назва** | **ID вимоги** | **Передумова** | **Тестові кроки** | **Очікуваний результат** | **Післяумова** |
| f\_13 | Некоректне введення даних | 13 | Відкрити програмний застосунок під час використання не операційної системи Windows | Запустити програмний застосунок | Програма не закуститься | Не використовувати програмний застосунок під час роботи на іншій операційній системі |

## Тестові дані

Опис тестових даних для об’єктів зображений на діаграмі нотації UML на рис. 3.1.

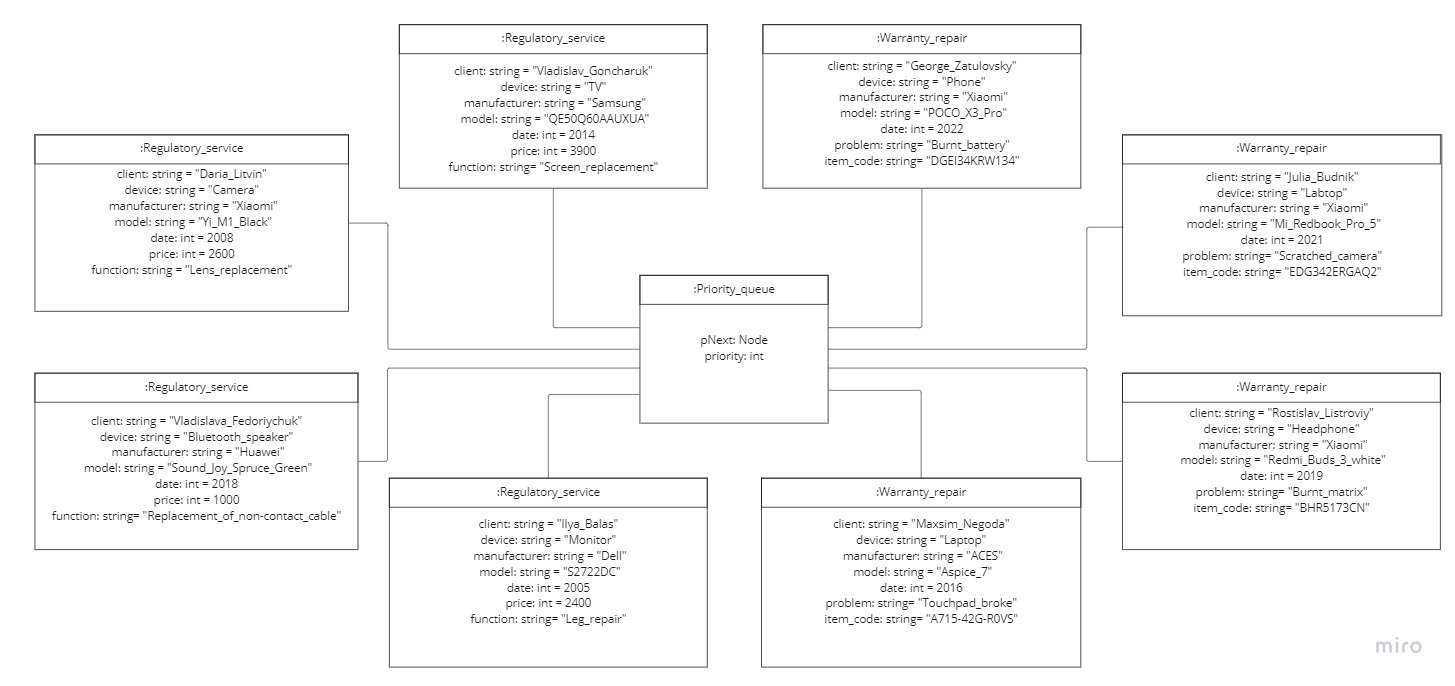


Рис. 3.1 – Діаграма для нотації UML

Вміст файлу, в якому зберігаються тестові дані для програмного застосунку представлені на рис 3.2.



Рис. 3.2. – Тестові дані збережені в файлі для програмного застосунку.

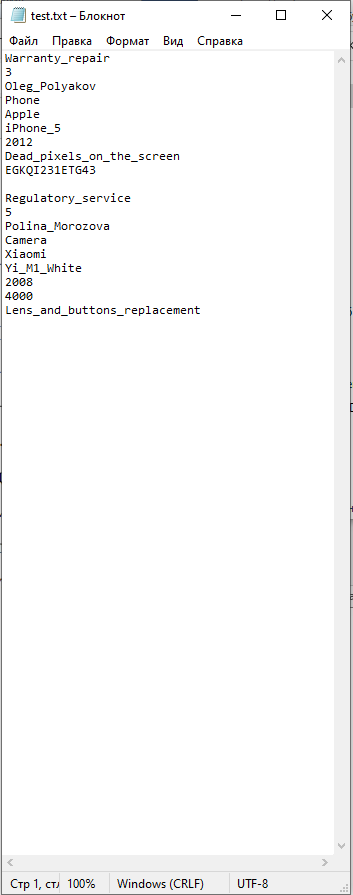


Рис. 3.3 - Тестові дані збережені в файлі під час роботи програмного застосунку (рис. 3.8)

## Поетапні результати роботи програмного застосунку

Рис. 3.3 - 3.23 демонструє роботу функції програмного застовунку. Лістинг програмного застосунку наведена у додатку А.

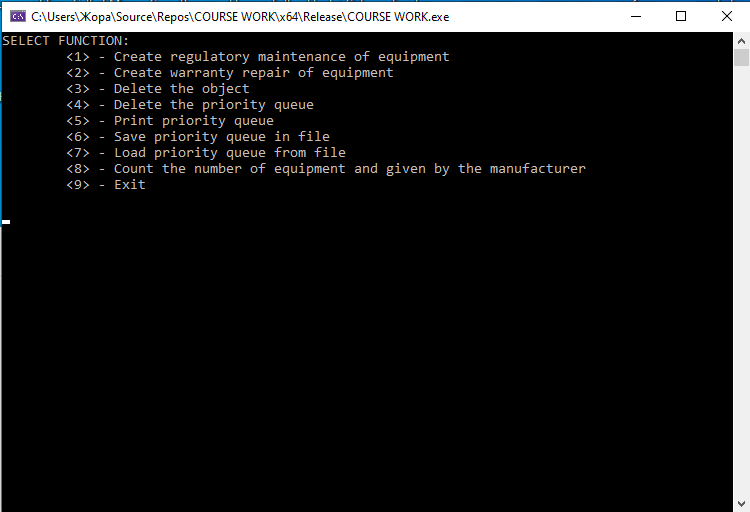


Рис. 3.4 – Головне меню програмного застосунку

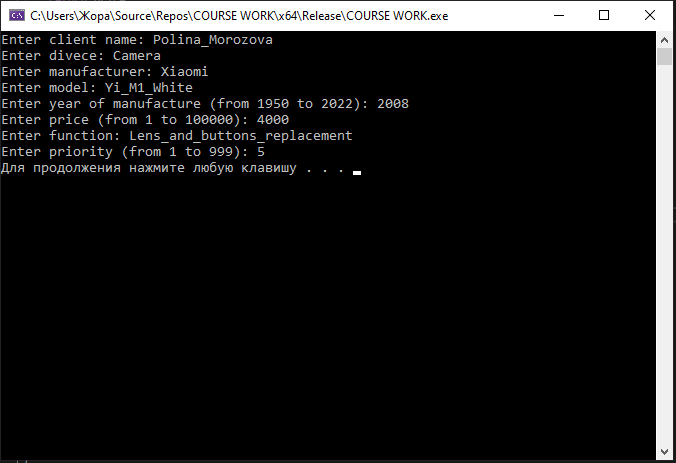


Рис 3.5. - Створення об’єкту «Регламентоване обслуговування»

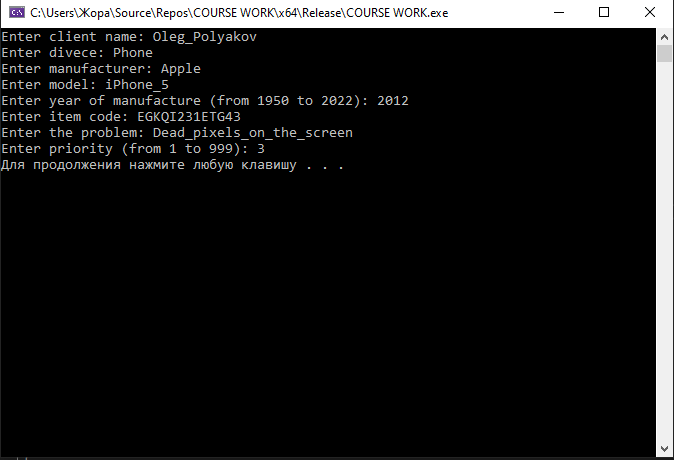


Рис 3.6 - Створення об’єкту «Гарантійне обслуговування»



Рис 3.7 – Виведення вмісту контейнеру в консоль

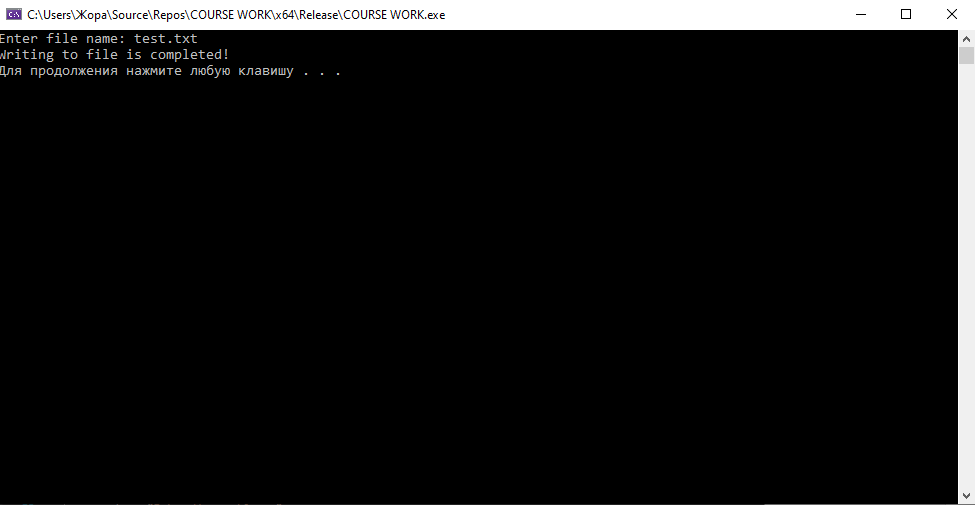


Рис 3.8 – Зберегаємо об’єкти у файл test.txt

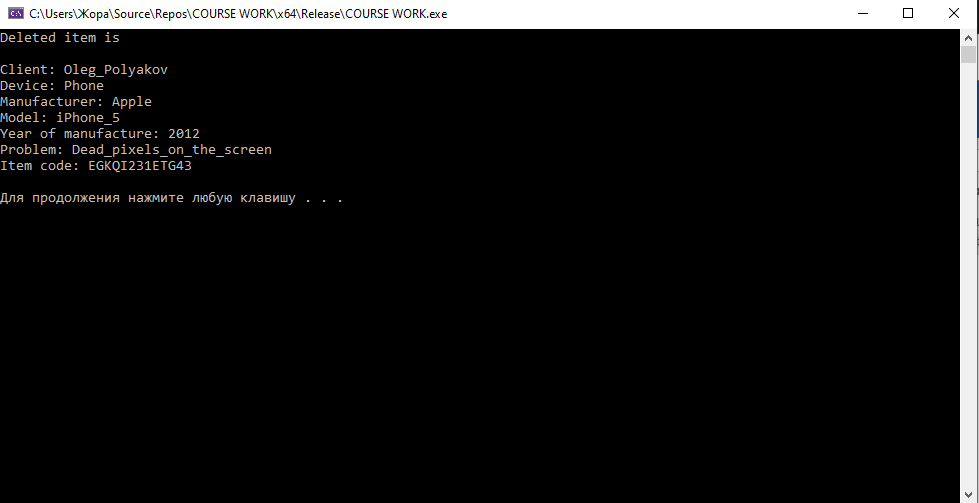


Рис 3.9 – Видалення першого об’єкту з контейнера



Рис 3.10 – Виведення вмісту контейнеру в консоль після видалення об’єкту

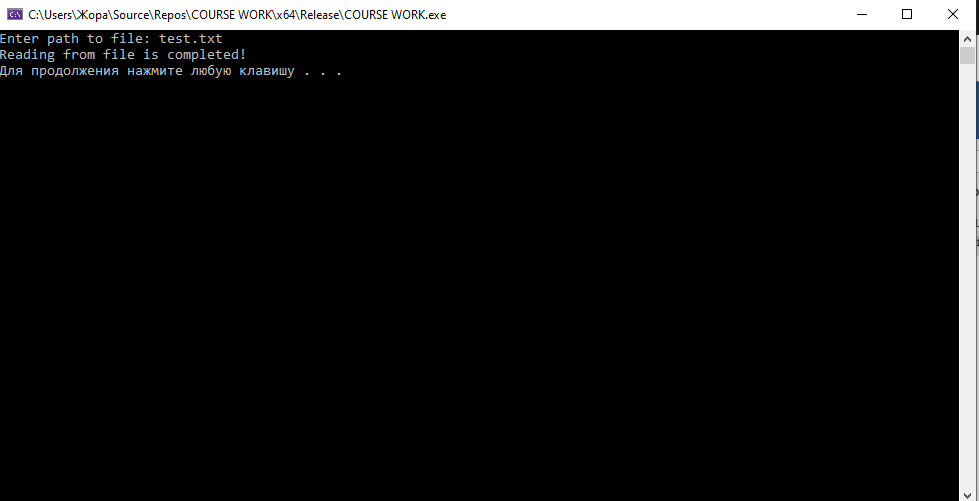


Рис 3.11 – Зчитування даних з файлу test.txt

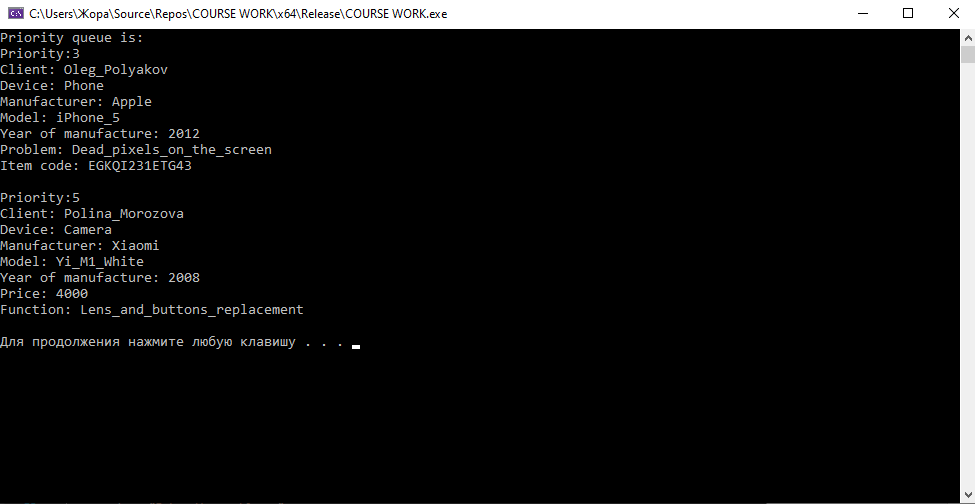


Рис 3.12 – Виведення вмісту контейнеру після зчитування з файлу test.txt

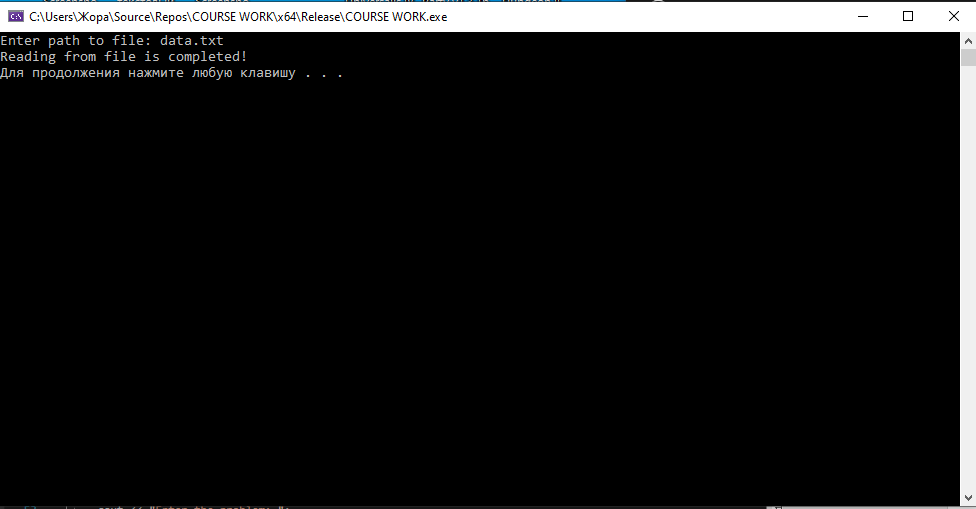


Рис 3.13 – Зчитування даних з файлу data.txt

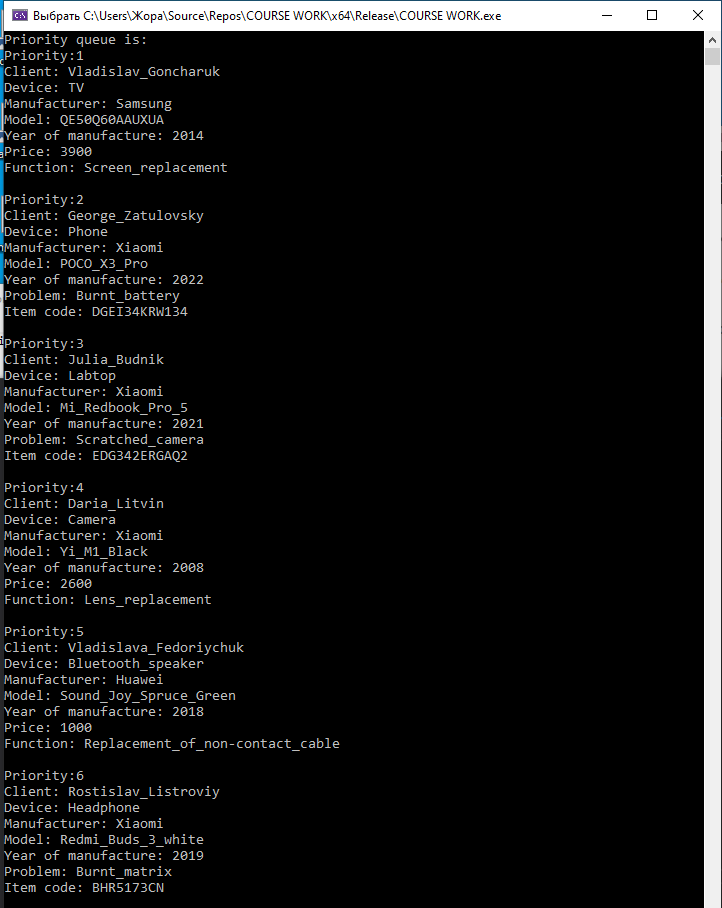


Рис 3.14 – Виведення вмісту контейнеру після зчитування з файлу data.txt

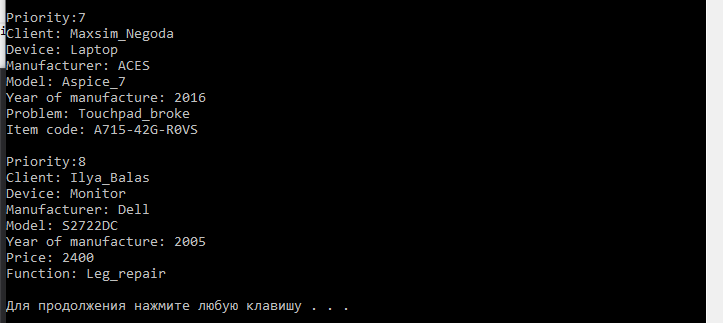


Рис 3.15 – Виведення вмісту контейнеру після зчитування з файлу data.txt (Продовження)

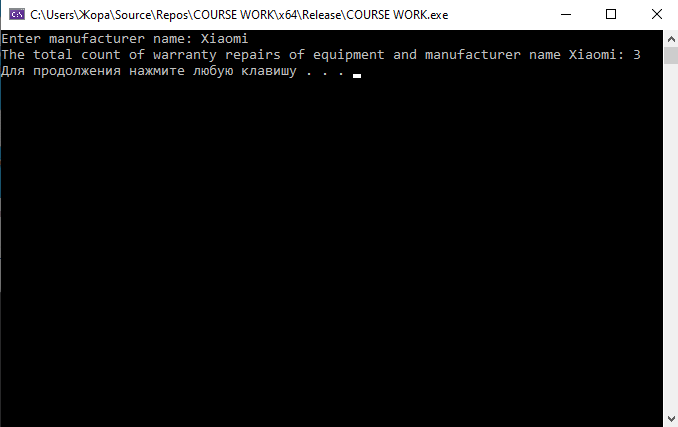


Рис 3.16 – Підрахунок кількості об’єктів з

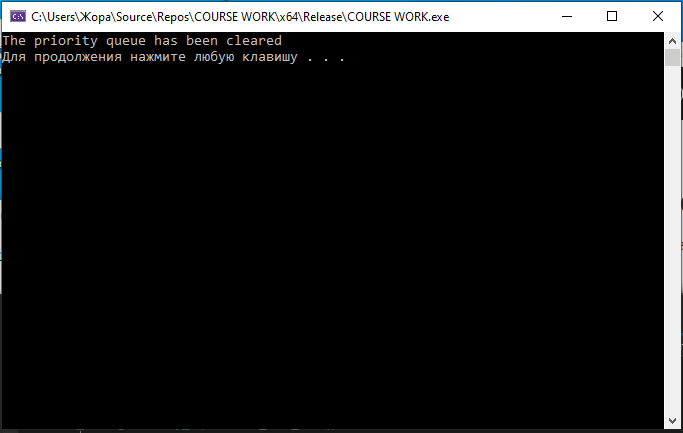


Рис 3.17 – Видалення всіх об’єктів з контейнеру

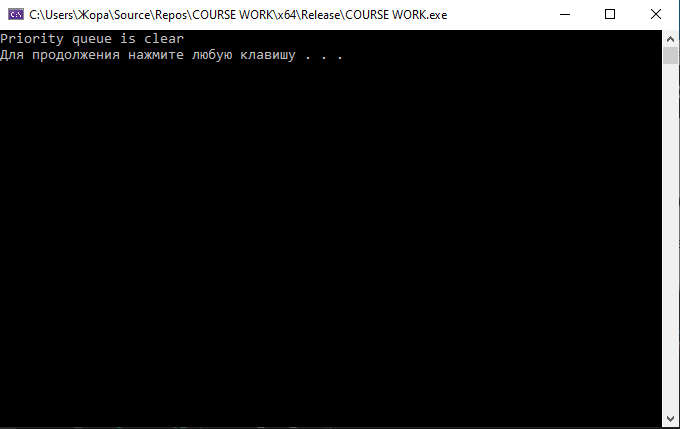


Рис 3.18 – Спроба видалення першого об’єкту у пустому контейнері

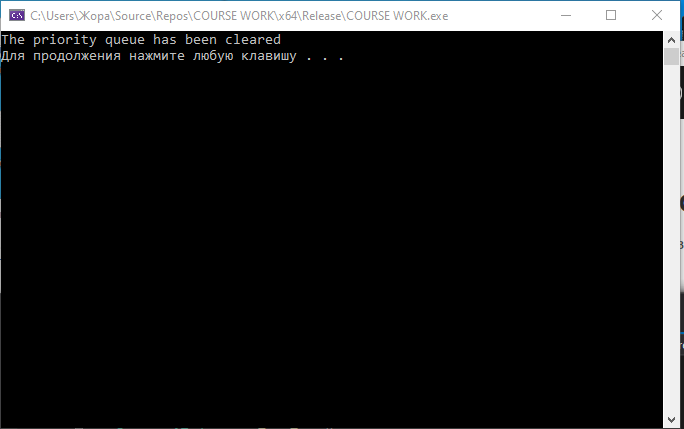


Рис 3.19 - Спроба видалення всіх об’єктів у пустому контейнері

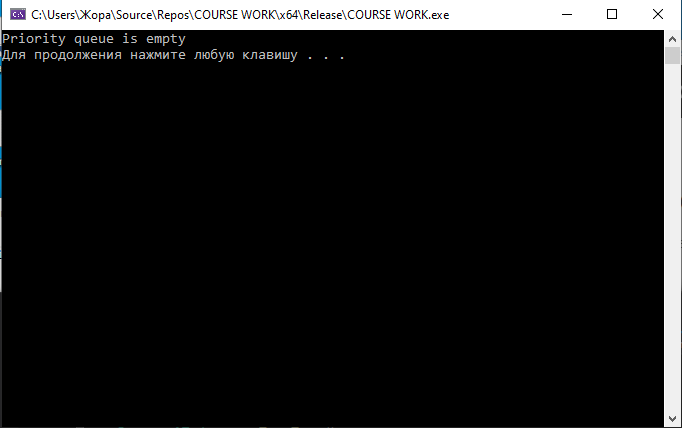


Рис. 3.20 – Спроба виведення пустого контейнера у консоль

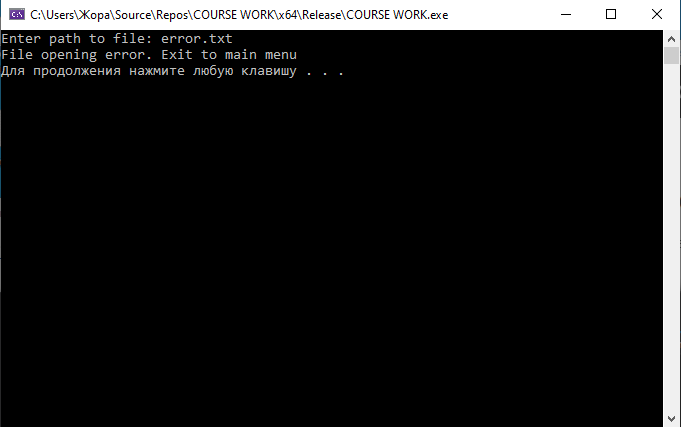
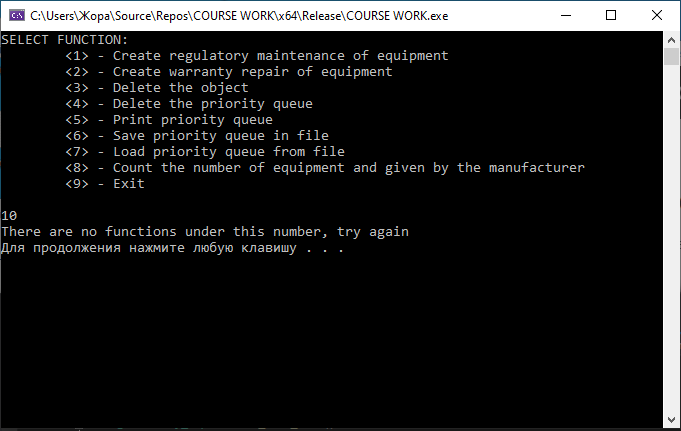


Рис 3.21 - Помилка при відкритті неіснуючого файлу error.txt

 Рис 3.22 – Некоректний ввід у меню

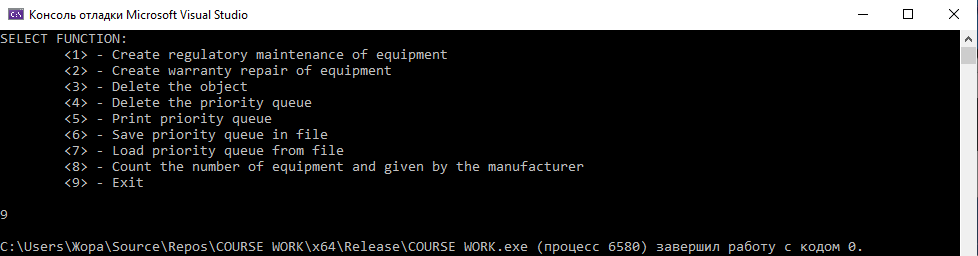


Рис 3.23 – Ініціалізувати вихід з програмного застосунку

## Модульне тестування

Результати виконання Unit тесту наведені на рис. 3.24. Лістінг коду Unit тесту наведено у додатку Б.

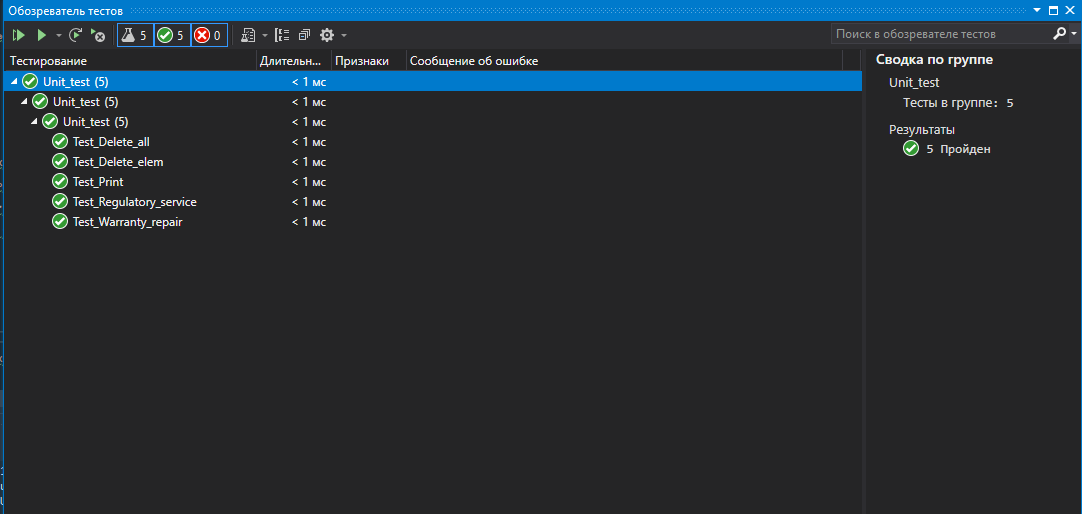


Рис. 3.24 – Результати модульного тесту Unit\_test

**ВИСНОВОК**

Під час виконання курсової роботи був реалізованний програмний застосунок, написаний мовою С++, з використанням основних принципів об’єктно-орієнтованого програмування.

У результаті отримано програмний застосунок з ієрархією класів, який складається з абстрактного класу “Задача сервісного центру” та класів “Гарантійний ремонт техніки” і “Регламентне обслуговування техніки” враховуючи парадигми поліморфізму та успадкування, використовуючи контейнер “Черга з пріоритетом “ для зберігання об’єктів класів. До цього програмного застосунку реалізувано меню користувача для роботи з програмою та завдання для розробки запиту, а саме підрахування загальної кількості гарантійних ремонтів техніки заданої фірми у контейнері.

Також проведено тестування програмного застосунку за допомогою модульних тестів (Unit тести) і меню користувача. Завдяки цим тестуванням можна впевнитись у коректності роботи програмного застосунку.

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Об’єктно-орієнтовне програмування : підручник. У 2-х ч. Ч. 2. Обєктно-орієнтовний підхід до розробки програмного забезпечення / С.М. Алхімова. — Київ: КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2019. — 192с.
2. Алхімова С.М. Алгоритмізація та програмування мовою високого рівня С++: комп’ютерний практикум / С. М. Алхімова. – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2018. – 156с.
3. Язык программирования С++. Лекции и упражнения, 5-е изд. Пер. с англ. ­ "И.Д. Вильяме", 2007. - 1 184 с.
4. Лафоре Р. Объектно-ориентированное программирование в С++, Классика Computer Science 4-е изд. - СПБ. Питер, 2011.- 928 с

**ДОДАТОК А**

## Код програмного продукту

Main.cpp

/\*

\* Реалізація головного меню Main

\*/

#include "Main.h"

#include <iostream>

#include <string>

using namespace std;

//Метод для створення об'єкта класа Regulatory\_service

void Regulatory\_service\_q(Priority\_queue\*& P\_q)

{

system("cls");

Regulatory\_service\* info = new Regulatory\_service();

int priority = 0;

info->input();

while (priority <= 0 || priority > 999)

{

cout << "Enter priority (from 1 to 999): ";

cin >> priority;

}

P\_q->insert\_q(info, priority);

system("pause");

system("cls");

}

//Метод для створення об'єкта класа Warranty\_repair

void Warranty\_repair\_q(Priority\_queue\*& P\_q)

{

system("cls");

Warranty\_repair\* info = new Warranty\_repair();

int priority = 0;

info->input();

while (priority <= 0 || priority > 999)

{

cout << "Enter priority (from 1 to 999): ";

cin >> priority;

}

P\_q->insert\_q(info, priority);

system("pause");

system("cls");

}

//Метод для видалення першого елементу контейнера

void Delete\_object(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->delete\_q();

system("pause");

system("cls");

}

//Метод для видалення всіх елементів контейнера

void Delete\_all\_object(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->delete\_all\_q();

cout << "The priority queue has been cleared" << endl;

system("pause");

system("cls");

}

//Метод для виведення елементів контейнера

void Print\_object(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->print\_q();

system("pause");

system("cls");

}

//Метод для збереження елементів контейнера в файл

void Save\_object(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->input\_in\_file();

system("pause");

system("cls");

}

//Метод для загруски елементів контейнера з файлу

void Load\_object(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->output\_from\_file();

system("pause");

system("cls");

}

//Метод для підрахунку елементів з однаковими даними

void Count(Priority\_queue\*& P\_q)

{

system("cls");

P\_q->count\_in\_q();

system("pause");

system("cls");

}

//Головне меню

int main()

{

Priority\_queue\* P\_q = new Priority\_queue;

int choice;

string cheak\_choice;

do

{

cout << "SELECT FUNCTION:" <<

"\n\t<1> - Create regulatory maintenance of equipment" <<

"\n\t<2> - Create warranty repair of equipment" <<

"\n\t<3> - Delete the object" <<

"\n\t<4> - Delete the priority queue" <<

"\n\t<5> - Print priority queue" <<

"\n\t<6> - Save priority queue in file" <<

"\n\t<7> - Load priority queue from file" <<

"\n\t<8> - Count the number of equipment and given by the manufacturer" <<

"\n\t<9> - Exit" << endl << endl;

try

{

cin >> choice;

}

catch (const exception&)

{

cout << "Incorrect data entry, try again" << endl;

system("pause");

system("cls");

}

switch (choice)

{

case 1:

{

Regulatory\_service\_q(P\_q);

break;

}

case 2:

{

Warranty\_repair\_q(P\_q);

break;

}

case 3:

{

Delete\_object(P\_q);

break;

}

case 4:

{

Delete\_all\_object(P\_q);

break;

}

case 5:

{

Print\_object(P\_q);

break;

}

case 6:

{

Save\_object(P\_q);

break;

}

case 7:

{

Load\_object(P\_q);

break;

}

case 8:

{

Count(P\_q);

break;

}

case 9:

{

break;

}

default:

{

cout << "There are no functions under this number, try again" << endl;

system("pause");

system("cls");

break;

}

}

} while (choice != 9);

return 0;

}

//Конструктор

Main::Main()

{

}

//Деструктор

Main::~Main()

{

P\_q->delete\_all\_q();

}

Main.h

/\*

\* Заголовний файл Main

\*/

#pragma once

#include "Priority\_queue.h"

#include "Warranty\_repair.h"

#include "Regulatory\_service.h"

class Main

{

private:

//Створення черги з пріорітетом

Priority\_queue\* P\_q;

public:

//Методи головного меню

Main();

~Main();

};

Priority\_queue.cpp

/\*

\* Реалізація контейнеру (черги з пріорітетом) Priority\_queue

\*/

#include "Priority\_queue.h"

//Конструктор

Priority\_queue::Priority\_queue()

{

}

//Деструктор

Priority\_queue::~Priority\_queue()

{

}

//Метод додавання об'єкта у контейнер

void Priority\_queue::insert\_q(Service\_center\* item, int priority)

{

Node\* tmp, \* pBack;

tmp = new Node;

tmp->data = item;

tmp->priority = priority;

if (node == NULL || priority <= node->priority)

{

tmp->pNext = node;

node = tmp;

}

else

{

pBack = node;

while (pBack->pNext != NULL && pBack->pNext->priority < priority)

{

pBack = pBack->pNext;

}

tmp->pNext = pBack->pNext;

pBack->pNext = tmp;

}

}

//Метод видалення об'єкта контейнера

void Priority\_queue::delete\_q()

{

Node \*tmp;

if (node == NULL)

{

cout << "Priority queue is clear" << endl;

}

else

{

tmp = node;

cout << "Deleted item is \n\n";

tmp->data->print();

node = node->pNext;

free(tmp);

}

}

//Метод виведення об'єктів контейнера у консоль

int Priority\_queue::print\_q()

{

Node\* ptr = node;

if (node == NULL)

{

cout << "Priority queue is empty " << endl;

return 0;

}

else

{

cout << "Priority queue is: " << endl;

while (ptr != NULL)

{

Service\_center\* Data\_info = ptr->data;

cout <<"Priority:"<< ptr->priority<< endl;

Data\_info->print();

ptr = ptr->pNext;

}

return 1;

}

}

//Метод видалення всіх об'єктів контейнера

void Priority\_queue::delete\_all\_q()

{

Node\* tmp;

while (node != NULL)

{

tmp = node;

node = node->pNext;

free(tmp);

}

}

//Метод підрахунок об'єктів у контейнері з однаковими параметрами

int Priority\_queue::count\_in\_q()

{

int count = 0;

Node\* tmp = node;

string find\_manufacturer = " ";

cout << "Enter manufacturer name: ";

cin >> find\_manufacturer;

while (tmp != NULL)

{

if (typeid(\*(tmp->data)) == typeid(Warranty\_repair) && (tmp->data)->GET\_manufacturer() == find\_manufacturer)

{

count++;

}

tmp = tmp->pNext;

}

cout << "The total count of warranty repairs of equipment and manufacturer name "<< find\_manufacturer<<": " << count << endl;

return count;

}

//Метод збереження об'єктів контейнеру у файл

void Priority\_queue::input\_in\_file()

{

Node\* ptr = node;

if (ptr != NULL)

{

string path;

cout << "Enter file name: ";

cin >> path;

ofstream fout;

fout.open(path);

if (!fout.is\_open())

{

cout << "File opening error. Exit to main menu" << endl;

return;

}

while (ptr != NULL)

{

if (typeid(\*(ptr->data)) == typeid(Regulatory\_service))

{

fout << "Regulatory\_service" << endl;

fout << (ptr->priority)<<endl;

fout << (\*((Regulatory\_service\*)(ptr->data)));

}

else

{

fout << "Warranty\_repair"<<endl;

fout << (ptr->priority) << endl;

fout << (\*((Warranty\_repair\*)(ptr->data)));

}

ptr = ptr->pNext;

fout << endl;

}

fout.close();

cout << "Writing to file is completed!" << endl;

}

}

//Метод зчитування об'єктів контейнеру з файлу

void Priority\_queue::output\_from\_file()

{

string path = " ";

cout << "Enter path to file: ";

cin >> path;

ifstream fin;

fin.open(path);

if (!fin.is\_open())

{

cout << "File opening error. Exit to main menu" << endl;

return;

}

else

{

delete\_all\_q();

string type;

int priority\_file = 0;

char cheak;

while (!fin.eof())

{

fin >> type;

fin >> priority\_file;

if (type == "Regulatory\_service")

{

Regulatory\_service\* R\_s\_file = new Regulatory\_service;

fin >> \*(R\_s\_file);

insert\_q(R\_s\_file, priority\_file);

}

else if(type == "Warranty\_repair")

{

Warranty\_repair\* W\_r\_file = new Warranty\_repair;

fin >> \*(W\_r\_file);

insert\_q(W\_r\_file, priority\_file);

}

type = "";

}

fin.close();

cout << "Reading from file is completed!" << endl;

return;

}

}

Priority\_queue.h

/\*

\* Заголовний файл Priority\_queue

\*/

#pragma once

#include <iostream>

#include <string>

#include <typeinfo>

#include "Service\_center.h"

#include "Regulatory\_service.h"

#include "Warranty\_repair.h"

using namespace std;

//Структура об'єкта

struct Node

{

Service\_center\* data;

struct Node\* pNext;

int priority;

};

class Priority\_queue

{

private:

Node\* node = NULL; //Cтворення об'єкта

public:

//Методи контейнера

Priority\_queue();

~Priority\_queue();

void insert\_q(Service\_center\* item, int priority);

void delete\_q();

int print\_q();

void delete\_all\_q();

int count\_in\_q();

void input\_in\_file();

void output\_from\_file();

};

Regulatory\_service.cpp

/\*

\* Реалізація класу нащадка Regulatory\_service

\*/

#include "Regulatory\_service.h"

//Конструктор

Regulatory\_service::Regulatory\_service(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, int c\_price, string c\_funtion):Service\_center(c\_client, c\_device,c\_manufacturer, c\_model, c\_date),price(c\_price), function(function)

{

}

//Конструктор

Regulatory\_service::Regulatory\_service()

{

}

//Деструктор

Regulatory\_service::~Regulatory\_service()

{

}

//Реалізація сеттерів

void Regulatory\_service::SET\_price(int price)

{

this->price = price;

}

void Regulatory\_service::SET\_funtion(string function)

{

this->function = function;

}

//Реалізація геттерів

int Regulatory\_service::GET\_price()

{

return price;

}

string Regulatory\_service::GET\_function()

{

return function;

}

//Реалізація функції введення параметрів

void Regulatory\_service::input()

{

Service\_center::input();

cout << "Enter price (from 1 to 100000): ";

cin >> price;

while (price < 1 || price > 100000)

{

cin >> price;

}

cout << "Enter function: ";

cin >> function;

}

//Реалізація функції виведення параметрів

void Regulatory\_service::print()

{

Service\_center::print();

cout << "\nPrice: " << price

<< "\nFunction: " << function << endl << endl;

}

//Запис даних полів класу у файловий потік виводу

ostream& operator<<(ostream& os, Regulatory\_service& data)

{

os << data.client << endl

<< data.device << endl

<< data.manufacturer << endl

<< data.model << endl

<< data.date << endl

<< data.price << endl

<< data.function << endl;

return os;

}

//Запис даних полів класу у файловий потік вводу

istream& operator>>(istream& is, Regulatory\_service& data)

{

is >> data.client >> data.device >> data.manufacturer >> data.model >> data.date >>data.price >> data.function;

return is;

}

Regulatory\_service.h

/\*

\* Заголовний файл Regulatory\_service

\*/

#pragma once

#include "Service\_center.h"

class Regulatory\_service: public Service\_center

{

private:

int price; //Ціна

string function; //Тип сервісу

public:

Regulatory\_service(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, int c\_price, string c\_function);

Regulatory\_service();

~Regulatory\_service();

//Сеттери класу

void SET\_price(int price);

void SET\_funtion(string function);

//Геттери класу

int GET\_price();

string GET\_function();

void input();

void print();

//Френдлі функції перевантаження операторів вводу та виводу

friend ostream& operator<<(ostream& os, Regulatory\_service& data);

friend istream& operator>>(istream& is, Regulatory\_service& data);

};

Service\_center.cpp

/\*

\* Реалізація абстрактного класу Service\_center

\*/

#include "Service\_center.h"

//Конструктор

Service\_center::Service\_center()

{

}

//Конструктор

Service\_center::Service\_center(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date):client(c\_client), device(c\_device), manufacturer(c\_manufacturer), model(c\_model), date(c\_date)

{

}

//Деструктор

Service\_center::~Service\_center()

{

}

//Реалізація геттерів

string Service\_center::GET\_device()

{

return device;

}

string Service\_center::GET\_manufacturer()

{

return manufacturer;

}

string Service\_center::GET\_model()

{

return model;

}

string Service\_center::GET\_client()

{

return client;

}

int Service\_center::GET\_date()

{

return date;

}

//Реалізація сеттерів

void Service\_center::SET\_divece(string device)

{

this->device = device;

}

void Service\_center::SET\_manufacturer(string manufacturer)

{

this->manufacturer = manufacturer;

}

void Service\_center::SET\_model(string model)

{

this->model = model;

}

void Service\_center::SET\_client(string client)

{

this->client = client;

}

void Service\_center::SET\_date(int date)

{

this->date = date;

}

//Реалізація функції виведення параметрів

void Service\_center::print()

{

cout << "Client: " << client

<< "\nDevice: " << device

<< "\nManufacturer: " << manufacturer

<< "\nModel: " << model

<< "\nEnter year of manufacture: " << date;

}

//Реалізація функції введення параметрів

void Service\_center::input()

{

cout << "Enter client name: ";

cin >> client;

cout << "Enter divece: ";

cin >> device;

cout << "Enter manufacturer: ";

cin >> manufacturer;

cout << "Enter model: ";

cin >> model;

cout << "Enter year of manufacture (from 1950 to 2022): ";

date = 0;

while (date < 1950 || date > 2022)

{

cin >> date;

}

}

Service\_center.h

/\*

\* Заголовний файл Service\_center

\*/

#pragma once

#include <string>

#include <iostream>

#include <fstream>

using namespace std;

class Service\_center

{

protected:

string client = " "; //Ім'я та призвище клієнта

string device = " "; //Тип техніки

string manufacturer = " "; //Ім'я виробника

string model; //Ім'я моделі

int date = 0; //Дата виробництва

public:

Service\_center();

Service\_center(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date);

virtual ~Service\_center();

//Геттери класу

string GET\_device();

string GET\_manufacturer();

string GET\_model();

string GET\_client();

int GET\_date();

//Сеттери класу

void SET\_divece(string device);

void SET\_manufacturer(string manufacturer);

void SET\_model(string model);

void SET\_client(string client);

void SET\_date(int date);

//функції

virtual void print();

virtual void input();

};

Warranty\_repair.cpp

/\*

\* Реалізація класу нащадка Warranty\_repair

\*/

#include "Warranty\_repair.h"

//Конструктор

Warranty\_repair::Warranty\_repair(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, string c\_item\_code, string c\_problem) :Service\_center(c\_client, c\_device, c\_manufacturer ,c\_model, c\_date), item\_code(c\_item\_code), problem(c\_problem)

{

}

//Конструктор

Warranty\_repair::Warranty\_repair()

{

}

//Деструктор

Warranty\_repair::~Warranty\_repair()

{

}

//Реалізація сеттерів

void Warranty\_repair::SET\_item\_code(string item\_code)

{

this->item\_code = item\_code;

}

void Warranty\_repair::SET\_problem(string problem)

{

this->problem = problem;

}

//Реалізація геттерів

string Warranty\_repair::GET\_problem()

{

return problem;

}

string Warranty\_repair::GET\_item\_code()

{

return item\_code;

}

//Реалізація функції введення параметрів

void Warranty\_repair::input()

{

Service\_center::input();

cout << "Enter item code: ";

cin >> item\_code;

cout << "Enter the problem: ";

cin >> problem;

}

//Реалізація функції виведення параметрів

void Warranty\_repair::print()

{

Service\_center::print();

cout << "\nProblem: " << problem

<<"\nItem code: "<< item\_code << endl << endl;

}

//Запис даних полів класу у файловий потік виводу

ostream& operator<<(ostream& os, Warranty\_repair& data)

{

os << data.client << endl

<< data.device << endl

<< data.manufacturer << endl

<< data.model << endl

<< data.date << endl

<< data.problem << endl

<< data.item\_code << endl;

return os;

}

//Запис даних полів класу у файловий потік вводу

istream& operator>>(istream& is, Warranty\_repair& data)

{

is >> data.client >> data.device >> data.manufacturer >> data.model >> data.date >> data.problem >> data.item\_code;

return is;

}

Warranty\_repair.h

/\*

\* Заголовний файл Warranty\_repair

\*/

#pragma once

#include "Service\_center.h"

class Warranty\_repair : public Service\_center

{

private:

string item\_code; //Серійний номер

string problem; //Опис проблеми

public:

Warranty\_repair(string c\_client, string c\_device, string c\_manufacturer, string c\_model, int c\_date, string c\_item\_code, string c\_problem);

Warranty\_repair();

~Warranty\_repair();

//Сеттери класу

void SET\_item\_code(string item\_code);

void SET\_problem(string problem);

//Геттери класу

string GET\_problem();

string GET\_item\_code();

void input();

void print();

//Френдлі функції перевантаження операторів вводу та виводу

friend ostream &operator<<(ostream& os, Warranty\_repair& data);

friend istream &operator>>(istream& is, Warranty\_repair& data);

};

## ДОДАТОК Б

## Код модульних тестів

Unit\_test.cpp

/\*

\*Релізація модульного тестування Unit\_test

\*/

#include "pch.h"

#include "CppUnitTest.h"

#include "../COURSE WORK/Main.h"

#include "../COURSE WORK/Priority\_queue.h"

#include "../COURSE WORK/Regulatory\_service.h"

#include "../COURSE WORK/Service\_center.h"

#include "../COURSE WORK/Warranty\_repair.h"

#include "../COURSE WORK/Main.cpp"

#include "../COURSE WORK/Priority\_queue.cpp"

#include "../COURSE WORK/Regulatory\_service.cpp"

#include "../COURSE WORK/Service\_center.cpp"

#include "../COURSE WORK/Warranty\_repair.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Unit\_test

{

TEST\_CLASS(Unit\_test)

{

public:

// Тестування методу створення об'єкта класа Regulatory\_service

TEST\_METHOD(Test\_Regulatory\_service)

{

Priority\_queue P\_q\_test;

Regulatory\_service\* Regulatory = new Regulatory\_service("George\_Zatulovsky", "Phone", "Xiaomi", "POCO\_X3\_Pro", 2022, 5000, "Screen\_replacement");

P\_q\_test.insert\_q(Regulatory, 1);

Assert::IsTrue(P\_q\_test.print\_q());

}

// Тестування методу створення об'єкта класа Warranty\_repair

TEST\_METHOD(Test\_Warranty\_repair)

{

Priority\_queue P\_q\_test;

Warranty\_repair\* Warranty = new Warranty\_repair("George\_Zatulovsky", "Phone", "Xiaomi", "POCO\_X3\_Pro", 2022,"Burn\_battery","WWWQIJSG323RF");

P\_q\_test.insert\_q(Warranty, 1);

Assert::IsTrue(P\_q\_test.print\_q());

}

// Тестування методу підрахунку однакових об'єктів (Виробником та типом сервісу)

//Тестування методу видалення всіх об'єктів з контейнеру

TEST\_METHOD(Test\_Delete\_all)

{

Priority\_queue P\_q\_test;

Warranty\_repair\* Warranty\_1 = new Warranty\_repair("George\_Zatulovsky", "Phone", "Xiaomi", "POCO\_X3\_Pro", 2022, "Burn\_battery", "WWWQIJSG323RF");

P\_q\_test.insert\_q(Warranty\_1, 1);

Warranty\_repair\* Warranty\_2 = new Warranty\_repair("Ivan\_Ivanovich", "TV", "LG", "Priar", 2009, "Burn\_battery", "GEHHEWSD323RF");

P\_q\_test.insert\_q(Warranty\_2, 2);

Regulatory\_service\* Regulatory\_3 = new Regulatory\_service("Maxim\_Sitnik", "Camera", "Sony", "Enry", 2015, 5600, "Broke\_lens");

P\_q\_test.insert\_q(Regulatory\_3, 3);

Regulatory\_service\* Regulatory\_4 = new Regulatory\_service("Vlada\_Perun", "Phone", "Apple", "Iphone\_13", 2021, 3500, "Burn\_matrix");

P\_q\_test.insert\_q(Regulatory\_4, 4);

P\_q\_test.delete\_all\_q();

Assert::IsTrue(P\_q\_test.print\_q()==0);

}

// Тестування методу видалення першого об'єкту з контейнеру

TEST\_METHOD(Test\_Delete\_elem)

{

Priority\_queue P\_q\_test;

Warranty\_repair\* Warranty\_1 = new Warranty\_repair("George\_Zatulovsky", "Phone", "Xiaomi", "POCO\_X3\_Pro", 2022, "Burn\_battery", "WWWQIJSG323RF");

P\_q\_test.insert\_q(Warranty\_1, 1);

P\_q\_test.delete\_q();

Assert::IsTrue(P\_q\_test.print\_q() == 0);

}

// Тестування методу вмведення об'єктів у консоль

TEST\_METHOD(Test\_Print)

{

Priority\_queue P\_q\_test;

Warranty\_repair\* Warranty\_1 = new Warranty\_repair("George\_Zatulovsky", "Phone", "Xiaomi", "POCO\_X3\_Pro", 2022, "Burn\_battery", "WWWQIJSG323RF");

P\_q\_test.insert\_q(Warranty\_1, 1);

Warranty\_repair\* Warranty\_2 = new Warranty\_repair("Ivan\_Ivanovich", "TV", "LG", "Priar", 2009, "Burn\_battery", "GEHHEWSD323RF");

P\_q\_test.insert\_q(Warranty\_2, 2);

Regulatory\_service\* Regulatory\_3 = new Regulatory\_service("Maxim\_Sitnik", "Camera", "Sony", "Enry", 2015, 5600, "Broke\_lens");

P\_q\_test.insert\_q(Regulatory\_3, 3);

Regulatory\_service\* Regulatory\_4 = new Regulatory\_service("Vlada\_Perun", "Phone", "Apple", "Iphone\_13", 2021, 3500, "Burn\_matrix");

P\_q\_test.insert\_q(Regulatory\_4, 4);

Warranty\_repair\* Warranty\_5 = new Warranty\_repair("Dasha\_Litvin", "Labtop", "Xiaomi", "Redmi\_node", 2019, "Screatch\_Web-camera", "GGWEQQHQSARF");

P\_q\_test.insert\_q(Warranty\_5, 5);

Assert::IsTrue(P\_q\_test.print\_q());

}

};

}