



AGENCJA MIESZKANIOWA

Projekt zaliczeniowy C++

Autorzy

Łukasz Konieczny, Katarzyna Trzaska, Anna Raputa

WIEiK Informatyka w Inżynierii Komputerowej

—Wstęp teoretyczny—

Nasz program został stworzony z myślą o kompleksowej obsłudze agencji mieszkaniowej, a nasze cele skupiają się na analizie oraz praktycznej implementacji kluczowych operacji związanych z zarządzaniem danymi. W ramach tego projektu koncentrujemy się na przeglądaniu list klientów, sprzedawców, mieszkań, oraz zarządzaniu zamówieniami.

Wykorzystaliśmy techniki programowania obiektowego, takie jak dziedziczenie, klasy wewnętrzne i abstrakcyjne, aby modelować rzeczywiste struktury danych związane z agencją mieszkaniową. W rezultacie nasza aplikacja oferuje klarowną reprezentację poszczególnych klas, takich jak klienci, sprzedawcy, mieszkania czy zamówienia, co znacznie ułatwia operacje na danych, sprawiając, że stają się intuicyjne.

Główne funkcje naszego programu obejmują:

- Płynne przeglądanie informacji: Umożliwiamy łatwe i intuicyjne przeglądanie danych dotyczących klientów, sprzedawców, mieszkań i zamówień.
- Efektywne zarządzanie zamówieniami: Zapewniamy operacje dodawania, usuwania zamówień oraz aktualizacji ich statusów, co przyczynia się do sprawnego zarządzania procesem.

Dodatkowo, nasza aplikacja opiera się na danych z plików:

- class Klient.txt: Lista klientów agencji mieszkaniowej.
- class Sprzedawca.txt: Lista sprzedawców agencji mieszkaniowej.
- class Mieszkanie.txt: Lista mieszkań oferowanych przez agencję mieszkaniową.
- class Zamowienie.txt: Lista zamówień (w trakcie realizacji oraz zrealizowanych).

To umożliwia elastyczne zarządzanie danymi i łatwą integrację z istniejącymi zbiorami informacji agencji mieszkaniowej.

Dodatkowo, nasza aplikacja została wzbogacona o funkcjonalność raportowania z działania programu. Aktywnie spisuje raporty, gromadząc istotne informacje na temat operacji wykonywanych w aplikacji. Te raporty nie tylko umożliwiają śledzenie bieżących działań, ale także stanowią wartościowe narzędzie do analizy danych oraz oceny efektywności agencji mieszkaniowej. Wszystkie te informacje są zapisywane w pliku log.txt.

—Opis implementacji z fragmentami kodu Źródłowego—

Nasz program został zorganizowany w kilka kluczowych klas, z których każda pełni określoną rolę w strukturze projektu:

Klasy reprezentujące dane:

Klient, Sprzedawca, Mieszkanie, oraz Zamowienie odpowiadają za reprezentację różnych rodzajów danych związanych z agencją mieszkaniową. Dane w każdej klasie są enkapsulowane za pomocą getterów i setterów, dzięki czemu nie zostaną one zmienione przez przypadek, lub niezgodnie z wymaganiami.

Klasa zarządzająca bazą danych (BazaDanych):

BazaDanych umożliwia efektywne zarządzanie danymi w programie, obsługując obiekty różnych typów. Klasy dziedziczące po *KlasaBazowa* muszą posiadać metody `serialize` i `deserialize`, co pozwala na skuteczny zapis i odczyt obiektów z pliku.

KlasaBazowa:

Jest klasą abstrakcyjną, nie pozwalającą na bezpośrednie tworzenie obiektów. Wymusza posiadanie numeru identyfikacyjnego (ID) oraz implementację metod:

- **toString** służy do czytelnego wypisania zawartości klasy,
- **serialize** przekształca dane klasy na string,
- **validate** pozwala na sprawdzenie poprawności przekazywanych danych.

To sprawia, że klasy dziedziczące po *KlasaBazowa* są jednolite i spójne w strukturze.

```
#pragma once
#include <string>
#include <regex>
class KlasaBazowa
{
protected:
    //Domyślnie id jest ustawione na -1, aby program wiedział, że obiekt nie ma jeszcze przyznanego unikatowego id
    long id = -1;
public:
    //Metody które mają nie zmieniać wartości zmiennych, można oznaczyć jako "const" (wtedy kompilator nie pozwoli im zmieniać wartości zmiennych)
    long getId() const;
    void setId(int noweId);

    //dalam to do sprawdzania poprawności zmiennych żeby nie trzeba było powtarzać w każdej klasie
    bool validate(std::string input, std::regex pattern);

    //Metoda abstrakcyjna (wirtualna), klasy które dziedziczą po "KlasaBazowa" muszą ją zaimplementować, a ona sama domyślnie nie ma implementacji
    virtual std::string toString() = 0;

    //Metoda do zamiany klasy na format, który damy radę odczytać z pliku
    virtual std::string serialize() = 0;
};
```

```
#pragma once
#include "KlasaBazowa.h"
class Klient : public KlasaBazowa
{
private:
    std::string imie;
    std::string nazwisko;
public:
    Klient(std::string imie, std::string nazwisko);

    std::string getImie() const;
    std::string getNazwisko() const;
    void setImie(std::string noweImie);
    void setNazwisko(std::string noweNazwisko);

    std::string toString() override;
    std::string serialize() override;

    static Klient& deserialize(std::string input);
};
```

Klasa obsługująca logowanie (Logger):

Logger służy do tworzenia i zapisywania raportu z działania programu. Posiada statyczną metodę log, umożliwiającą łatwe dodawanie nowych wpisów do raportu w dowolnym miejscu projektu.

```
#pragma once
#include <string>

class Logger {
private:
    static std::string raport;
public:
    static void log(const std::string& powiadomienie);
    static void flush();
};
```

```
#include "includes/Logger.h"
#include <fstream>

std::string Logger::raport = "";

void Logger::log(const std::string& powiadomienie)
{
    raport += powiadomienie + "\n";
}

void Logger::flush() {
    std::ofstream plik("log.txt");
    plik << raport;
    plik.close();
}
```

Klasy spinające całość (Menu oraz ProjektS1):

Menu i ProjektS1 odpowiadają za interakcję z użytkownikiem, umożliwiając przeglądanie zawartości bazy danych oraz zarządzanie zamówieniami za pomocą pętli i instrukcji switch-case.

Dodatkowo, klasy "bazodanowe" dziedziczące po *KlasaBazowa* posiadają metodę "deserialize", która umożliwia tworzenie nowych obiektów na podstawie stringa uzyskanego z metody "serialize". To kluczowe dla *BazaDanych*, która używa tych metod do zapisu i odczytu obiektów z plików. Odczyt i zapis z plików są zapewniane przez klasę *BazaDanych*, opartą na szablonach, umożliwiającą zarządzanie obiektami dowolnego typu. Sposób zarządzania obiektami w *BazaDanych* pozwala na proste rozszerzanie funkcjonalności zapisu i odczytu o nowe klasy.

Wszystkie te elementy są spinane w funkcjonalną całość przez klasę Menu, która pozwala użytkownikowi na przeglądanie zawartości bazy danych oraz zarządzanie zamówieniami.

—Zrzuty ekranu z działania programu—

```
C:\Users\lukim\source\repos\ X + v

|      |MENU GŁÓWNE|      |
[1]Wyświetl kategorie z bazy danych
[2]Operacje na zamówieniach
[3]Wyjście

Wybierz jedną z opcji podanych [1-3]
|
```

```
C:\Users\lukim\source\repos\ X + v

OPCJE KATEGORII DO ODCZYTU
[1]Wyświetl Klientów
[2]Wyświetl Sprzedawców
[3]Wyświetl Mieszkania
[4]Wyświetl Zamówienia
[5]Powrót do Menu Głównego

Wybierz jedną z opcji (1-5): |
```

```
C:\Users\lukim\source\repos\ X + v - □ X

WYŚWIETLANIE LISTY SPRZEDAWCÓW
ID Imię Nazwisko Email Numer telefonu
1 Artur Krawczyk artur.krawczyk@example.com 123456789
2 Kinga Malinowska kinga.malinowska@example.com 987654321
3 Damian Nowicki damian.nowicki@example.com 456789012
4 Aleksandra Piotrowska aleksandra.piotrowska@example.com 789012345
5 Rafał Kowal rafal.kowal@example.com 234567890
6 Julia Duda julia.duda@example.com 567890123
7 Adrian Zajac adrian.zajac@example.com 012345678
8 Dominika Szewczyk dominika.szewczyk@example.com 890123456
9 Michał Wójcik michal.wojcik@example.com 345678901
10 Kamila Nowakowska kamila.nowakowska@example.com 678901234

OPCJE KATEGORII DO ODCZYTU
[1]Wyświetl Klientów
[2]Wyświetl Sprzedawców
[3]Wyświetl Mieszkania
[4]Wyświetl Zamówienia
[5]Powrót do Menu Głównego

Wybierz jedną z opcji (1-5): |
```

```
C:\Users\lukim\source\repos\ X + v
|      |MENU GŁÓWNE|      |
[1]Wyświetl kategorie z bazy danych
[2]Operacje na zamówieniach
[3]Wyjście

Wybierz jedną z opcji podanych [1-3]
|
```

```
C:\Users\lukim\source\repos\ X + v
OPERACJE NA ZAMÓWIENIACH
[1]Dodawanie Zamówienia
[2]Usuwanie Zamówienia
[3]Zmiana Statusu Zamówienia
[4]Powrót do Menu Głównego

Wybierz jedną z opcji podanych (1-4)
|
```

C:\Users\lukim\source\repos\ X + v

DODAWANIE ZAMÓWIENIA

Podaj Status Zamówienia

Podaj numer od 0 do 1

[0]-W Trakcie Zamówienia

[1]-Zamówione

0

Podaj Id Mieszkania

2

Podaj Id Sprzedawcy

3

Podaj Id Klienta

8

Dodano nowe zamówienie

OPERACJE NA ZAMÓWIENIACH

[1]Dodawanie Zamówienia

[2]Usuwanie Zamówienia

[3]Zmiana Statusu Zamówienia

[4]Powrót do Menu Głównego

Wybierz jedną z opcji podanych (1-4)

|

```
C:\Users\lukim\source\repos\ X + v

ZMIANA STATUSU ZAMÓWIENIA
Podaj Id Zamówienia, którego status chcesz zmienić
8
Takiego zamówienia nie ma w bazie, podaj poprawne ID
Podaj Id Zamówienia, którego status chcesz zmienić
2
Podaj Status Zamówienia

Podaj Numer od 0 do 1
[0]-W Trakcie Zamówienia
[1]-Zamówione
0
Status został zaktualizowany
OPERACJE NA ZAMÓWIENIACH
[1]Dodawanie Zamówienia
[2]Usuwanie Zamówienia
[3]Zmiana Statusu Zamówienia
[4]Powrót do Menu Głównego

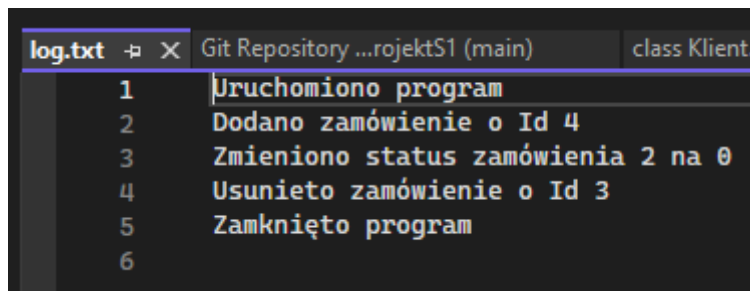
Wybierz jedną z opcji podanych (1-4)
|
```

```
C:\Users\lukim\source\repos\ X + v

USUWANIE ZAMÓWIENIA
Podaj Id Zamówienia, które chcesz usunąć
5
Takiego zamówienia nie ma w bazie, podaj poprawne ID
Podaj Id Zamówienia, które chcesz usunąć
3
Zamowienie zostało usunięte
OPERACJE NA ZAMÓWIENIACH
[1]Dodawanie Zamówienia
[2]Usuwanie Zamówienia
[3]Zmiana Statusu Zamówienia
[4]Powrót do Menu Głównego

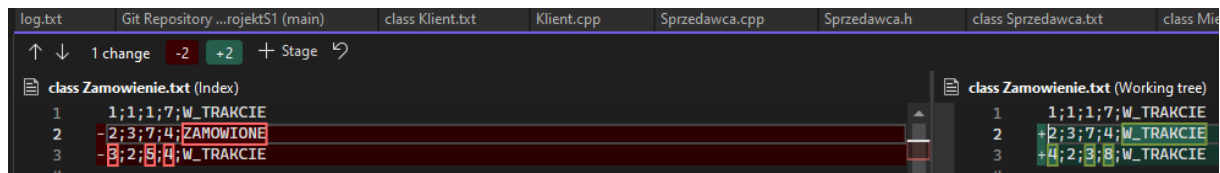
Wybierz jedną z opcji podanych (1-4)
|
```


Raport z działania:



```
log.txt  Git Repository ...rojektS1 (main)  class Klient.txt
1      Uruchomiono program
2      Dodano zamówienie o Id 4
3      Zmieniono status zamówienia 2 na 0
4      Usunięto zamówienie o Id 3
5      Zamknięto program
6
```

Wynik zarządzania zamówieniami:



```
log.txt  Git Repository ...rojektS1 (main)  class Klient.txt  Klient.cpp  Sprzedawca.cpp  Sprzedawca.h  class Sprzedawca.txt  class Mis
↑ ↓ 1 change -2 +2 + Stage ↺
class Zamowienie.txt (Index)
1      1;1;1;7;W_TRAKCIE
2      -2;3;7;4;ZAMOWIONE
3      -3;2;8;4;W_TRAKCIE
class Zamowienie.txt (Working tree)
1      1;1;1;7;W_TRAKCIE
2      +2;3;7;4;W_TRAKCIE
3      +4;2;3;8;W_TRAKCIE
```

—Wnioski—

Program agencji mieszkaniowej jest rezultatem określonych wymagań, skupiających się na zaawansowanych technikach programowania obiektowego. Kod aplikacji obejmuje dziedziczenie, klasy wewnętrzne i abstrakcyjne.

Warto podkreślić, że nasza aplikacja jest przygotowana do dalszego rozwoju. Dzięki elastycznej architekturze możliwe jest łatwe rozszerzanie funkcjonalności o zarządzanie innymi rodzajami danych czy dodatkowe operacje. Aplikacja stanowi solidny fundament gotowy na ewolucję zgodnie z rozwojem agencji mieszkaniowej czy zmieniającymi się potrzebami użytkowników.

Tworząc program zarządzający mieszkaniami, zdobyliśmy praktyczne doświadczenie w wykorzystywaniu kluczowych założeń programowania obiektowego, takich jak dziedziczenie, klasy abstrakcyjne i szablony. To cenne doświadczenie, które przyczyniło się do naszego rozwoju jako programistów.