

WZORCE PROJEKTOWE

SPRAWOZDANIE

ZADANIE ADAPTER BIBLIOTEKA LOGOWANIA

Patryk Figas
Informatyka, programowanie
Grupa 34_Inf_P_NW_6

1. Cel

Dokument powstał w celu przedstawienia sposobu integracji nowej biblioteki logowania z istniejącym systemem, który używa innego, starszego interfejsu logowania.

W ramach ćwiczenia zaimplementowano wzorzec projektowy **Adapter**, który umożliwia zachowanie istniejącego kodu aplikacji, a jednocześnie pozwala korzystać z nowej biblioteki bez jej bezpośredniego modyfikowania.

W ramach ćwiczenia zaprojektowano interfejsy i klasy za pomocą „pseudokodu”, diagramu UML i implementacji klas do programu oraz użycie jej w programie Main.

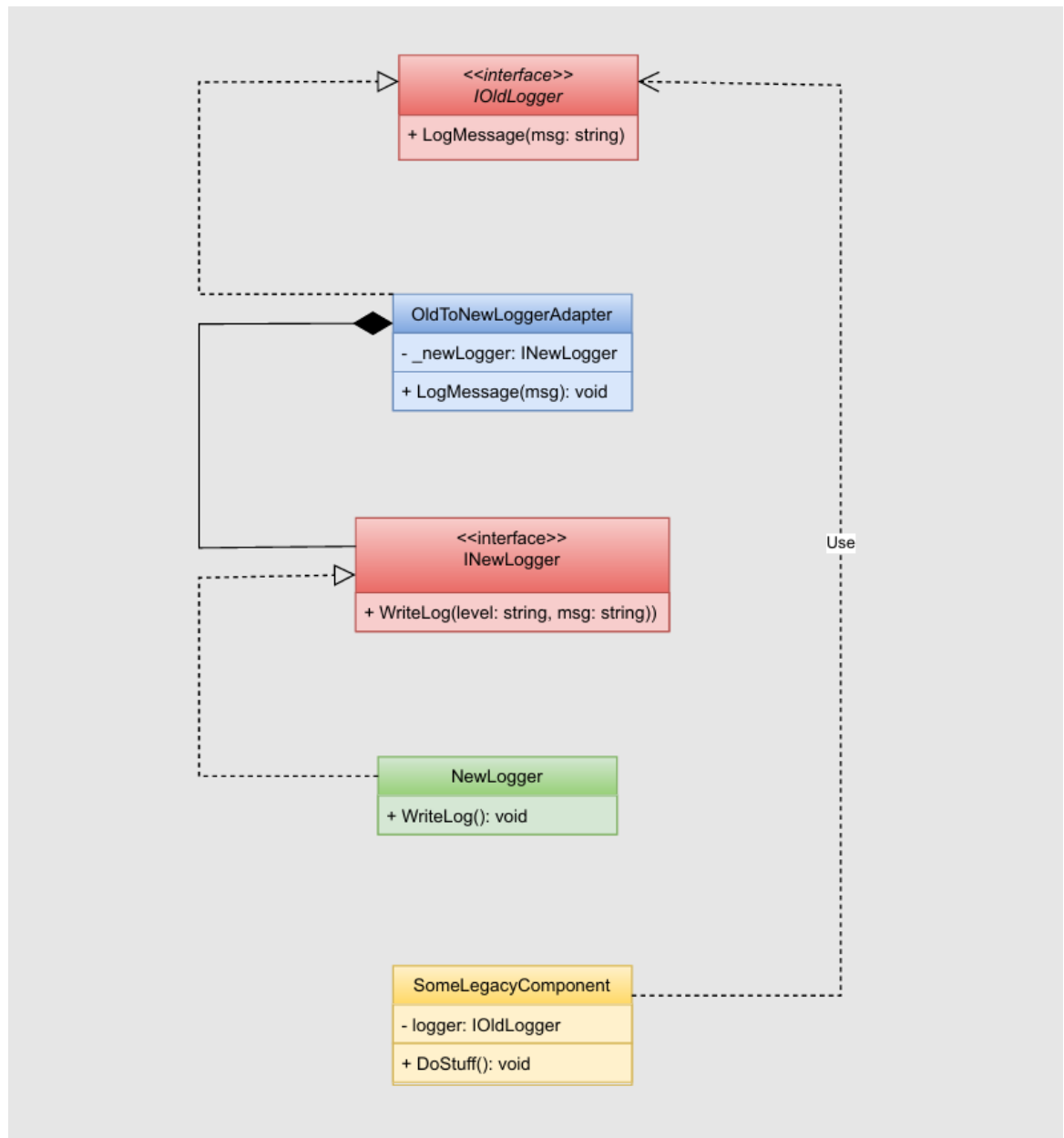
W ramach ćwiczenia zaimplementowano wzorzec projektowy **Adapter**, który umożliwia zachowanie istniejącego kodu aplikacji, a jednocześnie pozwala korzystać z nowej biblioteki bez jej bezpośredniego modyfikowania.

2. Opis rozwiązania

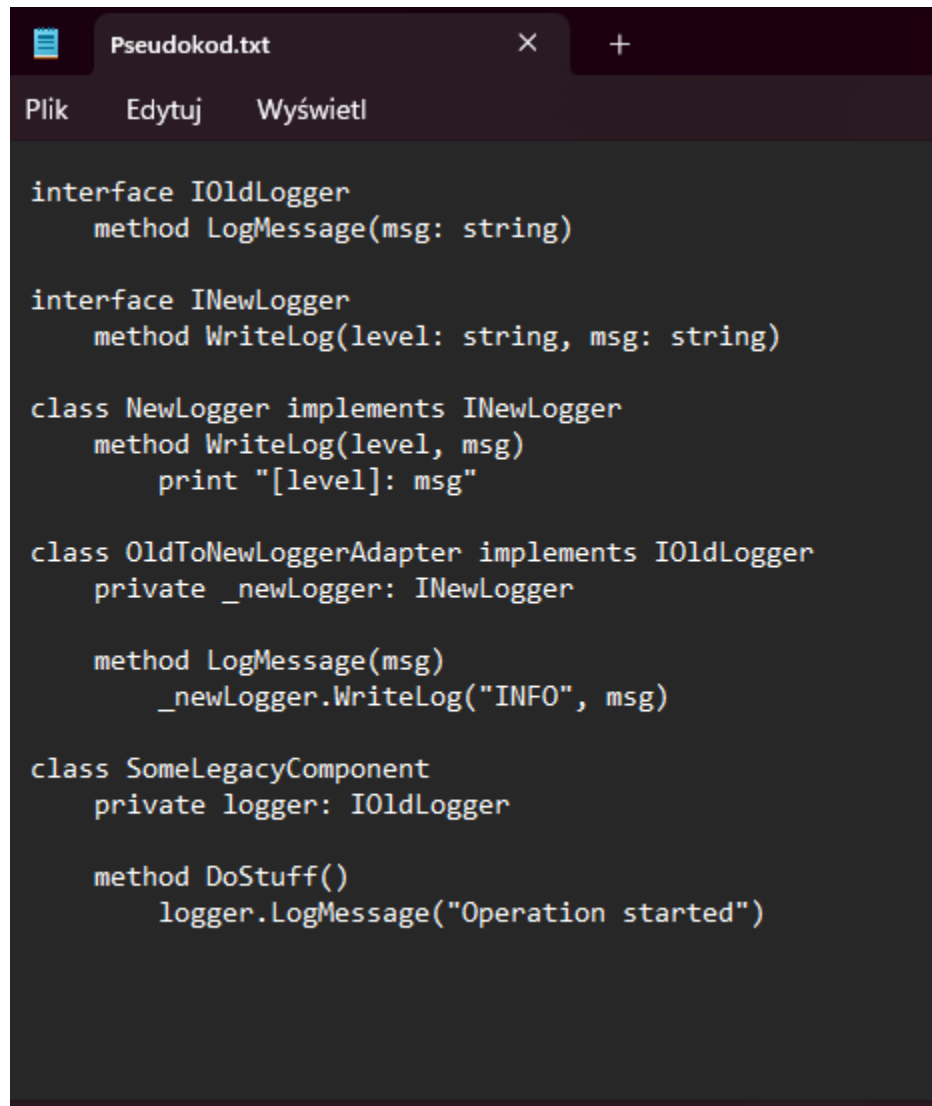
W istniejącym systemie funkcjonował interfejs **IOldLogger** z metodą **LogMessage(string)**. Nowa biblioteka logowania operuje na interfejsie **INewLogger**, który wymaga metody **WriteLog(string level, string message)**.

Zamiast zmieniać całą aplikację, zastosowano klasę adaptera **OldToNewLoggerAdapter**, która implementuje stary interfejs i wewnętrznie deleguje wywołania do nowej biblioteki. Dzięki temu możliwe jest stopniowe przejście na nowy system bez ryzyka błędów i naruszania istniejącego kodu.

- Propozycja diagramu klas



- Pseudokod



The image shows a code editor window titled "Pseudokod.txt" with a menu bar containing "Plik", "Edytuj", and "Wyświetl". The editor contains the following pseudocode:

```
interface IOldLogger
    method LogMessage(msg: string)

interface INewLogger
    method WriteLog(level: string, msg: string)

class NewLogger implements INewLogger
    method WriteLog(level, msg)
        print "[level]: msg"

class OldToNewLoggerAdapter implements IOldLogger
    private _newLogger: INewLogger

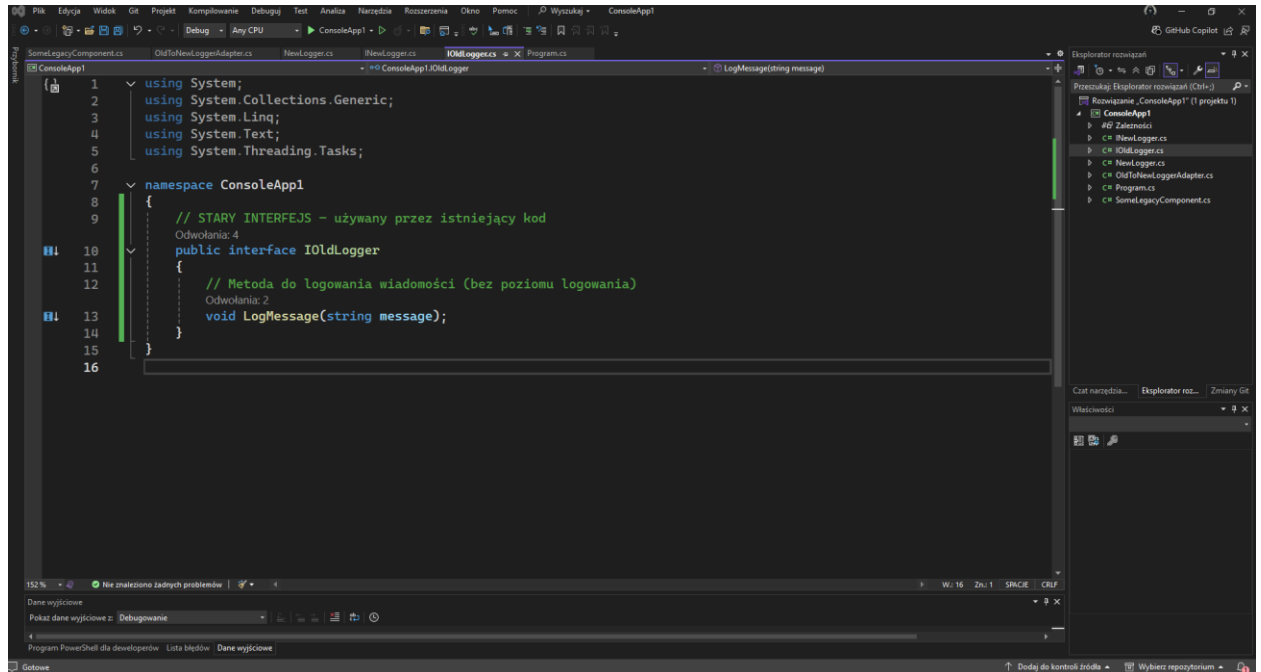
    method LogMessage(msg)
        _newLogger.WriteLog("INFO", msg)

class SomeLegacyComponent
    private logger: IOldLogger

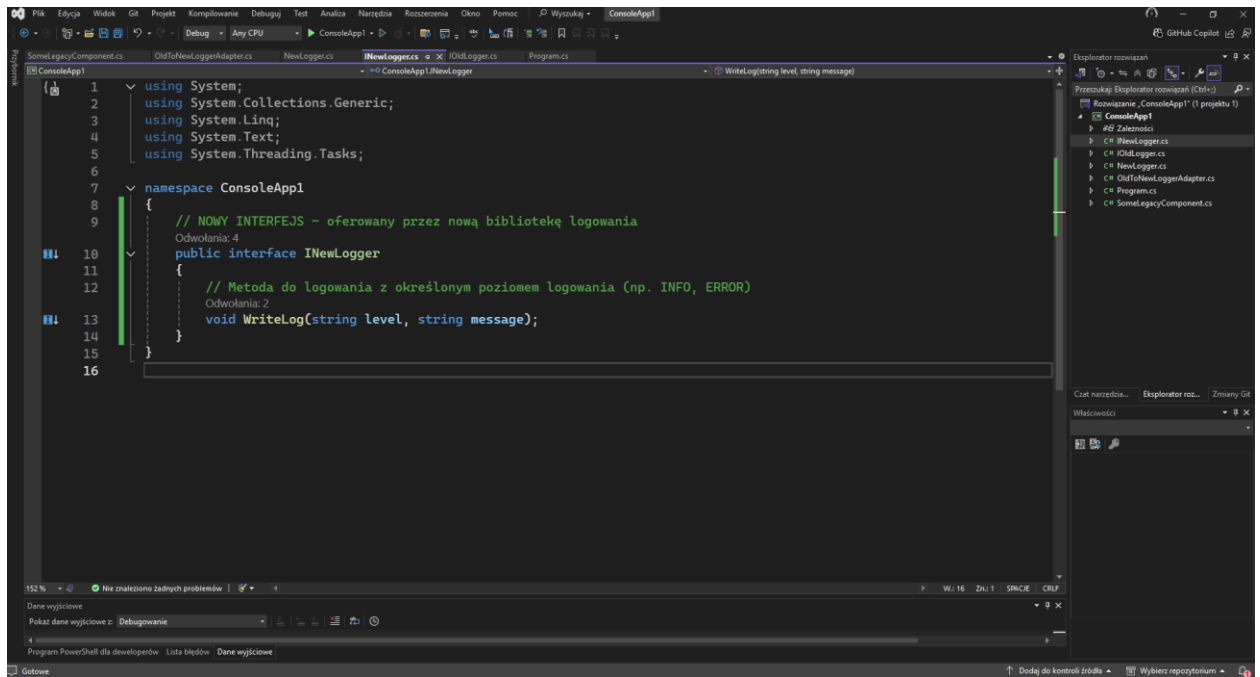
    method DoStuff()
        logger.LogMessage("Operation started")
```

3. Implementacja

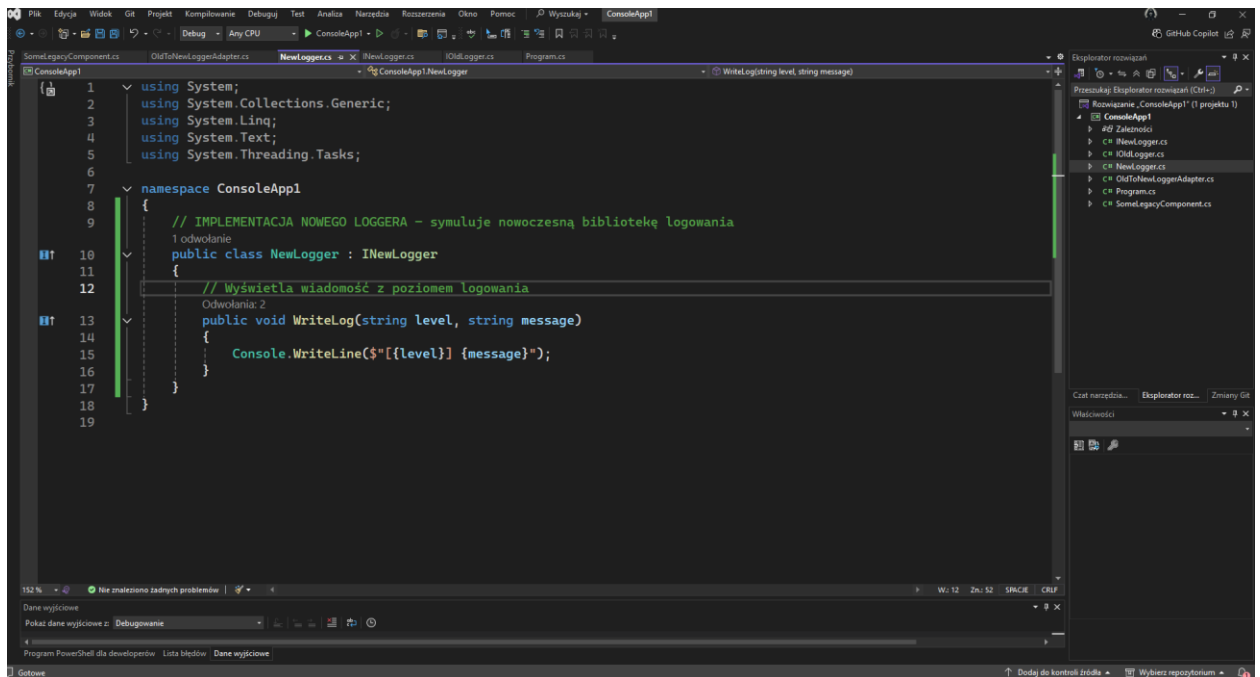
- Kod interfejsu `IOldLogger.cs`



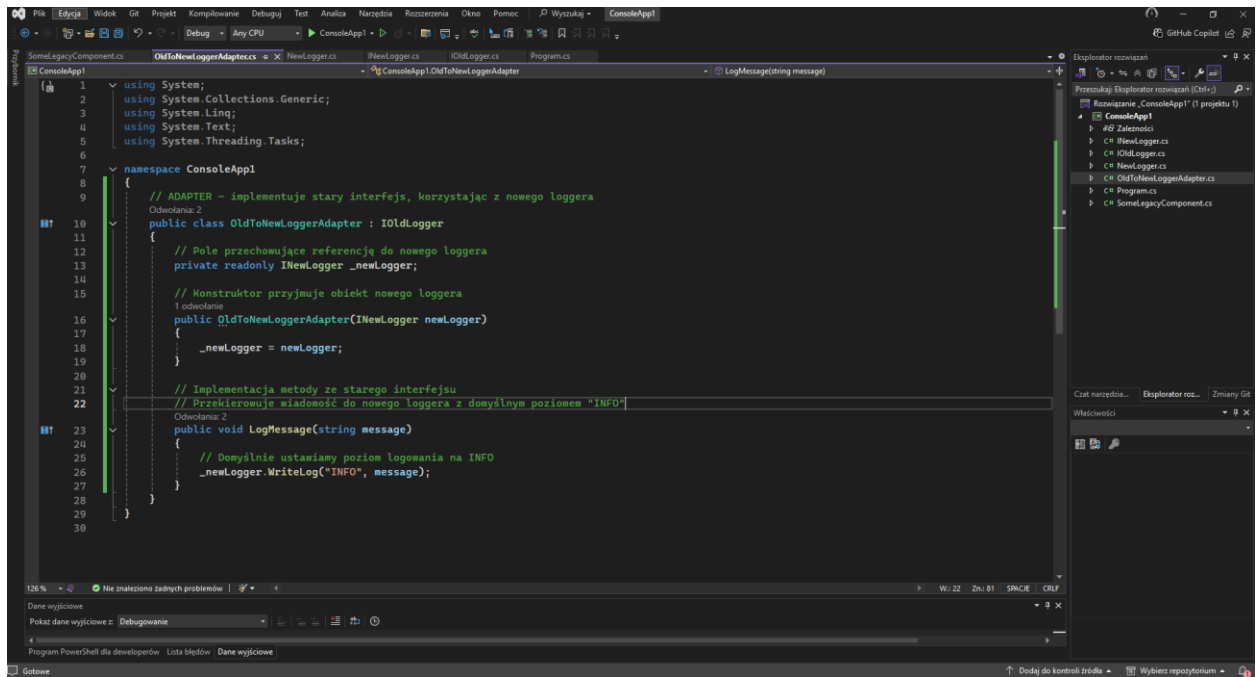
- Kod interfejsu `INewLogger.cs`



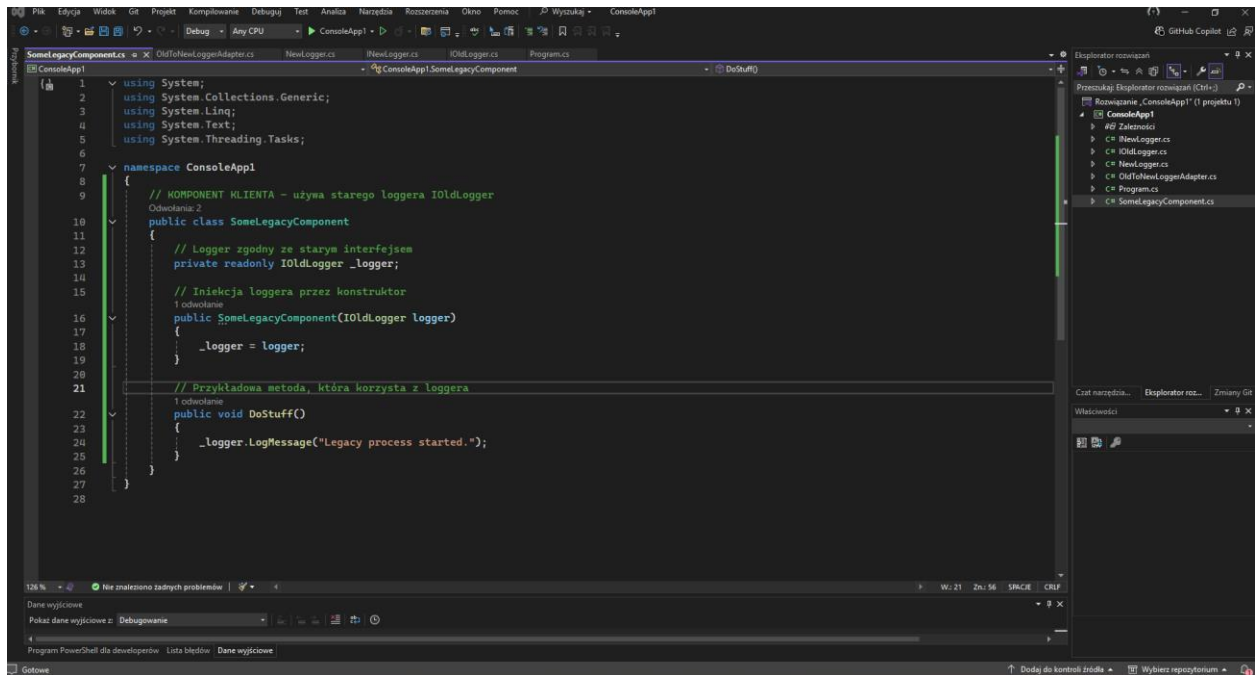
- Kod klasy `NewLogger.cs`



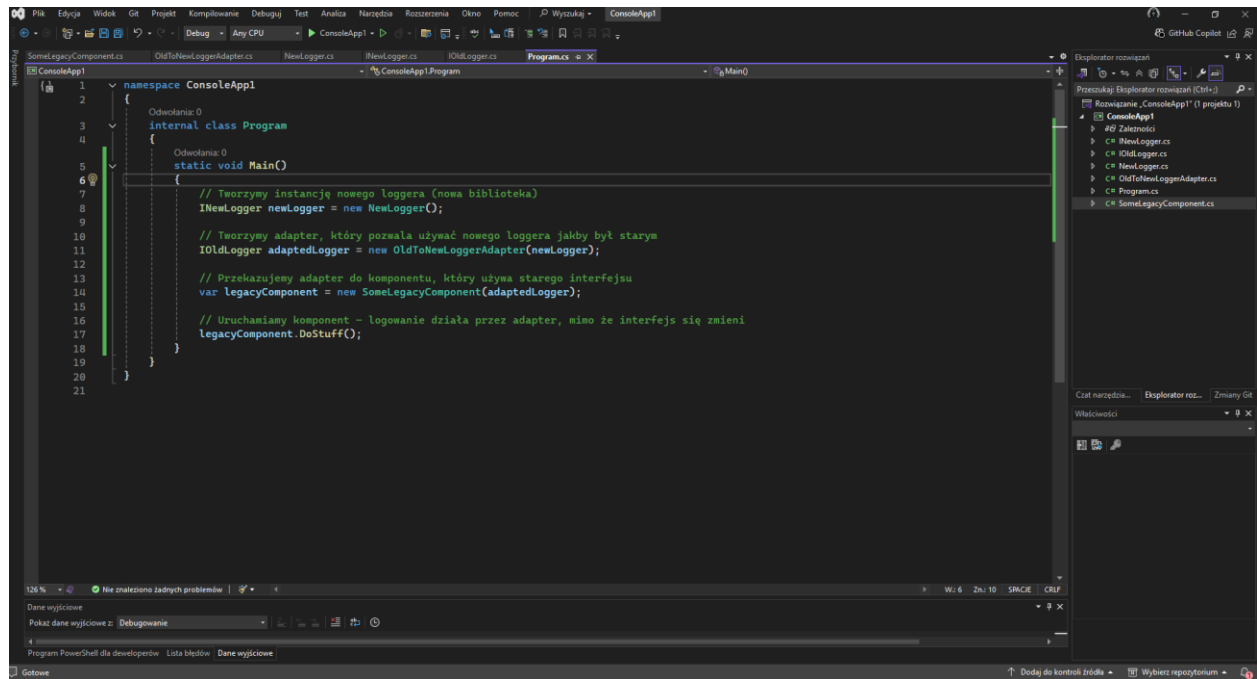
- Kod klasy OldToNewLoggerAdapter.cs



- Kod klasy SomeLegacyComponent.cs



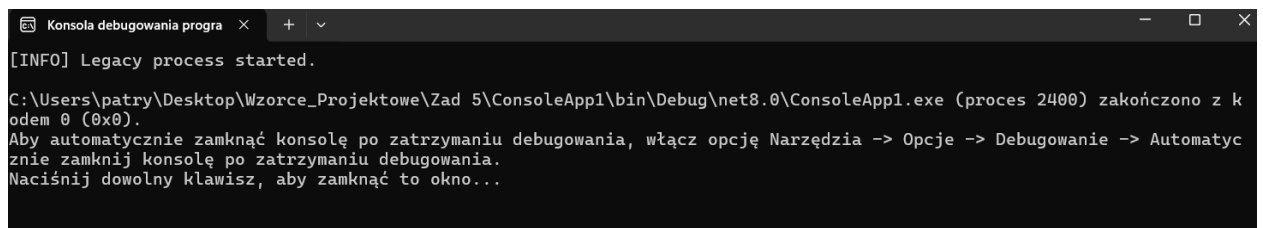
- Kod klasy Program.cs



4. Podsumowanie

W rozwiązaniu zastosowano wzorec projektowy **Adapter**, ponieważ idealnie odpowiadał on na potrzebę połączenia dwóch niespójnych interfejsów – nowego i starego loggera. Adapter pozwolił utrzymać działający kod klienta bez jego modyfikacji, co znacznie ułatwia migrację systemu.

Nowa biblioteka logowania została zintegrowana bez naruszania istniejącej architektury. Kod stał się bardziej elastyczny i zgodny z zasadą **Open/Closed** z zasad SOLID – można go rozszerzyć bez ingerencji w już działające komponenty.



```
Konsola debugowania progra x + v
[INFO] Legacy process started.
C:\Users\patry\Desktop\Wzorce_Projektowe\Zad 5\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe (proces 2400) zakończono z k
odem 0 (0x0).
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatyc
znie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Alternatywne wzorce, takie jak **Fasada** czy **Dekorator**, nie byłyby tu odpowiednie, ponieważ nie chodziło o uproszczenie interfejsu ani o dodanie funkcjonalności, lecz o dostosowanie interfejsu. Dlatego wzorec **Adapter** był najlepszym wyborem.

Lista załączników

Repozytorium GITHUB z projektem:

<https://github.com/PatrykFigas/Wzorce-projektowe.git>