

WZORCE PROJEKTOWE

SPRAWOZDANIE

ZADANIE SINGLETON/1

Patryk Figas
Informatyka, programowanie
Grupa 34_Inf_P_NW_6

1. Cel

Dokument powstał w celu przedstawienia rozwiązania zadania polegającego na implementacji klasy do logowania komunikatów z zastosowaniem wzorca projektowego Singleton.

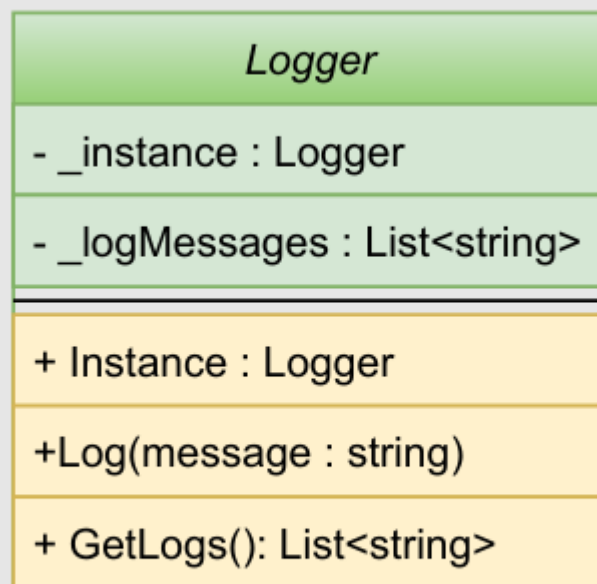
W ramach ćwiczenia zaprojektowano klasę za pomocą „pseudokodu”, diagramu UML i implementacji klasy do programu oraz użycie jej w programie Main.

Zastosowano wzorzec Singleton, który zapewnia, że klasa ma tylko jedną instancję i pozwala z niej korzystać z poziomu całego programu (static).

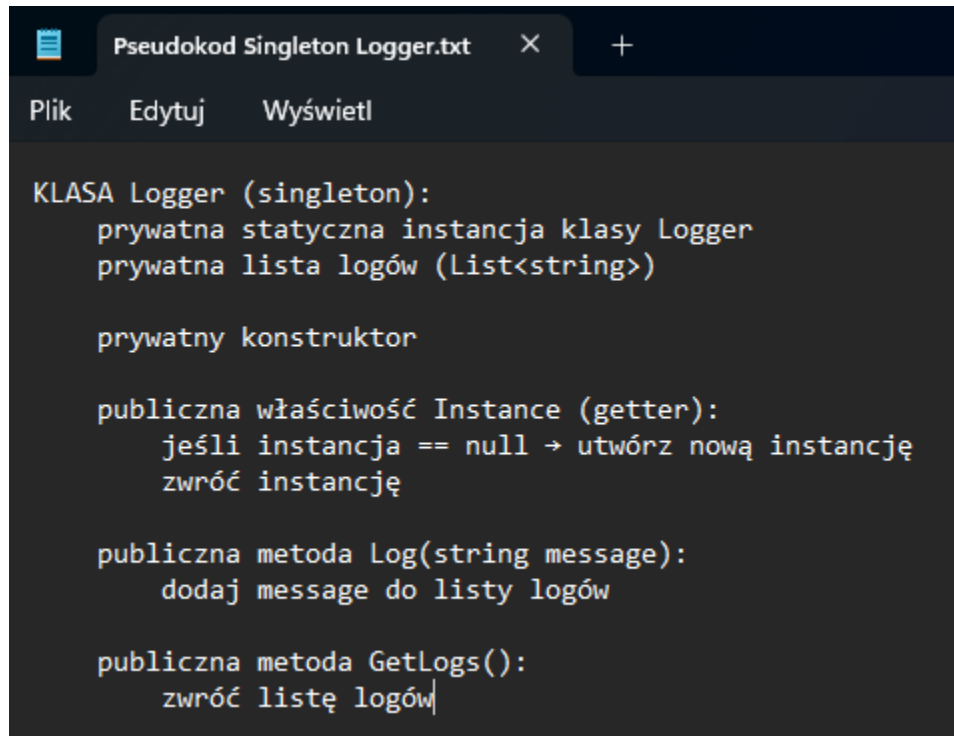
2. Opis rozwiązania

W ramach zadania zaprojektowano klasę **Logger**, której celem jest rejestrowanie komunikatów w aplikacji. Klasa została zaimplementowana zgodnie ze wzorcem projektowym **Singleton**, co oznacza, że przez cały czas działania programu istnieje tylko jedna instancja tej klasy. Dzięki temu wszystkie logi są centralnie gromadzone i dostępne z każdego miejsca w aplikacji. Komunikaty są przechowywane w liście typu **List<string>**, a ich dodawanie odbywa się za pomocą metody **Log()**.

- diagram klasy **Logger**



- pseudokod klasy **Logger**

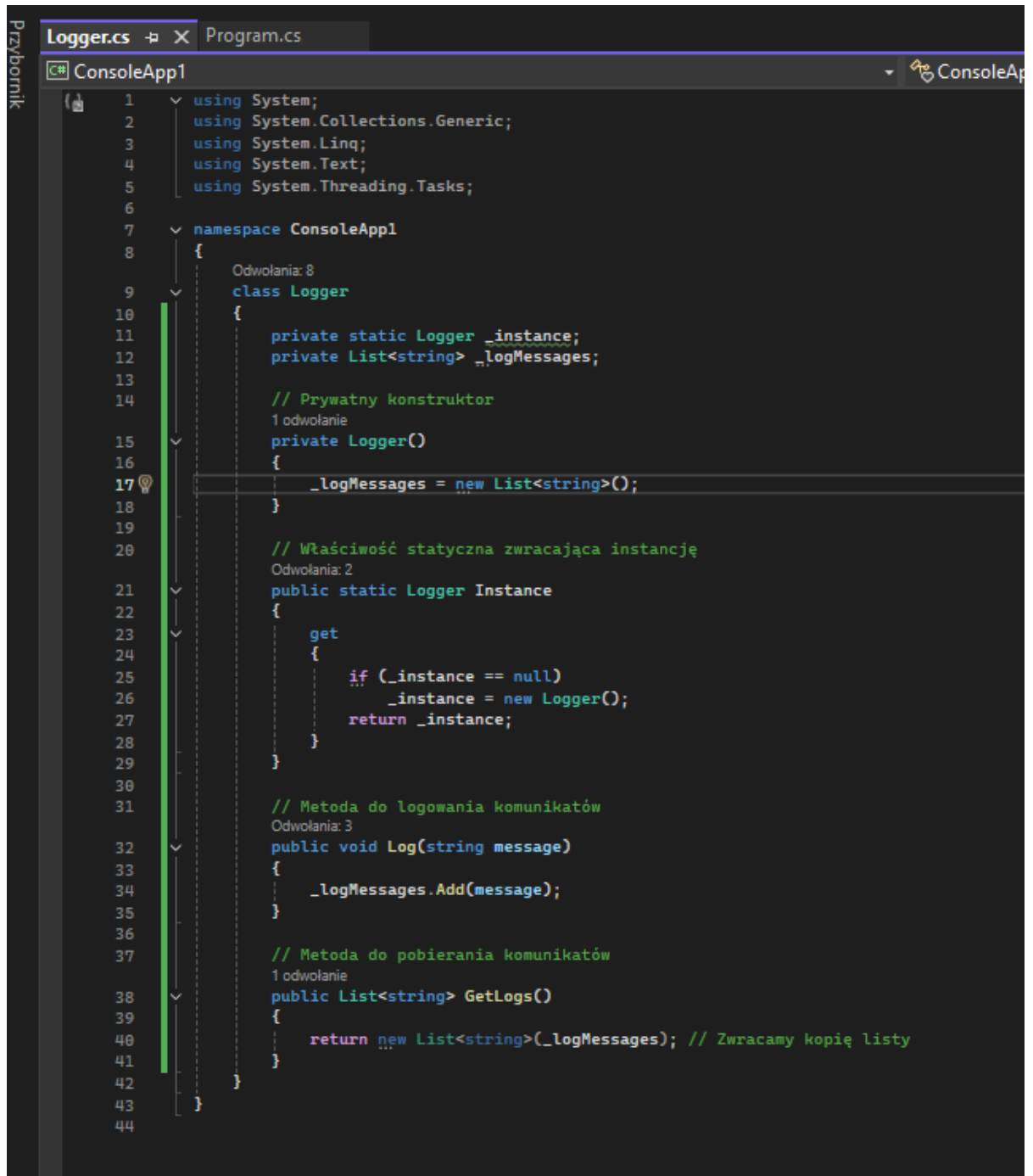


The image shows a code editor window with a dark theme. The title bar at the top reads "Pseudokod Singleton Logger.txt" and includes a close button (X) and a new file button (+). Below the title bar is a menu bar with three items: "Plik", "Edytuj", and "Wyświetl". The main area of the editor contains pseudocode for a class named "Logger" implemented as a singleton. The code is written in a light-colored font on a dark background. It defines a static instance, a private list of logs, a private constructor, a public static instance property with a getter, a public log method, and a public getlogs method.

```
KLASA Logger (singleton):  
    prywatna statyczna instancja klasy Logger  
    prywatna lista logów (List<string>)  
  
    prywatny konstruktor  
  
    publiczna właściwość Instance (getter):  
        jeśli instancja == null → utwórz nową instancję  
        zwróć instancję  
  
    publiczna metoda Log(string message):  
        dodaj message do listy logów  
  
    publiczna metoda Getlogs():  
        zwróć listę logów
```

3. Implementacja

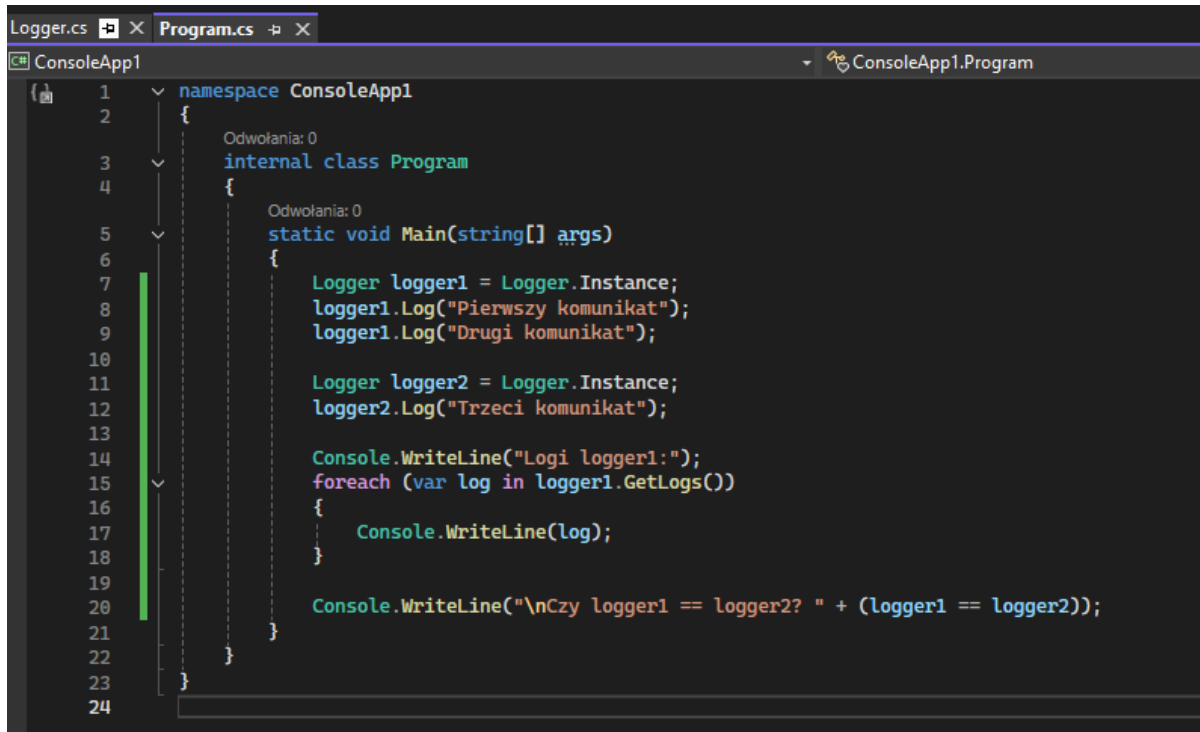
- kod klasy **Logger**:



The screenshot shows a Visual Studio editor window with two tabs: **Logger.cs** (active) and **Program.cs**. The **Logger.cs** file is part of a project named **ConsoleApp1**. The code defines a **Logger** class within the **ConsoleApp1** namespace. The class includes several static members and methods:

- Imports:** `using System;`, `using System.Collections.Generic;`, `using System.Linq;`, `using System.Text;`, and `using System.Threading.Tasks;`
- Namespace:** `namespace ConsoleApp1`
- Class:** `class Logger`
- Private Static Members:**
 - `private static Logger _instance;`
 - `private List<string> _logMessages;`
- Private Constructor:** `private Logger()` (marked with a comment: `// Prywatny konstruktor`). It initializes `_logMessages` with `new List<string>()`.
- Static Property:** `public static Logger Instance` (marked with a comment: `// Właściwość statyczna zwracająca instancję`). It has a `get` accessor that checks if `_instance` is null and creates a new `Logger` instance if necessary.
- Public Method:** `public void Log(string message)` (marked with a comment: `// Metoda do logowania komunikatów`). It adds the message to `_logMessages` using `Add`.
- Public Method:** `public List<string> GetLogs()` (marked with a comment: `// Metoda do pobierania komunikatów`). It returns a new `List<string>` containing a copy of `_logMessages` using `new List<string>(_logMessages)`.

- kod klasy **Program**



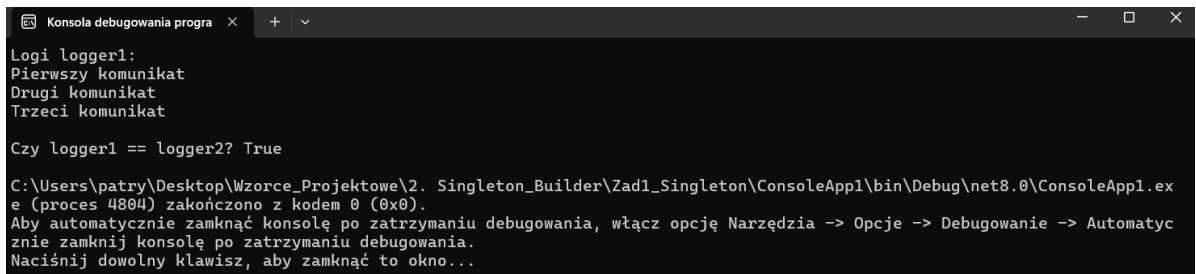
```
1 namespace ConsoleApp1
2 {
3     Odwołania: 0
4     internal class Program
5     {
6         Odwołania: 0
7         static void Main(string[] args)
8         {
9             Logger logger1 = Logger.Instance;
10            logger1.Log("Pierwszy komunikat");
11            logger1.Log("Drugi komunikat");
12
13            Logger logger2 = Logger.Instance;
14            logger2.Log("Trzeci komunikat");
15
16            Console.WriteLine("Logi logger1:");
17            foreach (var log in logger1.GetLogs())
18            {
19                Console.WriteLine(log);
20            }
21
22            Console.WriteLine("\nCzy logger1 == logger2? " + (logger1 == logger2));
23        }
24    }
```

4. Podsumowanie

W zadaniu zastosowano wzorec projektowy **Singleton**, ponieważ jego głównym celem jest zapewnienie, że dana klasa posiada tylko jedną instancję w całym programie. Jest to przydatne w przypadku klasy **Logger**, której instancja powinna być wspólna dla całej aplikacji, aby wszystkie logi były przechowywane w jednym miejscu.

Implementacja wzorca Singleton powiodła się, ponieważ:

- wszystkie odwołania do **Logger.Instance** zwracają tę samą instancję,
- rejestrowane komunikaty są przechowywane w jednej liście,
- test wykazał, że **logger1 == logger2**, co potwierdza jednolitość instancji.



```
Konsola debugowania progra
Logi logger1:
Pierwszy komunikat
Drugi komunikat
Trzeci komunikat

Czy logger1 == logger2? True

C:\Users\patry\Desktop\Wzorce_Projektowe\2. Singleton_Builder\Zad1_Singleton\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe (proces 4804) zakończono z kodem 0 (0x0).
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zaproponowany wzorec jest w tym przypadku najbardziej odpowiedni, ponieważ: inne wzorce, takie jak **Factory Method** czy **Prototype**, nie pasują do sytuacji, w której potrzebujemy jednej instancji. Z tego powodu wzorec Singleton jest najlepszym wyborem do realizacji tego typu zadania.

Lista załączników

Repozytorium GITHUB z projektem:

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