

WZORCE PROJEKTOWE

# SPRAWOZDANIE

ZADANIE CHAIN OF RESPONSIBILITY SYSTEM ZARZĄDZANIA BILETAMI

Patryk Figas  
Informatyka, programowanie  
Grupa 34\_Inf\_P\_NW\_6

## 1. Cel

Celem niniejszego dokumentu jest przedstawienie rozwiązania problemu obsługi zgłoszeń (tzw. „ticketów”) w systemie wsparcia technicznego poprzez zastosowanie wzorca projektowego **Chain of Responsibility**.

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

Zastosowano wzorec **łańcuch odpowiedzialności (Chain of Responsibility)**.

## 2. Opis rozwiązania

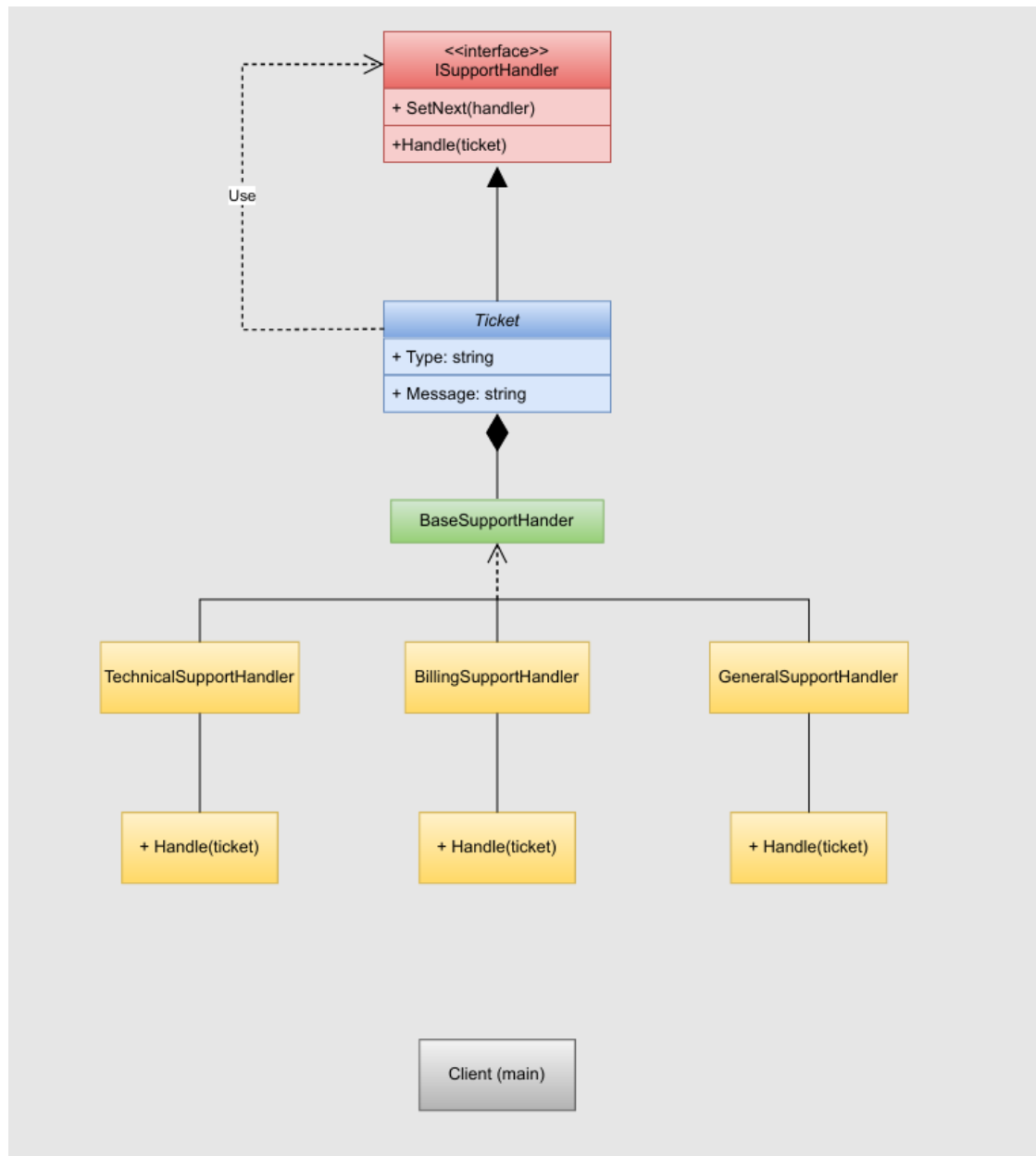
Rozwiązanie opiera się na przekazywaniu zgłoszeń między kolejnymi handlerami (klasami obsługi) do momentu, aż jeden z nich rozpozna typ zgłoszenia i je obsłuży. Każdy handler implementuje wspólny **interfejs `ISupportHandler`**, który zapewnia metodę **`Handle()`** oraz **`SetNext()`**, pozwalającą ustawić kolejnego handlera w łańcuchu.

System obsługuje trzy główne typy zgłoszeń:

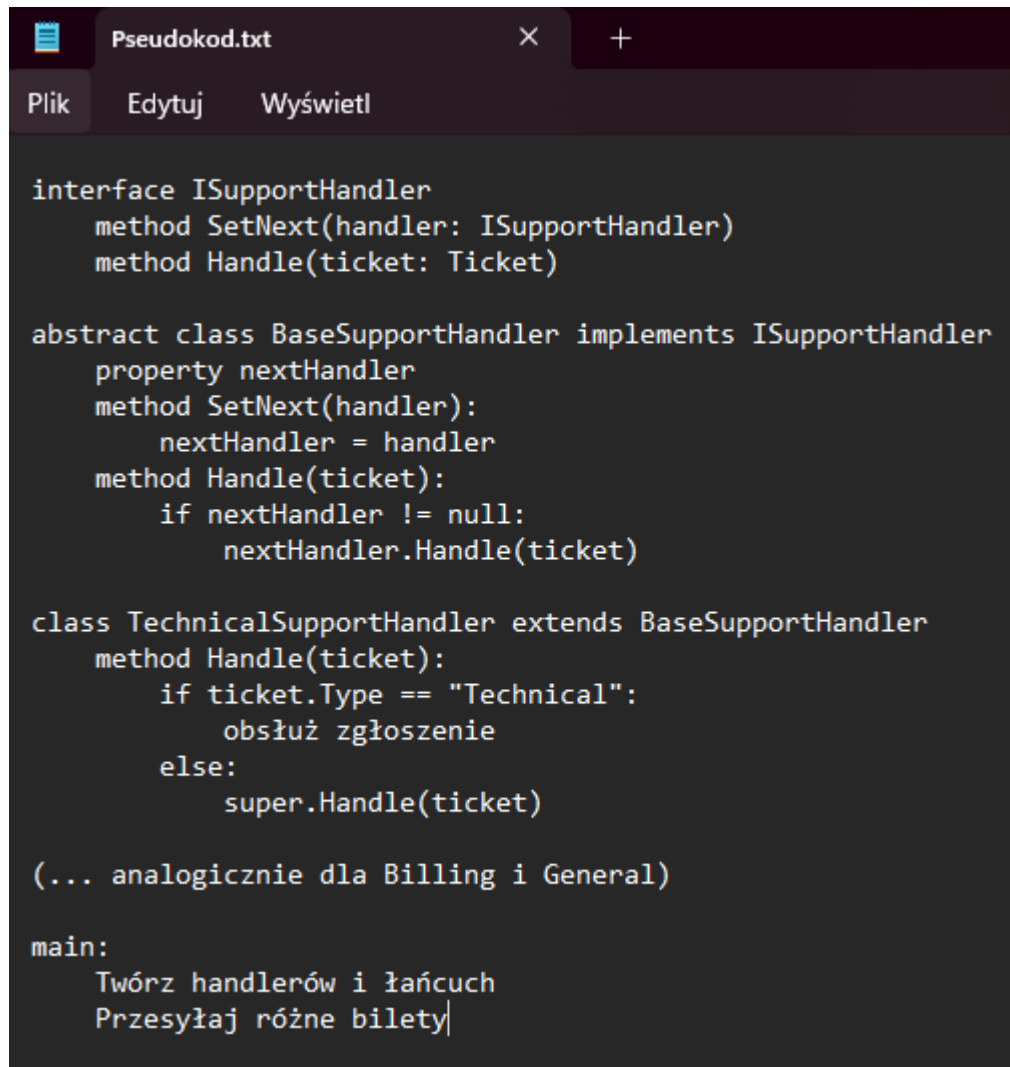
- **Technical** – obsługiwane przez **`TechnicalSupportHandler`**
- **Billing** – obsługiwane przez **`BillingSupportHandler`**
- **General** – obsługiwane przez **`GeneralSupportHandler`**

Zgłoszenia, które nie pasują do żadnego typu, są odrzucane z komunikatem o braku obsługi.

- Propozycja diagramu klas



- Pseudokod



```
interface ISupportHandler
    method SetNext(handler: ISupportHandler)
    method Handle(ticket: Ticket)

abstract class BaseSupportHandler implements ISupportHandler
    property nextHandler
    method SetNext(handler):
        nextHandler = handler
    method Handle(ticket):
        if nextHandler != null:
            nextHandler.Handle(ticket)

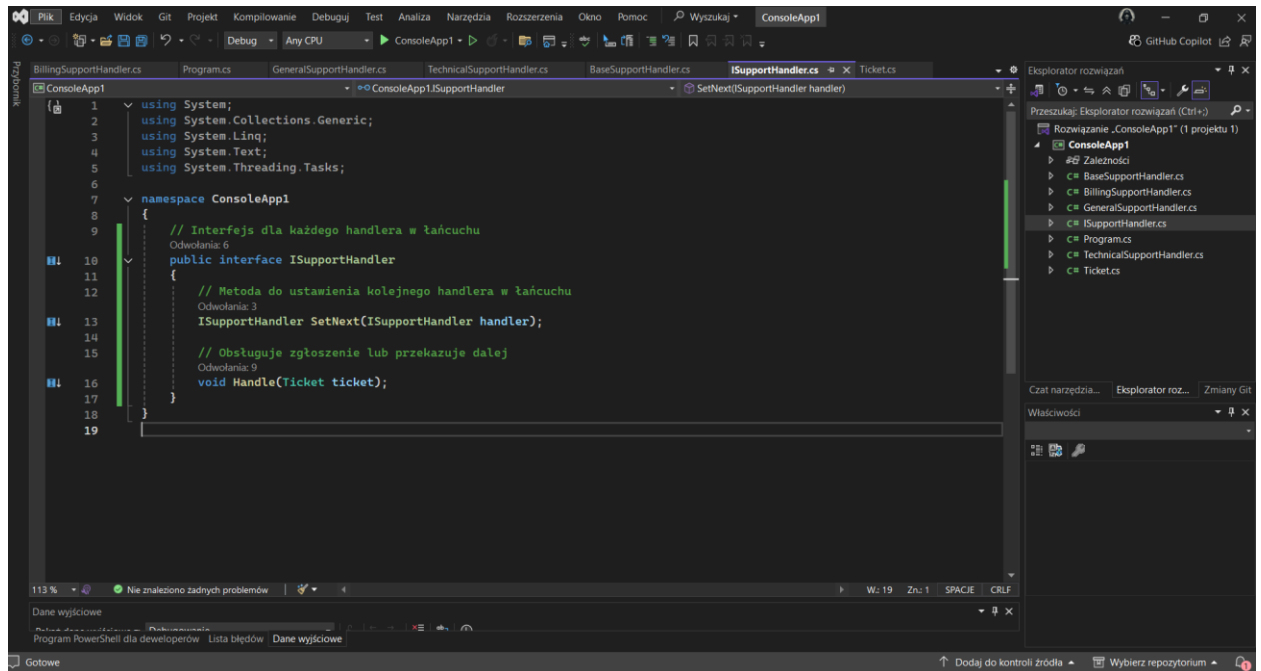
class TechnicalSupportHandler extends BaseSupportHandler
    method Handle(ticket):
        if ticket.Type == "Technical":
            obsłuż zgłoszenie
        else:
            super.Handle(ticket)

(... analogicznie dla Billing i General)

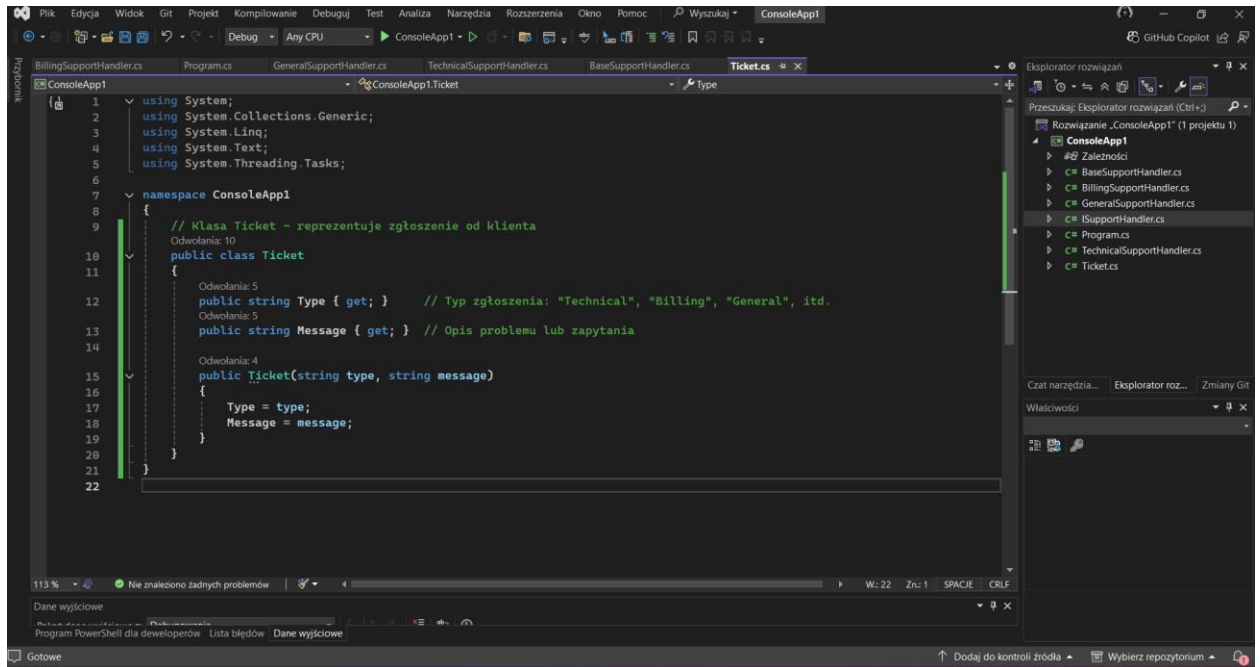
main:
    Twórz handlerów i łańcuch
    Przesyłaj różne bilety
```

### 3. Implementacja

- Kod interfejsu `ISupportHandler.cs`



- Kod klasy **Ticket.cs**

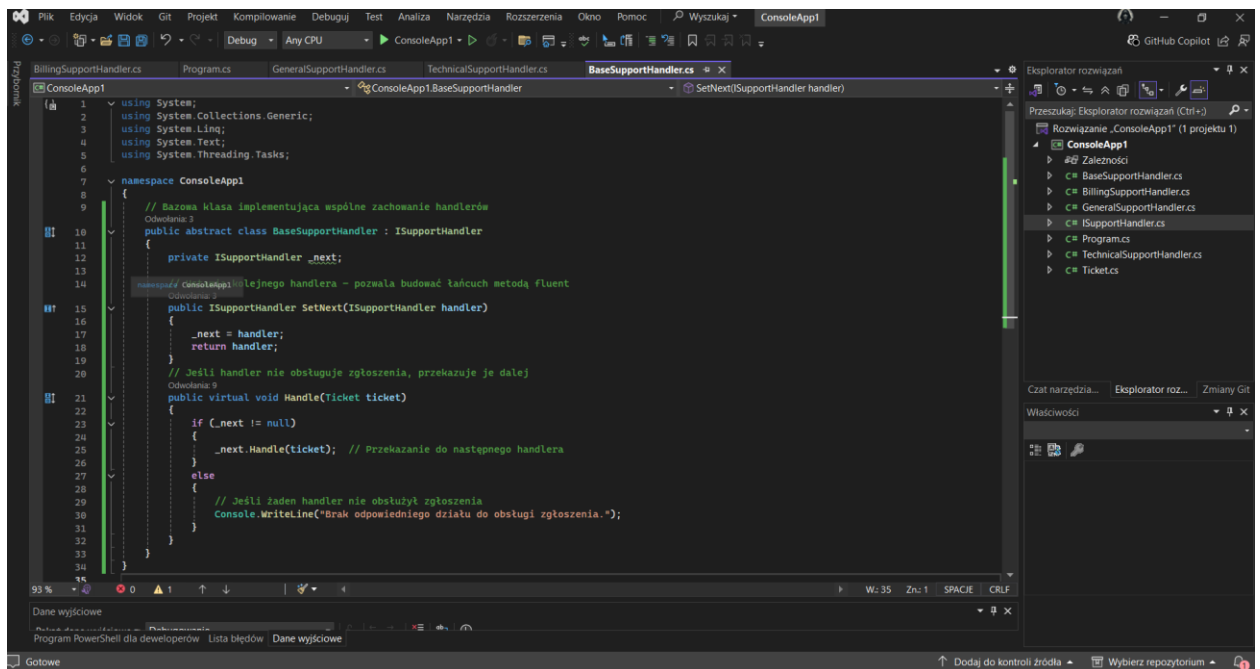


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp1
8  {
9      // Klasa Ticket - reprezentuje zgłoszenie od klienta
10     Odwołania: 10
11     public class Ticket
12     {
13         Odwołania: 5
14         public string Type { get; } // Typ zgłoszenia: "Technical", "Billing", "General", itd.
15         Odwołania: 5
16         public string Message { get; } // Opis problemu lub zapytania
17
18         Odwołania: 4
19         public Ticket(string type, string message)
20         {
21             Type = type;
22             Message = message;
23         }
24     }
25 }

```

- Kod klasy **BaseSupportHandler.cs**

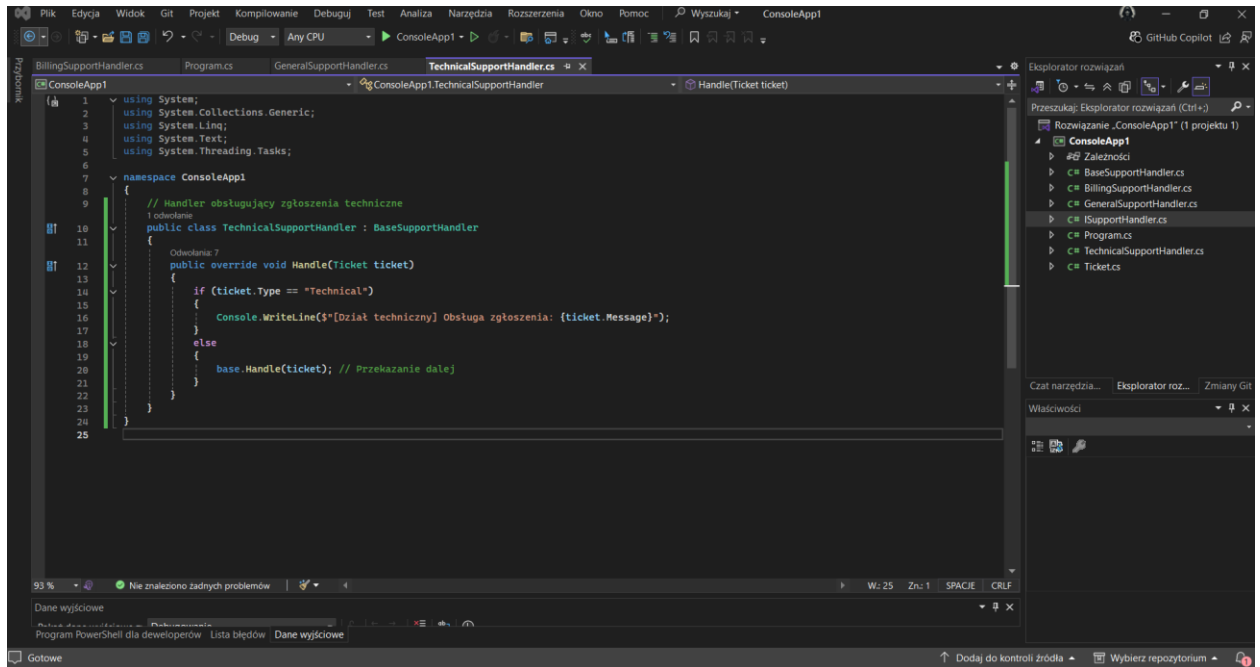


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp1
8  {
9      // Bazowa klasa implementująca wspólne zachowanie handlerów
10     Odwołania: 3
11     public abstract class BaseSupportHandler : ISupportHandler
12     {
13         private ISupportHandler _next;
14
15         // Następnego handlera - pozwala budować łańcuch metod fluent
16         Odwołania: 5
17         public ISupportHandler SetNext(ISupportHandler handler)
18         {
19             _next = handler;
20             return handler;
21         }
22
23         // Jeśli handler nie obsługuje zgłoszenia, przekazuje je dalej
24         Odwołania: 9
25         public virtual void Handle(Ticket ticket)
26         {
27             if (_next != null)
28             {
29                 _next.Handle(ticket); // Przekazanie do następnego handlera
30             }
31             else
32             {
33                 // Jeśli żaden handler nie obsłużył zgłoszenia
34                 Console.WriteLine("Brak odpowiedniego działu do obsługi zgłoszenia.");
35             }
36         }
37     }
38 }

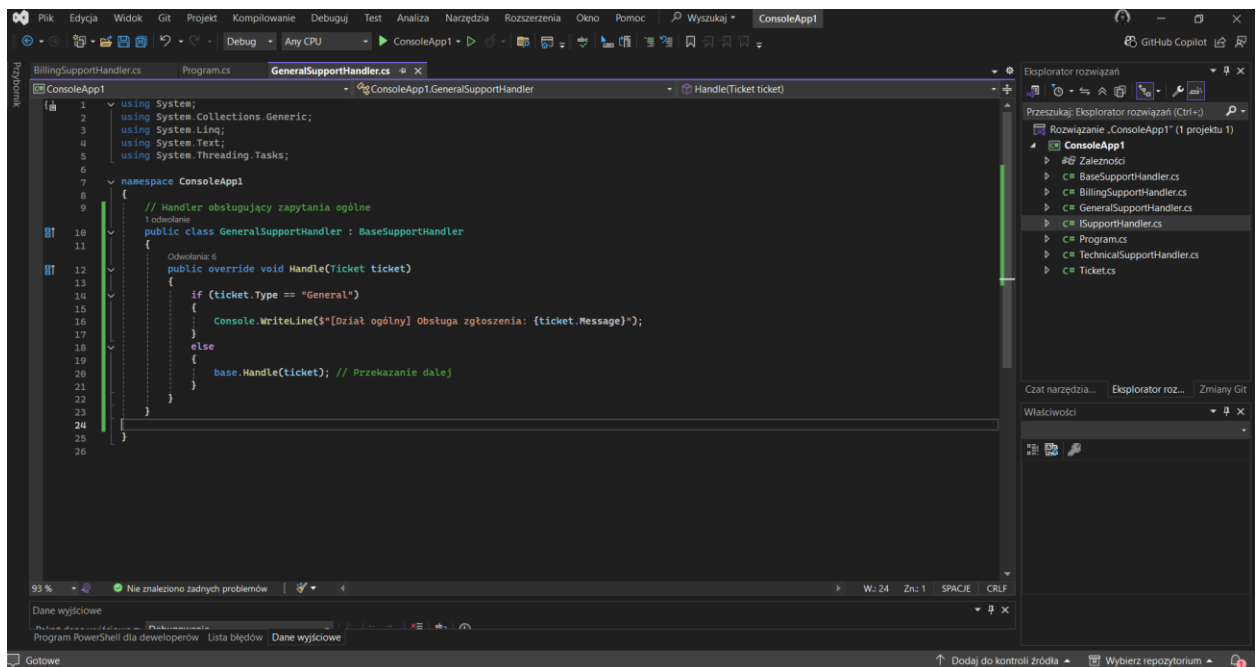
```

- Kod klasy **TechnicalSupportHandler.cs**



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApp1
8 {
9     // Handler obsługujący zgłoszenia techniczne
10     1 odwołanie
11     public class TechnicalSupportHandler : BaseSupportHandler
12     {
13         Odwołania: 7
14         public override void Handle(Ticket ticket)
15         {
16             if (ticket.Type == "Technical")
17             {
18                 Console.WriteLine($"[Dział techniczny] Obsługa zgłoszenia: {ticket.Message}");
19             }
20             else
21             {
22                 base.Handle(ticket); // Przekazanie dalej
23             }
24         }
25     }
26 }
```

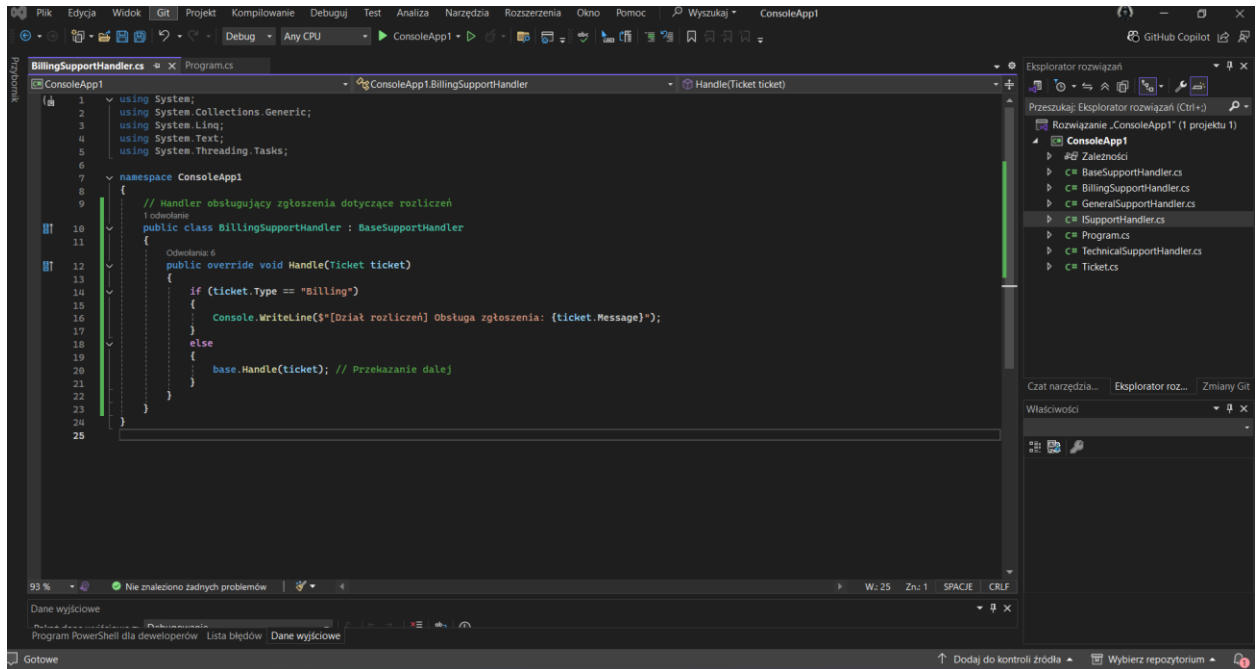
- Kod klasy **GeneralSupportHandler.cs**



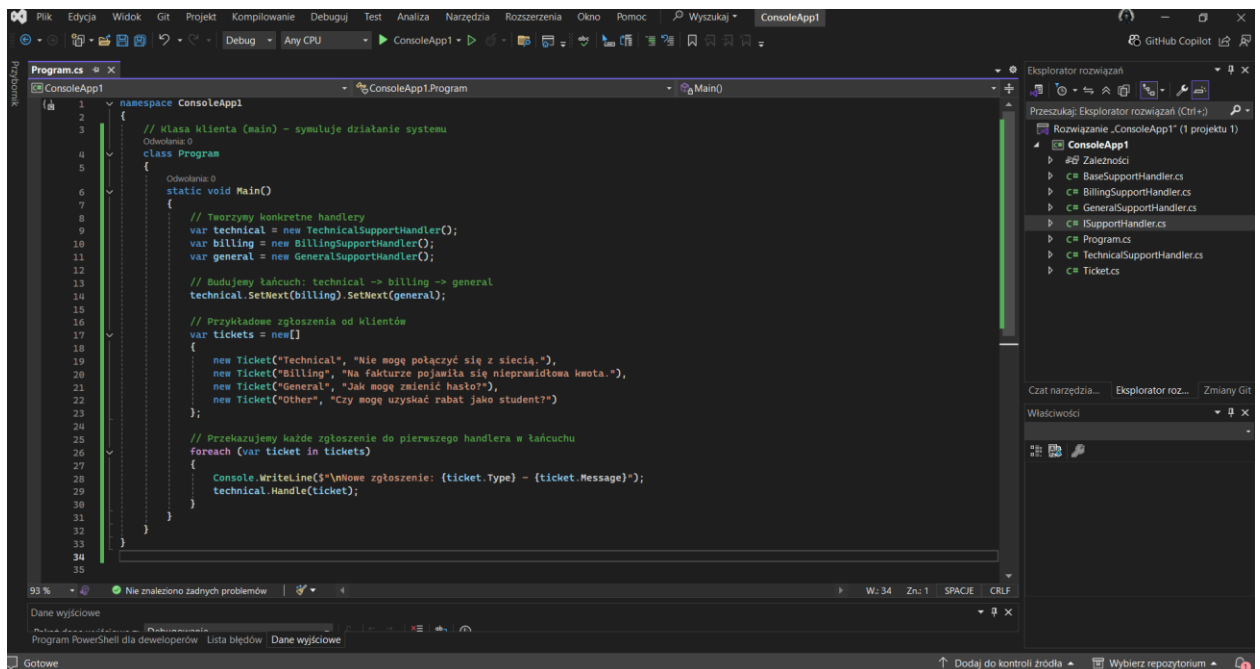
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApp1
8 {
9     // Handler obsługujący zapytania ogólne
10     1 odwołanie
11     public class GeneralSupportHandler : BaseSupportHandler
12     {
13         Odwołania: 6
14         public override void Handle(Ticket ticket)
15         {
16             if (ticket.Type == "General")
17             {
18                 Console.WriteLine($"[Dział ogólny] Obsługa zgłoszenia: {ticket.Message}");
19             }
20             else
21             {
22                 base.Handle(ticket); // Przekazanie dalej
23             }
24         }
25     }
26 }
```



- Kod klasy **BillingSupportHandler.cs**



- Kod klasy **Program.cs**



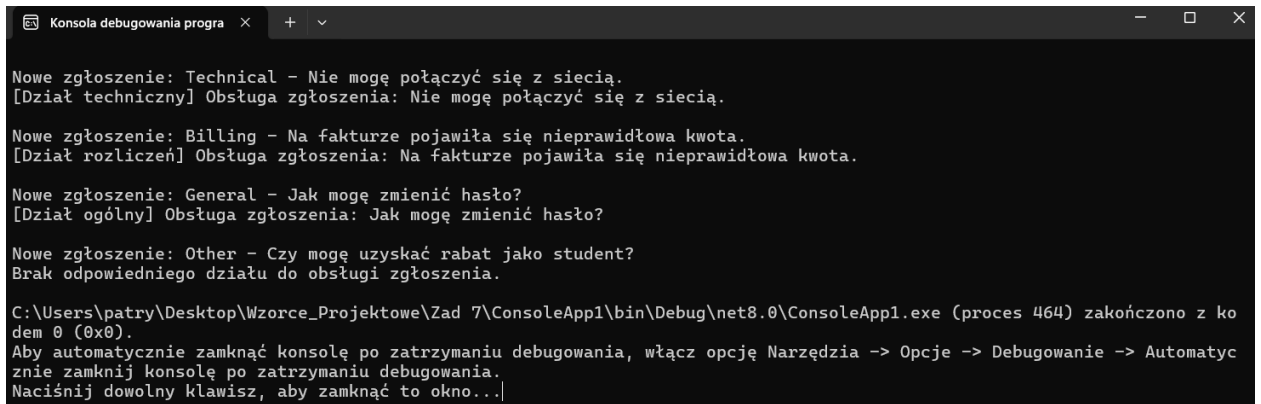


## 4. Podsumowanie

Do rozwiązania zadania zastosowano wzorzec **Chain of Responsibility**, ponieważ idealnie odwzorowuje proces przetwarzania zgłoszeń w systemie wsparcia technicznego. Umożliwia on modularne i rozszerzalne podejście – dodanie nowego typu zgłoszenia (np. Security, Legal) nie wymaga modyfikacji pozostałego kodu.

**Implementacja powiodła się**, ponieważ:

- Zgłoszenia są poprawnie obsługiwane przez odpowiednie handlers.
- Niezgodne zgłoszenia są obsługiwane zgodnie z logiką końcową („brak obsługi”).
- Kod jest elastyczny i zgodny z zasadą **Open/Closed (SOLID)**.



```
Konsola debugowania progra x + v

Nowe zgłoszenie: Technical - Nie mogę połączyć się z siecią.
[Dział techniczny] Obsługa zgłoszenia: Nie mogę połączyć się z siecią.

Nowe zgłoszenie: Billing - Na fakturze pojawiła się nieprawidłowa kwota.
[Dział rozliczeń] Obsługa zgłoszenia: Na fakturze pojawiła się nieprawidłowa kwota.

Nowe zgłoszenie: General - Jak mogę zmienić hasło?
[Dział ogólny] Obsługa zgłoszenia: Jak mogę zmienić hasło?

Nowe zgłoszenie: Other - Czy mogę uzyskać rabat jako student?
Brak odpowiedniego działu do obsługi zgłoszenia.

C:\Users\patry\Desktop\Wzorce_Projektowe\Zad 7\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe (proces 464) zakończono z ko
dem 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 (np. Strategy, Mediator) mogłyby być użyte do innej architektury przekazywania zgłoszeń, ale **Chain of Responsibility** jest tutaj najlepszym wyborem, ponieważ odwzorowuje naturalny przepływ decyzyjny i przetwarzania.

## Lista załączników

Repozytorium GITHUB z projektem:

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