Text

Description automatically generated with medium confidence

wzorce projektowe

Sprawozdanie

Ćwiczenie 7 - Chain of Responsibility

Adam Zajler

Informatyka, programowanie

34\_Inf\_P\_NW\_6

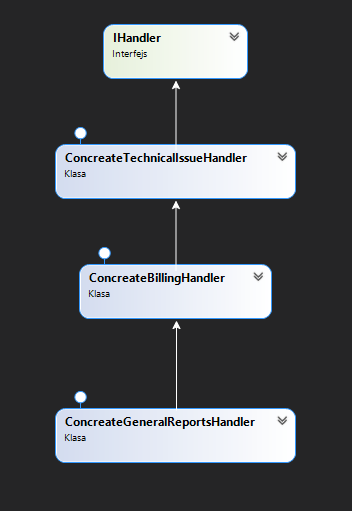
Rok akademicki 2023/24

# Cel

Celem sprawozdanie jest udokumentowanie wykonania ćwiczenia. W ramach zadania został utworzony program w języku c# który służy do obsługi zgłoszeń do różnych działów pomocy.

# Opis rozwiązania

Kod ilustruje wzorzec Decorator, pozwalający tworzyć spersonalizowaną kawę. Dodaje mleko, cukier lub syropy do bazowej klasy Coffee. Klasa CoffeeDecorator rozszerza funkcjonalność, dodając składniki. Dziedziczące po niej klasy dekoratorów (mleko, cukier, syrop) aktualizują cenę i listę dodatków.



# Implementacja

using System;

namespace Ćwiczenie\_7\_\_\_Chain\_of\_Responsibility

{

interface IHandler

{

void HandleTicketRequest(int code, string message);

}

class ConcreateTechnicalIssueHandler : IHandler

{

private IHandler successor;

public void SetSuccessor(IHandler successor)

{

this.successor = successor;

}

public void HandleTicketRequest(int code, string message)

{

if (code == 1)

{

Console.WriteLine("Obsługa wiadomości '{0}' przez pomoc techniczną\n", message);

}

else

{

successor.HandleTicketRequest(code, message);

}

}

}

class ConcreateBillingHandler : IHandler

{

private IHandler successor;

public void SetSuccessor(IHandler successor)

{

this.successor = successor;

}

public void HandleTicketRequest(int code, string message)

{

if (code == 2)

{

Console.WriteLine("Obsługa wiadomości '{0}' przez pomoc dział pomocy rozliczeniowej\n", message);

}

else

{

successor.HandleTicketRequest(code, message);

}

}

}

class ConcreateGeneralReportsHandler : IHandler

{

private IHandler successor;

public void SetSuccessor(IHandler successor)

{

this.successor = successor;

}

public void HandleTicketRequest(int code, string message)

{

if (code == 3)

{

Console.WriteLine("Obsługa wiadomości '{0}' przez ogólną pomoc raportów\n", message);

}

else

{

successor.HandleTicketRequest(code, message);

}

}

}

class Program

{

static void Main(string[] args)

{

var TechnicalHandler = new ConcreateTechnicalIssueHandler();

var BillingHandler = new ConcreateBillingHandler();

var GeneralHandler = new ConcreateGeneralReportsHandler();

TechnicalHandler.SetSuccessor(BillingHandler);

BillingHandler.SetSuccessor(GeneralHandler);

var requests = new int[] { 1, 2, 3 };

foreach(int request in requests)

{

Console.WriteLine("Obsługuje request: {0}", request);

TechnicalHandler.HandleTicketRequest(request, "Problem XYZ");

}

Console.ReadKey();

}

}

}

# Podsumowanie

# Zastosowano wzorzec łańcucha odpowiedzialności, aby umożliwić obsługę różnych typów zgłoszeń. Każdy handler w łańcuchu jest odpowiedzialny za obsługę określonego typu zgłoszenia. Jeśli handler nie jest w stanie obsłużyć zgłoszenia, przekazuje je dalej do następnego handlera w łańcuchu.

# Lista załączników

[AdamZajler/wzorce-projektowe (github.com)](https://github.com/AdamZajler/wzorce-projektowe/tree/main)