Programowanie Obiektowe Zadanie nr. 2

Oleksandr Denysiuk

Email: oleksandrdenysiuk2@gmail.com

Nr. Albumu: 143068

Data: 22.08.2023

Link do repozytorium : https://github.com/M1qtso1/Exercise2.git

Rozdział 1: Zmiana Console.WriteLine() na odczytanie z JSON pliku

ScreenLineEntry - Klasa ta reprezentuje pojedynczą linię na ekranie i zawiera trzy główne pola: BackgroundColor, ForegroundColor, oraz Text. BackgroundColor określa kolor tła linii, ForegroundColor określa kolor tekstu, a Text to treść linii.

```
public class ScreenLineEntry
{
    3 references
    public ConsoleColor BackgroundColor { get; set; }
    3 references
    public ConsoleColor ForegroundColor { get; set; }
    5 references
    public string Text { get; set; }

    0 references
    public ScreenLineEntry(ConsoleColor backgroundColor, ConsoleColor foregroundColor, string text)
    {
        BackgroundColor = backgroundColor;
        ForegroundColor = foregroundColor;
        Text = text;
    }

    0 references
    public override string ToString()
    {
        return $"Background Color: {BackgroundColor}, Foreground Color: {ForegroundColor}, Text: {Text}";
    }
}
```

ScreenDefinition - Klasa służy do definiowania ekranów poprzez przechowywanie listy obiektów ScreenLineEntry. Każdy obiekt tej klasy jest używany do określenia wyglądu i treści pojedynczej linii na ekranie.

```
public class ScreenDefinition
{
    5 references
    public List<ScreenLineEntry> LineEntries { get; set; } = new List<ScreenLineEntry>();
    2 references
    public ScreenDefinition()
    {
        LineEntries = new List<ScreenLineEntry>();
    }
}
```

IScreenDefinitionService - Interfejs, który definiuje zestaw metod, które muszą być zaimplementowane przez klasy obsługujące operacje na definicjach ekranów. Obejmuje metody do wczytywania i zapisywania definicji ekranów.

```
public interface IScreenDefinitionService
{
    7 references
    ScreenDefinition Load(string jsonFileName);
    3 references
    bool Save(ScreenDefinition screenDefinition, string jsonFileName);
}
```

ScreenDefinitionService - Klasa implementująca IScreenDefinitionService, odpowiedzialna za wczytywanie i zapisywanie definicji ekranów w formacie JSON. Zawiera także metody do wyświetlania linii zdefiniowanych w pliku JSON.

```
public class ScreenDefinitionService : IScreenDefinitionService
{
    7 references
    public ScreenDefinition Load(string jsonFileName)
    {
        if (!File.Exists(jsonFileName))
        {
            return null; // Return null when the file is not found
        }

        string json = File.ReadAllText(jsonFileName);
        return JsonConvert.DeserializeObject<ScreenDefinition>(json);
    }

    3 references
    public bool Save(ScreenDefinition screenDefinition, string jsonFileName)
    {
        try
        {
            string json = JsonConvert.SerializeObject(screenDefinition, Formatting.Indented);
            File.WriteAllText(jsonFileName, json);
            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error saving screen definition: {ex.Message}");
            return false;
        }
    }
```

```
    public Dictionary<string, string> LoadScreenDefinitions(string jsonFilePath)
    {
        // Check if the JSON file exists
        if (!File.Exists(jsonFilePath))
        {
            Console.WriteLine("Screen definition JSON file does not exist.");
            return new Dictionary<string, string>();
        }

        try
        {
            // Read the JSON content from the file
            string jsonContent = File.ReadAllText(jsonFilePath);

            // Deserialize the JSON content into a Dictionary
            return JsonConvert.DeserializeObject<Dictionary<string, string>>(jsonContent);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error loading screen definitions: {ex.Message}");
            return new Dictionary<string, string>();
        }
    }
```

```
    public string FindLineByText(string jsonFileName, string searchText)
    {
        var screenDefinition = Load(jsonFileName);
        var matchingLine = screenDefinition.LineEntries.FirstOrDefault(lineEntry => lineEntry.Text.Contains(searchText));

        if (matchingLine != null)
        {
            return matchingLine.Text;
        }
        else
        {
            throw new InvalidOperationException($"No matching line found for '{searchText}' in the JSON file.");
        }
    }
    97 references
    public void DisplayLines(string jsonFileName, int number)
    {
        ScreenDefinition screenDefinition = Load(jsonFileName);
        Console.ForegroundColor = screenDefinition.LineEntries[number].ForegroundColor;
        Console.BackgroundColor = screenDefinition.LineEntries[number].BackgroundColor;
        Console.WriteLine(screenDefinition.LineEntries[number].Text);
        Console.ResetColor();
    }
}
```

Screen - Klasa abstrakcyjna, która stanowi bazę dla ekranów w systemie. Zawiera podstawową logikę wyświetlania ekranu.

```
public abstract class Screen
{
    3 references
    protected IScreenDefinitionService ScreenDefinitionService { get; }
    3 references
    protected string ScreenDefinitionJson { get; }
    #region Public Methods

    /// <summary>
    /// Show the screen.
    /// </summary>
    7 references
    public Screen(IScreenDefinitionService screenDefinitionService, string screenDefinitionJson)
    {
        ScreenDefinitionService = screenDefinitionService;
        ScreenDefinitionJson = screenDefinitionJson;
    }
    // Add a property for the screen color
    1 reference
    public string? ScreenColor { get; set; }

    // Add a method for displaying the screen in the color specified in the settings
    0 references
    public virtual void Display()
    {
        // Set the background color of the console to the screen color
        Console.BackgroundColor = (ConsoleColor)Enum.Parse(typeof(ConsoleColor), ScreenColor);

        // Clear the console and write some text
        Console.Clear();
        Console.WriteLine($"This is the {this.GetType().Name} screen.");
    }
    #endregion // Public Methods
}
```

Wykorzystanie definicji ekranów w klasie DogsScreen innych ekranach

Klasa DogsScreen wykorzystuje definicje ekranów wczytane z pliku JSON. Podczas wyświetlania ekranu, klasa ta korzysta z definicji ekranów do ustalenia kolorów tła i tekstu na ekranie oraz do wyświetlenia odpowiednich komunikatów użytkownikowi. Na przykład, wykorzystuje różne kolory tła i tekstu do wyróżnienia poszczególnych opcji na ekranie

```
5 references
public sealed class DogsScreen : Screen
{
    private IDataService _dataService;
    private readonly ScreenDefinitionService _screenDefinitionService;
    private readonly string JsonFilePath = "DogsScreen.json";

    0 references
    public DogsScreen(IScreenDefinitionService screenDefinitionService, IDataService dataService, ScreenDefinitionService screenDefinitionServices) : base(screenDefinitionService, "DogsScreen.json")
    {
        _dataService = dataService;
        _screenDefinitionService = screenDefinitionServices;
    }
```

```
public void Show()
{

    Console.Clear();
    int selectedIndex = 1; // Track the currently selected line
    int startIndex = 0;
    int endIndex = 5;

    while (true)
    {
        Console.WriteLine();
        Console.BackgroundColor = ConsoleColor.Black;
        for (int i = startIndex; i <= endIndex; i++)
        {
            if (i == selectedIndex)
                Console.Write("-> "); // Indicate the selected line
            else
                Console.Write("   ");
            _screenDefinitionService.DisplayLines(JsonFilePath, i);
        }
```

```
var key = Console.ReadKey(intercept: true).Key;
switch (key)
{
    case ConsoleKey.UpArrow:
        selectedIndex = Math.Max(1, selectedIndex - 1);
        Console.Clear();
        break;
    case ConsoleKey.DownArrow:
        selectedIndex = Math.Min(5, selectedIndex + 1);
        Console.Clear();
        break;
    case ConsoleKey.Enter:
        switch (selectedIndex)
        {
            case 1:
                ListDogs();
                break;
            case 2:
                AddDog();
                break;
            case 3:
                DeleteDog();
                break;
            case 4:
                EditDogMain();
                break;
            case 5:
                _screenDefinitionService.DisplayLines(JsonFilePath, 10);
                Thread.Sleep(500);
                Console.Clear();
                return;
            default:
                break;
        }
        Console.BackgroundColor = ConsoleColor.Black;
        break;
    default:
        break;
}
```

Plik JSON

Tak wygląda plik JSON, z jakiego program bierze informację dla wyświetlania.

```
{
  "LineEntries": [
    {
      "Text": "Main Screen -> Animals Screen -> Mamamals Screen -> Dogs Screen",
      "BackgroundColor": "Black",
      "ForegroundColor": "Red"
    },
    {
      "Text": "List Dogs",
      "BackgroundColor": "Black",
      "ForegroundColor": "Yellow"
    },
    {
      "Text": "Create Dog",
      "BackgroundColor": "Green",
      "ForegroundColor": "White"
    },
    {
      "Text": "Delete Dog",
      "BackgroundColor": "Black",
      "ForegroundColor": "White"
    },
    {
      "Text": "Modify Dog",
      "BackgroundColor": "Green",
      "ForegroundColor": "White"
    },
```

Rozdział 2: Testy

Testy jednostkowe

W celu zapewnienia poprawności działania systemu, utworzymy testy jednostkowe przy użyciu narzędzia MSTest. Testy te sprawdzają różne funkcje klasy ScreenDefinitionService, takie jak wczytywanie poprawnych i niepoprawnych plików JSON oraz zapisywanie definicji ekranów. Przykłady testów:

• Load_ValidJsonFile_ReturnsScreenDefinition: Test sprawdzający, czy metoda Load klasy ScreenDefinitionService prawidłowo wczytuje definicję ekranu z poprawnego pliku JSON.

• Save_ValidScreenDefinition_ReturnsTrue: Test sprawdzający, czy metoda Save klasy ScreenDefinitionService prawidłowo zapisuje definicję ekranu w pliku JSON.

```
[TestClass]
0 references
public class ScreenDefinitionServiceTests
{
    [TestMethod]
    0 references
    public void Load_ValidJsonFile_ReturnsScreenDefinition()
    {
        // Arrange
        IScreenDefinitionService service = new ScreenDefinitionService();
        string jsonFileName = "validScreenDefinition.json"; // Provide a valid JSON file path

        // Act
        ScreenDefinition result = service.Load(jsonFileName);

        // Assert
        Assert.IsNotNull(result);
        // Add more assertions as needed
    }

    [TestMethod]
    0 references
    public void Save_ValidScreenDefinition_ReturnsTrue()
    {
        // Arrange
        IScreenDefinitionService service = new ScreenDefinitionService();
        ScreenDefinition screenDefinition = new ScreenDefinition();
        string jsonFileName = "validScreenDefinition.json"; // Provide a valid JSON file path

        // Act
        bool result = service.Save(screenDefinition, jsonFileName);

        // Assert
        Assert.IsTrue(result);
    }
```

• Load_InvalidJsonPath_ReturnsNull: Test sprawdzający, czy metoda Load obsługuje prawidłowo sytuację, gdy próbuje wczytać nieistniejący plik JSON.

• Save_InvalidScreenDefinition_ReturnsFalse: Test sprawdzający, czy metoda Save obsługuje prawidłowo sytuację, gdy próbuje zapisać nieprawidłową definicję ekranu.

```
    [TestMethod]
    0 references
    public void Load_InvalidJsonPath_ReturnsNull()
    {
        // Arrange
        IScreenDefinitionService service = new ScreenDefinitionService();
        string jsonFileName = "nonExistentFile.json"; // Provide a non-existent JSON file path

        // Act
        ScreenDefinition result = service.Load(jsonFileName);

        // Assert
        Assert.IsNull(result);
    }

    [TestMethod]
    0 references
    public void Save_InvalidScreenDefinition_ReturnsFalse()
    {
        // Arrange
        IScreenDefinitionService service = new ScreenDefinitionService();
        // Create an invalid screen definition (for example, with null properties)
        ScreenDefinition screenDefinition = new ScreenDefinition();
        string jsonFileName = ""; // Provide a valid JSON file path

        // Act
        bool result = service.Save(screenDefinition, jsonFileName);

        // Assert
        Assert.IsFalse(result);
    }
}
```

Rozdział 3: Strzałki

Metoda do wykorzystania strzałek

Tak wygląda metoda do poruszania się po ekranie za pomocą klawiszy strzałek.

```
for (int i = startIndex; i <= endIndex; i++)
{
    if (i == selectedIndex)
        Console.Write("-> "); // Indicate the selected line
    else
        Console.Write("   ");
    _screenDefinitionService.DisplayLines(JsonFilePath, i);
}
var key = Console.ReadKey(intercept: true).Key;
switch (key)
{
    case ConsoleKey.UpArrow:
        selectedIndex = Math.Max(1, selectedIndex - 1);
        Console.Clear();
        break;
    case ConsoleKey.DownArrow:
        selectedIndex = Math.Min(5, selectedIndex + 1);
        Console.Clear();
        break;
    case ConsoleKey.Enter:
        switch (selectedIndex)
        {
            case 1:
                ListDogs();
                break;
            case 2:
                AddDog();
                break;
            case 3:
                DeleteDog();
                break;
            case 4:
                EditDogMain();
                break;
            case 5:
                _screenDefinitionService.DisplayLines(JsonFilePath, 10);
                Thread.Sleep(500);
                Console.Clear();
                return;
            default:
                break;
        }
```