Programowanie Obiektowe Zadanie nr. 1

Oleksandr Denysiuk

Email: oleksandrdenysiuk2@gmail.com

Nr. Albumu: 143068

Data: 22.08.2023

Link do repozytorium : https://github.com/M1qtso1/Exercise1.git

Rozdział 1: Dodanie ustawień dotyczących kolorystyki poszczególnych ekranów

Interfejs ISettings oraz klasa Settings

Zostały zdefiniowane w celu przechowywania kolorów ekranów w formie słownika.

```
namespace SampleHierarchies.Data
{
    3 references
    public class Settings : ISettings
    {
        8 references
        public Dictionary<string, ConsoleColor> ScreenColors { get; set; }
    }
}
```

```
namespace SampleHierarchies.Interfaces.Data
{
    12 references
    public interface ISettings
    {
        8 references
        Dictionary<string, ConsoleColor> ScreenColors { get; set; }
    }
}
```

Interfejs ISettingsService oraz klasa SettingsService

Obsługa zapisu i odczytu ustawień została zaimplementowana w klasie SettingsService, która implementuje interfejs ISettingsService. W tej klasie wczytywane są ustawienia z pliku JSON lub tworzone są domyślne ustawienia, jeśli plik nie istnieje.

```
namespace SampleHierarchies.Services
{
    8 references
    public class SettingsService : ISettingsService
    {
        private ISettings _settings;
        0 references
        public SettingsService()
        {
            // Initialize settings (load from file or create defaults)
            _settings = Read("Settings.json") ?? new Settings
            {
            };
        }
        4 references
        public ISettings GetSettings()
        {
            return _settings;
        }
        3 references
        public ISettings? Read(string jsonPath)
        {
            if (!File.Exists(jsonPath))
            {
                return null;
            }
            string json = File.ReadAllText(jsonPath);
            return JsonConvert.DeserializeObject<Settings>(json);
        }
        2 references
        public void Write(ISettings settings, string jsonPath)
        {
            string json = JsonConvert.SerializeObject(settings, Formatting.Indented);
            File.WriteAllText(jsonPath, json);
        }
    }
}
```

```
namespace SampleHierarchies.Interfaces.Services;

6 references
public interface ISettingsService
{
    #region Interface Members

    /// <summary>
    /// Read settings.
    /// </summary>
    /// <param name="jsonPath">Json path</param>
    /// <returns></returns>
    3 references
    ISettings? Read(string jsonPath);
    /// <summary>
    /// Write settings.
    /// </summary>
    /// <param name="settings">Settings to written</param>
    /// <param name="jsonPath">Json path</param>
    2 references
    void Write(ISettings settings, string jsonPath);
    #endregion // Interface Members
}
```

Ekrany

Klasa MainScreen jest odpowiedzialna za wyświetlanie głównego ekranu aplikacji oraz obsługę ustawień kolorów. Kolorystyka ekranu głównego oraz ekranów zwierząt, ssaków i psów jest pobierana z ustawień.

```
public sealed class MainScreen : Screen
{
    #region Properties And Ctor

    /// <summary>
    /// Data service.
    /// </summary>
    private IDataService _dataService;
    private AnimalsScreen _animalsScreen;
    private ISettingsService _settingsService;
    private ISettings _settings;

    /// <summary>
    /// Ctor.
    /// </summary>
    /// <param name="dataService"></param>
    /// <param name="animalsScreen"></param>
    /// <param name="settingsService"></param>
    0 references
    public MainScreen(IDataService dataService, AnimalsScreen animalsScreen, SettingsService settingsService)
    {
        _dataService = dataService;
        _animalsScreen = animalsScreen;
        _settingsService = settingsService;
        _settings = settingsService.GetSettings();
    }


    #endregion Properties And Ctor

    #region Public Methods

    /// <inheritdoc/>
    2 references
    public override void Show()
    {
        while (true)
        {
            Console.ForegroundColor = _settings.ScreenColors["MainScreen"];
```

Zarządzanie ustawieniami kolorystyki

Zmiana kolorystyki ekranów poprzez menu "Settings Menu". Użytkownik może wybrać ekran do zmiany koloru oraz podać nowy kolor w formie ConsoleColor.

```
private void ShowSettingsMenu()
{
    while (true)
    {
        Console.ForegroundColor = _settings.ScreenColors["MainScreen"];
        Console.WriteLine();
        Console.WriteLine("Settings Menu:");
        Console.WriteLine("1. Change Colors");
        Console.WriteLine("2. Write Settings to JSON");
        Console.WriteLine("3. Read Settings from JSON");
        Console.WriteLine("0. Back to Main Menu");
        Console.Write("Enter your choice: ");
        string choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                ChangeColors();
                break;
            case "2":
                WriteSettingsToJson();
                break;
            case "3":
                ReadSettingsFromJson();
                break;
            case "0":
                return;
            default:
                Console.WriteLine("Invalid choice. Try again.");
                break;
        }
    }
}
```

```
private void ChangeColors()
{
    while (true)
    {
        Console.ForegroundColor = _settings.ScreenColors["MainScreen"];
        Console.WriteLine();
        Console.WriteLine("Select a screen to change its color:");
        Console.WriteLine("1. MainScreen");
        Console.WriteLine("2. AnimalScreen");
        Console.WriteLine("3. MammalsScreen");
        Console.WriteLine("4. DogsScreen");
        Console.WriteLine("0. Back to Settings Menu");
        Console.Write("Enter your choice: ");
        string choice = Console.ReadLine();

        if (choice == "0")
        {
            return;
        }

        if (int.TryParse(choice, out int screenChoice) && screenChoice >= 1 && screenChoice <= 4)
        {
            Console.Write("Enter a new color (e.g., Red, Green, Yellow): ");
            string newColor = Console.ReadLine();

            if (Enum.TryParse(newColor, out ConsoleColor color))
            {
                string[] screenNames = { "MainScreen", "AnimalScreen", "MammalsScreen", "DogsScreen" };
                string selectedScreen = screenNames[screenChoice - 1];
                _settings.ScreenColors[selectedScreen] = color;
                Console.WriteLine($"{selectedScreen} color has been updated to {color}.");
            }
            else
            {
                Console.WriteLine("Invalid color. Please enter a valid ConsoleColor.");
            }
        }
        else
        {
            Console.WriteLine("Invalid choice. Try again.");
        }
    }
}
```

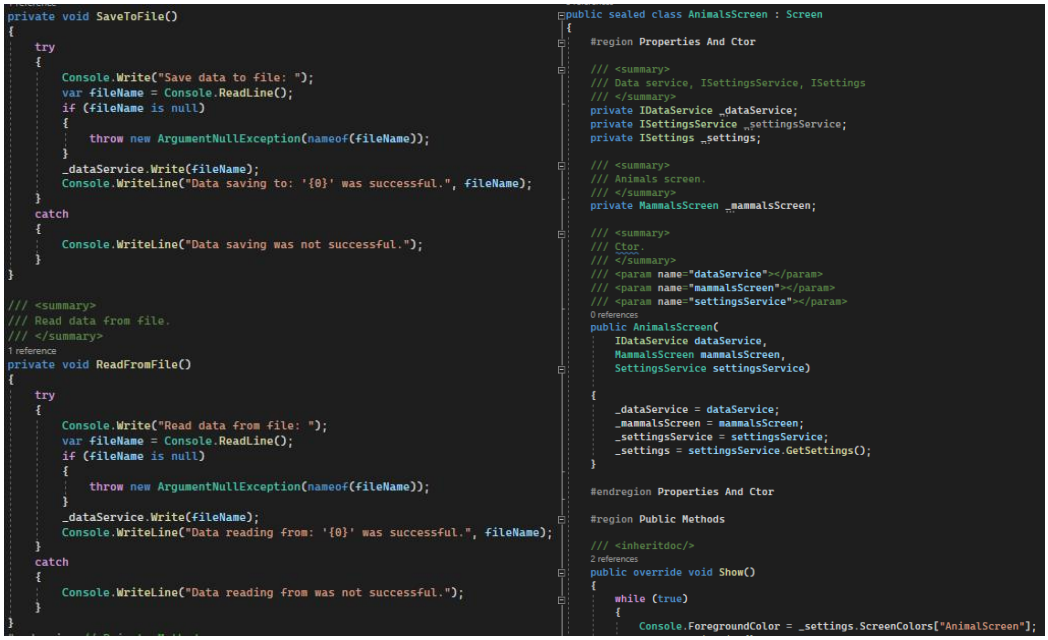
Zapis i odczyt ustawień

Klasa MainScreen pozwala również na zapis i odczyt ustawień kolorystyki do/z pliku JSON. Dla operacji zapisu i odczytu wykorzystywana jest klasa SettingsService.

```
1 reference
private void WriteSettingsToJson()
{
    _settingsService.Write(_settings, "Settings.json");
    Console.WriteLine("Settings have been written to Settings.json.");
}

1 reference
private void ReadSettingsFromJson()
{
    _settings = _settingsService.Read("Settings.json") ?? _settings;
    Console.WriteLine("Settings have been read from Settings.json.");
}
#endregion // Public Methods
```

Inne klasy

Tak samo wszystko było zrobione dla ekranu zwierząt, ekranu ssaków i ekranu psów.



```
private void SaveToFile()
{
    try
    {
        Console.Write("Save data to file: ");
        var fileName = Console.ReadLine();
        if (fileName is null)
        {
            throw new ArgumentNullException(nameof(fileName));
        }
        _dataService.Write(fileName);
        Console.WriteLine("Data saving to: '{0}' was successful.", fileName);
    }
    catch
    {
        Console.WriteLine("Data saving was not successful.");
    }
}

/// <summary>
/// Read data from file.
/// </summary>
1 reference
private void ReadFromFile()
{
    try
    {
        Console.Write("Read data from file: ");
        var fileName = Console.ReadLine();
        if (fileName is null)
        {
            throw new ArgumentNullException(nameof(fileName));
        }
        _dataService.Write(fileName);
        Console.WriteLine("Data reading from: '{0}' was successful.", fileName);
    }
    catch
    {
        Console.WriteLine("Data reading from was not successful.");
    }
}
```

```
public sealed class AnimalsScreen : Screen
{
    #region Properties And Ctor

    /// <summary>
    /// Data service, ISettingsService, ISettings
    /// </summary>
    private IDataService _dataService;
    private ISettingsService _settingsService;
    private ISettings _settings;

    /// <summary>
    /// Animals screen.
    /// </summary>
    private MammalsScreen _mammalsScreen;

    /// <summary>
    /// Ctor.
    /// </summary>
    /// <param name="dataService"></param>
    /// <param name="mammalsScreen"></param>
    /// <param name="settingsService"></param>
    0 references
    public AnimalsScreen(
        IDataService dataService,
        MammalsScreen mammalsScreen,
        SettingsService settingsService)

    {
        _dataService = dataService;
        _mammalsScreen = mammalsScreen;
        _settingsService = settingsService;
        _settings = settingsService.GetSettings();
    }

    #endregion Properties And Ctor

    #region Public Methods

    /// <inheritdoc/>
    2 references
    public override void Show()
    {
        while (true)
        {
            Console.ForegroundColor = _settings.ScreenColors["AnimalScreen"];
```

JSON plik z kolorami

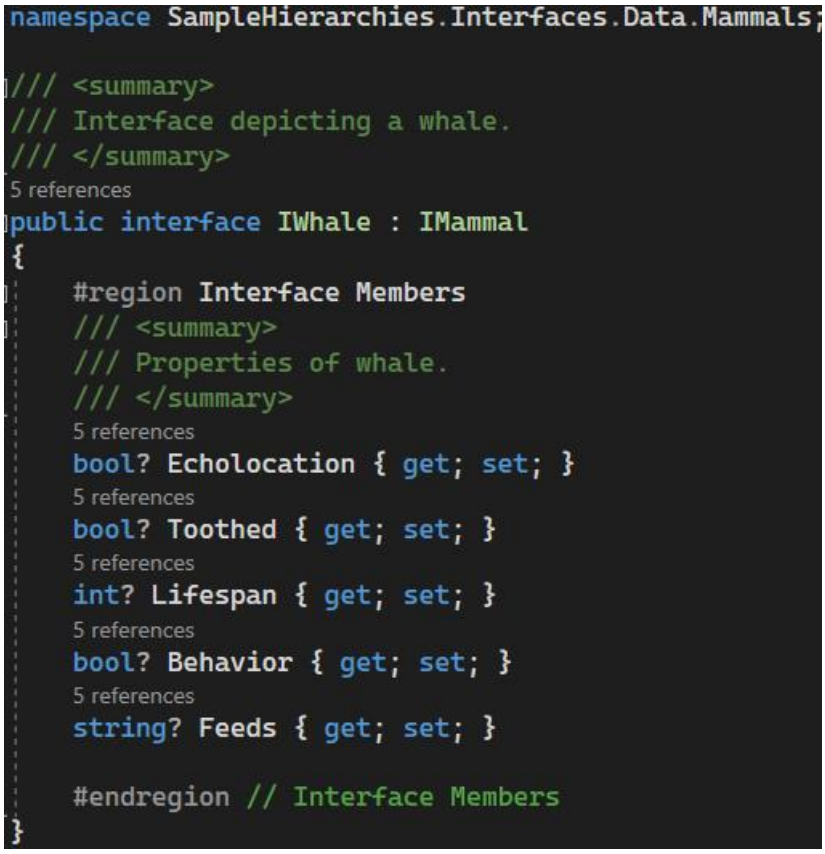
Tak wygląda JSON plik dla ustawienia kolorów ekranów.

```
{
  "ScreenColors": {
    "MainScreen": 12,
    "AnimalScreen": 13,
    "MammalsScreen": 1,
    "DogsScreen": 15
  }
}
```

Rozdzial 2: dodanie ssaków

IWhale

Dodanie struktury danych reprezentującej wieloryby. Tworzymy interfejs IWhale, który będzie zawierać właściwości specyficzne dla wielorybów.



```
namespace SampleHierarchies.Interfaces.Data.Mammals;

/// <summary>
/// Interface depicting a whale.
/// </summary>
5 references
public interface IWhale : IMammal
{
    #region Interface Members
    /// <summary>
    /// Properties of whale.
    /// </summary>
    5 references
    bool? Echolocation { get; set; }
    5 references
    bool? Toothed { get; set; }
    5 references
    int? Lifespan { get; set; }
    5 references
    bool? Behavior { get; set; }
    5 references
    string? Feeds { get; set; }

    #endregion // Interface Members
}
```

Whale

Klasa Whale implementującą interfejs IWhale. Klasa ta będzie przechowywać informacje o wielorybach, takie jak imię, wiek, obecność echolokacji, obecność zębów, długość życia, zachowanie i sposób odżywiania.

```
namespace SampleHierarchies.Data.Mammals;

/// <summary>
/// Very basic whale class.
/// </summary>
11 references
public class Whale : MammalBase, IWhale
{
    #region Public Methods

    /// <inheritdoc/>
    4 references
    public override void Display()
    {
        Console.WriteLine($"\nMy name is: {Name},\nMy age is: {Age},\nEcholocation: {Echolocation}," +
            $"\nToothed whale: {Toothed},\nLong lifespan: {Lifespan},\nSociable behavior: {Behavior}," +
            $"\nFeeds on squid: {Feeds}\n------------------\n");
    }

    /// <inheritdoc/>
    9 references
    public override void Copy(IAnimal animal)
    {
        if (animal is IWhale ad)
        {
            base.Copy(animal);
            Echolocation = ad.Echolocation;
            Toothed = ad.Toothed;
            Lifespan = ad.Lifespan;
            Behavior = ad.Behavior;
            Feeds = ad.Feeds;
        }
    }
```

```
#region Ctors And Properties

/// <inheritdoc/>
5 references
public bool? Echolocation { get; set; }
5 references
public bool? Toothed { get; set; }
5 references
public int? Lifespan { get; set; }
5 references
public bool? Behavior { get; set; }
5 references
public string? Feeds { get; set; }

/// <summary>
/// Ctor.
/// </summary>
/// <param name="name"></param>
/// <param name="age"></param>
/// <param name="echolocation"></param>
/// <param name="toothed"></param>
/// <param name="lifespan"></param>
/// <param name="behavior"></param>
/// <param name="feeds"></param>
1 reference
public Whale(string name, int age, bool? echolocation, bool? toothed, int? lifespan,
    bool? behavior, string? feeds) : base(name, age, MammalSpecies.Whale)
{
    Echolocation = echolocation;
    Toothed = toothed;
    Lifespan = lifespan;
    Behavior = behavior;
    Feeds = feeds;
}

#endregion // Ctors And Properties
```

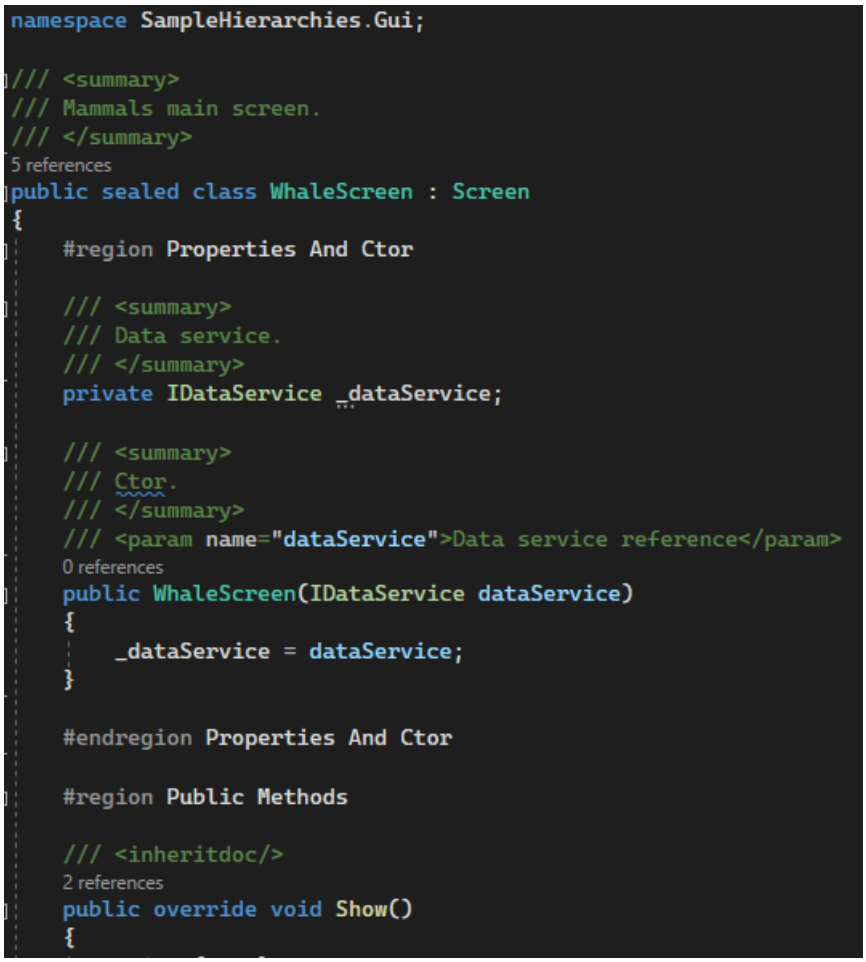
Mammals

Aby obsłużyć wieloryby, będziemy musieli dostosować serwis danych do przechowywania informacji o wielorybach. W klasie Mammals dodajmy listę wielorybów (List<IWhale> Whales) do struktury danych Mammals, która przechowuje informacje o różnych gatunkach ssaków.

```
namespace SampleHierarchies.Data.Mammals;

/// <summary>
/// Mammals collection.
/// </summary>
2 references
public class Mammals : IMammals
{
    #region IMammals Implementation

    /// <inheritdoc/>
    9 references
    public List<IDog> Dogs { get; set; }
    9 references
    public List<IOrangutan> Orangutans { get; set; }
    9 references
    public List<IChimpanzee> Chimpanzees { get; set; }
    9 references
    public List<IWhale> Whales { get; set; }

    #endregion // IMammals Implementation

    #region Ctors

    /// <summary>
    /// Default ctor.
    /// </summary>
    1 reference
    public Mammals()
    {
        Dogs = new List<IDog>();
        Orangutans = new List<IOrangutan>();
        Chimpanzees = new List<IChimpanzee>();
        Whales = new List<IWhale>();
    }

    #endregion // Ctors
}
```

WhaleScreen

Ekran o nazwie WhaleScreen, który będzie odpowiedzialny za zarządzanie danymi na temat wielorybów. Ten ekran będzie dziedziczył po klasie Screen i umożliwi użytkownikowi wykonywanie różnych operacji na danych wielorybów.



```
namespace SampleHierarchies.Gui;

/// <summary>
/// Mammals main screen.
/// </summary>
5 references
public sealed class WhaleScreen : Screen
{
    #region Properties And Ctor

    /// <summary>
    /// Data service.
    /// </summary>
    private IDataService _dataService;

    /// <summary>
    /// Ctor.
    /// </summary>
    /// <param name="dataService">Data service reference</param>
    0 references
    public WhaleScreen(IDataService dataService)
    {
        _dataService = dataService;
    }

    #endregion Properties And Ctor

    #region Public Methods

    /// <inheritdoc/>
    2 references
    public override void Show()
    {
```

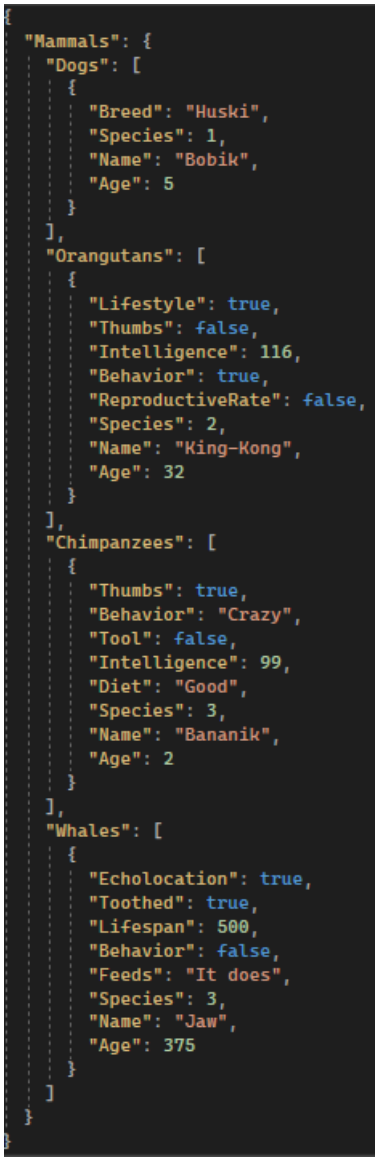
Na ekranie zaimplementujemy obsługę różnych operacji na wielorybach, takich jak wyświetlanie listy wielorybów, dodawanie nowych wielorybów, usuwanie istniejących wielorybów i edycja danych wielorybów. Każda z tych operacji będzie obsługiwana w osobnych metodach.

Dodamy również opcje zapisu i odczytu danych o wielorybach do/z pliku JSON na ekranie WhaleScreen. Do tych operacji wykorzystamy klasę DataService, która dostarcza dostęp do danych aplikacji.

Dla Chimpanzee i Orangutan były zrobione takie samy kroki.

JSON with animals

This is example JSON file with saved animals.



```
{
  "Mammals": {
    "Dogs": [
      {
        "Breed": "Huski",
        "Species": 1,
        "Name": "Bobik",
        "Age": 5
      }
    ],
    "Orangutans": [
      {
        "Lifestyle": true,
        "Thumbs": false,
        "Intelligence": 116,
        "Behavior": true,
        "ReproductiveRate": false,
        "Species": 2,
        "Name": "King-Kong",
        "Age": 32
      }
    ],
    "Chimpanzees": [
      {
        "Thumbs": true,
        "Behavior": "Crazy",
        "Tool": false,
        "Intelligence": 99,
        "Diet": "Good",
        "Species": 3,
        "Name": "Bananik",
        "Age": 2
      }
    ],
    "Whales": [
      {
        "Echolocation": true,
        "Toothed": true,
        "Lifespan": 500,
        "Behavior": false,
        "Feeds": "It does",
        "Species": 3,
        "Name": "Jaw",
        "Age": 375
      }
    ]
  }
}
```