Wyższa Szkoła Bankowa

Programowanie Obiektowe

Sprawozdanie do ćwiczenia nr 4

Imię i nazwisko: Mykhailo Zaliznyi

Adres email: zaliznyimh@gmail.com

Numer albumu: **144388**

Data: 27.08.2023

Link do repo na Github:
https://github.com/zaliznyimh/ProgramowanieObiektoweZadanie4

Rozdział 1 – Opis Hierarchii programu

W tym ćwiczeniu robiłem «System zarządzania pracownikami» pod nazwą **HRHub**.

Na potrzeby tego zadania zostały utworzone następujące projekty:

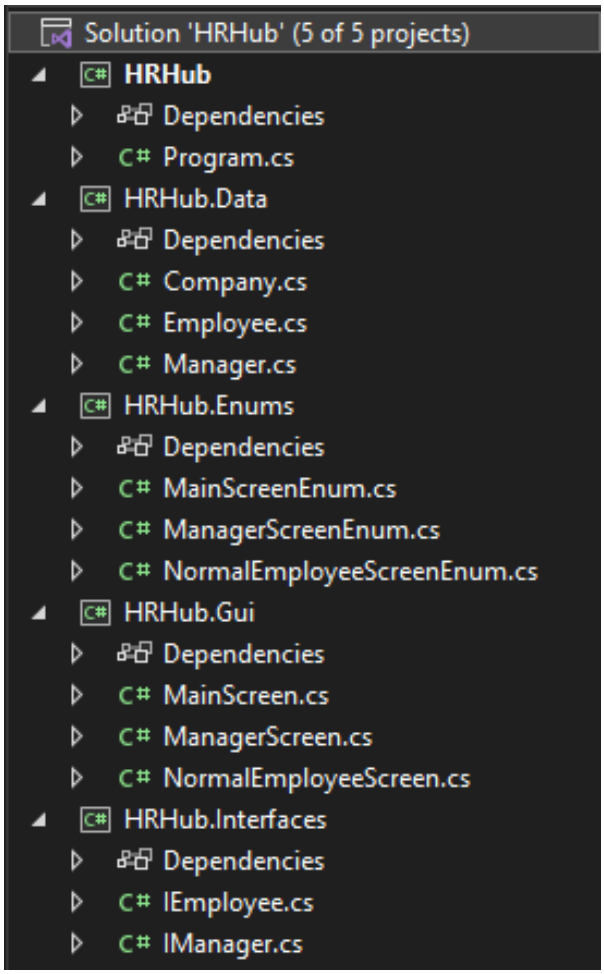
- HRHub.App – Tutaj znajduje się punkt wejścia do programu,

- HRHub.Data – Tutaj znajdują się definicję klasów programu,

- HRHub.Enums – Tutaj są Enum klasy które pomagają uprościć pracę z wyborami na każdym ekranie

- HRHub.Gui – Gui przechowuje definicje każdego z ekranów,

- HRHub.Interfaces – Tutaj znajdują się wszystkie interfejsy programu.



Rozdział 2 – Realizacja kodu

1) Klasie «Program.cs» znajduje się początek działania programu HrHub.

```
/// <summary>
/// The main entry point for the application.
/// </summary>

static void Main(string[] args)
{
    var host = CreateHostBuilder().Build();
    ServiceProvider = host.Services;

    var mainScreen = ServiceProvider.GetRequiredService<MainScreen>();
    mainScreen.Show();
}
```

2) W Interfejsach są dwa Interfejsy: IEmployee i IManager

IEmployee przechowuje właściwości i tworzy kontrakty dla realizacji dla klasy Employee:

```
/// <summary>
/// IEmployee interface members
/// </summary>
8 references
public int EmployeeID { get; set; }
7 references
public string? Name { get; set; }
6 references
public string? Position { get; set; }
6 references
public float HourlyRate { get; set; }
```

IManager przechowuje tylko właściwości jedną właściwośc dla klasy Manager:

```
/// <summary>
/// IManager interface member
/// </summary>
2 references
public int Bonus { get; set; }
```

3) Klasa Employee dziedziczy po Interfejsu IEmployee

```
#region Interface Members

/// <summary>
/// IEmployee interface implementation.
/// </summary>
8 references
public int EmployeeID { get; set; }
7 references
public string? Name { get; set; }
6 references
public string? Position { get; set; }
6 references
public float HourlyRate { get; set; }

#endregion // Interface Members
```

```
#region Ctor

/// <summary>
/// Ctor.
/// </summary>
/// <param name="employeeID"></param>
/// <param name="name"></param>
/// <param name="position"></param>
/// <param name="hourlyRate"></param>
2 references
public Employee(int employeeID, string name, string position, float hourlyRate)
{
    EmployeeID = employeeID;
    Name = name;
    Position = position;
    HourlyRate = hourlyRate;
}
```

Klasa Manager dziedziczy po Interfejsu IManager oraz klasie Employee

```
2 references
public class Manager : Employee, IManager
{
    #region Properties and Ctor

    /// <summary>
    /// IManager interface implementation
    /// </summary>
    2 references
    public int Bonus { get; set; }

    /// <summary>
    /// Ctor.
    /// </summary>
    /// <param name="employeeID"></param>
    /// <param name="name"></param>
    /// <param name="position"></param>
    /// <param name="hourlyRate"></param>
    /// <param name="bonus"></param>
    1 reference
    public Manager(int employeeID, string name, string position, float hourlyRate, int bonus) : base(employeeID, name, position, hourlyRate)
    {
        Bonus = bonus;
    }

    #endregion // Properties and Ctor
}
```

Również w projekcie HRHub.Data jest klasa **Company** która ma metody używane przez **ManagerScreen** i **NormalEmployeeScreen**

Takimi metodami są **AddEmployee()** która dodaje nowego pracownika, a później dodaje go do pliku JSON.

Poniżej jest tylko część kodu metody **AddEmployee():**

```
public void AddEmployee()
{
    try
    {
        Console.Write("Enter Employee ID: ");
        string? employeeIDasString = Console.ReadLine();

        Console.Write("Enter Employee Name: ");
        string? nameAsString = Console.ReadLine();

        Console.Write("Enter Employee Position: ");
        string? position = Console.ReadLine();

        Console.Write("Enter Employee Hourly Rate: ");
        string? hourlyRateAsString = Console.ReadLine();

        if (employeeIDasString == null) { throw new NotImplementedException(nameof(employeeIDasString)); }
        if (nameAsString == null) { throw new NotImplementedException(nameof(nameAsString)); }
        if (position == null) { throw new NotImplementedException(nameof(position)); }
        if (hourlyRateAsString == null) { throw new NotImplementedException(nameof(hourlyRateAsString)); }

        int employeeID = int.Parse(employeeIDasString);
        float hourlyRate = float.Parse(hourlyRateAsString);

        Employee addedEmployee;

        if (position.Equals("Manager", StringComparison.OrdinalIgnoreCase))
        {
            int bonus = 300;
            addedEmployee = new Manager(employeeID, nameAsString, position, hourlyRate, bonus);
        }
        else
        {
            addedEmployee = new Employee(employeeID, nameAsString, position, hourlyRate);
        }
```

Następną metodą jest **RemoveEmployee()** która usuwa pracownika z listy pracowników oraz z pliku JSON.

```
public void RemoveEmployee()
{
    try
    {
        Console.Write("Write the employee's ID: ");
        string? employeeIDAsString = Console.ReadLine();

        if (employeeIDAsString == null) { throw new NotImplementedException(nameof(employeeIDAsString)); }
        int employeeID = Int32.Parse(employeeIDAsString);

        if (File.Exists("EmployeeJson.json"))
        {
            string existingJson = File.ReadAllText("EmployeeJson.json");
            Employees = JsonConvert.DeserializeObject<List<Employee>>(existingJson);
        }

        // LINQ method for searching employee by ID
        if (Employees != null)
        {
            Employee? employeeToRemove = Employees.FirstOrDefault(e => e.EmployeeID == employeeID);

            if (employeeToRemove != null)
            {
                Employees.Remove(employeeToRemove);
            }
        }

        string json = JsonConvert.SerializeObject(Employees, Formatting.Indented);

        File.WriteAllText("EmployeeJson.json", json);

        Console.WriteLine("Employee removed successfully.");
    }
    catch
    {
        Console.WriteLine("Error during removing employee");
    }
}
```

ShowEmployeeInfo() wyświetla informacje o wybranym pracowniku

```
public void ShowEmployeeInfo()
{
    try
    {
        Console.WriteLine();
        Console.Write("Write the employee's ID: ");
        string? employeeIDAsString = Console.ReadLine();

        if (employeeIDAsString == null) { throw new NotImplementedException(nameof(employeeIDAsString)); }
        int employeeID = Int32.Parse(employeeIDAsString);

        if (File.Exists("EmployeeJson.json"))
        {
            string existingJson = File.ReadAllText("EmployeeJson.json");
            Employees = JsonConvert.DeserializeObject<List<Employee>>(existingJson);
        }

        if (Employees != null)
        {
            // LINQ Method for searching eployee by ID
            Employee? employeeShow = Employees.FirstOrDefault(e => e.EmployeeID == employeeID);

            if (employeeShow != null)
            {
                Console.WriteLine($"Employee ID: {employeeShow.EmployeeID}");
                Console.WriteLine($"Name: {employeeShow.Name}");
                Console.WriteLine($"Position: {employeeShow.Position}");
                Console.WriteLine($"Hourly Rate: {employeeShow.HourlyRate}");
            }
            else
            {
                Console.WriteLine("Employee not found.");
            }
        }
    }
```

ModifyEmployee() która zmienia informacje o pracowniku

CalculateSalary() która rachuje zaróbki pracowników w zależności od godzinnej stawki i ilości odpracowanych godzin.

Poniżej są zrzuty ekranu metody **CalcuteSalary()**

```
public void CalculateSalary()
{
    try
    {
        Console.WriteLine();
        Console.Write("Enter Employee ID: ");
        string? employeeIDAsString = Console.ReadLine();

        if (employeeIDAsString == null) { throw new NotImplementedException(nameof(employeeIDAsString)); }

        int employeeID = Int32.Parse(employeeIDAsString);

        if (File.Exists("EmployeeJson.json"))
        {
            string existingJson = File.ReadAllText("EmployeeJson.json");
            Employees = JsonConvert.DeserializeObject<List<Employee>>(existingJson);
        }
```

```
        // LINQ method for searching
        if (Employees != null)
        {
            Employee? employeeForSalary = Employees.FirstOrDefault(e => e.EmployeeID == employeeID);

            if (employeeForSalary != null)
            {
                float salary;
                Console.Write($"Enter hours worked by {employeeForSalary.Name} in the month: ");

                string? hoursAsString = Console.ReadLine();

                if (hoursAsString == null) { throw new NotImplementedException(nameof(hoursAsString)); }
                int hours = Int32.Parse(hoursAsString);

                if (employeeForSalary.Position != null)
                {

                    if (employeeForSalary.Position.Equals("Manager"))
                    {
                        salary = employeeForSalary.HourlyRate * hours + 300;
                    }
                    else
                    {
                        salary = employeeForSalary.HourlyRate * hours;
                    }

                    Console.WriteLine($"{employeeForSalary.Name}'s salary is {salary}");
                }
            }
            else
            {
                Console.WriteLine("Employee not found.");
            }
        }
```

4) Z Enum klasów tylko są trzy klasy dla każdego z ekranów:

MainScreena-a,

ManagerScreen-a,

NormalEmployeeScreen.

MainScrenEnum ma tylko trzy właściwości, bo same z tego ekranu użytkownik może dalej korzystać się z programu.

```
namespace HRHub.Enums;

/// <summary>
/// Enum for MainScreen
/// </summary>
5 references
public enum MainScreenEnum
{
    Exit = 0,
    Manager = 1,
    NormalEmployee = 2
}
```

NormalEmployee ma tylko takie właściwości, ktorę może zrobić zwykły użytkownik lub pracownik.

```
namespace HRHub.Enums;

/// <summary>
/// Enum for NormalEmployeeScreen
/// </summary>
5 references
public enum NormalEmployeeScreenEnum
{
    Exit = 0,
    Info = 1,
    Salary = 2
}
```

ManagerScreenEnum ma więcej właściwości, bo tylko Managery mogą modyfikować informację o pracownikach, dodawać nowych orazzwalniać niepotrzebnych pracowników.

```
/// <summary>
/// Enum list fot ManagerScreen
/// </summary>
8 references
public enum ManagerScreenEnum
{
    Exit = 0,
    Info = 1,
    Add = 2,
    Modify = 3,
    Fire = 4,
    Salary = 5
}
```

5) HrHub.Gui przechowuje wszystkie ekrany programu.

MainScreen ma metodę **Show()**

```
public void Show()
{
    Console.WriteLine("Welcome to HRHub programm");

    while (true)
    {
        try
        {
            Console.WriteLine();
            Console.WriteLine("Your available choises are:");
            Console.WriteLine("1. I'm manager");
            Console.WriteLine("2. I'm a normal employee");
            Console.WriteLine("3. Exit");
            Console.Write("Please enter your choise: ");

            string? userChoise = Console.ReadLine();

            if (userChoise == null)
            {
                throw new ArgumentNullException(nameof(userChoise));
            }

            MainScreenEnum choise = (MainScreenEnum)Int32.Parse(userChoise);

            switch (choise)
            {
                case MainScreenEnum.Manager:
                    CheckAndShow();
                    break;

                case MainScreenEnum.NormalEmployee:
                    _normalEmployeeScreen.Show();
                    Console.Clear();
                    break;
```

```
                case MainScreenEnum.Exit:
                    Console.WriteLine("Goodbye. Thanks for using HRHub app.");
                    return;
                default:
                    Console.WriteLine("Invalid input.");
                    break;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Invalid input. {ex.Message} ");
        }
    }
}
```

Oraz z tym tutaj istnieje prywatna methoda **CheckAndShow()** która sprawdza czy nazwa użytkownika i hasło jest poprawna dla tego żeby sprawdzić czy użytkownik jest Managerem.

ManagerScreen wywołuje metody z klasy Company() za pomocą wstrzykiwania zależności

```
1 reference
public void Show()
{
    while (true)
    {
        try
        {
            Console.WriteLine();
            Console.WriteLine("You are on the manager page. Make your choise");
            Console.WriteLine("0. Exit");
            Console.WriteLine("1. Show info about employee");
            Console.WriteLine("2. Add new employee");
            Console.WriteLine("3. Modify info about employee");
            Console.WriteLine("4. Fire the employee");
            Console.WriteLine("5. Calculate salary");
            Console.Write("Enter your choise: ");

            string? userChoise = Console.ReadLine();

            if (userChoise == null)
            {
                throw new ArgumentNullException(nameof(userChoise));
            }

            ManagerScreenEnum choise = (ManagerScreenEnum)Int32.Parse(userChoise);

            switch (choise)
            {
                case ManagerScreenEnum.Info:
                    _company.ShowEmployeeInfo();
                    break;

                case ManagerScreenEnum.Add:
                    _company.AddEmployee();
                    break;
```

```
                case ManagerScreenEnum.Modify:
                    _company.ModifyEmployeeInfo();
                    break;

                case ManagerScreenEnum.Fire:
                    _company.RemoveEmployee();
                    break;

                case ManagerScreenEnum.Salary:
                    _company.CalculateSalary();
                    break;

                case ManagerScreenEnum.Exit:
                    Console.WriteLine("Going back to main menu.");
                    return;

                default:
                    Console.WriteLine("Invalid input.");
                    break;
            }
        }
        catch
        {
            Console.WriteLine("Invalid input. Try again.");
        }
    }
}
```

NormalEmployeeScreen wywołuje metody właściwe tylko dla zwykłego pracownika

```
public void Show()
{
    while (true)
    {
        try
        {
            Console.WriteLine();
            Console.WriteLine("Welcome to Employee screen");
            Console.WriteLine("Your available choices are:");
            Console.WriteLine("0. Exit");
            Console.WriteLine("1. Show info about employee");
            Console.WriteLine("2. Calculate salary");
            Console.Write("Enter your choise: ");

            string? userChoise = Console.ReadLine();

            if (userChoise == null)
            {
                throw new ArgumentNullException(nameof(userChoise));
            }

            NormalEmployeeScreenEnum choise = (NormalEmployeeScreenEnum)Int32.Parse(userChoise);

            switch (choise)
            {
                case NormalEmployeeScreenEnum.Info:
                    _company.ShowEmployeeInfo();
                    break;

                case NormalEmployeeScreenEnum.Salary:
                    _company.CalculateSalary();
                    break;

                case NormalEmployeeScreenEnum.Exit:
                    Console.WriteLine("Going back to main menu");
                    return;

                default:
                    Console.WriteLine("Invalid input.");
                    break;
            }
```

```
            catch
            {
                Console.WriteLine("Invalid input. Try again.");
            }
        }
    }
```

6) Plik JSON z utworzonymi pracownikami nazywa się EmployeeJson.json:


```
[
  {
    "EmployeeID": 13377,
    "Name": "Stanislav",
    "Position": "Manager",
    "HourlyRate": 43.0
  },
  {
    "EmployeeID": 12345,
    "Name": "Brown",
    "Position": "Seller",
    "HourlyRate": 22.0
  },
  {
    "EmployeeID": 13532,
    "Name": "Mateusz",
    "Position": "Driver",
    "HourlyRate": 20.77
  },
  {
    "EmployeeID": 66666,
    "Name": "Barbara",
    "Position": "Packager",
    "HourlyRate": 25.0
  },
  {
    "EmployeeID": 32154,
    "Name": "Thomas",
    "Position": "Security guard",
    "HourlyRate": 22.0
  },
```

Rozdział 3 – Działanie programu

1) Uruchommy program w imieniu menedżera i sprawdźmy, czy pracownik o id **91234** istnieje.

```
Welcome to HRHub programm

Your available choises are:
1. I'm manager
2. I'm a normal employee
3. Exit
Please enter your choise: 1
Enter login: ManagerWSB
Enter password: qwerty123

You are on the manager page. Make your choise
0. Exit
1. Show info about employee
2. Add new employee
3. Modify info about employee
4. Fire the employee
5. Calculate salary
Enter your choise: 1

Write the employee's ID: 91234
Employee not found.
```

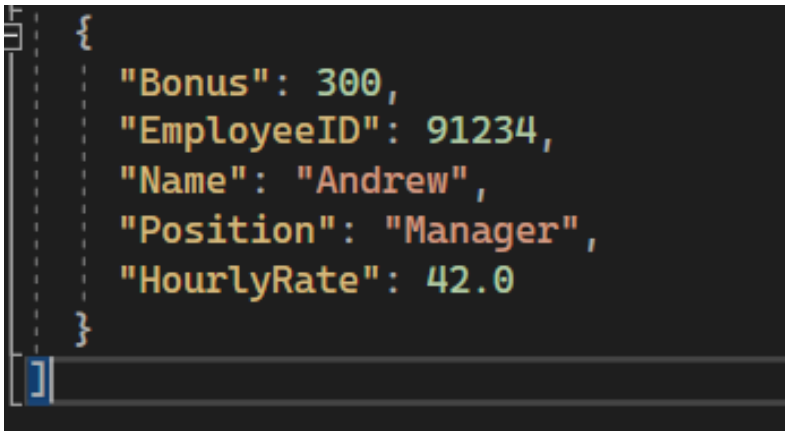
2) Stworzymy nowego pracownika z takim ID:

```
You are on the manager page. Make your choise
0. Exit
1. Show info about employee
2. Add new employee
3. Modify info about employee
4. Fire the employee
5. Calculate salary
Enter your choise: 2
Enter Employee ID: 91234
Enter Employee Name: Andrew
Enter Employee Position: Manager
Enter Employee Hourly Rate: 42
Andrew added successfully.

You are on the manager page. Make your choise
0. Exit
1. Show info about employee
2. Add new employee
3. Modify info about employee
4. Fire the employee
5. Calculate salary
Enter your choise: 1

Write the employee's ID: 91234
Employee ID: 91234
Name: Andrew
Position: Manager
Hourly Rate: 42
```

Również on był dodany do pliku JSON



3) Teraz sprawdźmy jak działa program dla zwykłego pracownika

```
Welcome to HRHub programm

Your available choises are:
1. I'm manager
2. I'm a normal employee
3. Exit
Please enter your choise: 2

Welcome to Employee screen
Your available choices are:
0. Exit
1. Show info about employee
2. Calculate salary
Enter your choise: 1

Write the employee's ID: 32154
Employee ID: 32154
Name: Thomas
Position: Security guard
Hourly Rate: 22
```

Policzymy zaróbki Thomasa:

```
Welcome to Employee screen
Your available choices are:
0. Exit
1. Show info about employee
2. Calculate salary
Enter your choise: 2

Enter Employee ID: 32154
Enter hours worked by Thomas in the month: 168
Thomas's salary is 3696
```

Stwierdzamy, że program działa poprawnie.