МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 2

з дисципліни “ООП”

тема “ C# .Net. Розширені можливості реалізації ООП у мові C#. Події”

Виконав
Студент 2 курсу
групи КП-03
Хоменко Максим Вячеславович
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___ р.
викладач
Заболотня Тетяна Миколаївна
(*прізвище, ім’я, по батькові*)

варіант №20

Київ 2021

Мета роботи

Ознайомитися з такими можливостями мови програмування C# як абстрактні класи, інтерфейси, делегати. Вивчити механізми оброблення подій у C#, а також можливості, які мають методи-розширення.

Постановка завдання

Постановка задачі

Для ієрархії класів, побудованої в лабораторній роботі №1, реалізувати:

- Множину інтерфейсів. При чому один з класів повинен реалізовувати щонайменше 2 інтерфейси. Також продемонструвати реалізацію explicit implementation інтерфейса, обгрунтувати її використання **(1 бал)**.
- Абстрактний клас. Забезпечити його наслідування. Наявність в цьому класі абстрактних методів - обов'язкова **(1 бал)**.
- Механізм «делегат – подія – обробник події» **(2 бали)**.
- Перетворити код, який забезпечує роботу з подіями та обробниками подій, на код, що використовує (*) **(2 бали)**:
  - анонімні методи;
  - lambda-вирази;
  - типи Action та Func (кожен з них).

  (*) - допускається реалізація коду однієї події різними способами, необов'язково різних подій.
- Механізм створення та оброблення власних помилок **(2 бали)**:
  - створити новий клас виключної ситуації;
  - створити новий клас аргументів для передачі їх до обробника виключної ситуації;
  - забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки;
  - реалізувати різні сценарії оброблення помилки.
- Метод-розширення будь-якого класу **(1 бал)**.

Фрагменти коду

1. реалізувати множину інтерфейсів. При чому один з класів повинен реалізовувати щонайменше 2 інтерфейси. Також продемонструвати реалізацію explicit implementation інтерфейса, обгрунтувати її використання

Parallelogram.cs

```
using Lab1_Encapsulation_Inheritance_polymorphism.interfaces;
using System;
using System.Threading;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
	public class Parallelogram : Rectangle, IBuildable, IBuildableQuickly, IMoveable
	{
		public Parallelogram(double aSide, double bSide) : base(aSide, bSide)
		{
			Console.WriteLine($"{nameof(Parallelogram)} -> derived ctor executed");
		}

		public void Move(int x, int y)
		{
			Console.WriteLine($"the figure placed into ({x};{y})");
		}
```

```
            // implements two interfaces at once
            //public void Build()
            //{
            //      throw new NotImplementedException();
            //}

            // implementing IBuildable
            public void Build()
            {
                  int proggressUpperbound = 100;
                  for (int i = 0; i < proggressUpperbound; i += 10)
                  {
                        Console.WriteLine($"Building triangle {i}% progress");
                        Thread.Sleep(2000);
                  }
            }

// explicit interface implementation allows us to use specifically this method through the interface-type reffernce
// as the result we can not just build but build quickly
            void IBuildableQuickly.Build()
            {
                  int proggressUpperbound = 100;
                  for (int i = 0; i < proggressUpperbound; i += 10)
                  {
                        Console.WriteLine($"Building triangle {i}% progress");
                        Thread.Sleep(100);
                  }
            }

      }
}
```

interfaces

```
namespace Lab1_Encapsulation_Inheritance_polymorphism.interfaces
{
      public interface IBuildable
      {
            void Build();
      }

      public interface IBuildableQuickly
      {
            void Build();
      }

      public interface IMoveable
      {
            void Move(int x, int y);
      }

      public interface IPaintable
      {
            void Paint();
      }
```

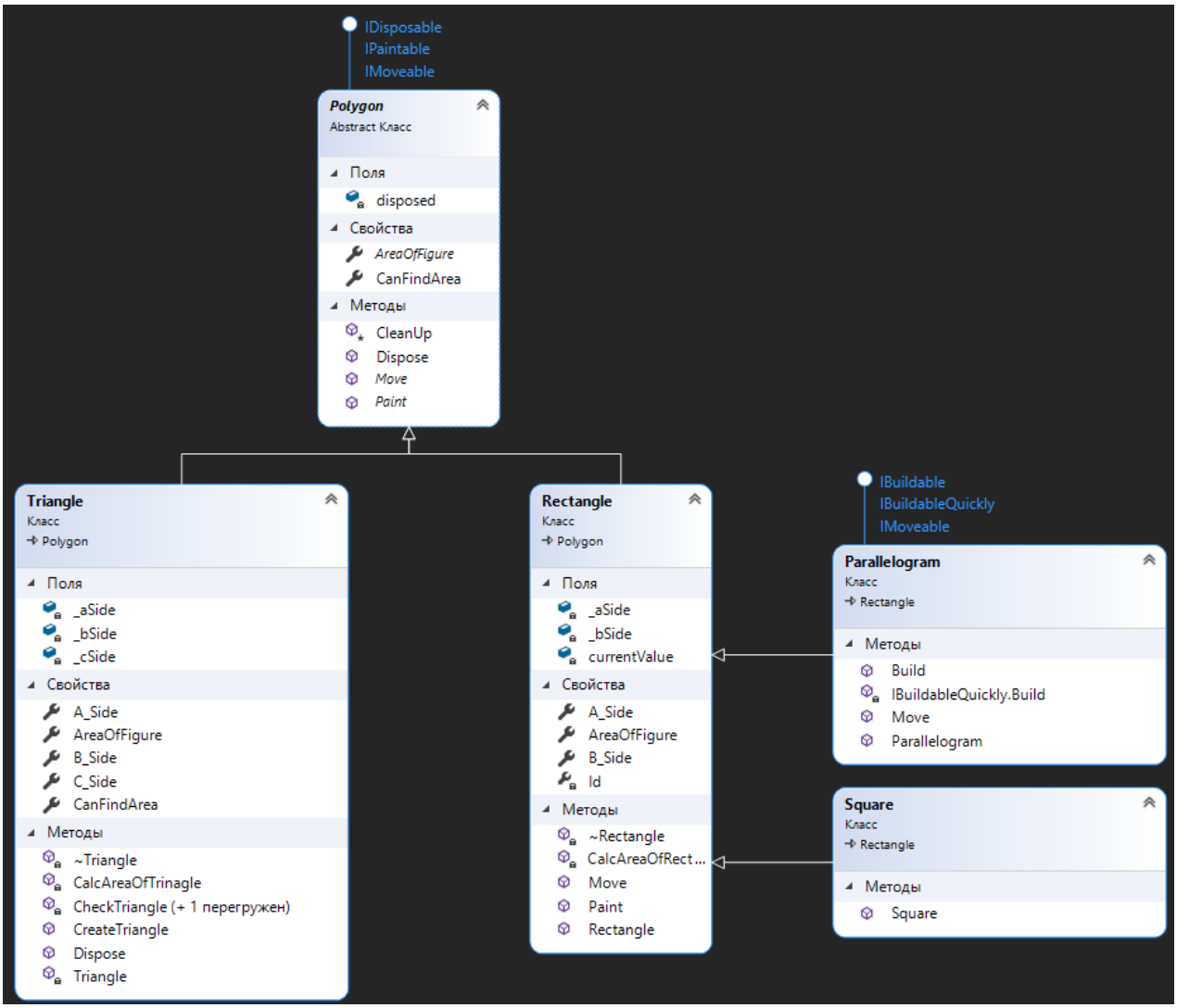
```
}
```

2. Абстрактний клас.
   Абстрактний клас

Polygon.cs

```
using Lab1_Encapsulation_Inheritance_polymorphism.interfaces;
using System;

namespace Lab1_Encapsulation_Inheritance_polymorphism.Models
{
	public abstract class Polygon : IDisposable, IPaintable, IMoveable
	{
		public virtual bool CanFindArea => true;
		public abstract double AreaOfFigure { get; }

		public abstract void Paint();

		public abstract void Move(int x, int y);

		// dispose
		private bool disposed = false;
		protected virtual void CleanUp(bool disposing) // in general
		{
			if (!disposed)
			{
				if (disposing)
				{
					//clean managed resources
				}
				// clean unmanaged resources
			}
			disposed = true;
		}
		public virtual void Dispose()
		{
			CleanUp(true);
			GC.SuppressFinalize(this);
		}

	}
}
```

Забезпечити його наслідування. Наявність в цьому класі *абстрактних методів* (абстрактні методи позначені курсивом: Move, Paint)



3. Механізм «делегат – подія – обробник події»

```
class Program
{
    Ссылок: 0
    static void Main(string[] args)
    {
        Rectangle r = new(5, 4);

        Client c = new();

        c.AddAvalableFigure(new KeyValuePair<string, Polygon>(nameof(Rectangle), r));

        r.FigureMoved += new Func<Point, Polygon, bool>(OnMoved); // delegate
        r.FigurePainted += OnPaintedLog;

        c.PaintRectangle();
    }
```

```
Ссылок: 14
public class Rectangle : Polygon
{
    public override event Func<Point, Polygon, bool> FigureMoved; // event
```

```
ссылка: 1
private static bool OnMoved(Point location, Polygon senderObj) // event handler
{
    if (!double.IsNaN(location.x) && !double.IsNaN(location.y))
    {
        Console.WriteLine( $"{senderObj.GetType()} has been moved at ({location.x};{location.y})");
        return true;
    }

    return false;
}
```

4. Перетворити код, який забезпечує роботу з подіями та обробниками подій, на код, що використовує

```
r.FigureMoved += new Func<Point, Polygon, bool>(OnMoved); // delegate func
r.FigurePainted += new Action<Polygon>(OnPaintedLog); // delegate action
r.FigureMoved += delegate (Point location, Polygon sender)  // subscription using anonymus method
{
    return OnMoved(location, sender);
};
r.FigurePainted += (obj) => OnPaintedLog(obj); // subscription using labda-expression
```

5. Механізм створення та оброблення власних помилок

створити новий клас виключної ситуації

FigureNotMovedException.cs

```
public class FigureNotMovedException : ApplicationException
{

}
```

створити новий клас аргументів для передачі їх до обробника виключної ситуації

FigureNotMovedException.cs

```
public record FigureNotMovedExceptionArgs(string Message, Exception InnerException);

       public class FigureNotMovedException : ApplicationException
       {
              public FigureNotMovedException() : base()
              {

              }

              public FigureNotMovedException(string message) : base(message)
              {

              }

              public FigureNotMovedException(string message, Exception innerException) : base(message,
innerException)
              {

              }

              public FigureNotMovedException(FigureNotMovedExceptionArgs arg) : base(arg.Message,
arg.InnerException)
              {

              }
       }
```

забезпечити ініціювання створеної виключної ситуації та продемонструвати, як працює обробник даної помилки

```
c.PaintRectangle();
c.MoveRectangle(double.NaN, double.PositiveInfinity); // initiation of exception
```

Program.cs

```
private static bool OnMoved(Point location, Polygon senderObj) // event handler
		{
			//if (!double.IsNaN(location.x) && !double.IsNaN(location.y))
			//{
			//	Console.WriteLine( $"{senderObj.GetType()} has been moved at
({location.x};{location.y})");
			//	return true;
			//}

			// code using app exception

			try
			{
				if (double.IsNaN(location.x) || double.IsNaN(location.y))
				{
					throw new FigureNotMovedException(new
FigureNotMovedExceptionArgs($"Figure is not moved due to invalid args!", null));
				}

				Console.WriteLine($"{senderObj.GetType()} has been moved at
({location.x};{location.y})");
			}
			catch (FigureNotMovedException e)
			{
				Console.WriteLine($"{nameof(FigureNotMovedException)}: Message: {e.Message};
Source: {e.Source}");
				return false;
			}

			return true;
		}
```

```
try
{
    if (double.IsNaN(location.x) || double.IsNaN(location.y))
    {
        throw new FigureNotMovedException(new FigureNotMovedExceptionArgs($"Figure is not moved due to invalid args!", null));
    }

    Console.WriteLine($"{senderObj.GetType()} has been moved at ({location.x};{location.y})");
}
catch (FigureNotMovedException e)
{
    Console.WriteLine($"{nameof(FigureNotMovedException)}: Message: {e.Message}; Source: {e.Source}");
    return false;
}

return true;
```

Console output

```
Rectangle -> base ctor 1 executed
Paint Rectangle
Rectangle has been painted
Move Rectangle
FigureNotMovedException: Message: Figure is not moved due to invalid args!; Source:
Lab1_Encapsulation_Inheritance_polymorphism
```

реалізувати різні сценарії оброблення помилки

```
try
{
    if (double.IsNaN(location.x) || double.IsNaN(location.y))
    {
        throw new FigureNotMovedException(
            new FigureNotMovedExceptionArgs($"Figure is not moved due to invalid args!", null));
    }

    Console.WriteLine($"{senderObj.GetType()} has been moved at ({location.x};{location.y})");
}
catch (FigureNotMovedException e)
{
    string output = $"{nameof(FigureNotMovedException)}:";

    if (e.Source != null)
    {
        output += $" Source: {e.Source};";
    }

    if (e.Message != null)
    {
        output += $" Message: {e.Message};";
    }

    if (e.InnerException != null)
    {
        output += $"Inner: {e.InnerException.Message};";
    }


    return false;
}

return true;
```

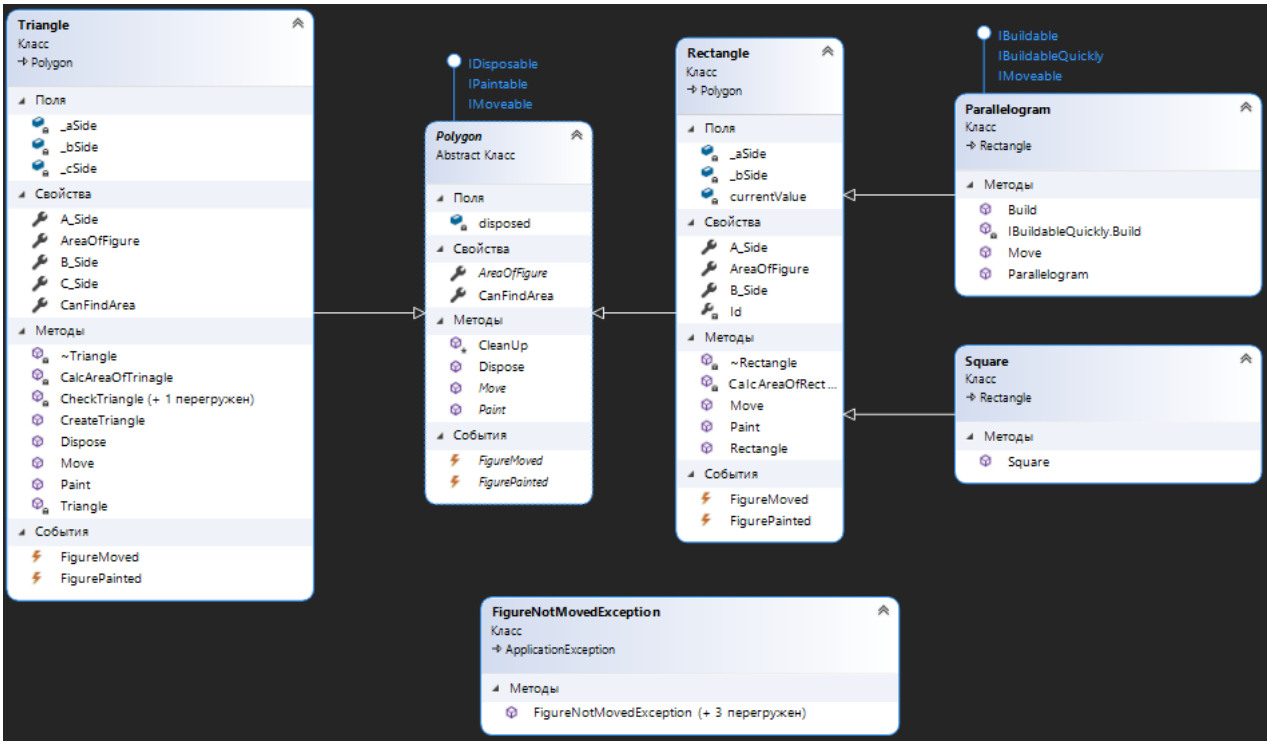
Метод-розширення будь-якого класу

Program.cs

```
public static class Extensions //
	{
		public static void AddAvaliableFigure(this Dictionary<string,Polygon> dictionary,
KeyValuePair<string, Polygon> figureForDict)
		{
			dictionary.Add(figureForDict.Key, figureForDict.Value);
		}
	}
```

```
public class Client
	{
		private Dictionary<string, Polygon> _avaliableFigures = new();

		public void AddAvalableFigure(KeyValuePair<string,Polygon> figure)
		{
			_avaliableFigures.AddAvaliableFigure(figure); // extension in use
		}
		 …
}
```

UML-діаграми



Висновок

Виконавши дану роботу, ми знайомитися з такими можливостями мови програмування C# як абстрактні класи, інтерфейси, делегати. Вивчили механізми оброблення подій у C#, а також можливості, які мають методи-розширення.