Лабораторна робота №8

Інтерфейси та внутрішні класи у мові C#

Мета лабораторної роботи:	1. Вивчити основні концепції використання інтерфейсів C#. Ознайомитись із методами використання внутрішніх класів. 2. Вивчити прийоми програмування з допомогою інтерфейсів мовою C#.

Зміст роботи.

1. Теоретичні відомості

1.1 Інтерфейси в C#

Інтерфейс - інший (короткий) метод завдання типу об'єктів. У ньому вказуються лише заголовки властивостей та методів. Щоб побудувати об'єкт цього типу, треба спочатку зробити клас, успадковуючи інтерфейс і реалізує всі оголошені в інтерфейсі члени. Об'єкти цього класу матимуть тип інтерфейсу та тип класу.

Так обходиться обмеження на спадкування: у класу може бути не більше одного безпосереднього класу-батька, але може бути ще багато додаткових спадкодавців-інтерфейсів.

Інтерфейси широко використовуються для завдання типів формальних параметрів у методів, призначених для обробки об'єктів із різних класів (аналогічно типу загального батька всіх цих класів, але батько - один, а інтерфейсів - багато).

Приклад використання інтерфейсу

2. Завдання на лабораторну роботу

Реалізувати абстрактні класи та інтерфейси, а тестову програму для класів, згідно з варіантом.

Умову завдання згідно з варіантом можна подивитися тут.

3. Звіт

Повинен містити: постановку завдання, програмний код розв'язання, результат роботи написаної програми

Умова Завдання

Реалізувати абстрактні класи та інтерфейси, а також успадкування та поліморфізм для наступних класів. Розробити тестову програму.

1. **interface Абітурієнт←abstract class Студент ← class Студент-Заочник**.
2. **interface Співробітник←abstract class Інженер ← class Керівник**.
3. **interface Будівля← abstract class Громадська Будівля ← class Театр**.
4. **interface Mobile ← abstract class Siemens Mobile ← class Model.**
5. **interface Корабель ← abstract class Військовий Корабель ← class Авіаносець.**
6. **interface Лікар ← abstract class Хірург ← class Нейрохірург**.
7. **interface Корабель ← abstract class Вантажний Корабель ← class Танкер**.
8. **interface Меблі ← abstract class Шафа ← class Книжкова Шафа**.
9. **interface Фільм ← abstract class Вітчизняний Фільм ← class Комедія**.
10. **interface Тканина ← abstract class Одяг ← class Костюм**.
11. **interface Техніка ← abstract class Плеєр ← class Відеоплеєр**.
12. **interface Транспортний Засіб ← abstract class Громадський Транспорт ← class Трамвай**.
13. **interface Пристрій Друку ← class Принтер ← class Лазерний Принтер**.
14. **interface Папір ←abstract class Зошит ← class Зошит Для Малювання**.
15. **interface Джерело Світла ← abstract class Лампа ← class Настільна Лампа**.
16. **interface Будівля ← abstract class Громадська Будівля ← class Театр**.
17. **interface Mobile ← abstract class Siemens Mobile ← class Model**.
18. **interface Співробітник ← abstract class Інженер ← class Керівник**.
19. **interface Будівля ← abstract class Громадська Будівля ← class Театр**.
20. **interface Техніка ← abstract class Плеєр ← class Аудіоплеєр**.

Приклад використання інтерфейсу

```
using System;
namespace InterfacesGuide
{
    /* A class can be treated as some sort of type definition
     * that may also containe деякі parts of implementation.
     * Interfaces define types more briefly (as completely
     * Abstract classes without any implementation).
     * Interfaces and interface members are abstract;
     * interfaces do not provide a default implementation.
     *
     * Interfaces може бути зроблено з методів, properties, events, indexers,
     * або будь-яка комбінація з чотирма member types.
     * An interface can not contain fields.
     * Interface members є автоматично public.
     *
     * An interface cannot be instantiated безпосередньо.
     * A class can inherit from a single base class only,
     * but it can inherit from (one or many) interfaces (structs too).
     * To implement an interface member,
     * the corresponding member on the class must be public, non-static,
     * and have same name and signature as the interface member.
     * Interfaces може вносити інші interfaces.
     *
     * Add when neccesary:
     * The IComparable interface announces to user of the object
     * that the object can compare itself to other objects of the same type,
     * and the user of the interface does не потребує знати, що це implemented.
     */

    interface I1
    {
        void Meth1();
        void Meth2();
        void Paint();
    }
```

```
interface I2
{
    void Meth3();
    void Paint();
}
public class CC: I1, I2
{
    public void Meth1()
    {
        Console.WriteLine("It is Meth1()");
    }
    public void Meth2()
    {
        Console.WriteLine("It is Meth2()");
    }
    public void Meth3()
    {
        Console.WriteLine("It is Meth3()");
    }
    public void Paint()
    {
        Console.WriteLine(" :-) ");
    }
}

class Program
{
    static void Main()
    {
        I1 = new CC(); // Creates an object of class CC and assigns it to a variable of type I1
        a.Paint();
        a.Meth1();

        // a.Meth3(); //Error: I1 не має значення Meth3()
        I2 b = (I2)a; // Use cast to do type conversion explicitly
        b.Meth3(); // It works! So a:I1 hides Meth3 but does not remove it.

        Console.WriteLine("\n Function FF is of type (I1 -> void ):");
        Console.WriteLine("\nFunction call FF(I1 a) :");
```

```
            FF(a);
            Console.WriteLine("\nFunction call FF(CC c ) where C:I1 :");
            CC c = new CC();
            FF(c);

            Console.WriteLine("\n Function GG is of type (I2 -> void):");
            Console.WriteLine("\nFunction call GG(CC c ) where C:I2 :");
            GG(c);


        }

        static void FF(I1 x)
        {
            x.Meth1(); x.Meth2(); x.Paint();
        }

        static void GG(I2 x)
        {
            x.Meth3(); x.Paint();
        }
    }
}
```

Можна з користю успадковувати від двох інтерфейсів з однаково описаним у них методом, пропонуючи не загальну реалізацію цього, а дві різні - дивимося приклад.

```
using System;
namespace InterfacesExplImpl
{
/* Explicit interface implementation also allows the programmer
* implement two interfaces that have the same member names
* and give each interface member a separate implementation.
*/
// Declare the English units interface:
interface IEnglishDimensions
{
float Length();
```

```
float Width();
}
// Declare the metric units interface:
interface IMetricDimensions
{
float Length();
float Width();
}
// Declare the Box class that implements the 2 interfaces:
// IEnglishDimensions and IMetricDimensions:
class Box : IEnglishDimensions, IMetricDimensions
{
float lengthInches;
float widthInches;
Public Box(float length, float width)
{
lengthInches = length;
widthInches = width;
}
// Normal implementation (for default inches):
public float Length()
{
return lengthInches;
}
public float Width()
{
return widthInches;
}
// Explicit implementation (for cm.):
float IMetricDimensions.Length()
{
return lengthInches * 2.54f;
}
float IMetricDimensions.Width()
{
return widthInches * 2.54f;
}
}
class Program
```

```
{
static void Main()
{
Box box1 = новий Box(30.0f, 20.0f);
IMetricDimensions mDimensions = (IMetricDimensions)box1;
System.Console.WriteLine("Length(in): {0}", box1.Length());
System.Console.WriteLine("Width(in): {0}", box1.Width());
System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
System.Console.WriteLine("Width(cm): {0}", mDimensions.Width());
}
}
}
```