Лабораторна робота №7

Тема: Використання інтерфейсів у C#.

Мета роботи: Набуття навичок розробки класів з використанням інтерфейсів та базового принципу об'єктно-орієнтованого програмування – успадкування.

Теоретичні відомості. Інтерфейс – це «полегшений» клас, де всі функції є віртуальними і відсутні поля (але можуть бути властивості). Кількість функцій, визначених у конкретному інтерфейсі, залежить від того, яку поведінку ми намагаємося змоделювати за допомогою цього інтерфейсу. З погляду синтаксису інтерфейси C# визначаються наступним чином:

```
public interface IPoint
{
  byte GetNumberOfPoints();
}
```

В цьому прикладі функція *GetNumberOfPoints* автоматично стає віртуальною. Інтерфейси *.NET* також можуть підтримувати будь-яку кількість властивостей (і подій). Наприклад, інтерфейс *IPoints* містить властивість *Points* для читання та запису:

```
public interface IPoint
{
  int Points (get; set;)
}
```

У будь-якому випадку інтерфейс – це не більше ніж іменований набір абстрактних членів, а це означає, що будь-який клас, що реалізує цей інтерфейс, повинен самостійно повністю визначати кожен із членів цього інтерфейсу. Більше того, кожен метод, що визначається, автоматично стає віртуальним – оператор *override* використовувати не треба. Таким чином, інтерфейси – це ще один спосіб реалізації поліморфізму у застосунку: оскільки в різних класах функції одних і тих же інтерфейсів будуть реалізовані по-різному, тому в результаті ці класи будуть реагувати на ті самі виклики по-своєму. Інтерфейс – це чиста синтаксична конструкція, яка призначена лише для певних цілей. Інтерфейси в жодному разі не є типами даних, і у них не буває

реалізацій методів за умовчанням. Кожен член інтерфейсу (будь то властивість чи метод) автоматично стає віртуальним. У C# множинне спадкування заборонено; водночас реалізація у класі кількох інтерфейсів (тобто успадкування кількох інтерфейсів) – це звичайна справа.

Реалізація інтерфейсу. Коли в C# який-небудь клас має реалізовувати потрібні нам інтерфейси, назви цих інтерфейсів мають бути записані (через кому) після двокрапки, що іде за ім'ям класу. Ім'я базового класу (якщо воно є) має стояти перед іменами будь-яких інтерфейсів:

```
public class Triangle: Shape, IPoint
{
  public Triangle (){}
  public Triangle (string name) : base(name){}
  //Реализация IPoint
  public byte GetNumberOfPoints()
  {
    return 3;
  }
}
```

В класі мають бути реалізовані всі методи використовуваного інтерфейсу, інакше станеться помилка компіляції.

Успадкування інтерфейсів та успадкування реалізації. Успадкування інтерфейсів (успадкування від інтерфейсу) дозволяє визначити всю реалізацію з нуля, також у наслідуванні інтерфейсів немає проблеми з однаковими іменами функцій. Добре все починати з чистого листа, але це складно і на практиці нерідко необхідно використовувати раніше розроблену реалізацію, використовуючи звичайне успадкування. Але у спадкуванні реалізації є деякі проблеми. Часто при успадкування реалізації нам необхідна лише частина функцій з успадкованого класу, решта реалізації стає тягарем, причому не тільки для нашого класу, але і для тих класів, які будуть успадковані від нашого. Розглянемо ситуації, коли вибирається успадкування інтерфейсів, а коли успадкування реалізації.

Приклад1. Успадкування реалізації: Клас *Control*, від нього успадковані *Button, CheckBox, ListBox*. У класі *Control* реалізована велика функціональність, а класи *Button, CheckBox, ListBox* успадковують усю цю функціональність та вносять деякі невеликі доповнення. Таким чином, класи *Button, CheckBox, ListBox* сильно зв'язані і основний код міститься в єдиному вигляді в реалізації класу *Control*. І тут виправдано використання успадкування реалізації.

Приклад 2. Наслідування інтерфейсу: Класи ArrayList, Queue, HashTable реалізують в собі кілька інтерфейсів: *IList, IEnumerable, ICollection*. Усередині класи *ArrayList, Queue, HashTable* реалізовані абсолютно по-різному, тобто. слабо зв'язані, тому тут виправдано використання успадкування інтерфейсу.

Отримання посилання на інтерфейс. Перший спосіб – скористатись явним приведенням типів:

```
Triangle tr = new Triangle();
IPoint p = (IPoint) tr;
Console.WriteLine (p.GetNumberOfPoints());
```

Однак якщо ми спробуємо застосувати те саме приведення типів до об'єкта класу, що не підтримує *IPont*, ми отримаємо повідомлення про помилку часу виконання. Коли ми намагаємося отримати посилання на інтерфейс шляхом явного приведення типів для об'єкта класу, що не підтримує цей інтерфейс, система генерує виняток *InvalidCastExeption*. Щоб уникнути проблем із винятком, потрібно його перехопити:

```
Triangle tr = new Triangle ();
IPoint p;
try
{
  p = (IPoint)tr;
  Console.WriteLine (p.GetNumberOfPoints() );
}
catch (InvalidCastExeption e)
{
  Console.WriteLine ("Not point");
}
```

Другий спосіб отримати посилання на інтерфейс – використати ключове слово *as*:

```
Triangle tr = new Triangle();
IPoint p;
p = tr as IPoint;
```

```
if ( p != null)
  Console.WriteLine(p.GetNumberOfPoints());
else Console.WriteLine("Not point");
```

Якщо при використанні ключового слова *as* ми спробуємо створити посилання на інтерфейс через об'єкт, який цей інтерфейс не підтримує, посилання просто буде встановлене в *null*, і при цьому жодних винятків не буде генеруватися.

Третій спосіб отримання посилання на інтерфейс – скористатися операцією *is*. Якщо об'єкт не підтримує інтерфейс, умова дорівнює *false*:

```
Triangle tr = new Triangle();
if (tr is IPoint)
  Console.WriteLine(p.GetNumberOfPoints());
else Console.WriteLine("Not point");
```

Вирішення проблеми з однаковими іменами функцій. Опишемо два інтерфейси з функцією *Menu*. У інтерфейсів вона виконує різні функції. В C# є можливість реалізувати обидва інтерфейси в одному класі, розрізняючи два *Menu*, що відносяться до різних інтерфейсів:

```
interface IRestaurant
{
  void Menu();
}
interface IWindow
{
  void Menu();
}
class RestaurantForm: Form, IRestaurant, IWindow
{
  void IRestaurnt.Menu(){...}//функція Menu,що відноситься
                                до інтерфейсу IRestaurant
  void IWindow.Menu(){...} //функція Menu, що відноситься до
                                інтерфейсу IWindow
```

```
  public void Menu(){...}//функція Menu саме RestaurantForm
}
```

Як цім скористатись?

```
void f (IReataurant r)
{
  r.Menu();
}
RestaurantForm r = new RestaurantForm();
f(r); //викликається функція Menu, що відноситься до
         інтерфейсу IRestaurant
```

Якщо в програмі написати просто *r.Menu()*, то викликається *Menu* класа *RestaurantForm*, а не якого-небудь інтерфейсу. Виклик *(r as IWindow).Menu();* приведе к виклику функції *Menu* интерфейсу *IWindow*.

Якщо ж у класі визначена тільки остання функція *Menu*, то у всіх трьох прикладах викличеться саме вона.

Завдання для самостійної роботи

У кожному із завдань для тестування створити програму-клієнт.

1. Створити клас, що містить методи знаходження площі, периметра та типу опуклого чотирикутника, який може задаватися довжинами сторін або координатами вершин. Створити відповідні інтерфейси для випадку завдання чотирикутника довжинами сторін та координатами вершин.
2. Створити клас, що містить методи знаходження площі, периметра, радіуса вписаного і описаного кола для рівностороннього, рівнобедреного, різностороннього та прямокутного трикутників. Для різного типу трикутників передбачити відповідні інтерфейси.
3. Створити клас, що містить методи визначення паралельності та перпендикулярності прямих на площині, що задані різними способами. Для різних способів задання прямих передбачити відповідні інтерфейси.
4. Створити клас, що містить методи знаходження розв’язків системи двох лінійних рівнянь з двома невідомими та системи трьох лінійних рівнянь з трьома невідомими (коефіцієнти можуть бути задані як десяткові дробів або як раціональні дроби). Для системи двох

лінійних рівнянь з двома невідомими та системи трьох лінійних рівнянь з трьома невідомими передбачити відповідні інтерфейси.

5. Створити клас, що містить методи розв'язання квадратного рівняння (знаходження розв'язків розглядати як у R, так і просторі комплексних чисел). Для різних випадків передбачити відповідні інтерфейси.
6. Створити клас, що містить методи додавання, віднімання, множення та ділення раціональних дробів та такі ж методи для роботи з комплексними числами. Для випадку раціональних дробів та випадку комплексних чисел передбачити відповідні інтерфейси.
7. Створити клас, що містить методи для знаходження відсотку від числа, збільшення, зменшення числа на певну кількість відсотків (число може бути десятковим або раціональним дробом). При створенні класу використати відповідні інтерфейси.
8. Створити клас, що містить методи для знаходження суми і різниці чисел, які можуть бути або раціональними дробами, або числами, записаними за допомогою римських цифр. При створенні класу використати відповідні інтерфейси.
9. Створити клас, що містить методи для знаходження суми і різниці цифр, які можуть бути арабськими, римськими або задані прописом. При створенні класу використати відповідні інтерфейси.
10. Створити клас, що містить методи знаходження суми, добутку та середнього арифметичного елементів масиву, який може бути статичним або динамічно створеним. При створенні класу використати відповідні інтерфейси.
11. Створити інтерфейс, що містить опис методів для роботи з векторами (знаходження довжини вектора, суми векторів, множення вектора на число, скалярного добутку) та класи, які містять реалізації методів інтерфейсу у випадку двовимірних та тривимірних векторів.
12. Створити інтерфейс, що містить опис методів для роботи з двовимірними векторами (знаходження довжини вектора, суми векторів, множення вектора на число, скалярного добутку) та класи, які містять реалізації методів інтерфейсу у випадку, коли вектор задано його координатами та координатами початку і кінця.
13. Створити інтерфейс, що містить опис методів для знаходження n-го члена та суми перших n членів прогресії. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку арифметичної та геометричної прогресій.
14. Створити інтерфейс, що містить опис методів для знаходження суми цифр числа, а визначення кількості нулів у записі числа. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку цілого та дійсного чисел

15. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку елементів, що є цілими числами типу Byte. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у множині та одновимірному масиві.
16. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку елементів, що є цілими числами. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у стеку та черзі.
17. Створити інтерфейс, що містить опис методів додавання, вилучення та пошуку дійсних чисел, які зберігаються у файлі. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи числа у текстовому та типізованому файлі дійсних чисел.
18. Створити інтерфейс, що містить опис методів роботи з матрицями (знаходження детермінанта, сліду, рангу матриці), та класи, що містять реалізації методів інтерфейсу у випадку квадратних матриць порядку 2 і 3.
19. Створити інтерфейс, що містить опис методів знаходження суми, найбільшого, найменшого елементів. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу, зберігаючи елементи у одновимірному та двовимірному масивах.
20. Створити інтерфейс, що містить опис методів визначення відстані від точки до площини в R^3 та метод перевірки належності точки деякій площині. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку задання площини канонічним рівнянням та за допомогою трьох точок.
21. Створити інтерфейс, що містить опис методів знаходження площі поверхні, площі основи та об’єму правильної піраміди. На основі цього інтерфейсу створити класи, що містять реалізації методів інтерфейсу у випадку трикутної та чотирикутної пірамід.

Контрольні запитання

1. Що таке інтерфейс?
2. Що спільного між інтерфейсами та абстрактними класами?
3. Чим відрізняються інтерфейси від абстрактних класів?
4. Який інтерфейс є базовим для всіх інтерфейсів?
5. Що повинен містити клас, який реалізує інтерфейс?
6. Чи обов’язково клас, який реалізує інтерфейс має містити реалізації всіх його методів?
7. Чи можна змінній типу інтерфейс надавати значення об’єкта типу класу, в якому реалізовано цей інтерфейс?
8. Чи може інтерфейс мати багато реалізацій?