Практична робота №7

Класи, структури та колекції C#

Мета роботи. Розглянути класи, опис їх полів, методів і властивостей, їх відмінності від структур, колекції.

Постановка завдання. У середовищі Microsoft Visual Studio виконати завдання №1, 2, за інструкціями в теоретичній частині.

Теоретичні відомості

Класи

Мова C# повною мірою є об'єктно-орієнтованою. З представленого вище матеріалу видно, що всі дії, здійснювані в C#, є зверненнями до ресурсів якихось класів. Іноді для цього необхідно створювати об'єкти класів, іноді (при виклику статичних методів) створення об'єктів не потрібне.

Однією з найважливіших особливостей C# є те, що тут не можна створити процедуру, або функцію що окремо існує, – вона обов'язково повинна входити в який-небудь клас.

Клас – це шаблон, який визначає властивості і методи об'єктів, що створюються як екземпляри даного класу.

Основне призначення класу – створювати якийсь тип даних, який задає реалізацію деякої абстракції даних, характерної для завдання, на користь якого створюється програмна система. Таким чином, класи – не просто цегла, з якої будується система. Кожна цеглинка тепер має важливу змістовну начинку, що відповідає за виконання своєї операції.

Визначення класу в C# практично ідентично визначенню такого ж класу в C++. Проте в C# існують додаткові елементи, що значно полегшують практичне використання класів. Розглянемо і засвоїмо основні правила побудови і використання класів, виконуючи послідовно завдання.

Завдання 1

Для системи обробки банківських операцій, здійснюваних клієнтами, необхідно створити клас Client, що містить інформацію про ім'я, паспорт і дату народження позичальника, властивості для доступу до цих даних, а також метод, за допомогою якого можливо редагувати інформацію про клієнта.

Визначення класу почнемо з оголошення його імені, а також завдання його полів і необхідних методів (конструкторів). Визначення класу виглядатиме таким чином:

```
public class Client
{
  private string Name;
  private string Passport;
```

```
  private DateTime BirthDate;

  public Client()
  {
  }
}
```

Тут визначені поля класу, а також конструктор за умовчанням (без параметрів). Для класів C# можна визначити будь-яку кількість конструкторів. Якщо в класі не визначений жоден конструктор, використовується конструктор за умовчанням, який при створенні об’єкту автоматично привласнить всім змінним – членам класу безпечні значення. В даному випадку, конструктор за умовчанням задається явно.

Зона видимості полів класу відповідно до правил має бути визначена або як закрита, або як захищена (у випадку, якщо передбачається спадкоємство його полів). Доступ же до полів – членам класу має бути організований або за допомогою методів, або за допомогою властивостей класу. Створимо властивості класу Client, забезпечуюче читання і запис значень полів класу.

```
public string passport
{
  get
  {
    return Passport;
  }
  set
  {
    Passport = value;
  }
}

public string name
{
  get
  {
    return Name;
  }
  set
  {
    Name = value;
  }
}

public int age
{
```

```
  get
  {
    int a;
    a = DateTime.Now.Year – BirthDate.Year;
    return a;
  }
}

public DateTime birthdate
{
  get
  {
    return BirthDate;
  }
  set
  {
    if (DateTime.Now > value)
      BirthDate = value;
    else
      throw new Exception(«Введена неправильна дата народження»);
  }
}
```

Як видно з даного прикладу, властивість складається з методів *set і get.* При цьому властивість повинна містити хоч би один з методів.

Set дозволяє змінювати значення поля класу, *get* – набувати значення. У метод *Set* передається значення параметра за допомогою змінної *value*. Обидва методи можуть містити довільну кількість операторів, що описують алгоритм виконання дій в процесі читання або запису значення в полі класу. У даному прикладі властивості *passport і name* дозволяють просто дістати доступ до полів класу, прочитуючи або встановлюючи значення відповідних змінних.

Властивість *birthdate* також призначена для читання і запису значення змінної – члена класу *BirthDate*. При цьому при читанні значення (операція get) відбувається просто передача значення змінній *BirthDate*, при спробі ж запису нового значення в цю змінну відбувається перевірка допустимості встановлюваного значення змінної. В даному випадку перевірка зводиться до порівняння нового значення дати народження з поточною датою. Якщо встановлюване значення дати народження більше або дорівнює поточній даті, генерується виключення, яке не дозволяє записати нове значення в змінну, – член класу.

Властивість *age* застосовується для отримання поточного віку клієнта. Воно призначене тільки для читання значення із змінної, тому містить лише метод *get*. При використанні властивості *age* відбувається обчислення поточного значення віку

клієнта в літах шляхом віднімання року народження з поточного значення року. Використання властивостей аналогічно використанню змінних.

Завдання 2

Створити об'єкт c1 класу Client. Поля даного об'єкту заповнити значеннями з використанням властивостей. Організувати на екран вивод значеннь полів, для цього також застосовуються властивості класу:

```
Client c1=new Client();
c1.name = "Іваненко";
c1.passport = "9002";
c1.birthdate = new DateTime(1967, 08, 03);
Response.Write("Ім'я=" + c1.name+"</br>");
Response.Write("Паспорт=" + c1.passport+"</br>");
Response.Write("Вік="+c1.age);
```

Окрім змінних – членів класу і властивостей для доступу до цих змінних практично будь-який клас містить методи, призначені для опису алгоритмів виконання класом дій. Один з обов'язкових методів класу – конструктор за умовчанням.

Проте класи в C# можуть містити довільну кількість конструкторів, призначених для ініціалізації об'єкту даного класу.

Створимо конструктор з параметрами, що дозволяє ініціалізувати об'єкт класу, вводячи значення полів цього об'єкту як параметри конструктора. Зробити це можна таким чином:

```
public Client(string ClientName, string ClientPassport
DateTime ClientBirthDate)
{
  name = ClientName;
  passport = ClientPassport;
  birthdate = ClientBirthDate;
}
```

Видно, що конструктор містить три параметри. У тілі конструктора відбувається запис переданих як параметри значень у відповідні поля класу за допомогою властивостей даного класу. У випадку з датою народження це дозволяє не дублювати процедуру перевірки введеної дати, а скористатися алгоритмом, реалізованим у властивості birthdate.

Створимо також метод, що дозволяє змінити значення полів об'єкту класу Client:

```
public void EditClient(string ClientName, string
 ClientPassport,DateTime ClientBirthDate)
{
```

```
  Name = ClientName;
  Passport = ClientPassport;
  birthdate = ClientBirthDate;
 }
```

Як видно з прикладу, початковий код цього методу практично повністю ідентичний конструктору з параметрами з різницею в імені, а також в типі повертаного значення. Звичайно, в даному випадку можна було б обійтися і використанням властивостей для зміни значень полів класу, проте, іноді буває корисно, щоб такого роду зміни були реалізовані в рамках одного методу, тим більше, якщо алгоритм змін є нестандартним.

Тепер, з використанням конструктора з параметрами, можна створити і відразу ж ініціалізувати об'єкт класу Client:

```
Client c1=new Client("Іваненко","9002",new DateTime(1967,08,03));
```

Структури

У багатьох відношеннях, структури можна розглядати як особливий різновид класів.

Для структур можна визначати конструктори, реалізовувати інтерфейси. Проте для структур в C# не існує базового класу, тому всі структури є похідними від типу **ValueType.**

Простий приклад структури можна представити так:

```
public struct Employee
{
  public string name;
  public string type;
  public int deptID;
}
```

Використання структури можливе таким чином. Спочатку її необхідно створити. У момент створення структури для неї виділяється пам'ять в області стека. Надалі до елементів структури можливе звернення шляхом вказівки імені структури і імені елементу, розділених крапкою:

```
Employee Alex;
Alex.name = "Alex";
Alex.type = "manager";
Alex.deptID = 2310;
```

У реальній системі для зручнішого використання структур доцільно визначити конструктор або декілька конструкторів. При цьому необхідно пам'ятати, що в структурах неможливо перевизначити конструктор за умовчанням (без параметрів).

Всі визначені в структурі конструктори повинні приймати параметри. Нижче представлений приклад конструктора структури.

```
public Employee(int DEPTID, string Name, string EmpType)
{
  deptID = DEPTID;
  type = EmpType;
  name = Name;
}
```

Для виклику конструктора структури необхідно використовувати ключове слово new.Використання конструктора виглядає таким чином:

```
Employee Nick=new Employee(278,"Nick","worker");
```

Аналогічним чином стає можливим створення і використання методів структур.

Колекції

Раніше був розглянутий клас Array, в якому реалізований цілий набір часто використовуваних операцій над елементами масиву, таких як сортування, клонування, перерахування і розстановка елементів в зворотному порядку. Проте при роботі з масивами виникає і цілий ряд інших завдань. Вирішенням цих завдань і займаються **колекції**, що дозволяють організовувати елементи спеціальним чином і проводити згодом над ними певний набір операцій.

Всі визначені в .NET Framework колекції розташовані в просторі імен **System.Collections**.

Найбільш часто використовувані колекції цього простору імен представлені нижче:

ArrayList	Масив об'єктів, що динамічно змінює свій розмір.
Hashtable	Набор взаємозв'язаних ключів і значень, заснованих на хэш-коде ключа.
Queue	Стандартна черга, реалізована за принципом "першим увійшов, першим вийшов" (FIFO – First In First Out).
SortedList	Представляє клас, елементами якого можуть бути пари "ключ-значення", відсортовані за значенням ключа які надають доступ до елементів по їх порядковому номеру (індексу).
Stack	Черга, реалізована за принципом "першим увійшов, останнім вийшов" (LIFO – Last In First Out).

У просторі імен System.Collections.Specialized розташовані класи, призначені для використання в спеціальних випадках. Так, клас StringCollection представляє набір

рядків. Класом StringDictionary є набір строкових значень, що складаються з пар "ключ-значення", як значення яких використовується рядок.

Розглянемо застосування колекцій на прикладі класу ArrayList.

Для цього скористаємося розглянутим раніше класом Client. Дуже часто виникає ситуація, при якій необхідно буває поміщати об'єкти в список або масив для їх подальшої обробки. Звичайно, при цьому може бути достатньо тих можливостей, які надають стандартні масиви C#, розглянуті раніше. Проте колекції, володіють ширшими можливостями, отже, їх застосування переважне в порівнянні з масивами.

Завдання 3

Для обробки даних про клієнтів створити клас *Clients*, призначений для зберігання списку клієнтів і його обробки. При цьому необхідно врахувати, що клас *Clients* повинен забезпечувати можливість додавання, видалення і нумерації елементів (клієнтів).

Для реалізації цієї функціональності необхідно використовувати клас *ArrayList* як вкладеного в клас *Clients*. Таким чином, необхідно визначити в даному класі елемент, що дозволяє вести список клієнтів, а також реалізувати набір відкритих методів, які передаватимуть виклики на виконання різних дій внутрішньому класу, похідному від *ArrayList*. Приклад початкового коду класу *Clients* приведений нижче.

```
public class Clients
{
 private ArrayList ClientsList;
 public Clients()
 {
  ClientsList=new ArrayList();
 }
 public int ClientsCount
 {
  get
  {
   return ClientsList.Count;
  }
 }
  public void AddClient(Client з)
 {
  ClientsList.Add(c);
 }
 public void RemoveClient(int ClientToRemove)
```

```
  {
    ClientsList.RemoveAt(ClientToRemove);
  }
  public Client GetClient(int CLIENTID)
  {
    return (Client) ClientsList[CLIENTID];
  }
}
```

Використання такого класу представляється дуже простим:

```
Clients cl=new Clients();
cl.AddClient(new Client("Сидоров","9002",new
 DateTime(1980,12,21)));
cl.AddClient(new Client("Петренко","9004",new
 DateTime(1975,03,15)));
```

Тепер стає можливим звернення до будь-якого класу Client, поміщеного в колекцію Clients за допомогою методу GetClient:

```
Client MyClient = cl.GetClient(1);
Response.Write(MyClient.name);
```

Результатом роботи цього фрагмента програми буде рядок "Петренко".

Проте існує декілька незручностей від використання класу Clients в його нинішній реалізації. По-перше, звичнішим зверненням до елементу масиву або списку в мові C# є використання індексаторів, що позначаються у вигляді квадратних дужок. Для реалізації цієї можливості в класі Clients необхідно створити індексатор:

```
public Client this[int pos]
{
  get
  {
    return ((Client)ClientsList[pos]);
  }
  set
  {
    ClientsList[pos]= value;
  }
}
```

Як видно, від звичайного методу індексатор відрізняється ім'ям (this), а також наявністю квадратних дужок, в яких указується параметр – номер витягуваного елементу. За допомогою індексатора стає можливим витягання елементу з колекції, а також запис елементу в колекцію під певним номером:

```
Response.Write(cl[1].name);
cl[1].name = "Сидоров";
Response.Write(cl[1].name);
```

Тепер можна організувати обробку елементів масиву в циклі. При роботі з колекціями досить зручною є можливість використання циклу forech. Для реалізації такої функціональності при роботі з класом Clients необхідно використовувати метод GetEnumerator() інтерфейсу IEnumerator – задати спадкоємство класом Clients інтерфейсу IEnumerator і реалізувати метод, як показано нижче.

```
public class Clients:IEnumerable
{
  public IEnumerator GetEnumerator()
  {
    return ClientsList.GetEnumerator();
  }
}
```

Приклад застосування циклу foreach з використанням класу Clients показаний нижче.

```
foreach (Client з in cl)
{
  Response.Write("Імя="+c.name+" ");
  Response.Write("Паспорт="+c.passport+" ");
  Response.Write("Возраст=" + з.age);
  Response.Write("</br>");
}
```

Контрольні запитання:

1. Які типи данних описуються за допомогою класів C# ?
2. Сформулюйте правила передачі параметрів в процедури і функції ?
3. Що таке класс? Основне призначення класів C#?
4. Як відбувається визначення класу. Призначення полів і методів класу?
5. Що таке структура? Надайте приклад використання структур у C#?
6. Які колекції розташовані в просторі імен ***System.Collections***?
7. Вкажіть різницю між методом та індексатором класу. Для чого використують індексатори класів?