Bài giảng môn học

Bài 3: Kiến thức cơ bản về C# (tt)
Class

Lương Trần Hy Hiến

FIT, HCMUP

Lập trình Windows Form với C#

Nội dung

1. Định nghĩa lớp
2. Từ khóa truy cập (Access Modifiers)
3. Constructors
4. Properties
5. Thành phần Static
6. Inheritance, Polymorphism
7. Interfaces
8. Generic Classes

Định nghĩa lớp

Classes in OOP

- **Classes model real-world objects and define**
  - Attributes (state, properties, fields)
  - Behavior (methods, operations)
- **Classes describe structure of objects**
  - Objects describe particular instance of a class
- **Properties hold information about the modeled object relevant to the problem**
- **Operations implement object behavior**

Classes in C#

- **Classes in C# could have following members:**
  - Fields, constants, methods, properties, indexers, events, operators, constructors, destructors
  - Inner types (inner classes, structures, interfaces, delegates, ...)
- **Members can have access modifiers (scope)**
  - `public`, `private`, `protected`, `internal`
- **Members can be**
  - `static` (common) or specific for a given object

Định nghĩa lớp

Begin of class definition

```
public class Cat : Animal
{
    private string name;
    private string owner;

    public Cat(string name, string owner)
    {
        this.name = name;
        this.owner = owner;
    }

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

Inherited (base) class

Fields

Constructor

Property

Định nghĩa lớp (2)

```
    public string Owner
    {
        get { return owner;}
        set { owner = value; }
    }

    public void SayMiau()
    {
        Console.WriteLine("Miauuuuuuu!");
    }
}
```

Method

End of class definition

Class Definition and Members

- **Class definition consists of:**
  - Class declaration
  - Inherited class or implemented interfaces
  - Fields (static or not)
  - Constructors (static or not)
  - Properties (static or not)
  - Methods (static or not)
  - Events, inner types, etc.

Thuộc tính truy cập (Access Modifiers)

Thuộc tính	Giới hạn truy cập
Public	Không hạn chế
Private	Chỉ trong lớp (mặc định)
Protected	Trong lớp và lớp con
Internal	Trong chương trình
Protected internal	Trong chương trình và lớp con

→ tính đóng gói (encapsulation)

Từ khóa 'this'

- **Từ khóa this dùng để tham chiếu đến thể hiện hiện hành của một đối tượng**

```
public void SetName( int name)
{
   this.Name = name;
}
```

- **Tham chiếu this này được xem là con trỏ ẩn đến tất các phương thức không có thuộc tính tĩnh trong một lớp.**

Task: Define Class Dog

- **Our task is to define a simple class that represents information about a dog**
  - The dog should have name and breed
  - If there is no name or breed assigned to the dog
    - It should be named "Balkan"
    - Its breed should be "Street excellent"
  - It should be able to view and change the name and the breed of the dog
  - The dog should be able to bark

Defining Class Dog – Example

```
public class Dog
{
    private string name;
    private string breed;

    public Dog()
    {
        this.name = "Balkan";
        this.breed = "Street excellent";
    }

    public Dog(string name, string breed)
    {
        this.name = name;
        this.breed = breed;
    }
```

(example continues)

Defining Class Dog – Example (2)

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

    public string Breed
    {
        get { return this.breed; }
        set { this.breed = value; }
    }

    public void SayBau()
    {
        Console.WriteLine("{0} said: Bauuu!", this.name);
    }
}
```

Lớp và đối tượng

- Phương thức thiết lập (Constructor)
- Phương thức thiết lập sao chép
- Phương thức huỷ
- Sử dụng using

(hàm huỷ tự đông gọi trong thời gian sớm nhất)

```
using System.Drawing;
class Tester
{
   public static void Main( )
   {
      using (Font theFont = new Font("Arial", 10.0f))
      {
         // sử dụng theFont
      } // phương thức Dispose của theFont được gọi
      Font anotherFont = new Font("Courier",12.0f);
      using (anotherFont)
      {
         // sử dụng anotherFont
      } // phương thức Dispose của anotherFont được gọi
   }
}
```

Constructors

Phương thức khởi tạo

- **Hàm tạo mặc định: giống C++**
- **Hàm tạo có tham số: tương tự C++ nhưng không có tham số mặc định**

```
public class MyClass
{
    public MyClass() // zero-parameter constructor
    {
            // construction code
    }
    public MyClass(int number)   // another overload
    {
            // construction code
    }
}
```

Phương thức khởi tạo

- C# không cho phép khởi tạo sao chép
- Chú ý với hàm tạo có tham số: hãy luôn luôn có hàm tạo mặc định để tránh lỗi biên dịch.

VD: Lớp điểm trong mặt phẳng Oxy

```
public class Point
{
    private int xCoord;
    private int yCoord;

    // Simple default constructor
    public Point()
    {
        xCoord = 0;
        yCoord = 0;
    }

    // More code ...
}
```

VD: Lớp người

```
public class Person
{
    private string name;
    private int age;
    // Default constructor
    public Person()
    {
        name = "[no name]";
        age = 0;
    }
    // Constructor with parameters
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    // More code ...
}
```

Gọi lại Constructor

```
public class Point
{
    private int xCoord;
    private int yCoord;

    public Point() : this(0,0) // Reuse construc
    {
    }

    public Point(int xCoord, int yCoord)
    {
        this.xCoord = xCoord;
        this.yCoord = yCoord;
    }

    // More code ...
}
```

Phương thức hủy

- C# cung cấp cơ chế thu dọn (garbage collection) và do vậy **không cần** phải khai báo tường minh các phương thức hủy.
- Phương thức **Finalize** sẽ được gọi bởi cơ chế thu dọn khi đối tượng bị hủy.
- Phương thức kết thúc chỉ giải phóng các tài nguyên mà đối tượng nắm giữ, và không tham chiếu đến các đối tượng khác.

Phương thức hủy (tt)

```
~Class1()
{
  // Thực hiện một số công việc
}
Class1.Finalize()
{
  // Thực hiện một số công việc
  base.Finalize();
}
```

Hàm hủy

```
class MyClass : IDisposable
{
  public void Dispose()
  {
    // implementation
  }
}
```

Hàm hủy (tt)

- Lớp sẽ thực thi giao diện *System.IDisposable*, tức là thực thi phương thức *IDisposable.Dispose().*
- Không biết trước được khi nào một Destructor đượcgọi.
- Có thể chủ động gọi thu dọn rác bằng cách gọi phương thức *System.GC.Collect().*
- *System.GC* là một lớp cơ sở .NET mô tả bộ thu gom rác và phương thức *Collect()* dùng để gọi bộ thu gom rác.

Thuộc tính (Property)

- **Đóng gói dữ liệu với Property**
  - VD: Lớp người có chuỗi m_sHoten
  - Cài đặt Property HoTen:

```
public string HoTen
{
    get { return m_sHoTen; }
    set { m_sHoTen = value; }
}
```

  - Sử dụng Property:

```
Nguoi A = new Nguoi();
A.HoTen = "Lương Trần Hy Hiến";   //đặt giá trị
string tentoi = A.HoTen;          //lấy giá trị
```

Thuộc tính (Property)

- **Nếu câu lệnh Property chỉ có đoạn lệnh get → thuộc tính *chỉ đọc* (Read Only)**
- **Nếu câu lệnh Property chỉ có đoạn lệnh set → thuộc tính *chỉ ghi* (Write Only)**

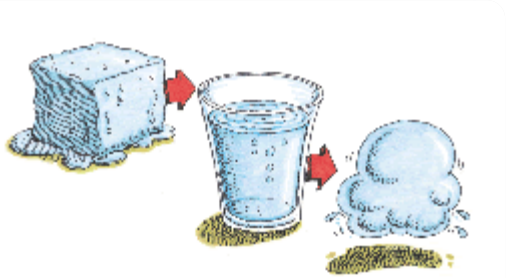
Ví dụ: Lớp điểm

```
public class Point
{
    private int xCoord;
    private int yCoord;

    public int XCoord
    {
        get { return xCoord; }
        set { xCoord = value; }
    }

    public int YCoord
    {
        get { return yCoord; }
        set { yCoord = value; }
    }

    // More code ...
}
```



Dynamic Properties

- **Properties được tính toán tự động mà không dựa vào một Field nào**

```
public class Rectangle
{
    private float width;
    private float height;

    // More code ...

    public float Area
    {
        get
        {
            return width * height;
        }
    }
}
```

Automatic Properties

```
class UserProfile
{
    public int UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}


UserProfile profile = new UserProfile() {
    FirstName = "Hien",
    LastName = "Luong",
    UserId = 91112
};
```

Thành viên static

- Thành viên tĩnh được xem như một phần của lớp.
- Có thể truy cập đến thành viên tĩnh của một lớp thông qua tên lớp
- C# không cho phép truy cập đến các phương thức tĩnh và các biến thành viên tĩnh thông qua một thể hiện.
- Không có friend
- Phương thức tĩnh hoạt động ít nhiều giống như phương thức toàn cục

Static vs. Non-Static

- **Static**:
  - Associated with a type, not with an instance
- **Non-Static**:
  - The opposite, associated with an instance
- **Static**:
  - Initialized just before the type is used for the first time
- **Non-Static**:
  - Initialized when the constructor is called

Static Members – Example

```
public class SqrtPrecalculated
{
    public const int MAX_VALUE = 10000;

    // Static field
    private static int[] sqrtValues;

    // Static constructor
    private static SqrtPrecalculated()
    {
        sqrtValues = new int[MAX_VALUE + 1];
        for (int i = 0; i < sqrtValues.Length; i++)
        {
            sqrtValues[i] = (int)Math.Sqrt(i);
        }
    }
```

//(example continues)

Static Members – Example (2)

```
    // Static method
    public static int GetSqrt(int value)
    {
        return sqrtValues[value];
    }

    // The Main() method is always static
    static void Main()
    {
        Console.WriteLine(GetSqrt(254));
    }
}
```

Hướng đối tượng

```
public class BankAccount{
    protected string ID;
    protected string Owner;
    protected decimal _Balance;
    public BankAccount(string ID,
    string Owner) {
        this.ID = ID;
        this.Owner = Owner;
        this._Balance = 0;
    }
    public void Deposit(decimal
    Amount) {
        _Balance+=Amount;
    }
    public void Withdraw(decimal
                        Amount) {
        _Balance-=Amount;
    }
    public decimal Balance {
        get {
            return _Balance;
        }
    }
}
```

Fields

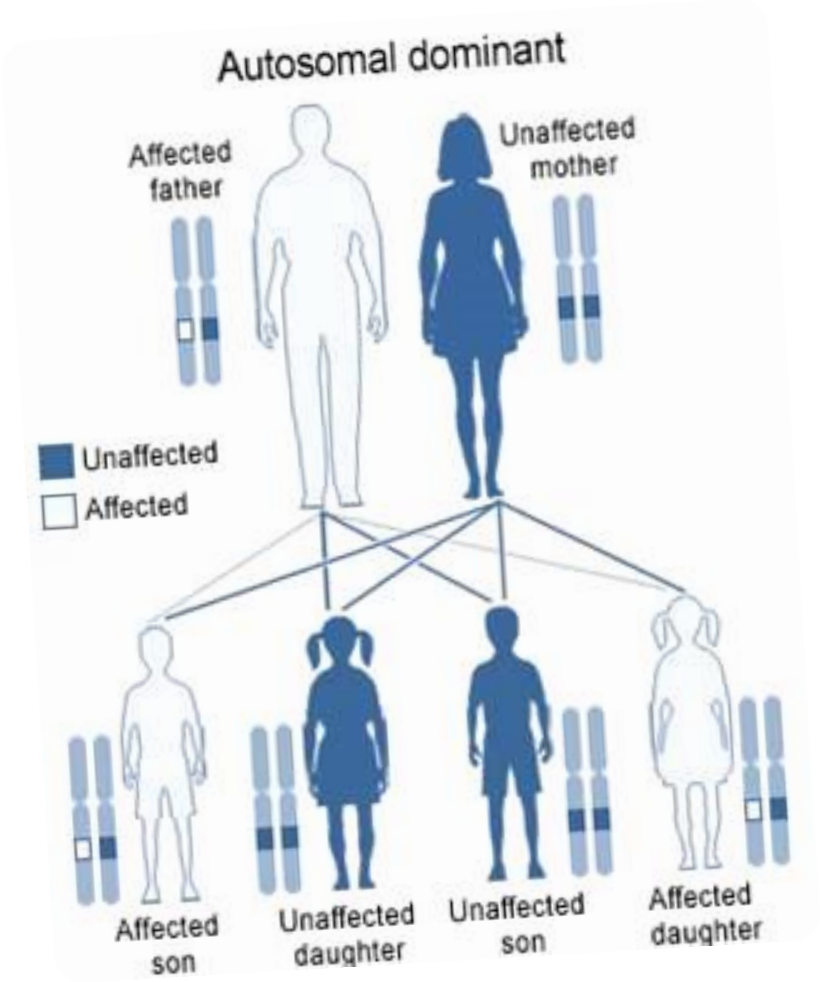
Thuộc tính chỉ đọc

Hướng đối tượng

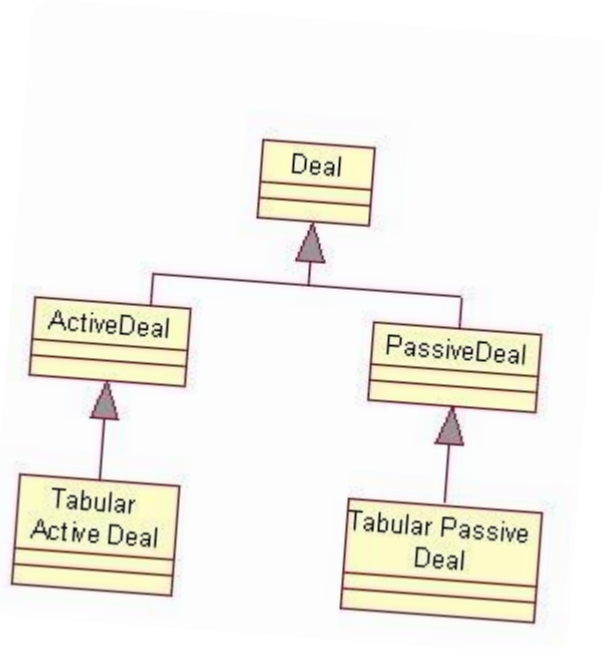
```
class Program
{
   static void Main(string[] args)
   {
      BankAccount acc= new Account(“1606201007676", “Hy Hien");
      acc.Deposit(1000);
      acc.Withdraw(100);
      Console.WriteLine("Balance: {0}", acc.Balance);
      //myAcct.Balance=10000;
      Console.ReadLine();
   }
}
```

Chồng hàm (overload)

- Không chấp nhận hai phương thức chỉ khác nhau về kiểu trả về.
- Không chấp nhận hai phương thức chỉ khác nhau về đặc tính của một thông số đang được khai báo như *ref* hay *out*.



Inheritance



Sự thừa kế

- 1 class chỉ có thể kế thừa từ ***1 class cơ sở***
- 1 class có thể kế thừa từ ***nhiều Interface***
- Từ khóa **sealed** được dùng trong trường hợp khai báo class mà ***không cho phép*** class khác ***kế thừa***.
- Cú pháp:

```
public class Derived_Class : Base_Class
{
   ...
}
```

Đa hình

- **Để tạo một phương thức hỗ tính đa hình:**
  - khai báo khóa virtual trong phương thức của lớp cơ sở
- **Để định nghĩa lại các hàm virtual, hàm tương ứng lớp dẫn xuất phải có từ khóa override**

Phương thức Override

```
class MyBaseClass
{
   public virtual string VirtualMethod()
   {
         return "This method is virtual and defined in
   MyBaseClass";
   }
}
class MyDerivedClass : MyBaseClass
{
   public override string VirtualMethod()
   {
         return "This method is an override defined in
   MyDerivedClass";
   }
}
```

Gọi các hàm ở lớp cơ sở

- Cú pháp: base.<methodname>()

```
class CustomerAccount
{
    public virtual decimal CalculatePrice()
    {
            // implementation
    }
}
class GoldAccount : CustomerAccount
{
    public override decimal CalculatePrice()
    {
            return base.CalculatePrice() * 0.9M;
    }
}
```

Lớp cơ sở trừu tượng

```
abstract class Building
{
   public abstract decimal CalculateHeatingCost();
   // abstract method
}
```

- **Một lớp abstract không được thể hiện và một phương thức abstract không được thực thi mà phải được overriden trong bất kỳ lớp thừa hưởng không abstract nào**
- **Nếu một lớp có phương thức abstract thì nó cũng là lớp abstract**
- **Một phương thức abstract sẽ tự động được khai báo *virtual*.**

Abstract class

```
public abstract class BankAccount {
   …
   public abstract bool IsSufficientFund(decimal
   Amount);
   public abstract void AddInterest();
   …
}
```

- Không thể new một abstract class
- Chỉ có lớp abstract mới có thể chứa abstract method

Abstract Class – Example

```
abstract class MovableShape : IShape, IMovable
{
    private int x, y;
    public void Move(int deltaX, int deltaY)
    {
        this.x += deltaX;
        this.y += deltaY;
    }
    public void SetPosition(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
    public abstract int CalculateSurface();
}
```

Giao diện - Interface

- **Một "interface" được định nghĩa như một "hợp đồng", do đó, nếu một class hoặc một struct cài đặt 1 interface thì phải cài đặt tất cả các tính năng được khai báo trong interface đó.**
- **Có thể hiểu interface như là một lớp trừu tượng hoàn toàn (tất cả các phương thức đều trừu tượng). Khi một class cài đặt 1 interface thì coi như nó được kế thừa từ lớp trừu tượng nói trên.**

Giao diện

- **Tạo một giao diện**

```
interface IStorable
{
   void Read( );
   void Write(object);
}
```

```
interface ICompressible
{
    void Compress();
    void Decompress();
}
```

- **Mở rộng giao diện & Kết hợp giao diện**

```
interface ILoggedCompressible : ICompressible
{
   void LogSavedBytes( );
}

interface IStorableCompressible: IStoreable,ILoggedCompressible
{
  void LogOriginalSize( );
}
```

Giao diện

- **Cách sử dụng :**

```
Document doc = new Document("Test Document");
IStorable isDoc = (IStorable) doc;
isDoc.Read( );
ICompressible icDoc = (ICompressible) doc;
icDoc.Compress( );
```

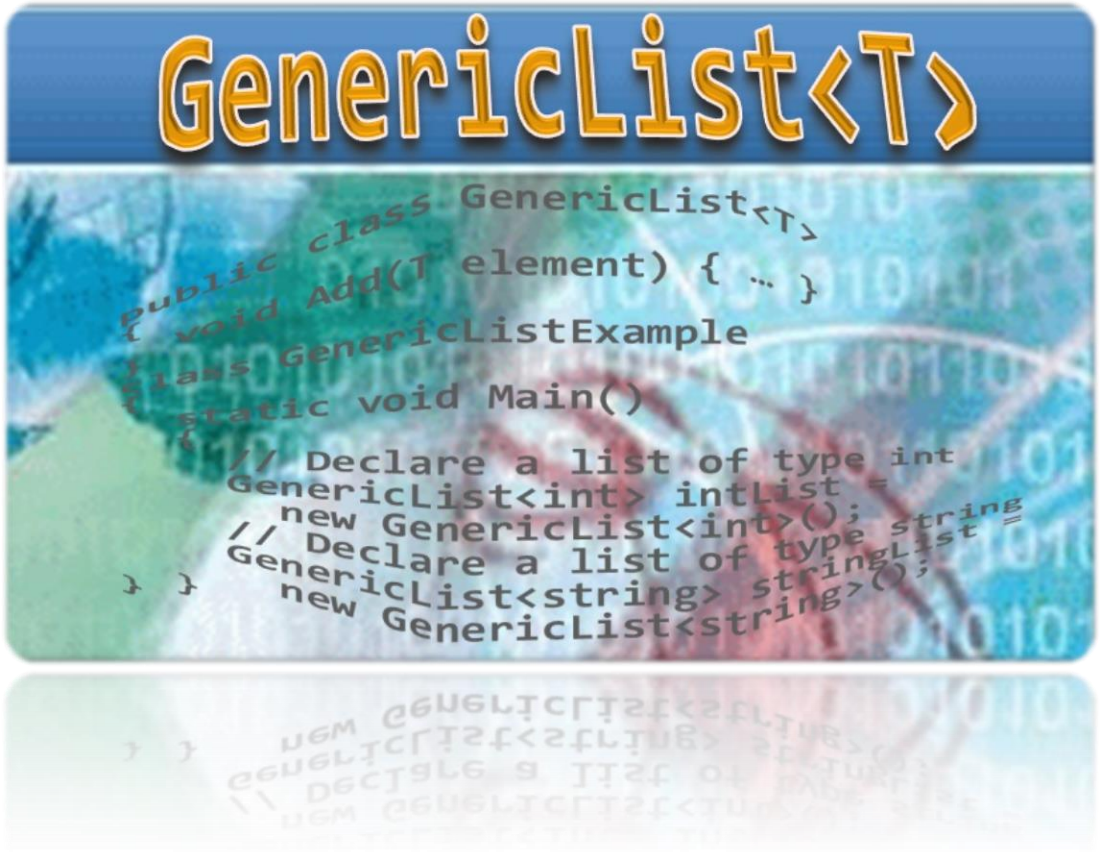
- **Toán tử is: kiểm tra xem đối tượng có được hỗ trợ giao diện hay không (VD doc is IStorable)**
- **Toán tử as: Kiểm tra và gán**

```
Document doc = new Document("Test Document");
IStorable isDoc = doc as IStorable;
if (isDoc != null)
  isDoc.Read( );
else
  Console.WriteLine("IStorable not supported");
```

Một số giao diện chuẩn sau

Giao diện	Ý nghĩa
IEnumerable	Khi một lớp cài đặt giao diện này, đối tượng thuộc lớp có được dùng trong câu lệnh foreach
ICollection	Được cài đặt bởi tất cả các lớp túi chứa có thành viên CopyTo(), Count, IsReadOnly(), IsSyncronize(), SyncRoot()
IComparer	So sánh hai đối tượng trong túi chứa
IList	Dùng bởi các lớp túi chứa truy xuất phần tử thông qua chỉ mục (số)
IDictionary	Dùng bởi các lớp túi chứa truy xuất phần tử thông qua quan hệ khóa/giá trị như Hashtabe, StoredList.
IDictionaryEnumerator	Cho phép duyệt đối với các túi chứa cài đặt IDictionary

Generic Classes



Generics?

- **Generics cho phép định nghĩa lớp khi6ng phụ thuộc kiểu dữ liệu**
  - Các lớp có thể được khởi tạo với một số kiểu dữ liệu khác nhau
  - Ví dụ: `List<T>` → `List<int>` / `List<string>` / `List<Student>`
- **Generics được biết như là "parameterized types" hay "template types"**
  - Tương tự kiểu templates trong C++
  - Tương tự kiểu generics trong Java

Ví dụ Generics

```
public class GenericList<T>
{
  public void Add(T element) {
}

class GenericListExample
{
  static void Main()
  {
    // Declare a list of type int
    GenericList<int> intList =
      new GenericList<int>();

    // Declare a list of type string
    GenericList<string> stringList =
      new GenericList<string>();
  }
}
```

T is an unknown type, parameter of the class

T can be used in any method in the class

T can be replaced with int during the instantiation


STOP