PHÁT TRIỂN ỨNG DUNG CSDL 1

Tháng 9/2014

LẬP TRÌNH ỨNG DỤNG VỚI NGÔN NGỮ C#

Lập trình hướng đối tượng

Tóm tắt nội dung bài thực hành:

Hướng dẫn về lập trình hướng đối tượng trong C#: khai báo lớp đối tượng, kế thừa, lớp trìu tượng, interface

Bộ môn **Hệ thống thông tin**

Khoa Công nghệ thông tin

ĐH Khoa hoc tư nhiên TP HCM

MỤC LỤC

1 Mục tiêu

Sau khi hoàn thành bài tập này sinh viên có thể với ngôn ngữ C#:

- Định nghĩa một lớp đối tượng
- Sử dụng các kỹ thuật của hướng đối tương: kế thừa, abstract class, inteface.

2 Định nghĩa lớp đối tượng

Cú pháp:

```
public class TenLop
{
     //Khai báo các thuộc tính
     //Khai báo các phương thức
      //Các phương thức khởi tạo
      //Các phương thức get/set
      //Các phương thức xử lý nghiệp vụ
      //Các phương thức overload
      //Các phương thức static
}
```

Ví dụ:

```
public class PhanSo
{
    //Khai báo các thuộc tính
    private int tuSo;
    private int mauSo;
    //Phương thức khởi tạo mặc định
    public PhanSo()
    {
        this.TuSo = 1;
        this.MauSo = 2;
    }
    //Phương thức khởi tạo đầy đủ tham số
    public PhanSo(int tuSo, int mauSo)
    {
        this.tuSo = tuSo;
        this.mauSo = mauSo;
    }
    //Phương thức khởi tạo sao chép
    public PhanSo(PhanSo ps)
    {
        this.tuSo = ps.tuSo;
        this.mauSo = ps.mauSo;
    }
}
```

Ví dụ:

```
public class PhanSo
{
    //Khai báo các thuộc tính
    private int tuSo;
    private int mauSo;
    //Phương thức khởi tạo đối tượng
    //Các phương thức get/set
    public int getTuSo()
    {
        return tuSo;
    }
    public void setTuSo(int tuSo)
    {
        this.tuSo = tuSo;
    }
    public int getMauSo()
    {
        return mauSo;
    }
    public void setMauSo(int mauSo)
    {
        if (mauSo != 0)
        {
            this.mauSo = mauSo;
        }
    }
}
```

Vi dụ:

```
public class PhanSo
 {
      //Khai báo các thuộc tính
      private int tuSo;
      private int mauSo;
      //...
      public PhanSo cong(PhanSo ps)
      {
           PhanSo kq = new PhanSo();
           kq.tuSo = this.tuSo * ps.mauSo + this.mauSo *
     ps.tuSo;
           kq.mauSo = this.mauSo * ps.mauSo;
           return kq;
      }
 }
```

Ví dụ:

```
public static void main(String[] args)
{
       PhanSo ps1=new PhanSo(1,2);
       PhanSo ps2=new PhanSo(3,4);
       PhanSo ps3=ps1.cong(ps2);
       Console.WriteLine(ps3.getTuSo() + "/" +
ps3.getMauSo());
}
```

3 Đối tượng

Đối tượng là một sự thể hiện của lớp. Một lớp có thể có nhiều sự thể hiện khác nhau.

Khai báo đối tượng:

```
TenLop TenDoiTuong;
TenDoiTuong = new TenLop;
```

Ví dụ:

```
PhanSo a = new PhanSo();
PhanSo x = new PhanSo(1, 2);
```

4 Phương thức

Phương thức là khả năng mà một đối tượng thuộc về lớp có thể thực hiện.

Cú pháp khai báo:

```
public KieuDuLieu TenPhuongThuc(KieuDuLieu
ThamSo1, KieuDuLieu ThamSo2,…)
{
     //Định nghĩa nội dung phương thức
}
```

Ví dụ:

```
public class PhanSo
{
    //Khai báo các thuộc tính
    private int tuSo;
    private int mauSo;
    //...
    public PhanSo cong(PhanSo ps)
    {
        PhanSo kq = new PhanSo();
        kq.tuSo = this.tuSo * ps.mauSo + this.mauSo *
     ps.tuSo;
        kq.mauSo = this.mauSo * ps.mauSo;
        return kq;
    }
}
```

5 Phương thức khởi tạo

Các phương thức thiết lập của một lớp có nhiệm vụ thiết lập thông tin ban đầu cho các đối tượng thuộc về lớp ngay khi đối tượng được khai báo. Là phương thức đầu tiên được triệu gọi và chỉ gọi một lần khi khởi tạo đối tượng, nó nhằm thiết lập các tham số đầu tiên cho đối tượng. Tên hàm trùng tên lớp; còn các mặt khác như phương thức bình thường. Các đặc điểm của phương thức khởi tạo:

- Phương thức thiết lập của lớp được định nghĩa thông qua toán tử new.
- Không có giá trị trả về.
- Được tự động gọi thực hiện ngay khi đối tượng được khai báo.
- Có thể có nhiều phương thức thiết lập trong một lớp.
- Trong một quá trình sống của đối tượng thì chỉ có 1 lần duy nhất phương thức thiết lập được gọi thực hiện, đó là khi đối tượng được khai báo.
- Nếu lớp không định nghĩa hàm khởi tạo, trình biên dịch tự động tạo một hàm khởi tạo mặc định. Khi đó các biến thành viên sẽ được khởi tạo theo các giá trị mặc định:

Kiểu	Giá trị mặc định
số (int, long, …)	0
bool	false
char	'\0' (null)
enum	0
Tham chiếu	null

Các loại phương thức khởi tạo:

- Phương thức thiết lập mặc định.
- Phương thức thiết lập sao chép.
- Phương thức thiết lập nhận tham số đầu vào

Ví dụ:

```
public class PhanSo
 {
      //Các thuộc tính
      private int Tu;
      private int Mau;

      // Phương thức thiết lập mặc định
      public PhanSo()
      {
           Tu = 0;
           Mau = 1;
      }

      // Phương thức thiết lập khi biết tử số
      public PhanSo(int t)
      {
           Tu = t;
           Mau = 1;
      }

      // Phương thức thiết lập khi biết đầy đủ thông
 tin
      public PhanSo(int t, int m)
      {
           Tu = t;
           Mau = m;
      }

      // Phương thức thiết lập sao chép
      public PhanSo(PhanSo ps)
      {
           Tu = ps.Tu;
           Mau = ps.Mau;
      }
 }
```

Gọi phương thực khởi tạo:

```
PhanSo a = new PhanSo();
PhanSo b = new PhanSo(1);
PhanSo c = new PhanSo(1, 2);
PhanSo d = new PhanSo(c);
```

6 Phương Thức Hủy

C# cũng cung cấp bộ thu dọn rác tự động nó sẽ ngầm hủy các biến khi không dùng. Tuy nhiên trong một số trường hợp ta cũng cần hủy tường minh, khi đó chỉ việc cài đặt phương thức Finalize(), phương thức này sẽ được gọi bởi bộ thu dọn rác. Ta không cần phải gọi phương thức này. Đặc điểm của phương thức hủy:

- Phương thức hủy của lớp được định nghĩa thông qua toán tử Finalize.
- Không có giá trị trả về.
- Không có tham số đầu vào.
- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng.
- Phương thức hủy thuộc nhóm các phương thức xử lý.
- Có duy nhất một phương thức hủy trong 1 lớp mà thôi

7 Thành phần tĩnh

Một thành phần (phương thức/thuộc tính) mặc định thường được gọi từ một đối tượng của lớp . Tuy nhiên, thành phần tĩnh có thể được gọi thông qua lớp mà không cần qua bất kì đối tượng nào của lớp. Thành phần tĩnh là thành phần chung của lớp. Lưu ý khi định nghĩa phương thức tĩnh chỉ được sử dụng những thành phần tĩnh của lớp.

Cú pháp khai báo:

```
//Khai báo thuộc tính tĩnh
public static KieuDieuLieu TenThuocTinh
//Khai bao phương thức tĩnh
public static KieuDieuLieu TenPhuongThuc (
KieuDuLieu ThamSo1, KieuDuLieu ThamSo2,..)
{
     //không được sử dụng các thành phần không tĩnh
     //của lớp.
}
```

Ví dụ:

```
public class PhanSo
{
    private static int soLuongPhanSo = 0;
    private int tu;
    private int mau;
    public PhanSo(int t, int m)
    {
        tu = t;
        mau = m;
        soLuongPhanSo++;
    }
    public static void XuatSLPhanSo()
    {
        Console.WriteLine("So Phan So da tao la
{0}", soLuongPhanSo);
    }
}
```

Cú pháp truy cập thành phần tĩnh:

```
TenLop.ThanhPhanTinh

Hoặc

TenDoiTuong.ThanhPhanTinh
```

Ví dụ

```
PhanSo a = new PhanSo(1,3);
PhanSo b = new PhanSo(4,5);
PhanSo c = new PhanSo(1, 2);
//gọi thành phần tĩnh từ lớp:
PhanSo.XuatSLPhanSo();// xuất:So Phan So da tao
la 3
//gọi thành phần tĩnh từ đối tượng của lớp:
c.XuatSLPhanSo();//xuất: So Phan So da tao la 3
```

8 Kế thừa:

Một lớp có thể có những phương thức, thuộc tính của một lớp khác thông qua cơ chế kế thừa.

Cú pháp:

```
public class TenLopCha
{
    //Khai báo các thuộc tính
    //Khai báo các phương thức
}

public class TenLopCon: TenLopCha
{
     //Khai báo các thuộc tính
     //Khai báo các phương thức
}
```

Ví dụ:

```
public class GiangVien
{
    //Khai báo các thuộc tính
    //Khai báo các phương thức
}

public class GiangVienCoHuu : GiangVien
{
    //Khai báo các thuộc tính
    //Khai báo các phương thức
}
```

Lưu ý :

- Lớp con có đầy đủ những thành phần(phương thức/thuộc tính) của lớp cha
- *Thành phần protected của lớp cha*: được sử dụng trực tiếp các lớp con
- *Thành phần private của lớp cha*:lớp con không được sử dụng trực tiếp, có thể sử dụng gián tiếp thông qua các phướng thức get/set của lớp cha trên thành phần đó.
- *Thuộc tinh và phương thức public*: được sử dụng trực tiếp ở các lớp con

9 Các từ khóa thông dụng:

Truy xuất lớp hiện tại: `this`

Truy xuất đến lớp cha: `base`

10 Phương thức bi ghi đè (Override)

Lớp con có thể cài đặt lại một phương thức mà nó kế thừa của lớp cha.

Cú pháp:

```
public override KieuDuLieu tenPhuongthuc
(KieuDuLieu TS1, KieuDuLieu TS2,…)
{
     //Định nghĩa lại phương thức đã kế thừa của
lớp cha
}
```

Lớp con chỉ có thể ghi đè phương thức được khai báo bằng từ khóa `virtual` ở lớp cha.

Ví dụ:

```
public class Window
{
     public virtual void DrawWindow()
     {
          //.....
     }
}
public class Button : Window
{
     public override void DrawWindow()
     {
          //...
          //có thể dùng từ khóa đề truy cập
phương thức bị ghi đè của lớp cha.
          base.DrawWindow();
          //...

     }
}
```


Lưu ý:

- Phương thức `static` không được phép `override`
- Gọi phương thức bị ghi đè của lớp cha: `base.tenPhuongThuc(...)`

Sealed Method là một phương được khai báo với từ khóa `sealed`. Sealed Method không được phép `override` ở lớp kế thừa

Cú pháp:

```
sealed public void TenPhuongThuc(…)
{
      //...
}
```

Sealed Class là một lớp được khai báo với từ khóa sealed. Sealed Class không cho phép kế thừa

Cú pháp:

```
public sealed class TenLop
{
    //...
}
```

11 Abstract Class

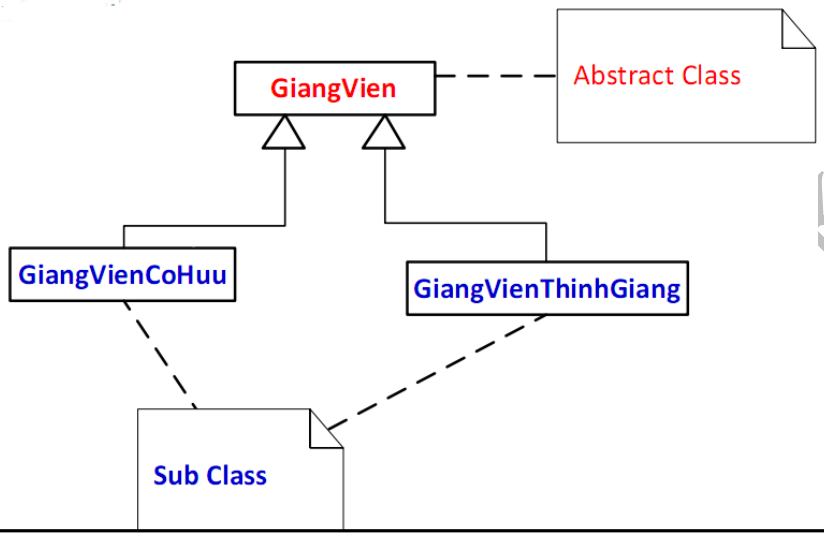
Abstract class là một lớp được khai báo với từ khóa abstract.

Abstract Class có thể chứa các phương thức abstract hoặc phương thức có cài đặt. Phương thức abstract là phương thức không có cài đặt :

Ví dụ:

```
public abstract class GiangVien
{
    //. . .
    public void nhap(){...}
    public abstract double tinhLuong();
}
```

Abstract Class không thể tạo thể hiện nhưng abstract Class có thể giữ tham chiếu đối tượng của các Sub Class không phải abstract Class.




```
public class GiangVienCoHuu: GiangVien
{
   //các phương thức riêng của GiangVienCoHuu
  //Ghi đè phương thức tinhLuong của GiangVien
    public override double tinhLuong()
    {
        //. . .
    }
}

public class GiangVienThinhGiang: GiangVien

{

  //các phương thức riêng của GiangVienThinhGiang
  //Ghi đè phương thức tinhLuong của GiangVien
    public override double tinhLuong()
    {
        //. . .
    }
}
```

```
public class Main
{
     public static void main(String[] args)
    {
       GiangVien gv = new GiangVienCoHuu();
       GiangVien gv1 = new GiangVienThinhGiang();


    }
```

Nếu các Sub Class không phải là `abstract` class thì:

- Bắt buộc phải cài đặt lại tất cả các phương thức `abstract` của lớp cha.
- Không bắt buộc cài đặt lại các phương thức không là `abstract` của lớp cha.

Nếu các Sub Class là Abstract Class thì không bắt buộc phải cài đặt lại phương thức abstract / không abstract của lớp cha

12 Interface

Một interface chỉ gồm các phương thức `abstract`.

So với với `abstract` class, interface không chứa bất kỳ một phương thức nào có cài đặt. Giống với abstract class, không thể tạo thể hiện của một interface nhưng một interface có thể giữ tham chiếu đối tượng của 1 lớp cài đặt lại interface đó.

Cú pháp:

```
public interface TenInterface
{
    //khai báo các phương thức abstract
}
```

Ví dụ:

```
public interface Hinh
{
    double tinhDienTich();
    double tinhchuVi();

}
```

Một lớp có thể cài đặt lại từ nhiều interface

Ví dụ:

```
public interface IB
{
     //. . .
}
public interface IC
{
     //. . .
}
public class B: IB, IC
{
     //. . .
}
```

Nếu một lớp khi cài đặt lại 1 `interface` thì lớp đó phải cài đặt lại tất cả các phương thức được khai báo trong `interface` đó.

Ví du:

```
public class HinhTamGiac:Hinh
{

     //Các thành phần riêng của lớp
HinhTamGiac

     //Cài đặt lại các thành phần của
interface Hinh
     public double tinhDienTich()
     {
          //. . .
     }
     public double tinhchuVi()
     {
          //. . .
     }
}
```

```
public class HinhChuNhat:Hinh
{

      //Các thành phần riêng của lớp
HinhChuNhat
      public Diem timGiaoDiemDgCheo(…)

      //Cài đặt lại các thành phần của
interface Hinh
      public double tinhDienTich()
      {
            //. . .
      }
      public double tinhchuVi()
      {
            //. . .
      }
}
```

```
public class Main
{
      public static void main(String[] args)
        {
            Hinh htg =  new HinhTamGiac ();
            Hinh hcn =  new HinhChuNhat ();

      }

}
```

13 Đa hình:

Đa hình là khả năng quyết định trong lúc runtime phương thức nào sẽ được thực thi khi phương thức của lớp cơ sở bị ghi đè (`override`) ở các lớp dẫn xuất.

Các bước đa hình thái:

- Xây dựng lớp cơ sở (lớp bình thường, lớp trìu tượng, giao diện)

- Xây dựng các lớp dẫn xuất ghi đè các phương thức của lớp cơ sở.
- Thực hiện tạo ra đa hình thái bằng cơ chế lớp cơ sở có thể tham chiếu tới 1 đối tượng của lớp dẫn xuất và quyết định phương thức thực thi sẽ là phương thức đã ghi đè, hiện thực hóa ở lớp dẫn xuất tương ứng.

Ví dụ:

```
public class Main
{
     public static void main(String[] args)
     {

          GiangVien gv = new GiangVienCoHuu();
          gv.tinhLuong()//goi phương thức tính lương
của GiangVienCoHuu
          gv = new GiangVienThinhGiang();
           gv.tinhLuong()//goi phương thức tính lương
     của GiangVienThinhGiang
          Hinh h =  new HinhTamGiac ();
          h.tinhDienTich()//goi phương thức tính diện
     tích của HinhTamGiac
          h =  new HinhChuNhat ();
          h.tinhDienTich()//goi phương thức tính diện
     tích của HinhChuNhat


     }

}
```

Lưu ý: khi lớp cơ sở tham chiếu tới 1 đối tượng của lớp dẫn xuất thì chỉ truy xuất được các phương thức được định nghĩa của lớp cơ sở và các phương thức đã được ghi đè, hiện thực hóa ở lớp dẫn xuất tương ứng. Tuy nhiên, khi lớp cơ sở tham chiếu 1 đối tượng của lớp dẫn xuất thì không thể truy xuất được các phương thức được định nghĩa riêng của lớp dẫn xuất. Trong trường hợp này, muốn truy cập các phương thức được định nghĩa riêng của lớp dẫn xuất, phải ép kiểu từ lớp cơ sở về lớp dẫn xuất.

Ví dụ:

```
public class Main
{
      public static void main(String[] args)
      {


            Hinh h =  new HinhChuNhat ();
            h.tinhDienTich()//goi phương thức tính diện
      tích của HinhChuNhat- đã được ghi đè ở lớp
      HinhChuNhat.
            //Gọi phương thức định nghĩa riêng
      timGiaoDienDgCheo() của lớp HinhChuNhat
            //Ép kiểu h thành con trỏ của HinhChuNhat
            HinhChuNhat k=(HinhChuNhat) h;
             k. timGiaoDienDgCheo();

      }

}
```

:

14 Object:

Tất cả các lớp đều được thừa kế từ lớp *Object* cho dù một lớp có khai báo kế thừa từ nó hay không. Do đó, ngoài những thành phần được định nghĩa của 1 lớp, thì nó còn có những thành phần được kế thừa từ lớp *Object*. Hơn nữa, khi định nghĩa 1 lớp, chúng ta có thể ghi đè các phương thức được khai báo virtual của lớp Object:

Các phương thức của lớp Object:

Phương thức	Ý nghĩa
public virtual string ToString()	Trả về một chuỗi mô tả của đối tượng
public virtual int GetHashTable()	Được sử dụng theo từ điển
public virtual bool Equals(object obj)	So sánh thông tin của các đối tượng. Các đối tượng có chứa thông tin giống nhau không?
public static bool Equals(object objA, object objB)	So sánh thông tin của các đối tượng . Các đối tượng có chứa thông tin giống nhau không?
public static bool ReferenceEquals(object objA, object objB)	So sánh tham chiếu của các đối tượng. Các con trỏ có tham chiếu tới một đôí tương hay không?
public Type GetType()	Trả về chi tiết kiểu của một đối tượng.
public object MemberwiseClone()	Tạo ra một bản copy của đối tượng
protected virtual void Finalize()	Phương thức hủy của 1 đối tượng.

15 ArrayList

ArrayList la 1 mảng các đối tượng của lớp Object có kích thước thay đổi động. Do đó, khi khai báo ArrayList, chúng ta không cần phải xác định trước số lượng phần tử tối đa của mảng. Do tất cả các lớp đều hiển nhiên kế thừa từ lớp Object nên 1 phần tử trong ArrayList có thể tham chiếu tới 1 đối tượng của bất kỳ lớp nào.

Thư viện: `using System.Collections;`

Khởi tạo tạo arrayList rỗng:

```
ArrayList list = new ArrayList();
```

Khởi tạo tạo arrayList từ 1 Collection khác:

```
ArrayList list = new ArrayList(Collection  c);
```

Ví dụ:

```
ArrayList list = new ArrayList();
int[] a = new int[16];
//...
ArrayList list1 = new ArrayList(a);
ArrayList list2 = new ArrayList(list);
```

Thêm vào cuối: Add (E element)

```
ArrayList arr = new ArrayList();
PhanSo ps = new PhanSo();
arr.Add(ps);
HocSinh hs = new HocSinh();
arr.Add(hs);
int i = 1;
arr.Add(i);
string s = "ABC";
arr.Add(s);
```

Thêm vào 1 vị trí: Insert (int index, E element);

```
ArrayList arr = new ArrayList();
/…
string r = "BCD";
arr.Insert(0, r);
```

Xóa: remove (E element), remove (int index)

Xóa toàn bộ: clear ()

```
ArrayList arr = new ArrayList();
PhanSo ps = new PhanSo();
arr.Add(ps);
HocSinh hs = new HocSinh();
arr.Add(hs);
string s = "ABC";
arr.Add(s);
arr.RemoveAt(0);
arr.Remove(hs);
arr.Clear();
```

Lấy kích thước: Count

```
ArrayList arr = new ArrayList();
/…
int n=arr.Count;
```

Lấy phần tử: (KieuDuLieu) TenArrayList[index]

```
ArrayList arr = new ArrayList();
PhanSo ps = new PhanSo();
arr.Add(ps);
HocSinh hs = new HocSinh();
arr.Add(hs);
int i =1;
arr.Add(i);
PhanSo ps1 = (PhanSo)arr[0];
HocSinh hs1 = (HocSinh)arr[1];
int i1 = (int)arr[2];
```