Giao diện

v 2.0 - 04/2013

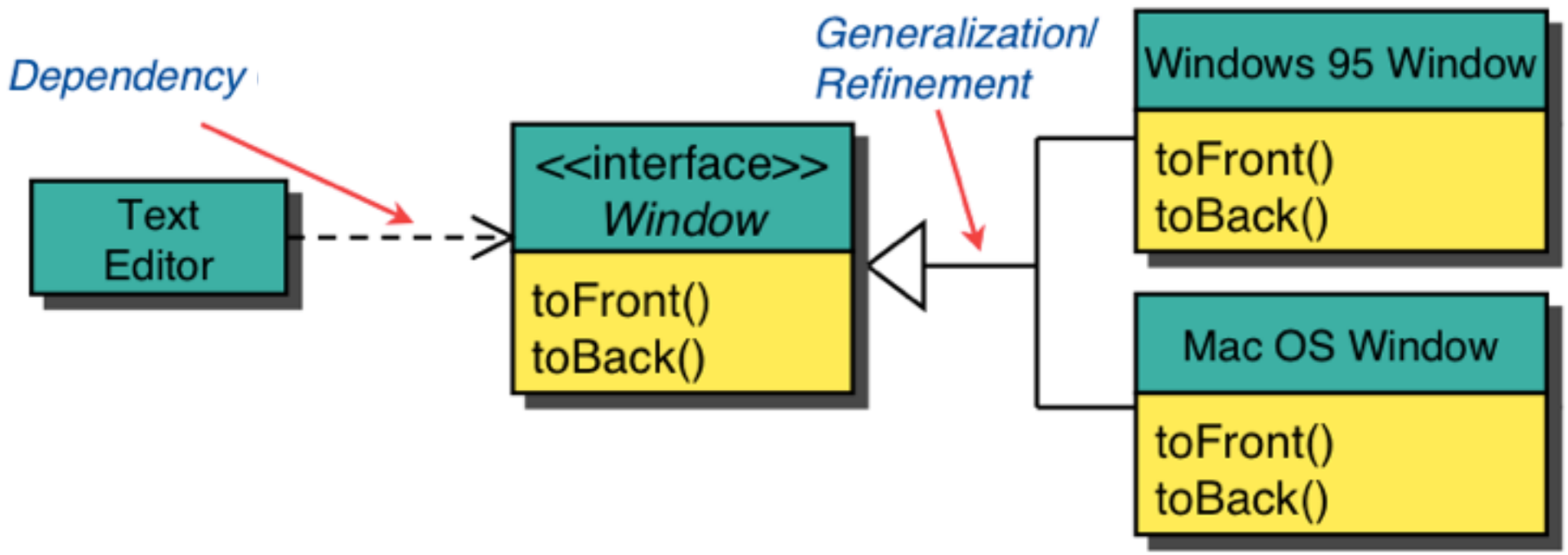
Nội dung

1. Giới thiệu Giao diện (`Interface`)
2. Cài đặt một số giao diện chuẩn

Giao diện

Interface - giao diện

- **Giao diện** là một lớp trừu tượng không **có cài đặt**
  - Giao diện được cài đặt thông qua các lớp khác
  - Các cài đặt có thể được thay đổi mà không cần những thay đổi của đối tượng sử dụng
- Ví dụ : Một chương trình xử lý văn bản hiển thị cửa sổ của nó sử dụng một giao diện cửa sổ mà nó được cài đặt khác nhau cho Windows 95 và Mac OS



Giao diện

- Là một tập các thành phần trừu tượng
- Giao diện cung cấp các hành vi mà một lớp hoặc cấu trúc nào đó lựa chọn để hỗ trợ
- Lớp không bị hạn chế về số lượng giao diện mà nó hỗ trợ

Giao diện	**Lớp trừu tượng**
Tất cả các thành phần đều là trừu tượng	Có thể định nghĩa một số thành phần trừu tượng
Giao diện không phụ thuộc vào cây phân cấp → có thể được thừa kế từ bất kỳ lớp nào	Chỉ những lớp phái sinh mới nạp chồng được các thành phần trừu tượng → phải thuộc cây phân cấp
Cho phép chỉ hỗ trợ cho vài lớp cần thiết trong cây phân cấp. Ví dụ : `HavingPoints` cho kiểu `Hexagon`	Tất cả các lớp phái sinh phải nạp chồng các thành phần trừu tượng

Định nghĩa Giao diện

- Sử dụng từ khoá `interface`, tên giao diện bắt đầu bằng chữ `I`
- Không xác định lớp cơ sở (kể cả `System.Object`)
- Các thành phần cũng không xác định khả năng truy xuất, mặc định là `public` và `abstract`
- Không có biến thành phần và cấu tử
- Không cài đặt cho hàm thành phần
- Có thể định nghĩa các *nguyên mẫu thuộc tính*

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

```
public interface IPointy
{
    byte Points { get; }
}
```

Cài đặt một giao diện

- Cài đặt một giao diện cho một lớp sử dụng cú pháp thừa kế
  - Lớp cơ sở phải được đứng trước

```
class Pencil : IPointy {...}
class SwitchBlade : object, IPointy {...}
class Fork : Utensil, IPointy {...}
struct Arrow : ICloneable, IPointy {...}
```

- Bắt buộc phải nạp chồng toàn bộ các thành phần của giao diện

```
class Hexagon : Shape, IPointy
{
   ...
   //cài đặt IPointy
   public byte Points
   {
      get { return 6; }
   }
}
```

Sử dụng Giao diện

- Không thể cấp phát bộ nhớ cho kiểu giao diện

```
IPointy p = new IPointy(); // Error
```

- Truy xuất các thành phần của giao diện thông qua lớp

```
Hexagon hex = new Hexagon("hex");
Console.WriteLine(hex.Points());
```

- Sử dụng toán tử as và is để kiểm tra một lớp có cài đặt giao diện hay không

```
Shape[] myShapes = {new Hexagon(), new Circle(),
                    new Triangle(), new Circle()};
foreach (Shape s in myShapes)
   if (s is IPointy)
      Console.WriteLine(((IPointy)s).Points());
```

- Giao diện làm đối số của các hàm

```
private static void PointMe(IPointy p)
{
   Console.WriteLine(c.Points());
}
```

Sử dụng Giao diện

- Giao diện là kiểu trả về

```
static IPointy FindFirstPointyShape(Shape[] shapes)
{
   foreach (Shape s in shapes)
      if (s is IPointy) return s as IPointy;
   return null;
}
```

- Sử dụng cú pháp tường minh để tránh xung đột về tên

```
public interface IDrawForm
{
   void Draw();
}
public interface IDrawToMemory
{
   void Draw();
}
class Octagan : IDrawToForm, IDrawToMemory
{
   void IDrawToForm.Draw() {...}

   void IDrawToMemory.Draw() {...}
}
```

Cài đặt một số giao diện chuẩn

Tạo các kiểu cho phép lặp

- Bạn có một lớp chứa một tập hợp các phần tử
- Bạn muốn câu lệnh `foreach` chạy được với lớp này

```
// Lớp Garage chứa một mảng các thành phần kiểu Car
Garage g = new Garage();
...
foreach (Car c in g)
   ...
```

- Cài đặt giao diện `IEnumerable`

```
public interface IEnumerable
{
   IEnumerator GetEnumerator();
}
public interface IEnumerator
{
   bool MoveNext(); //chuyển đến thành phần tiếp theo
   object Current { get; }; // lấy ra thành phần hiện tại
   void Reset(); // chuyển về thành phần đầu tiên
}
```

Cài đặt IEnumerable

```
public class Garage : IEnumerable
{
   private Car[] carArr;
   ...
   // cài đặt IEnumerable
   public IEnumerator GetEnumerator()
   {
      return carArr.GetEnumerator();
   }
}
```

Từ khoá `yield`

- Từ khoá `yield` xác định giá trị được trả ra cho câu lệnh `foreach`
- Khi câu lệnh `yield` được thực thi, vị trí hiện tại trong lớp chứa sẽ được lưu trữ và thực thi được bắt đầu lại tại vị trí này cho lần lặp tiếp theo

```
public class Garage
{
   private Car[] carArr = new Car[4];
   ...
   // cài đặt IEnumerable
   public IEnumerator GetEnumerator()
   {
      yield return carArr[3];
      yield return carArr[2];
      yield return carArr[1];
      yield return carArr[0];
   }
}
```

Cài đặt `ICloneable`

- Giao diện `ICloneable` cung cấp hàm trừu tượng `Clone()` cho phép sao chép từng thành phần của một đối tượng

```
public interface ICloneable
{
    object Clone();
}

public class Point : ICloneable
{
    ...
    public object Clone()
    {
        return new Point(this.X, this.Y);
    }
}
```

```
Point p = new Point(10, 10);
Point p1 = (Point)p.Clone();
```

Cài đặt ICloneable

- Nếu lớp của bạn không có thành phần kiểu tham chiếu
  - Sử dụng hàm protected MemberwiseClone của lớp System.Object
    - MemberwiseClone sẽ sao chép giá trị trên bộ nhớ stack cho đối tượng mới

```
public object Clone()
{
   return this.MemberwiseClone();
}
```

- Nếu có các thành phần kiểu tham chiếu
  - Đầu tiên, dùng MemberwiseClone để sao chép
  - Sau đó, tạo ra các bản sao đối tượng cho các thành phần kiểu tham chiếu

```
public object Clone()
{
   Point newPoint = (Point)this.MemberwiseClone();
   PointDescription currentDesc = new PointDescription(this.desc.Name);
   newPoint.desc = currentDesc;
   return newPoint;
}
```

Cài đặt `IComparable`

- Khi muốn tạo ra các lớp mà nó sẽ hỗ trợ hàm `Sort` của các tập hợp

```
public interface IComparable
{
    int CompareTo(object o);
}
public class Car : IComparable
{
    ...
    int CompareTo(object o)
    {
        Car temp = o as Car;
        if (temp != null)
        {
            if (this.CarID > temp.CarID) return 1;
            if (this.CarID < temp.CarID) return -1;
            else return 0;
        }
        throw new ArgumentException("not a Car");
    }
}
```

Cài đặt IComparable

- hoặc

```
public class Car : IComparable
{
   ...
   int CompareTo(object o)
   {
      Car temp = o as Car;
      if (temp != null)
      {
         return this.CarID.CompareTo(temp.CarID);
      }
      throw new ArgumentException("not a Car");
   }
}
```

```
static void Main()
{
   Car[] myAutos = new Car[5];
   ...
   myAutos.Sort();
}
```

Cài đặt IComparer

```
public interface IComparer
{
    int Compare(object o1, object o2);
}
```

- Giao diện này không được cài đặt ngay trong lớp mà được cài đặt trên các lớp trợ giúp
  - Mỗi lớp trợ giúp cho một cách sắp xếp

```
public class Car : IComparable
{
    ...
    public class PetNameComparer : IComparer
    {
        int Compare(object o1, object o2)
        {
        ...
        }
    }
}
```

Cài đặt IComparer

```
public class Car : IComparable
{
   ...
   public class PetNameComparer : IComparer
   {
      int Compare(object o1, object o2)
      {
         Car t1 = o1 as Car;
         Car t2 = o2 as Car;
         if (t1 != null && t2 != null)
            return String.Compare(t1.PetName, t2.PetName);
         else
            throw new ArgumentException("not a Car");
      }
   }
}
```

```
static void Main()
{
   ...
   //gọi hàm tĩnh Sort, cung cấp đối tượng lớp PetNameComparer
   Array.Sort(myAutos, new Car.PetNameComparer());
   ...
}
```

Cài đặt IComparer

```
public class Car : IComparable
{
   ...
   public class PetNameComparer : IComparer
   {
      int Compare(object o1, object o2)
      {
         Car t1 = o1 as Car;
         Car t2 = o2 as Car;
         if (t1 != null && t2 != null)
            return String.Compare(t1.PetName, t2.PetName);
         else
            throw new ArgumentException("not a Car");
      }
   }

   public static IComparer SortByPetName
   {
      get { return (IComparer)new PetNameComparer(); }
   }
}
```

```
//lấy đối tượng lớp PetNameComparer thông qua thuộc tính
Array.Sort(myAutos, Car.SortByPetName);
```

Cảm ơn sự chú ý

Câu hỏi ?