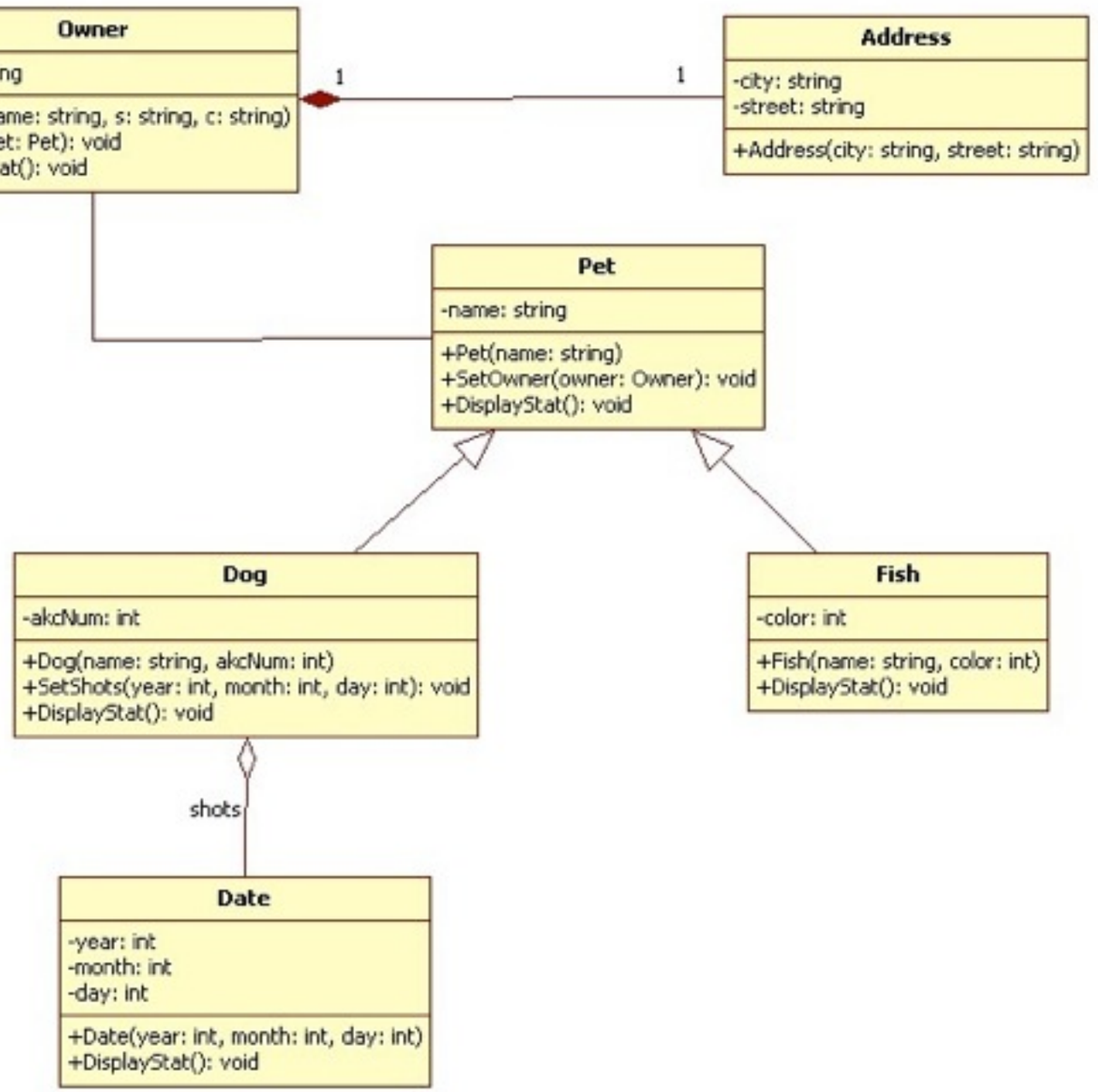
Đa hình

v 2.0 - 04/2013

các bạn đã có thể...

cài đặt mô hình

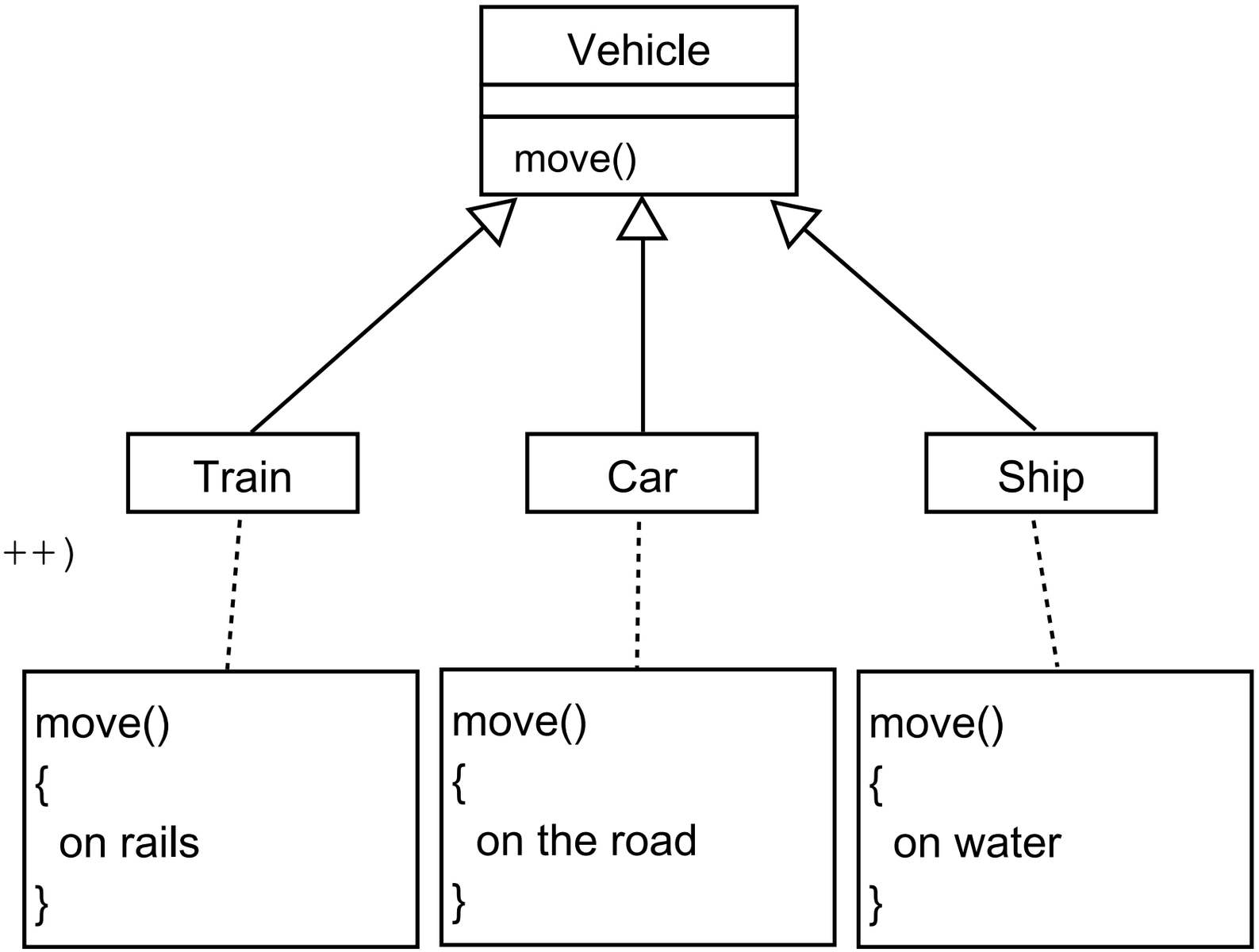


bằng C#

chúng ta sẽ học...

```
Vehicle veh [ 3 ] = {
    Train(“TGV”),
    Car(“twingo”),
    Ship(“Titanic”)
};

for (int i = 0; i < 3; i++)
{
   veh[ i ].move();
}
```



Nội dung

1. Nhắc lại về thừa kế
2. Đa hình
3. Lớp cơ sở trừu tượng
4. Một số vấn đề khác

Nhắc lại về thừa kế

Thừa kế

- Là khả năng lớp con thừa kế từ lớp cha tất cả những thành phần dữ liệu, thuộc tính và hàm thành phần của lớp cha
  - Ngoại trừ : cấu tử, hủy tử, toán tử =
- Cú pháp :

```
class Student : HCEPerson
{
    // Khai báo của lớp Student
}
```

- Khai báo và định nghĩa lớp cơ sở như bình thường
- Toán tử truy xuất
  - private : chỉ cho phép truy xuất bên trong lớp, KHÔNG bao gồm các lớp con
  - protected : chỉ cho phép truy xuất bên trong lớp và cả từ các lớp con của nó

Lớp con

- Trong phần định nghĩa cấu tử
  - Chứa lời gọi đến cấu tử của lớp cha (lớp cơ sở)

Student.cs

```
12          public Student() {}
13          public Student(int id, string name, string address, int course, int year)
14              : base(id, name, address)
15          {
16              this.course = course;
17              this.year = year;
18              this.classesTaken = new List<Class>();
19          }
```

HCEPerson.cs

```
10
11          public HCEPerson() {}
12          public HCEPerson(int id, string name, string address)
13          {
14              this.id = id;
15              this.name = name;
16              this.address = address;
17          }
18
```

Lớp con

- Có thể định nghĩa lại các hàm thành phần của lớp cha

HCEPerson.cs

```
19          public string displayProfile()
20          {
21              return string.Format("[Name : {0}; ID : {1}; Address : {2}]",
22                                   this.name, this.id, this.address);
23          }
24
25          public void changeAddress(string newAddress)...
29      }
30  }
```

Student.cs

```
21          public new string displayProfile()
22          {
23              return string.Format("[Name : {0}; ID : {1}; Address : {2}; Course : {3};" +
24                                   "Year : {4}; Num Of Classes Taken : {5}]",
25                                  this.name, this.id, this.address, this.course,
26                                  this.year, this.classesTaken.Count);
27          }
28
29          public void addClassTaken(Class newClass)...
```

Sử dụng

Program.cs

```
HCEPerson binh = new HCEPerson(901289, "Hoang Van Binh", "1 Le Loi");
Student an = new Student(971232, "Nguyen Van An", "100 Phung Hung", 43, 2);
binh.displayProfile();

binh = an; // chuyển đổi kiểu ngầm định, ngược lại phải viết tường minh

binh.displayProfile();

Class c1 = new Class(“HTTT4253”);
// trình biên dịch sẽ báo lỗi, vì không tồn tại hàm addClassTaken() trong
lớp HCEPerson
binh.addClassTaken(c1);
```

```
[Name : Hoang Van Binh; ID : 901289; Address : 1 Le Loi]
[Name : Nguyen Van An; ID : 971232; Address : 100 Phung Hung]
```

- Kiểu khai báo và kiểu hiện thời
- Điều này không hợp lý
  - Bởi vì nó không phù hợp với kiểu hiện thời mà nó đang nhận
  - Giải pháp cho điều này sẽ tạo ra kỹ thuật đa hình

Đa hình

Đa hình

- Khả năng của kiểu dữ liệu A được xem và được sử dụng như kiểu dữ liệu B
  - Ví dụ : đối tượng kiểu Student có thể được sử dụng thay cho một đối tượng kiểu HCEPerson

```
HCEPerson an =
        new Student(971232, "Nguyen Van An", "100 Phung Hung", 43, 2);

an.displayProfile();
```

```
[Name : Nguyen Van An; ID : 971232; Address : 100 Phung Hung; Course :
43; Year : 2; Num Of Class Taken : 0]
```

- Việc lựa chọn hàm chính xác được thực hiện tại thời gian chạy và dựa trên đối tượng mà một biến đang chứa

Từ khoá `virtual` và `override`

- Sử dụng từ khoá `virtual` để định nghĩa một *hàm thành phần* của *lớp cơ sở* là có thể được *nạp chồng* bởi lớp con
  - Hàm thành phần của lớp cơ sở lúc này được gọi là **hàm ảo**
- Sử dụng từ khoá `override` khi *lớp thừa kế* muốn thay đổi cài đặt của một *hàm ảo*
- Dùng từ khoá `base` để truy xuất những cài đặt của lớp cơ sở

Từ khoá `virtual` và `override`

HCEPerson.cs

```
14          public HCEPerson() { }
15          public HCEPerson(int id, string name, string address)...
21
22          public virtual string displayProfile()
23          {
24              return string.Format("[Name:{0}; ID:{1}; Address:{2}]",
25                                   this.name, this.id, this.address);
26          }
```

Student.cs

```
17              this.year = year;
18          }
19
20          public override string displayProfile()
21          {
22              return string.Format("[Name:{0}; ID:{1}; Address:{2}; Course:{3}, Year:{4}]",
23                                   this.name, this.id, this.address, this.course, this.year);
24          }
25      }
```

Từ khoá `sealed`

- Sử dụng thêm từ khoá `sealed` nếu bạn muốn ngăn không cho nạp chồng các *hàm ảo*

```
20          public override sealed string displayProfile()
21          {
22              return string.Format("[Name:{0}; ID:{1}; Address:{2}; Course:{3}, Year:{4}]",
23                                   this.name, this.id, this.address, this.course, this.year);
24          }
```

Ứng dụng

- Sử dụng một biến của lớp cơ sở để tham chiếu đến một đối tượng của lớp con

```
HCEPerson an =
    new Student(971232, "Nguyen Van An", "100 Phung Hung", 43, 2);

an.displayProfile();
```

- Viết các hàm xử lý cho một lớp các đối tượng

```
Render(Shape s)
{...}
```

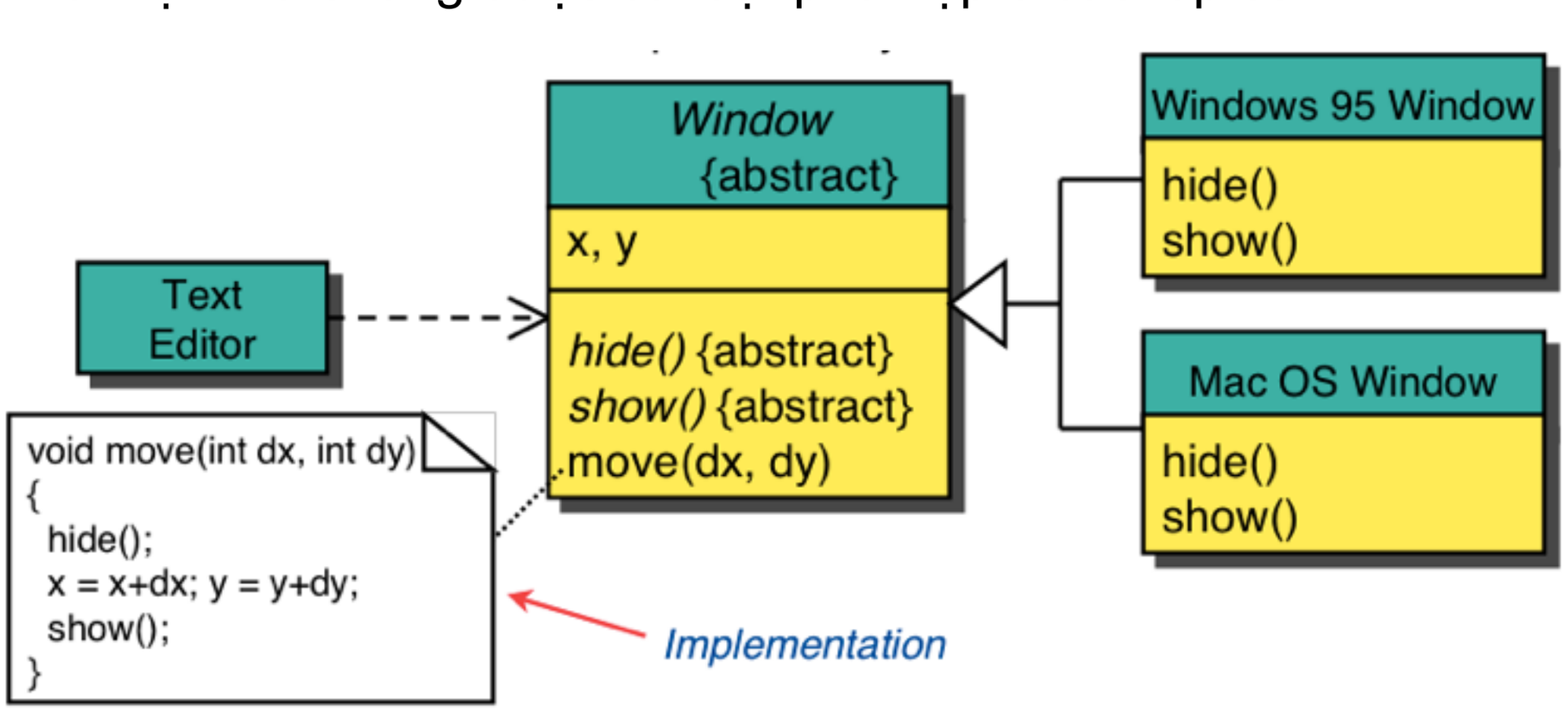
```
Circle c;
Hexagon h;

Render(c);
Render(e);
```

Lớp trừu tượng

Abstract class - lớp trừu tượng

- **Lớp trừu tượng** là một lớp
  - **Phần cài đặt của một số phương thức** bị **bỏ qua**
  - Những phương thức bị bỏ qua chỉ được cài đặt tại các lớp con
- Ví dụ : Thao tác di chuyển cửa sổ được cài đặt sử dụng hai phương thức ẩn và hiện mà chúng được cài đặt phù hợp ở các lớp con



Lớp trừu tượng

- Là lớp với một hoặc nhiều *hàm trừu tượng* và được định nghĩa bằng từ khoá `abstract`

HCEPerson.cs

```
 8    abstract class HCEPerson
 9    {
10        protected string name;
11        protected int id;
12        protected string address;
13
14        public HCEPerson() { }
15        public HCEPerson(int id, string name, string address)...
```

- Không thể khởi tạo

```
HCEPerson p = new HCEPerson(); // không thể làm điều này
```

  - Chỉ có ý nghĩa trong ngữ cảnh thừa kế
    - tổ chức những tính năng chung cho nhiều lớp
    - khai báo giao diện mà mỗi lớp con phải cung cấp
- Lớp con phải cài đặt tất cả các *hàm trừu tượng*

Hàm trừu tượng

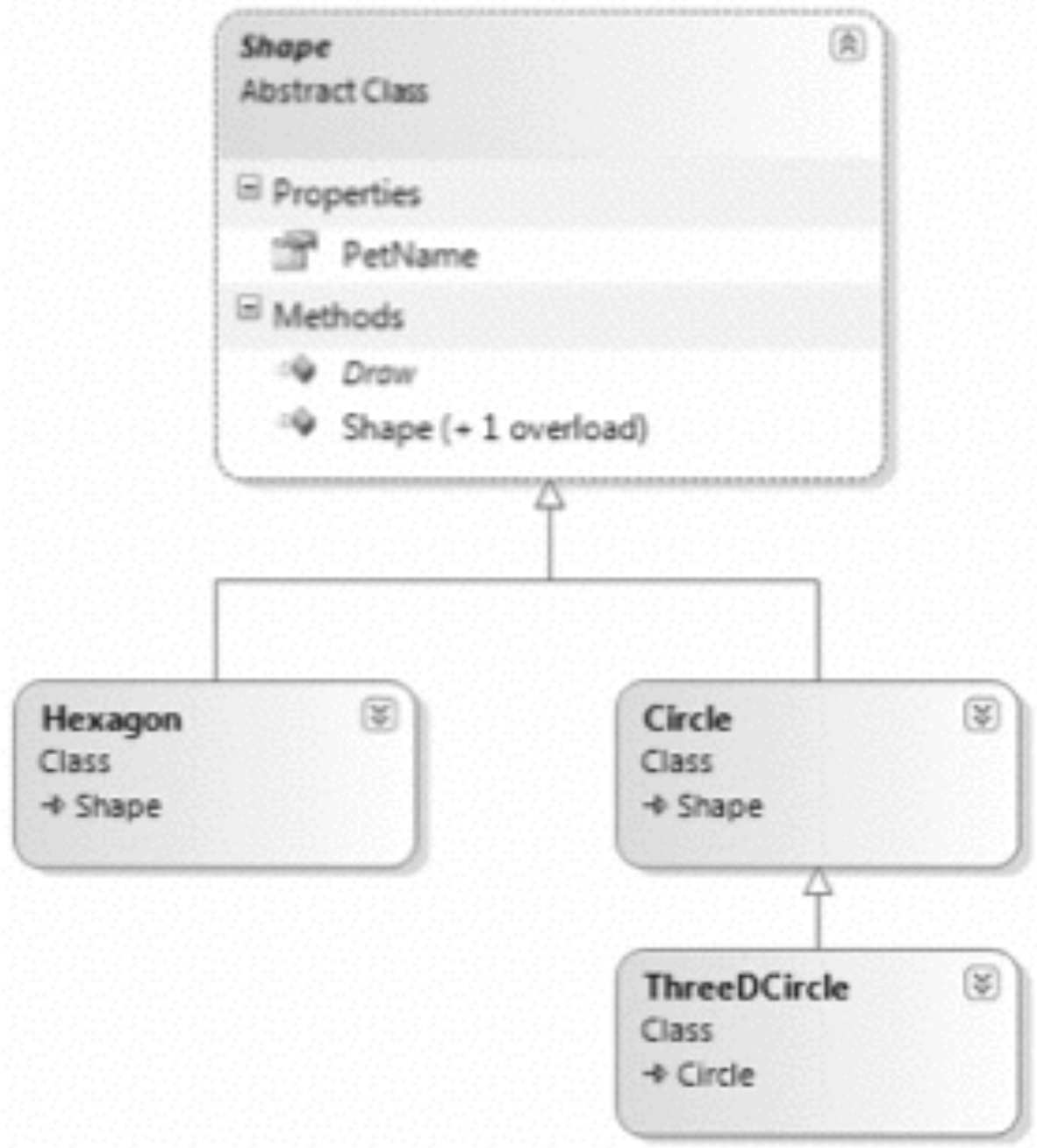
- Là hàm thành phần *không có cài đặt* và sử dụng từ khoá `abstract` trong nguyên mẫu hàm

HCEPerson.cs

```
14          public HCEPerson() { }
15          public HCEPerson(int id, string name, string address)...
21
22          public abstract string displayProfile();
23
24          public void changeAddress(string newAddress)
25          {
26              this.address = newAddress;
27          }
```

- Sử dụng từ khoá `override` khi nạp chồng các hàm trừu tượng

Ví dụ


Shape
Abstract Class
Properties
PetName
Methods
Draw
Shape (+ 1 overload)
Hexagon
Class
Shape
Circle
Class
Shape
ThreeDCircle
Class
Circle

Ví dụ

Shape.cs

```
 8   abstract class Shape
 9   {
10       public string Name { get; set; }
11
12       public Shape(string name = "No Name")
13       {
14           Name = name;
15       }
16
17       public abstract void Draw();
18   }
```

Circle.cs

```
 8   class Circle : Shape
 9   {
10       public Circle() { }
11       public Circle(string name) : base(name) { }
12
13       public override void Draw()
14       {
15           Console.WriteLine("Circle {0}", this.Name);
16       }
17   }
```

Cấu tử trong lớp trừu tượng

- Có ý nghĩa gì không khi định nghĩa một cấu tử ?
- Vì lớp này sẽ không bao giờ được khởi tạo !

- Đúng ! Bạn vẫn nên tạo ra một cấu tử để khởi tạo các thành phần dữ liệu của nó, vì chúng sẽ được thừa kế bởi các lớp con của nó.

Một số vấn đề khác

Kỹ thuật che phủ (shadowing)

- Kỹ thuật đối lập về mặt logic với *chồng hàm*
- Khi lớp con định nghĩa một thành phần trùng lặp với một thành phần của lớp cha, thì lớp con đã *che phủ* phiên bản của lớp cha
  - Hữu ích khi bạn thừa kế từ một lớp mà bạn không tạo ra
    - Vd : Bạn nhận được lớp `Circle` từ người khác và bạn có lớp `ThreeDCircle` mà bạn sẽ cho thừa kế từ lớp `Circle`. Vấn đề là lớp `ThreeDCircle` định nghĩa hàm `Draw` trùng tên với một hàm thành phần của `Circle`
      - Giải pháp là dùng từ khoá `override` hoặc `new` cho hàm `Draw` (của lớp `ThreeDCircle`)
      - Nhưng muốn dùng `override` thì phải có khả năng sửa hàm `Draw` của lớp `Circle` thành `virtual`
- Từ khoá `new` cho phép bỏ qua phiên bản cài đặt ở lớp cơ sở
  - Có thể áp dụng từ khoá `new` cho bất kỳ thành phần nào : biến thành phần, hằng, biến thành phần tĩnh, thuộc tính, hàm thành phần
- Bạn có thể truy xuất phiên bản cài đặt của lớp cha bằng chuyển đổi kiểu tường minh

```
ThreeDCircle o = new ThreeDCircle();
((Circle)o).Draw();
```

Luật chuyển đổi kiểu

- Nhắc lại :
  - Chuyển đổi ngầm định : gán đối tượng lớp con cho biến lớp cha
  - Ngược lại là chuyển đổi tường minh
- System.Object là lớp cơ sở cao nhất trong hệ thống kiểu dữ liệu của nền tảng .NET
  - Mọi thứ đều thừa kế từ Object
  - Có thể chứa một đối tượng kiểu bất kỳ trong một biến kiểu object
  - Một biến kiểu lớp cơ sở có thể chứa bất kỳ đối tượng nào của lớp phái sinh
    - Lợi ích (Xem ứng dụng thứ 2 trong slide 15)

```
object frank = new Hexagon("frank");
```

```
Shape frank = new Hexagon("frank");
Circle jill = new ThreeDCircle("jill");
```

Từ khoá as và is

```
object frank = new Manager(“frank”);
// lỗi tại thời gian chạy
Hexagon hex = (Hexagon)frank; // chuyển đổi tường minh
```

- Chuyển đổi tường minh được thực hiện tại thời gian chạy
  - Nên khi chạy, chương trình trên mới phát sinh lỗi
- Từ khoá as cho phép ép kiểu và nếu không được sẽ trả ra giá trị null

```
Hexagon hex = frank as Hexagon;
if (hex == null)
   // thực hiện điều gì đó
```

- Từ khoá is cho phép kiểm tra một đối tượng thuộc lớp nào
  - Trả ra giá trị true, false

```
static void Draw(Shape s)
{
   if (s is Hexagon)
      // thực hiện thao tác với Hexagon
   if (s is Circle)
      // thực hiện thao tác với Circle
}
```

System.Object

- Trong .NET, mọi kiểu dữ liệu đều thừa kế từ lớp `System.Object`
  - Kể cả các kiểu dữ liệu mà bạn định nghĩa ra
  - Khi một lớp được tạo ra mà không được xác định rõ lớp cơ sở, trình biên dịch sẽ tự động cho phái sinh từ `System.Object`
- `System.Object` định nghĩa một tập các thành phần chung cho mọi kiểu :

```
public class Object
{
   //virtual members
   public virtual bool Equals(object obj);
   public virtual void Finalize();
   public virtual int GetHashCode();
   public virtual string ToString();

   //instance-level, non-virtual members
   public Type GetType();
   protected object MemberwiseClone();

   //static members
   public static bool Equals(object objA, object objB);
   public static bool ReferenceEquals(object objA, object objB);
}
```

System.Object

`Equals()`	Thực hiện so sánh trên các *tham chiếu* của đối tượng. Nếu hai biến cùng tham chiếu đến một đối tượng thì trả ra kết quả `true`, ngược lại là `false` Thường được nạp chồng để thực hiện so sánh trên các giá trị nội tại của đối tượng Khi nạp chồng hàm này, bạn cũng nên nạp chồng hàm `GetHashCode()`
`GetHashCode()`	Hàm này trả ra một số nguyên (`int`) thể hiện cho một đối tượng nhất định
`ToString()`	Hàm này trả ra một *thể hiện chuỗi* của đối tượng
`GetType()`	Hàm này trả ra một đối tượng kiểu `Type` mô tả đối tượng mà bạn đang tham chiếu
`MemberwiseClone()`	Hàm này trả ra một sao chép từng thành phần của đối tượng, được sử dụng khi sao chép (*clone*) một đối tượng
`Equals (object, object)`	Hàm tĩnh, so sánh từng thành phần dữ liệu của hai đối tượng
`ReferenceEquals (object, object)`	Hàm tĩnh, so sánh hai đối tượng có cùng tham chiếu

Ví dụ

```
class Person { }

class Program
{
   static void Main()
   {
      Person p1 = new Person();

      Console.WriteLine(p1.ToString());
      Console.WriteLine(p1.GetHashCode());
      Console.WriteLine(p1.GetType());

      Person p2 = p1;
      object o = p2;
      if (o.Equals(p1) && p2.Equals(o))
         Console.WriteLine("Same instance !");
   }
}
```

Nạp chồng `ToString()`

- Có thể nạp chồng hàm này để trả ra một chuỗi thể hiện nội dung các biến thành phần
  - Hữu ích cho gỡ rối (debug)

```
class Person
{
   public string FirstName { get; set; }
   public string LastName { get; set; }

   public Person() { }
   public Person(string fName, string lName)
   {
      FirstName = fName;
      LastName = lName;
   }

   public override string ToString()
   {
      return string.Format("[FirstName:{0}; LastName:{1}]",
                           FirstName, LastName);
   }
}
```

Nạp chồng Equals()

```
class Person
{
    ...
    public override bool Equals(object obj)
    {
        //kiểm tra obj có phải kiểu Person ?
        if (obj is Person && obj != null)
        {
            Person temp = (Person)obj;
            //so sánh bằng cho từng thành phần dữ liệu
            if (temp.FirstName == this.FirstName
                && temp.LastName == this.LastName)
                return true;
            else
                return false;
        }
        return false;
    }
    ...
}
```

Nạp chồng Equals()

- Nếu đã có nạp chồng hàm ToString()

```
class Person
{
   ...
   public override bool Equals(object obj)
   {
      //không cần kiểm tra hay chuyển đổi kiểu
      return obj.ToString() == this.ToString();
   }
   ...
}
```

Nạp chồng `GetHashCode()`

- Khi đã nạp chồng hàm `Equals()` thì bạn cũng phải nạp chồng hàm `GetHashCode()`
- Mã băm (*hash*) là một giá trị số thể hiện *đối tượng ở tình trạng cụ thể*
  - Hai đối tượng chứa cùng chuỗi Hello thì có mã băm giống nhau và ngược lại
- Mặc định, hàm này sử dụng vị trí của đối tượng trong bộ nhớ để làm mã băm
- Nếu bạn tạo ra một kiểu dự định lưu trữ một giá trị kiểu `Hashtable`, bạn nên nạp chồng hàm này
- Nên lấy mã băm từ các thành phần dữ liệu duy nhất
  - Ví dụ : ID, số bảo hiểm xã hội

Nạp chồng GetHashCode()

```
class Person
{
    ...
    public override int GetHashCode()
    {
        return SSN.GetHashCode();
    }
    ...
}
```

hoặc

```
class Person
{
    ...
    public override int GetHashCode()
    {
        return this.ToString().GetHashCode();
    }
    ...
}
```

Cảm ơn sự chú ý

Câu hỏi ?