Quan hệ giữa các lớp

v 2.0 - 04/2013

các bạn đã có thể...

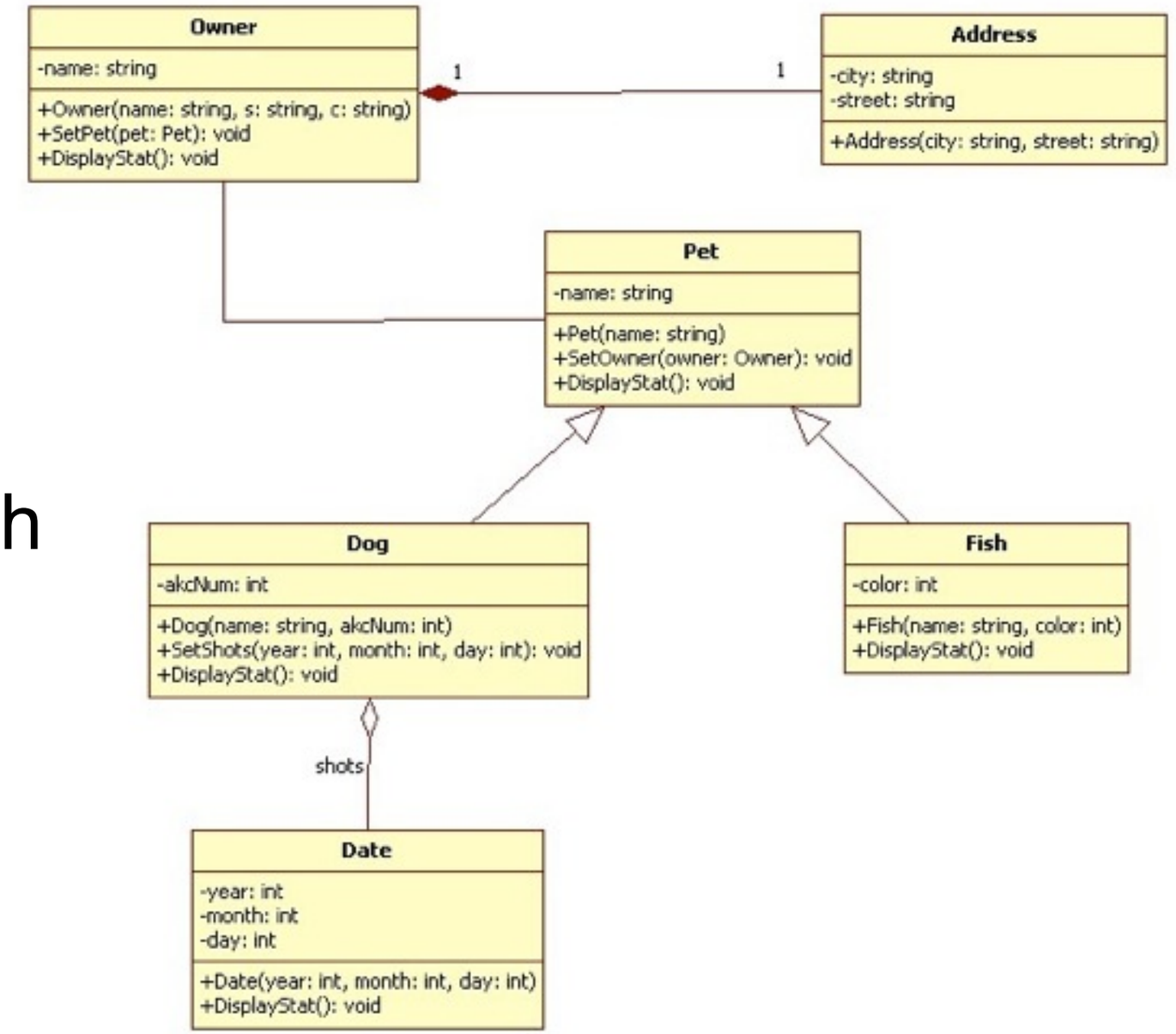
Virus
- reproductionRate : float - resistance : float -instance defaultReproductionRate : float = 0.1
<<constructor>> + Virus(newResistance : float) + Virus(newReproductionRate : float, newResistance : float) <<process>> + reproduce(immunity : float) : Virus* + survive(immunity : float) : bool

Virus.cs

```
using System;

namespace virus
{
    class Virus
    {
        float reproductionRate; // rate of reproduction, in %
        float resistance; // resistance against drugs, in %
        const float defaultReproductionRate = 0.1f;

        public Virus(float newResistance)...

        public Virus(float newReproductionRate, float newResistance)...

        // If this virus cell reproduces,
        // returns a new offspring with identical genetic info.
        // Otherwise, returns NULL.
        public Virus Reproduce(float immunity)...

        // Returns true if this virus cell survives, given the patient's immunity
        public bool Survive(float immunity)...
    }
}
```

chúng ta sẽ học...

cài đặt mô hình



bằng C#

Nội dung

1. Các mối quan hệ lớp
2. Thừa kế
3. Một số vấn đề khác
4. Ví dụ - Pet

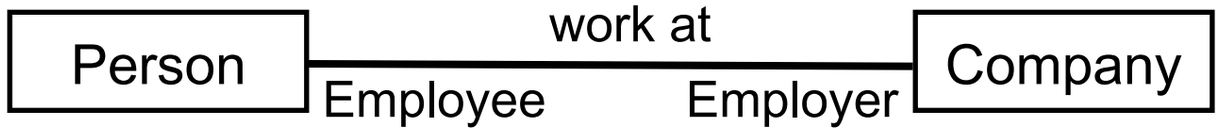
Các mối quan hệ lớp

Mối quan hệ lớp

- Cho phép các đối tượng tương tác với nhau
  - Các đối tượng *giao tiếp* bằng cách *gởi thông điệp* thông qua các kết nối
  - Các đối tượng có thể truy xuất các hàm, thuộc tính của các đối tượng mà nó kết nối
- Được thể hiện thông qua các đồ thị liên kết
  - Các nút / đỉnh là các lớp (hình chữ nhật)
  - Các cạnh / cung là các quan hệ
- Các mối quan hệ lớp
  - Association - q.h kết hợp
  - Aggregation - q.h thu nạp
  - Composition - q.h thành phần
  - Generization - tổng quát hoá (*kỹ thuật thừa kế*)

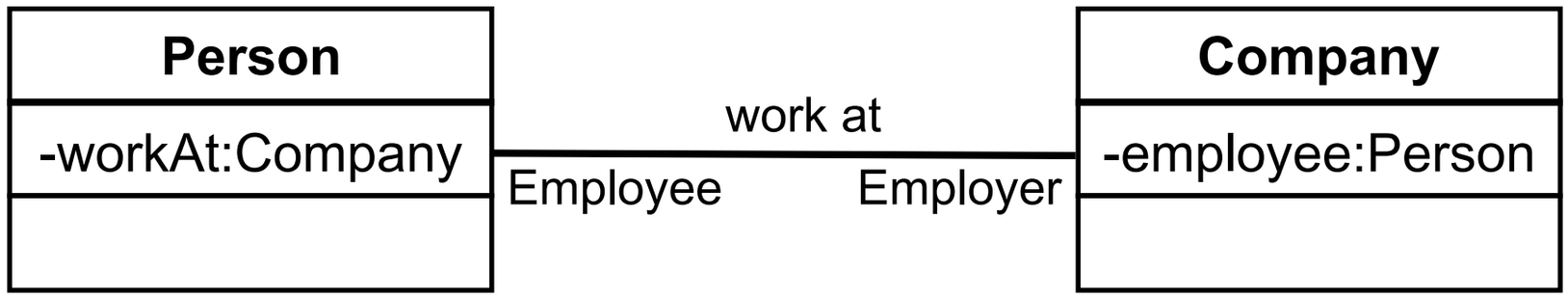
Association - q.h kết hợp

- Mối quan hệ kết hợp thể hiện các mối quan hệ giữa các lớp



- Quan hệ kết hợp cho phép các đối tượng gọi các hàm, thuộc tính lẫn nhau
- Các đối tượng kết hợp không phải tồn tại mãi và không bắt buộc được tạo ra cùng lúc
  - Cho phép null
  - Không được phép hủy cấp phát bộ nhớ của đối tượng nó tham chiếu đến
  - Tạo ra đối tượng mới bên ngoài lớp rồi mới gán vào cho lớp để lưu trữ
  - Khi gán đối tượng mới thì không xoá đối tượng cũ

Ví dụ - q.h kết hợp



Person.cs

```
using System;

namespace AssociationEx
{
    class Person
    {
        string name;
        Company workAt;

        ...
    }
}
```

Company.cs

```
using System;

namespace AssociationEx
{
    class Company
    {
        string name;
        Person employee;

        ...
    }
}
```

Ví dụ - q.h kết hợp

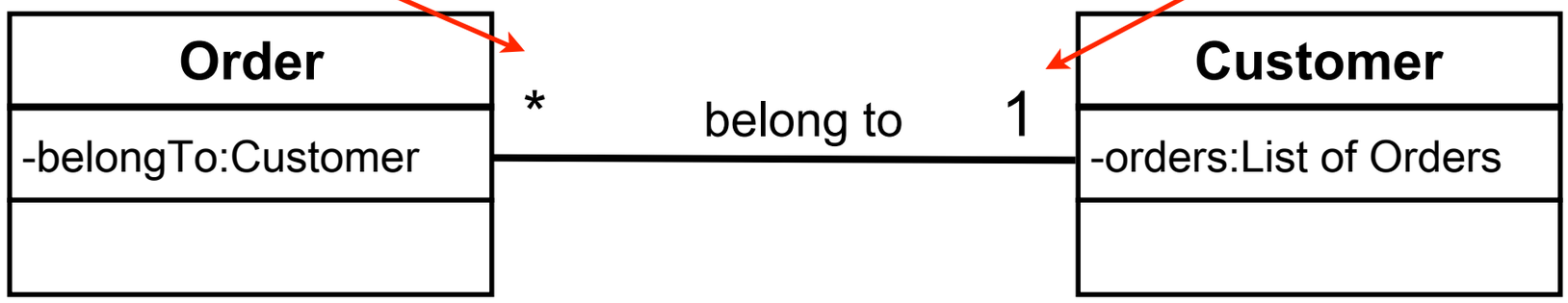
Person.cs

```
using System;

namespace AssociationEx
{
   class Person
   {
      string name;
      Company workAt;

      // Có thể cho phép gán đối tượng ngay tại cấu tử
      public Person (string name) {...}
      public Person (string name, Company c) {...}
      public Person () {}

      public string Name { get {...} set {...} }
      // hoặc gán đối tượng mới thông qua thuộc tính
      public Company WorkAt { get {...} set {...} }

      ...
   }
}
```

Multiplicity - bản số

Một khách hàng có thể có nhiều đơn hàng

Một đơn hàng chỉ thuộc về một khách hàng



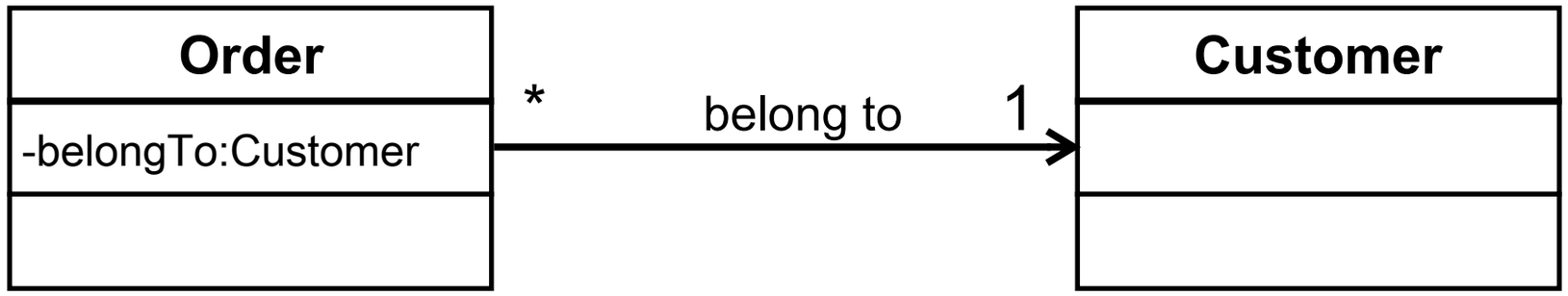
`Order.cs`

```
using System;

namespace AssociationEx
{
    class Order
    {
        int id;
        Customer belongTo;

        ...
    }
}
```

`Customer.cs`

```
using System;
using System.Collections.Generic;

namespace AssociationEx
{
    class Customer
    {
        string name;
        List<Order> orders;

        ...
    }
}
```

Navigability - tính khả điều hướng



- Đơn hàng **biết** nó thuộc về khách hàng nào, nhưng khách hàng không biết nó có đơn hàng nào

Order.cs

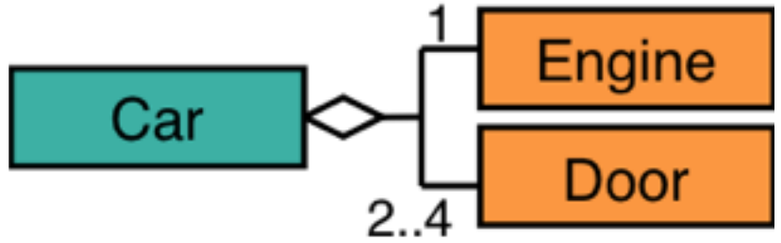
```
using System;

namespace AssociationEx
{
   class Order
   {
      int id;
      Customer belongTo;

      ...
   }
}
```

Customer.cs

```
using System;

namespace AssociationEx
{
   class Customer
   {
      string name;

      ...
   }
}
```

Aggregation - q.h thu nạp

- Quan hệ thu nạp là quan hệ **part-of**
- Quan hệ thu nạp và các đặc tính :
  - Đặc tính mô tả thuộc tính của đối tượng, như tốc độ, giá, chiều dài
  - Quan hệ thu nạp mô tả kết cấu của đối tượng

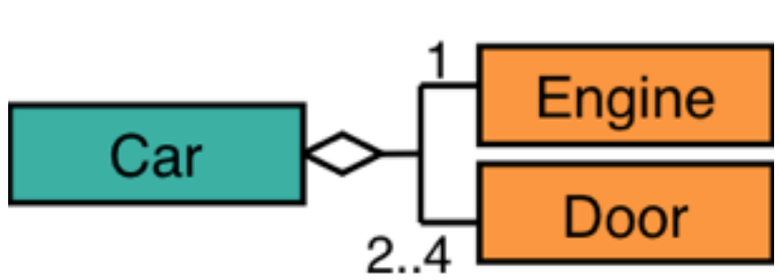


- Cài đặt quan hệ
  - Sử dụng các *liên kết yếu*
    - Các *thành phần* và *toàn thể* có vòng đời độc lập
    - Tạo mối quan hệ khi cần thiết
    - Gán đối tượng mới thì xoá đối tượng cũ
    - Tạo ra đối tượng mới của lớp *thành phần* bên trong hoặc bên ngoài lớp *toàn thể*
    - Xoá đối tượng lớp *thành phần* trong hàm huỷ tử của lớp *toàn thể*

Ví dụ - q.h thu nạp

Car.cs

```
using System;
using System.Collections.Generic;

namespace AggregationEx
{
    class Car
    {
        Engine engine;
        List<Door> doors;

    }
}
```



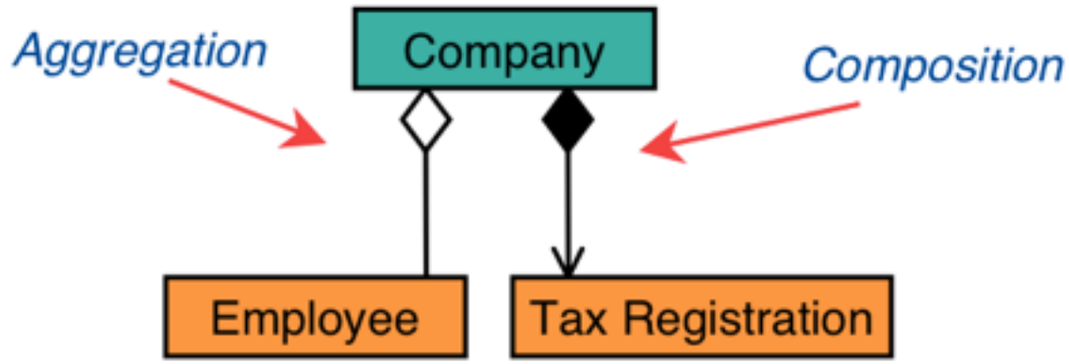
Ví dụ - q.h thu nạp

Car.cs

```
using System;

namespace AggregationEx
{
   class Car
   {
      Engine engine;

      public Car () {}
      // Xoá đối tượng trong hàm huỷ tử
      ~Car () { engine = null; }

      // Tạo đối tượng mới bên ngoài lớp rồi gán thông qua thuộc tính
      // không cần thao tác xoá đối tượng cũ
      public Engine CarEngine { get {...} set {...} }

      // Tạo đối tượng mới bên trong lớp
      public void SetEngine(string nameEngine)
      {
         engine = new Engine(nameEngine);
      }
      ...
   }
}
```

Composition - q.h thành phần

- Quan hệ thành phần là biến thể **mạnh hơn** của quan hệ thu nạp
  - Một thành phần **chỉ thuộc về** một toàn thể
  - Các thành phần thường sống và **chết** theo toàn thể

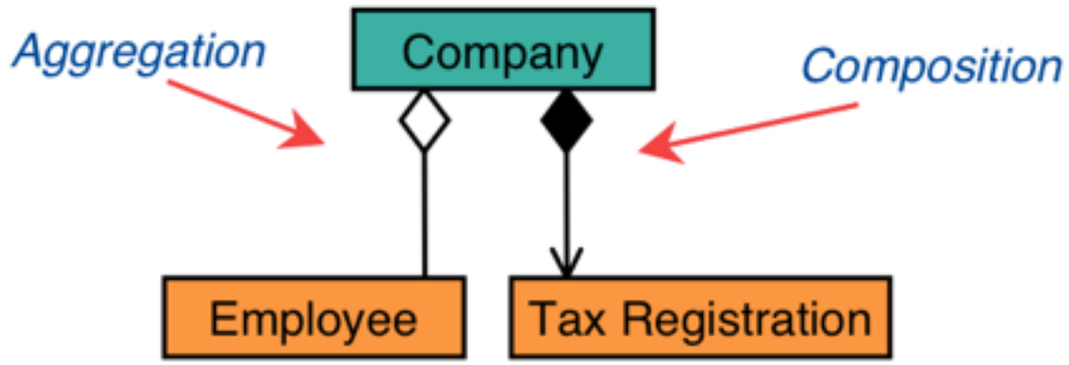


- Cài đặt quan hệ
  - Sử dụng *liên kết mạnh*
  - Các *thành phần* và *toàn thể* có đời sống trùng lặp nhau
  - Khởi tạo các *thành phần* trong *cấu tử của toàn thể*
  - Các *thành phần* chỉ thuộc về một *toàn thể*
  - Các *thành phần* không thể thay đổi trong suốt quá trình thực thi
  - Tạo đối tượng lớp *thành phần* bên trong lớp *toàn thể*
  - Không gán đối tượng mới, chỉ cho phép thay đổi dữ liệu
  - Xoá đối tượng lớp *thành phần* trong hàm huỷ tử của lớp *toàn thể*

Ví dụ - q.h thành phần

Company.cs

```
using System;
using System.Collections.Generic;

namespace CompositionEx
{
    class Company
    {
        List<Employee> employees;
        TaxRegistration taxReg;

        ...
    }
}
```



Ví dụ - q.h thành phần

Company.cs

```
using System;

namespace CompositionEx
{
   class Company
   {
      TaxRegistration taxReg;

      public Company () {}
      // Tạo đối tượng mới bên trong cấu tử
      public Company (string id, int year, int month, int day)
      { taxReg = new TaxRegistration( id, year, month, day ); }
      // Xoá đối tượng trong hàm huỷ tử
      ~Car () { taxReg = null; }

      // Chỉ cho phep thay đổi dữ liệu, không gán đối tượng mới
      public string TaxReg { get {...} set { taxReg.id = value; } }

      ...
   }
}
```

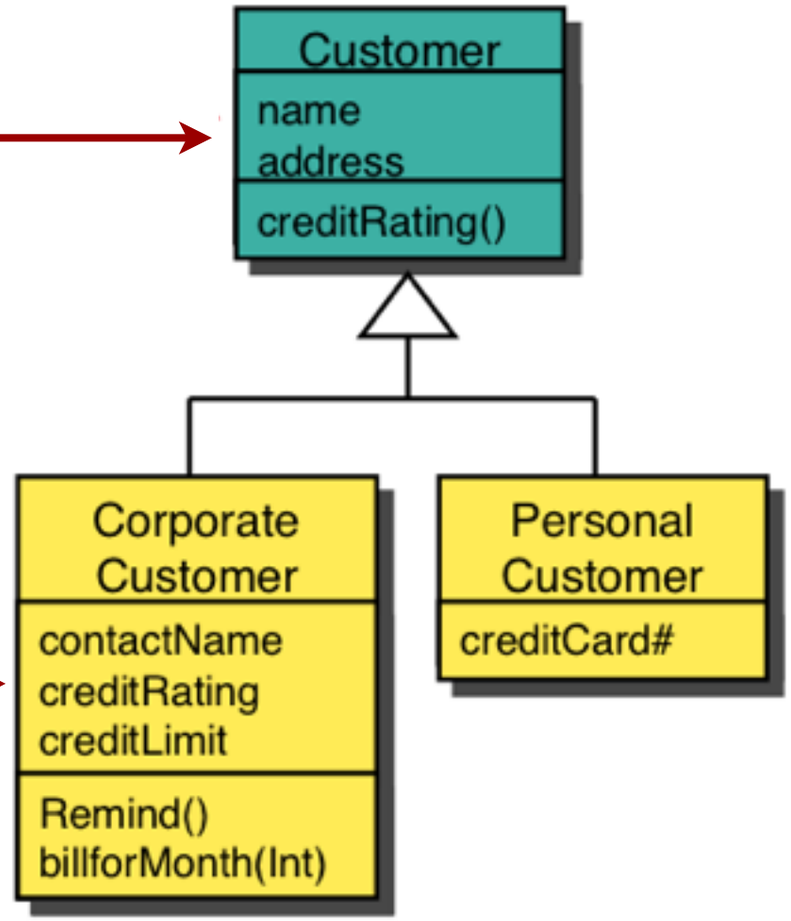
Inheritance - Thừa kế

Generalization - tổng quát hoá

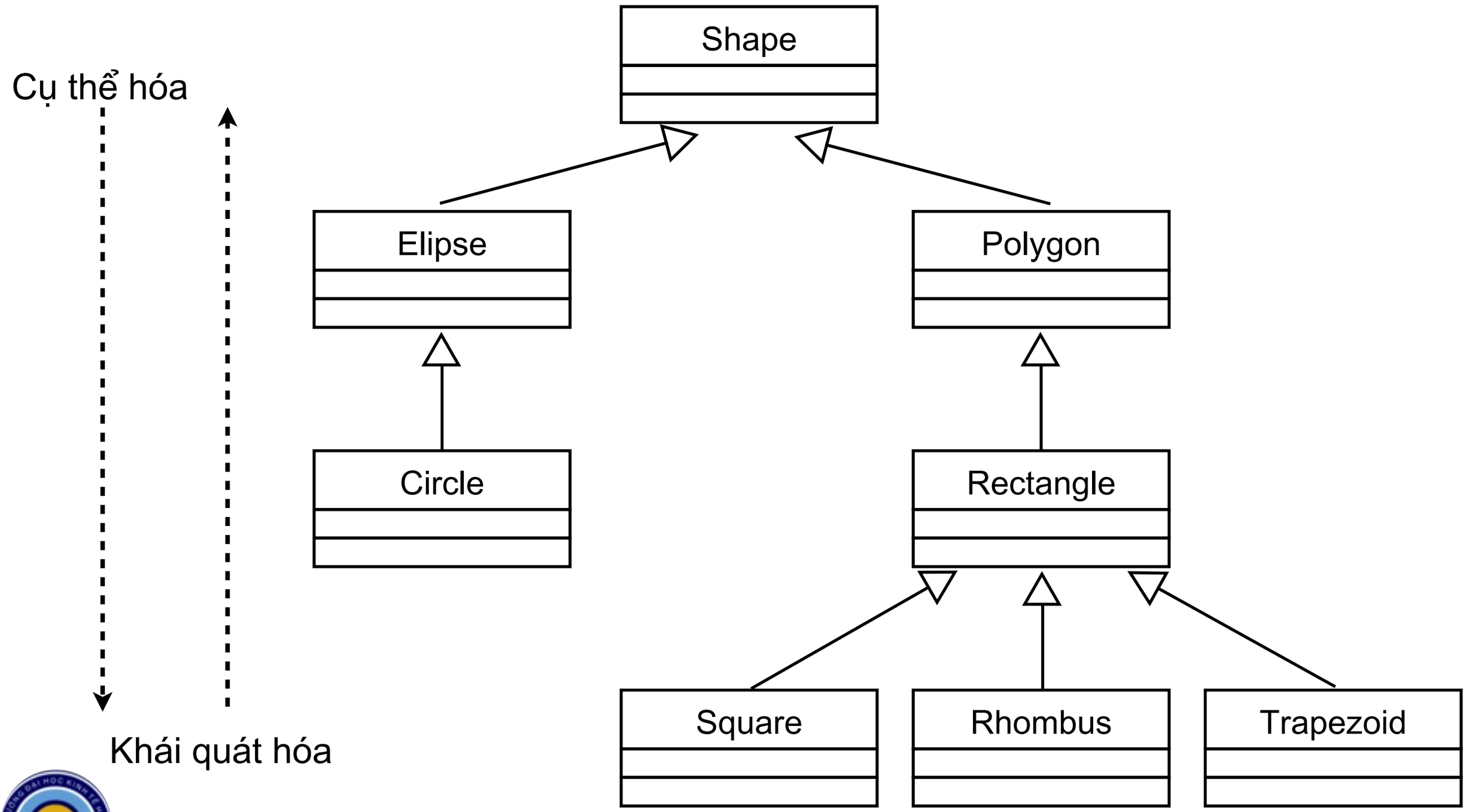
- **Tổng quát hóa** gom **những thứ giống nhau** giữa vài lớp trong một *lớp cha* (superclass)
- **Cụ thể hóa** (specialization) thêm **những thứ khác nhau** vào trong *lớp con*

Những đặc tính giống nhau được đặt ở lớp cha

Những đặc tính khác nhau được tách ra đặt ở các lớp con



Phân cấp thừa kế



Thừa kế

- Là một quan hệ giữa một lớp và một phiên bản cụ thể hơn
- Sự trừu tượng cho phép chia sẻ những điểm tương tự giữa các lớp trong khi ngăn chặn những điểm khác biệt
  - Cơ chế cho phép sử dụng lại mã nguồn
  - Sự đơn giản hóa về khái niệm bằng cách làm giảm số lượng đặc tính riêng
- *Lớp con* (lớp phái sinh) thừa kế tất cả các *đặc tính* của *lớp cha* (lớp cơ sở)
- Một thể hiện của lớp con là một thể hiện của cả lớp cha của nó
- *Nạp chồng* - lớp con định nghĩa các hàm thành phần cùng tên và cùng tham số với các hàm thành phần trong lớp cha

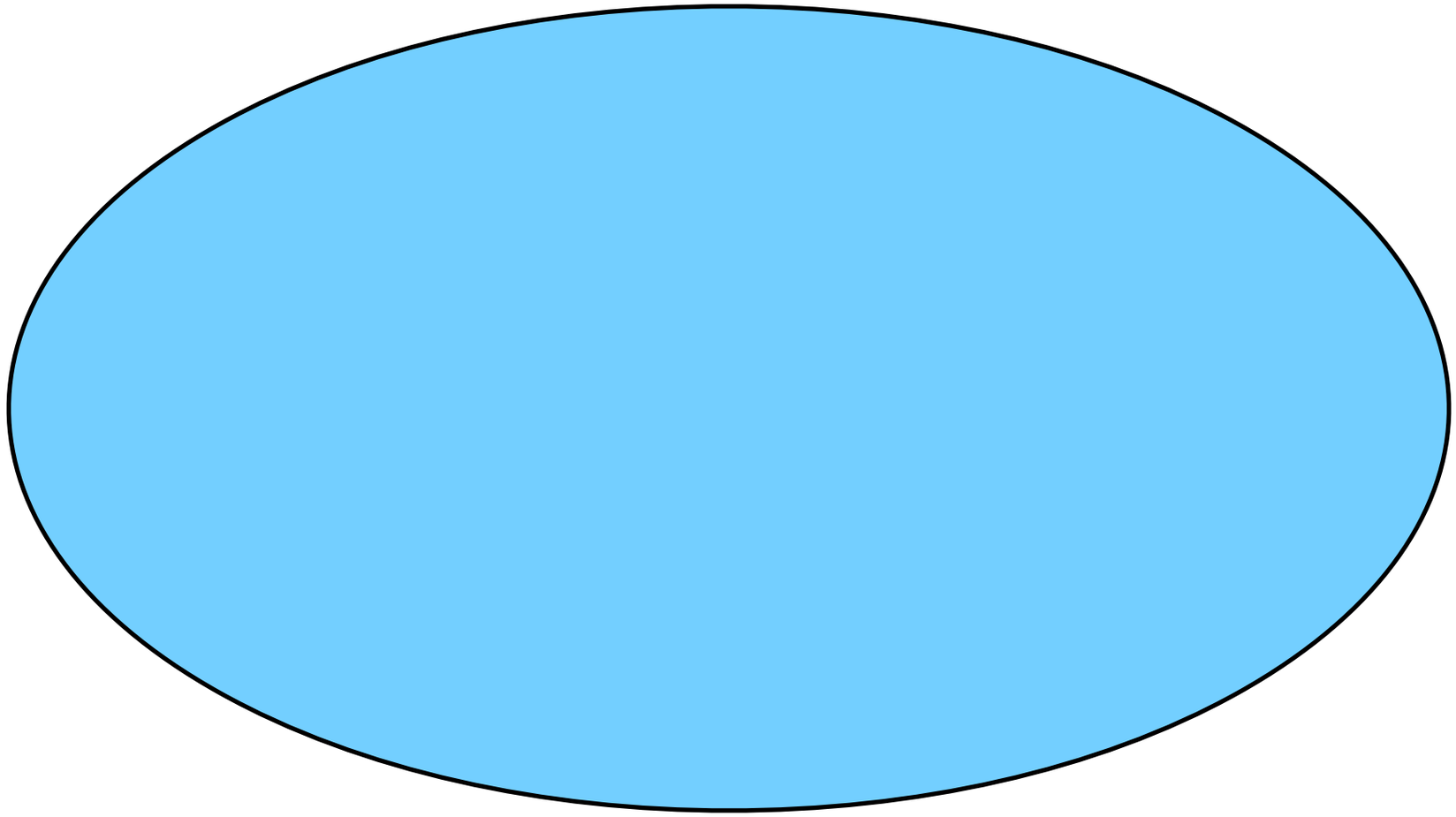
Cái gì được thừa kế ?

- Được thừa kế
  - Các thành phần dữ liệu
  - Hầu hết các hàm thành phần, thuộc tính
- Các hàm không được thừa kế
  - Cấu tử
  - Hủy tử
  - Toán tử gán (=)
- Tất cả các cấu tử và hủy tử được thực thi theo cây phân cấp
  - Các cấu tử thực thi từ trên xuống
  - Các hủy tử thực thi từ dưới lên

Ví dụ

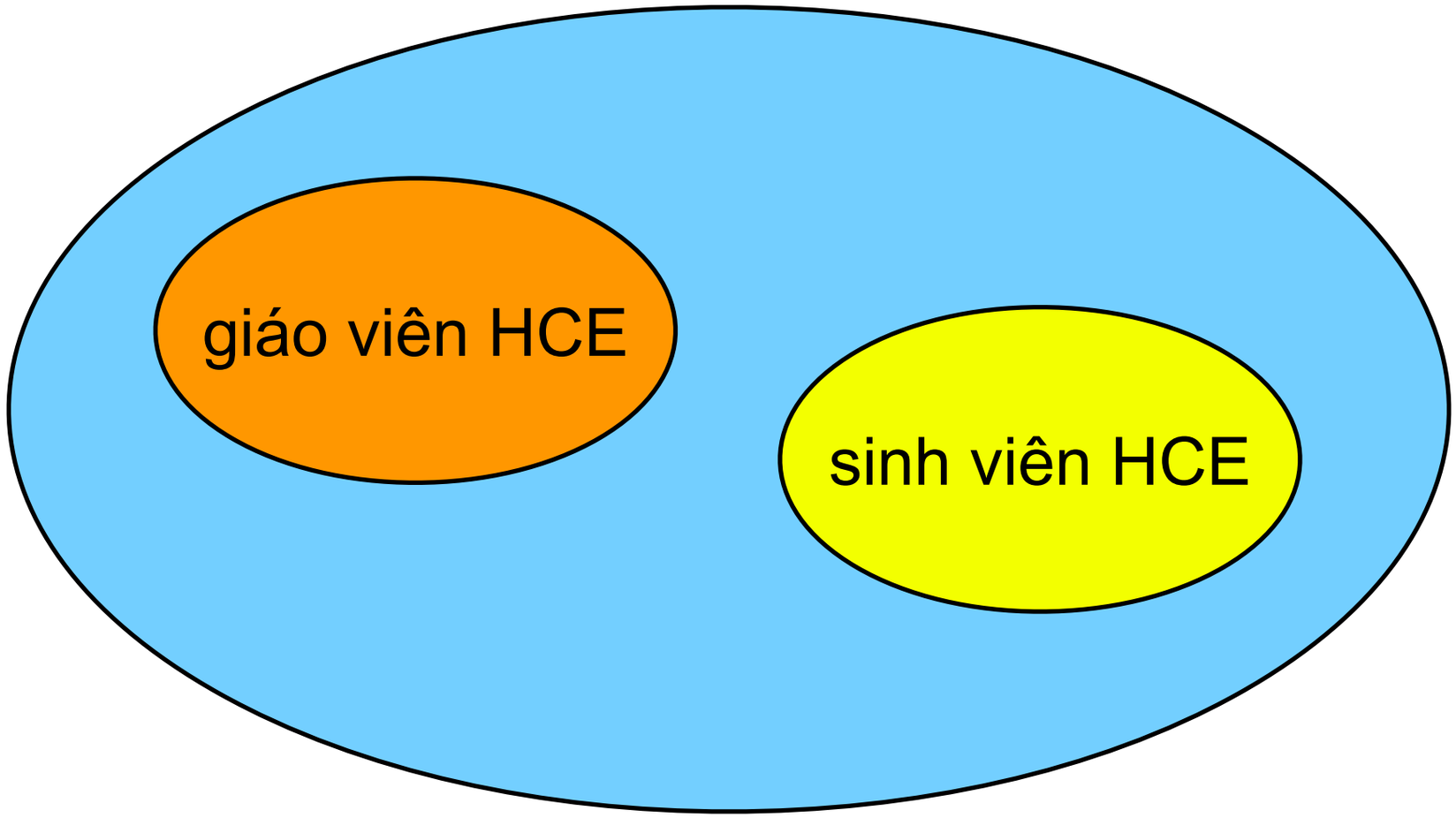
Kiểu dữ liệu

- Một lớp định nghĩa một tập các đối tượng (hay một kiểu dữ liệu)



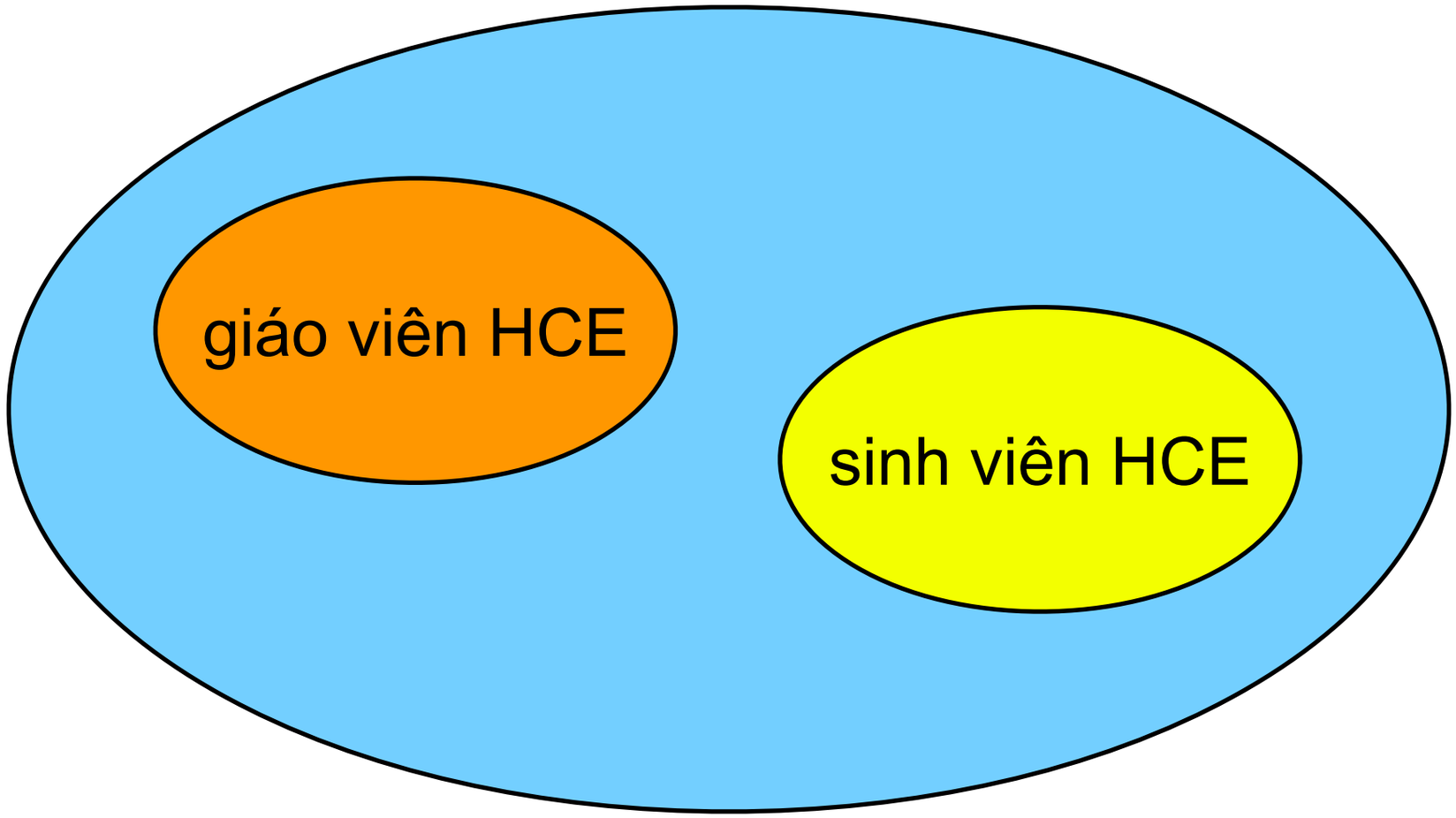
con người ở HCE

Kiểu d.l bên trong một kiểu d.l

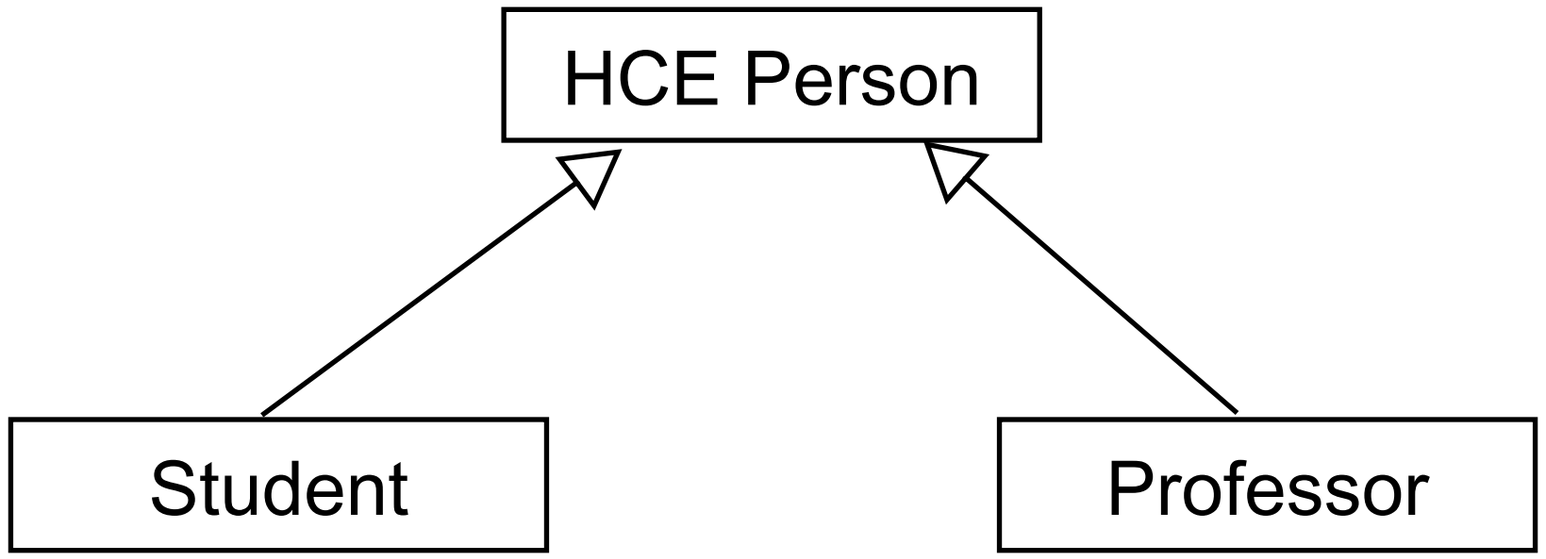


Kiểu dữ liệu con

- Giáo viên và sinh viên là các kiểu dữ liệu con của con người



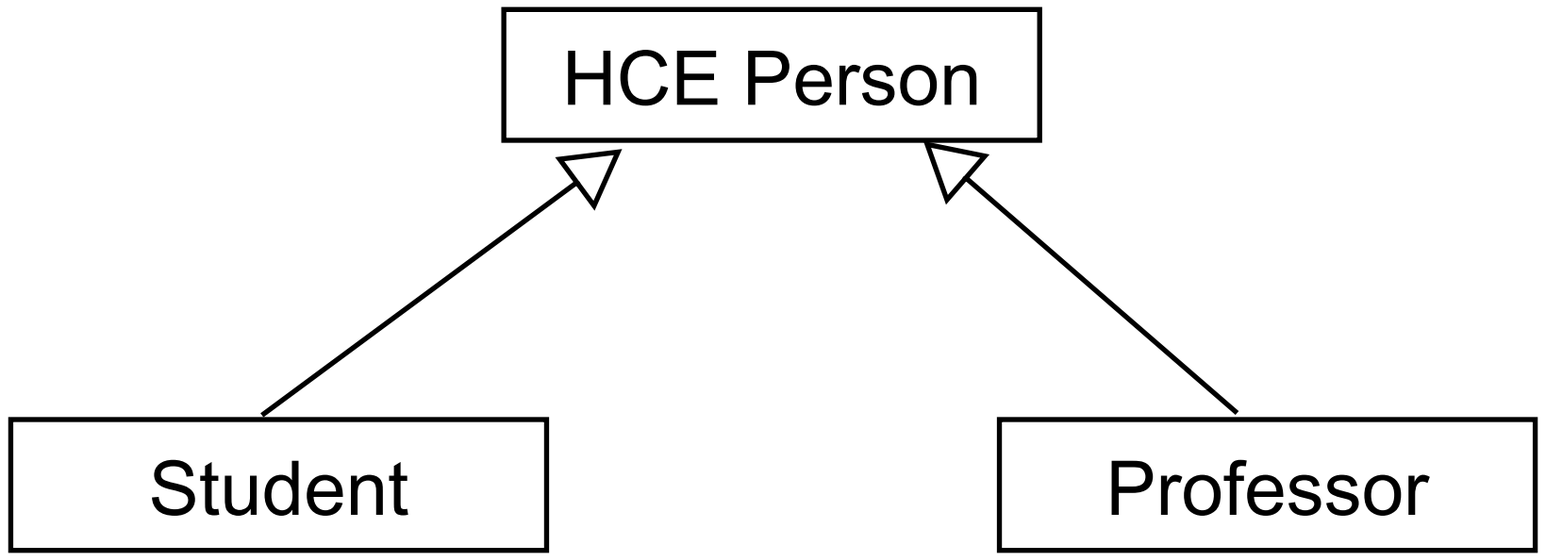
Cây phân cấp kiểu (1/3)



Các đặc tính / hành vi nào mà tất cả con người ở HCE đều có ?

- tên, mã số, địa chỉ
- thay đổi địa chỉ, hiển thị thông tin

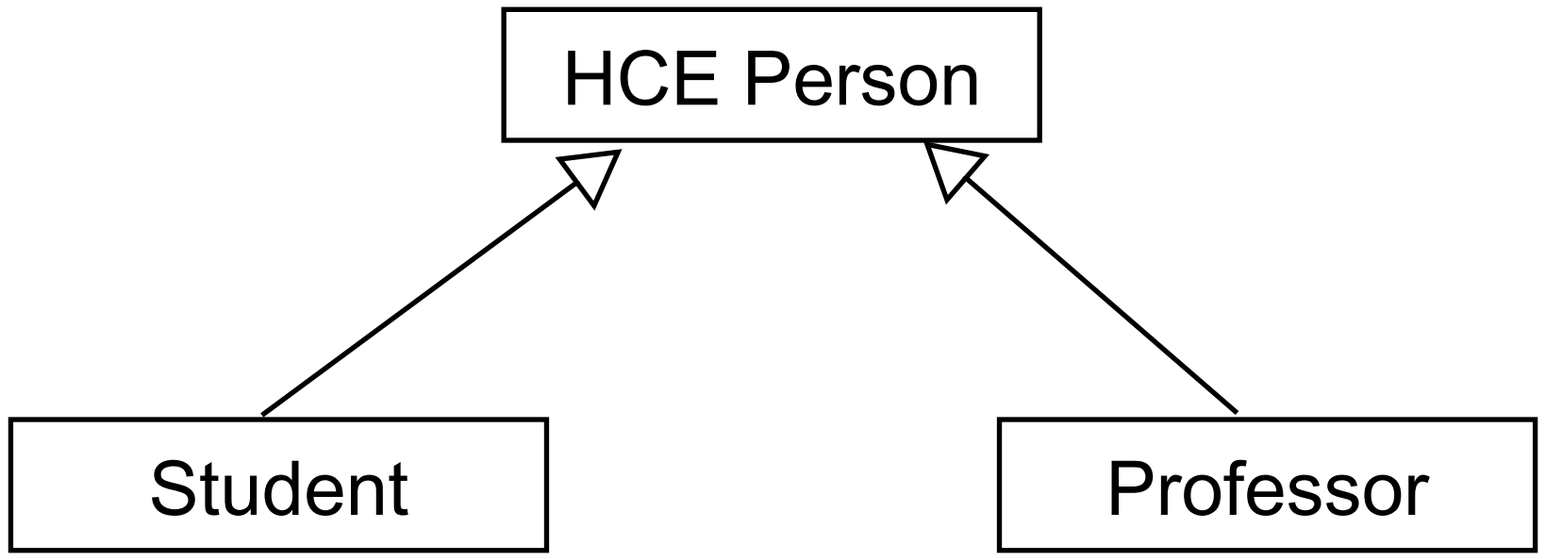
Cây phân cấp kiểu (2/3)



Các đặc tính / hành vi nào là cụ thể cho sinh viên ?

- khóa học, các lớp học đang theo học, năm học
- thêm một lớp học, thay đổi năm học

Cây phân cấp kiểu (3/3)



Các đặc tính / hành vi nào là cụ thể cho giáo viên ?

- các lớp đang dạy, thứ hạng (giáo sư, trợ lý giáo sư)
- thêm một lớp dạy, lên chức

Thừa kế

- Một *lớp con* thừa kế các đặc tính và hành vi của *lớp cha*

- Ví dụ : Mỗi sinh viên HCE có

Đặc tính :	*Hành vi :*
name	display profile
ID	change address
address	add a class taken
course number	change course
year	
classes taken	

Lớp cơ sở : HCEPerson

HCEPerson.cs

```
1  using System;
2
3  namespace HCE
4  {
5      class HCEPerson
6      {
7          protected int id;
8          protected string name;
9          protected string address;
10
11         public HCEPerson() {}
12         public HCEPerson(int id, string name, string address)...
18
19         public string displayProfile()...
24
25         public void changeAddress(string newAddress)...
29     }
30 }
31
```

Lớp cơ sở : HCEPerson

HCEPerson.cs

```
1  using System;
2
3  namespace HCE
4  {
5      class HCEPerson
6      {
7          protected int id;
8          protected string name;
9          protected string address;
10
11         public HCEPerson() {}
12         public HCEPerson(int id, string name, string address)...
18
19         public string displayProfile()...
24
25         public void changeAddress(string newAddress)...
29     }
30 }
31
```

Toán tử truy xuất `protected`

Toán tử truy xuất

- `public`
  - cho phép truy xuất bởi bất cứ *ai*
- `protected`
  - cho phép truy xuất bên trong lớp và bởi tất cả các lớp con của nó
- `private`
  - chỉ cho phép truy xuất bên trong lớp, KHÔNG bao gồm các lớp con

Lớp phái sinh : Student

Student.cs

```
1  using System;
2  using System.Collections.Generic;
3
4  namespace HCE
5  {
6      class Student : HCEPerson
7      {
8          int course;
9          int year;
10         List<Class> classesTaken;
11
12         public Student() {}
13         public Student(int id, string name, string address, int course, int year)
14             : base(id, name, address)...
20
21         public new string displayProfile()...
26
27         public void addClassTaken(Class newClass)...
31
32         public void changeCourse(int newCourse)...
36     }
37 }
38
```

Lớp phái sinh : Student

Student.cs

```
1  using System;
2  using System.Collections.Generic;
3
4  namespace HCE
5  {
6      class Student : HCEPerson
7      {
8          int course;
9          int year;
10         List<Class> classesTaken;
11
12         public Student() {}
13         public Student(int id, string name, string address, int course, int year)
14             : base(id, name, address)...
20
21         public new string displayProfile()...
26
27         public void addClassTaken(Class newClass)...
31
32         public void changeCourse(int newCourse)...
36     }
37 }
38
```

Cú pháp thừa kế

Khả năng truy xuất từ lớp con

Thành phần lớp cơ sở	Truy xuất từ lớp	Thành phần lớp phái sinh
`public`	ĐƯỢC	`public`
`protected`	ĐƯỢC	`private`
`private`	KHÔNG	

Protected

- Lợi ích : các kiểu phái sinh không còn phải truy xuất gián tiếp các thành phần sử dụng các hàm `public` hoặc thuộc tính
- Nguy cơ : có thể bỏ qua các kiểm tra hợp lý dữ liệu (`business rules`) trong các thuộc tính
- Không nên tạo ra các biến thành phần `protected`, nhưng có thể tạo ra một số hàm thành phần `protected`

Khởi tạo một đối tượng của lớp con

Student.cs

```
1  using System;
2  using System.Collections.Generic;
3
4  namespace HCE
5  {
6      class Student : HCEPerson
7      {
8          int course;
9          int year;
10         List<Class> classesTaken;
11
12         public Student() {}
13         public Student(int id, string name, string address, int course, int year)
14             : base(id, name, address)...
20
21         public new string displayProfile()...
26
27         public void addClassTaken(Class newClass)...
31
32         public void changeCourse(int newCourse)...
36     }
37 }
38
```

Khởi tạo một đối tượng của lớp con

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
```

Khởi tạo một đối tượng của lớp con

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          }
```

Lời gọi cấu tử của lớp cơ sở

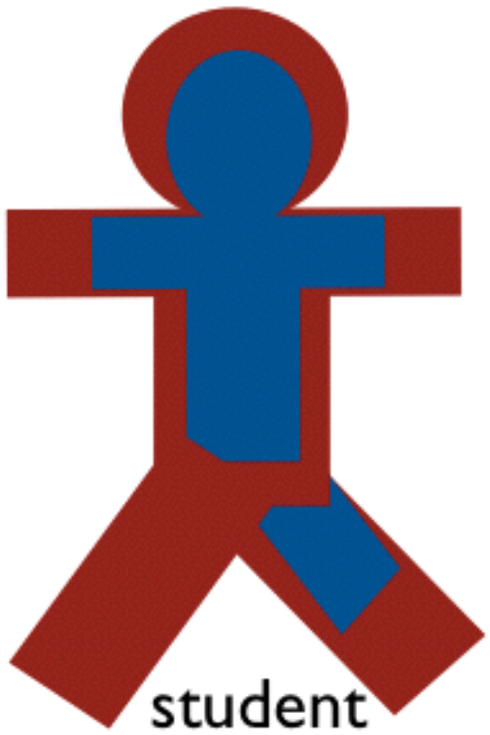
HCEPerson.cs

```
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
```

Khởi tạo một đối tượng của lớp con

Program.cs

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



name = “Nguyen Van An”

ID = 971232

address = “100 Phung Hung”

course = 43

year = 2

classes taken

Nạp chồng một hàm của lớp cơ sở

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)...
```

Nạp chồng hàm để hiển thị thêm thông tin

Nạp chồng một hàm của lớp cơ sở

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)...
```

Tạo ra phiên bản mới sử dụng từ khoá new

Nạp chồng một hàm của lớp cơ sở

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  }
```

Sử dụng từ khoá `base` để gọi phiên bản của lớp cơ sở

Student.cs

```
21          public new string displayProfile()
22          {
23              return base.displayProfile()
24                     + string.Format("[Course : {0}; Year : {1}; Num Of Classes Taken : {2}]",
25                                     this.course, this.year, this.classesTaken.Count);
26          }
27
28          public void addClassTaken(Class newClass)...
32
```

Nạp chồng một hàm của lớp cơ sở

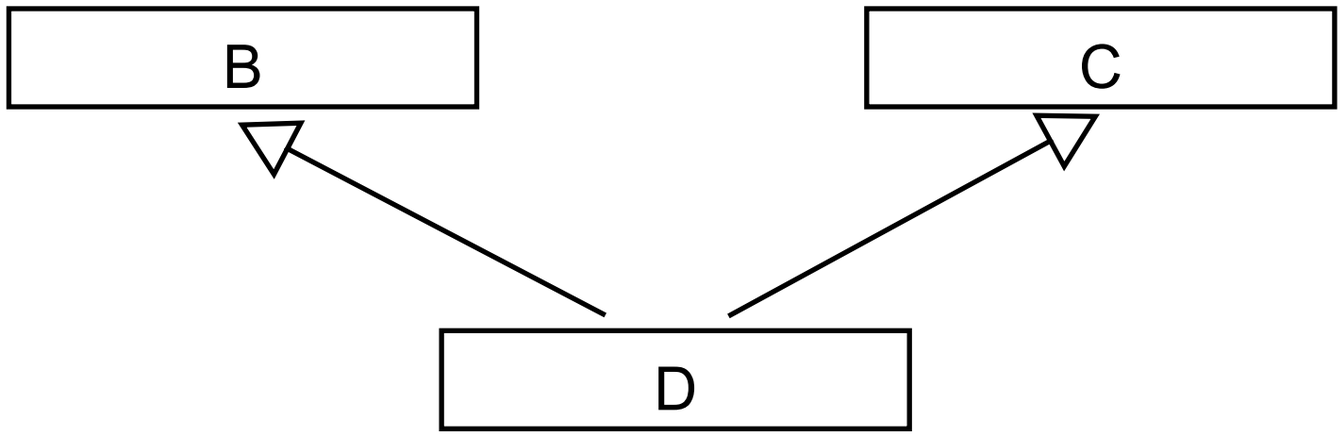
Program.cs

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

Class c1 = new Class("HTTT4253");
an.addClassTaken(c1);

binh.displayProfile();
an.displayProfile();
```

```
[Name : Hoang Van Binh; ID : 901289; Address : 1 Le Loi]
[Name : Nguyen Van An; ID : 971232; Address : 100 Phung Hung; Course : 43;
Year : 2; Num Of Clasess taken : 1]
```

Một số vấn đề khác

Đa thừa kế



- C# KHÔNG hỗ trợ đa thừa kế lớp
- Chỉ hỗ trợ đa thừa kế hành vi thông qua giao diện (`Interface`)

Kiểu hiện thời và kiểu khai báo

- Mỗi biến có một kiểu khai báo tại thời điểm biên dịch
- Nhưng trong thời gian chạy, biến đó có thể tham chiếu đến một đối tượng có kiểu hiện thời
  - Có thể là cùng kiểu hoặc kiểu con của kiểu khai báo

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

- Đâu là kiểu khai báo của biến `binh` và `an` ?
- Đâu là kiểu hiện thời của chúng ?

Gọi hàm được nạp chồng

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

an.displayProfile();
```

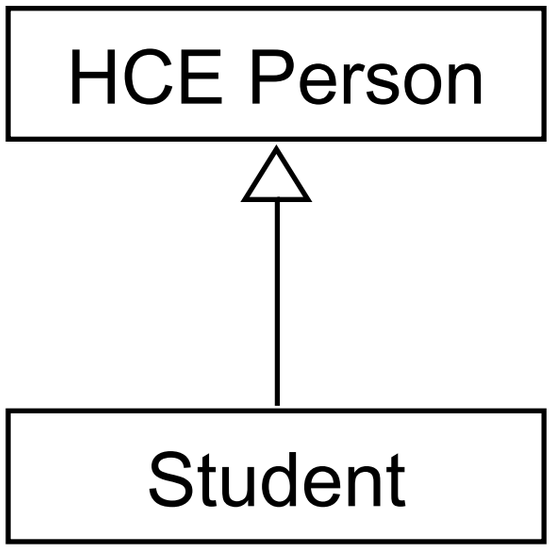
Gọi hàm được nạp chồng

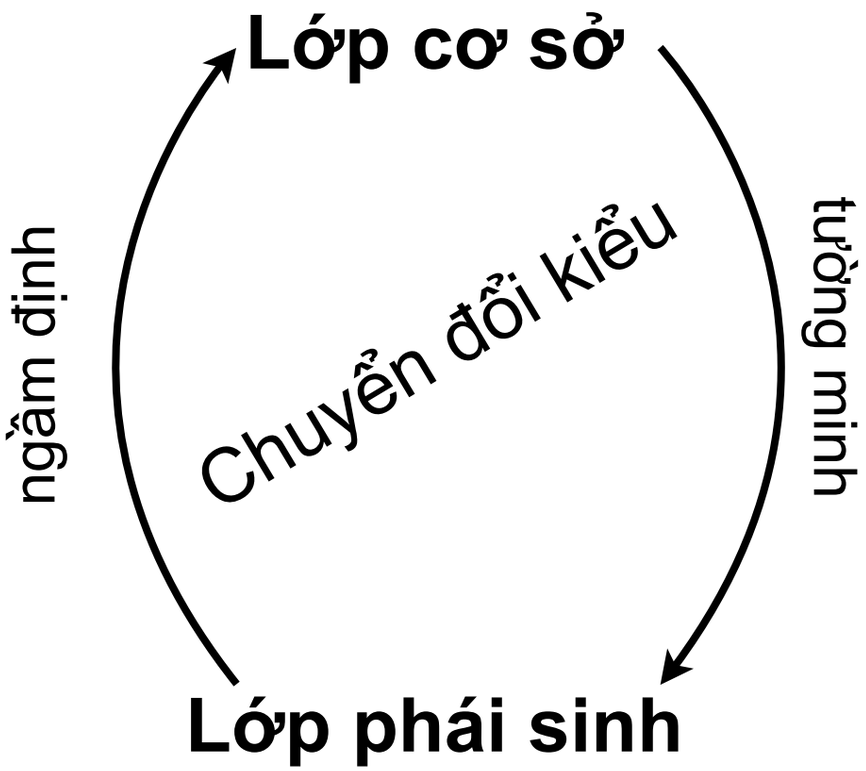
```
HCEPerson an =
   new Student(971232, "Nguyen Van An", "100 Phung Hung", 43, 2);

an.displayProfile();
```

```
[Name : Nguyen Van An; ID : 971232; Address : 100 Phung Hung]
```

- Vì sao khóa học và lớp học tham dự không được in ra ?

Chuyển đổi kiểu






```
Student s;
HCEPerson p = s;
Student s1 = (Student)p;
```

Từ khoá `sealed`

```
sealed class SelectionStudent : Student
{
    ...
}
```

- Ngăn chặn việc thừa kế
- Phù hợp cho các lớp tiện ích (`utility class`)
  - `System.String`
- C# `struct` ngầm định là `sealed` nên không có khả năng thừa kế

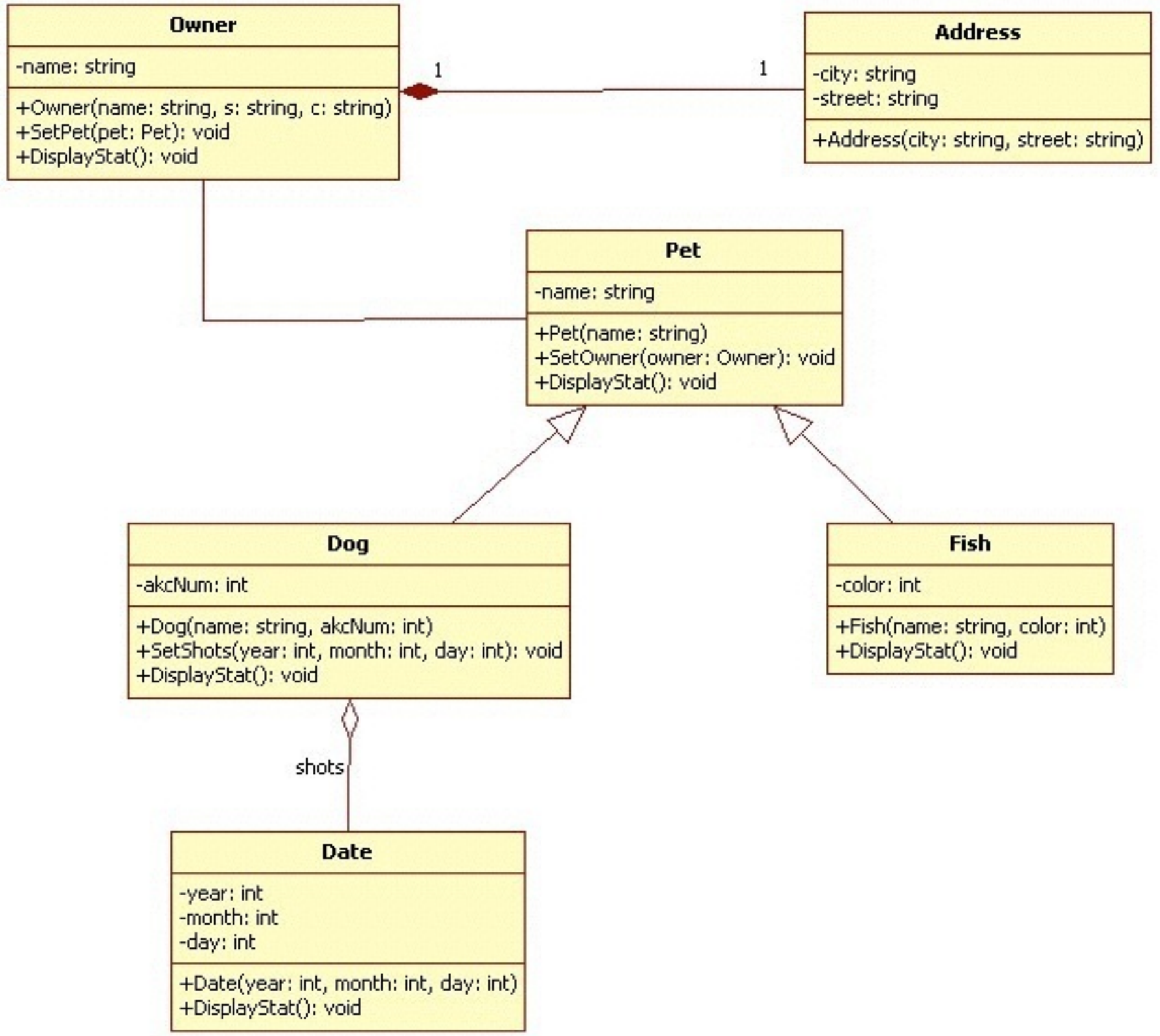
Kiểu lồng nhau

- C# cho phép định nghĩa một kiểu (`enum`, `class`, `interface`, `struct`,...) ngay bên trong phạm vi của một lớp hoặc `struct`

```
class Employee
{
   public class BenefitPackage
   {
      public double ComputePayDeduction() { return 125.0; }
      public enum BenefitPackageLevel { Standard, Gold, Platinum }
   }
   ...
}
```

- Lý do sử dụng :
  - Cho phép điều khiển hoàn toàn trên tất cả các cấp độ truy xuất của lớp nội tại
  - Bởi vì lớp nội tại là thành viên của lớp chứa nên nó có thể truy xuất các thành viên `private` của lớp chứa
  - Lớp nội tại chỉ hữu dụng như một lớp trợ giúp (`helper class`) và không được dự định cho bên ngoài sử dụng

Ví dụ - Pet


Owner
-name: string
+Owner(name: string, s: string, c: string)
+SetPet(pet: Pet): void
+DisplayStat(): void
1
1
Address
-city: string
-street: string
+Address(city: string, street: string)
Pet
-name: string
+Pet(name: string)
+SetOwner(owner: Owner): void
+DisplayStat(): void
Dog
-akcNum: int
+Dog(name: string, akcNum: int)
+SetShots(year: int, month: int, day: int): void
+DisplayStat(): void
Fish
-color: int
+Fish(name: string, color: int)
+DisplayStat(): void
shots
Date
-year: int
-month: int
-day: int
+Date(year: int, month: int, day: int)
+DisplayStat(): void

Cảm ơn sự chú ý

Câu hỏi ?