



CHỦ ĐỀ 5

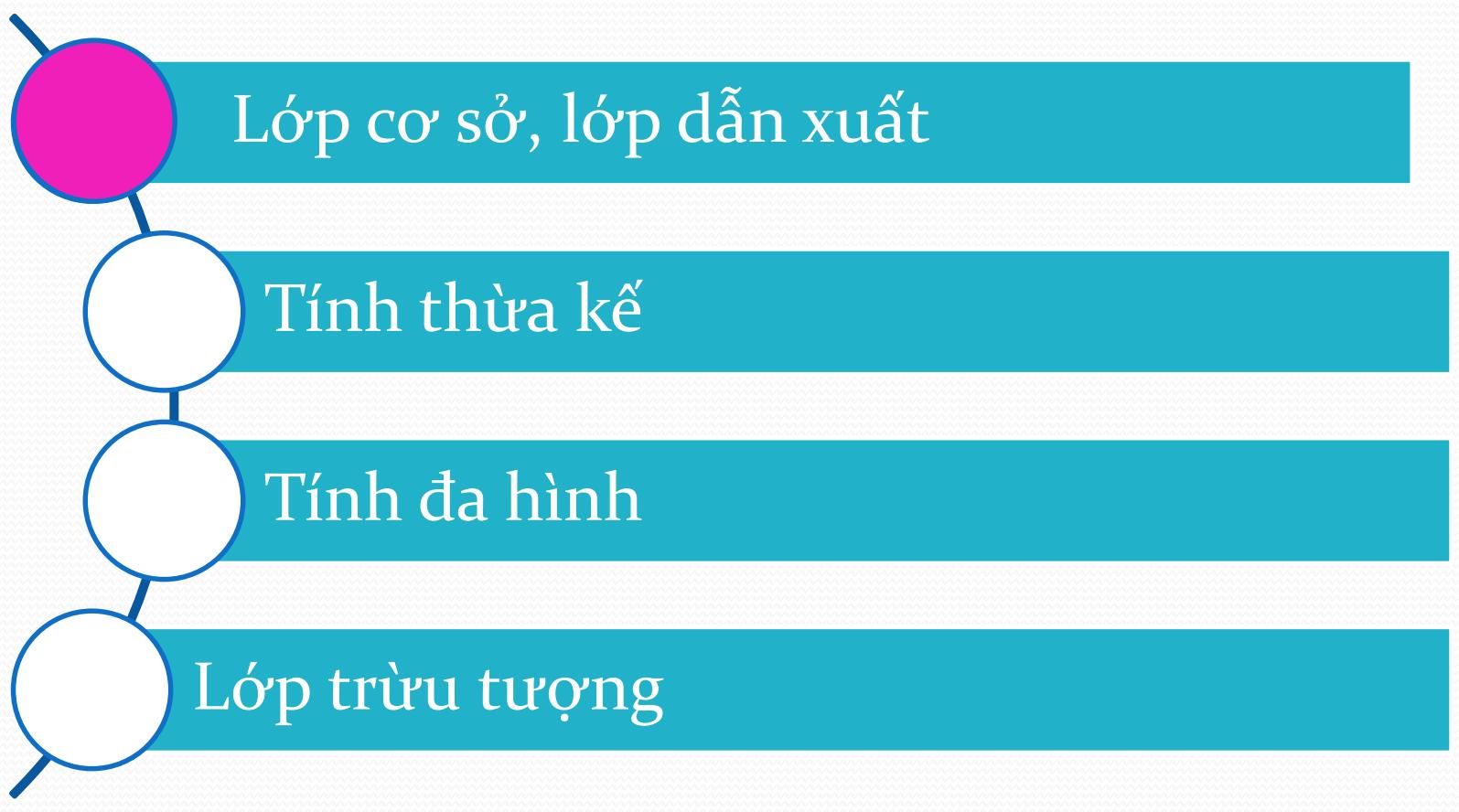


THÙA KẾ VÀ ĐA HÌNH

GVGD: PHẠM THỊ KIM NGOAN

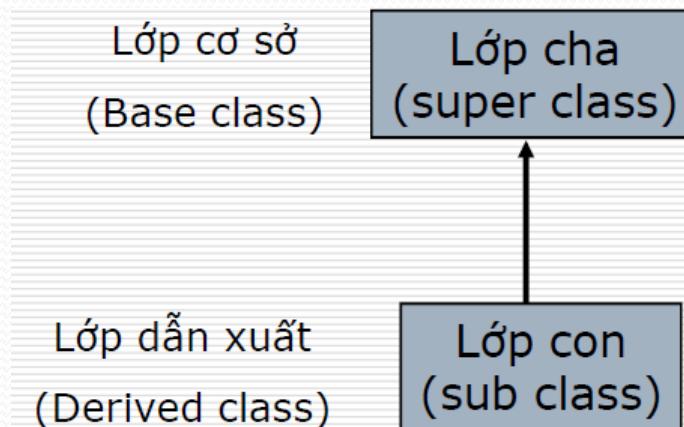
Email: ptkngoan@gmail.com

Nội dung



Lớp cơ sở, lớp dẫn xuất

- Tính kế thừa là một khái niệm nền tảng cho phép tái sử dụng mã đang tồn tại.
- Các lớp (class) có thể kế thừa từ lớp khác (định nghĩa class mới dựa trên class đã có). Lớp mới được gọi là lớp dẫn xuất (lớp con), lớp đã có là lớp cơ sở (lớp cha)



Lớp cơ sở, lớp dẫn xuất

- Lớp dẫn xuất thừa kế các thành phần của lớp cơ sở + các thành phần mới.
- Ngầm định một lớp được tạo ra sẽ kế thừa từ lớp `System.Object`.
- Một lớp chỉ được phép kế thừa tối đa 1 lớp, trừ khi được khai báo `sealed`.
- C# cho phép kế thừa phân cấp.

Bài tập 1

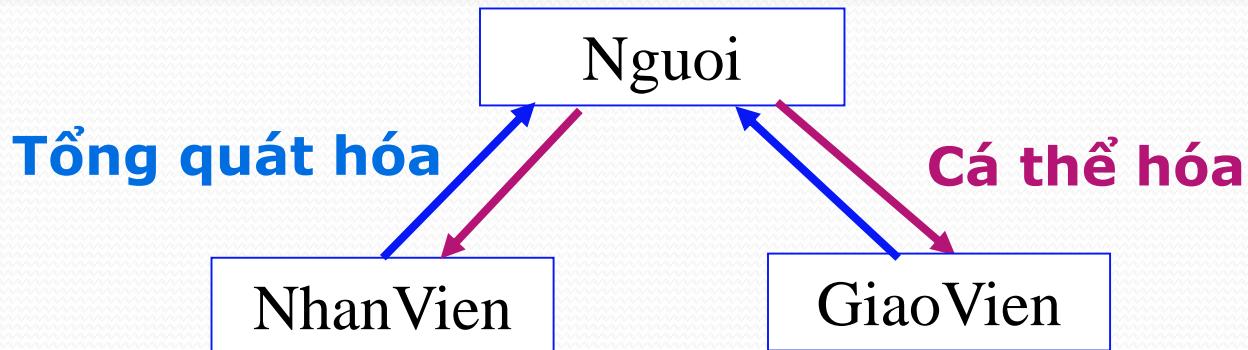
Xây dựng chương trình quản lý cán bộ nhân viên đơn giản cho một trung tâm ngoại ngữ. Trung tâm có 2 loại cán bộ:

- Nhân viên văn phòng có các thông tin: mã số, họ tên, giới tính, số ngày công, lương tháng; tiền lương được tính: số ngày công * lương tháng/26
- Giáo viên có các thông tin: mã số, họ tên, giới tính, số tiết giảng, thù lao 1 tiết giảng, tiền lương được tính: số tiết * tiền thù lao 1 tiết.

➤ Chương trình thực hiện các chức năng sau:

- Nhập vào thông tin 01 cán bộ nhân viên (cho phép chọn loại cán bộ nhân viên khi nhập)
- In thông tin và tiền lương cán bộ nhân viên (mã số, họ tên, tiền lương)

Lớp cơ sở, lớp dẫn xuất



- Lớp cơ sở (lớp cha): Nguoi
- Lớp dẫn xuất (lớp con): NhanVien, GiaoVien

Chú ý: Lớp dẫn xuất không thể bỏ đi các thành phần đã được khai báo trong lớp cơ sở.

Tính thừa kế

- Hai điểm mạnh nhất trong lập trình hướng đối tượng là **tính kế thừa** và **đa hình**
- Cú pháp

```
<access> class <Derived class>:[access]<base class>
{
    - Attributes
    + Methods
    ...
}
```

Tính thừa kế

(1) Quyền truy xuất tp đó ở lớp cha (lớp cơ sở):

(2) Kiểu dẫn xuất

		Thừa kế private	Thừa kế protected	Thừa kế public
(1) private	-	-	-	
protected	private	protected	protected	
public	private	protected	public	

Quyền truy xuất ở lớp con

Tính thừa kế

➤ Ví dụ:

```
class Person
{   string id, name;
    bool male;
    ...
}
class Employee: Person
{   byte days;
    float m_salary;
    ...
}
class Teacher: Person
{   byte hours;
    float h_salary;
    ...
}
```



Cài đặt lớp cơ sở

```
class Person
{ string id, name;
  bool male;

  public void Input()
  {
    Console.Write("nhap ma so:");
    id = Console.ReadLine();
    Console.Write("nhap ho ten:");
    name = Console.ReadLine();
    Console.Write("nhap gioi tinh:");
    male = Convert.ToBoolean(Console.ReadLine());
  }
  ...
}
```

Định nghĩa lại phương thức ở lớp con

```
class Employee: Person
{ byte days;
  float m_salary;

  public new void Input()
  {
    base.Input();
    Console.Write("nhap so ngay cong:");
    days =Convert.ToByte(Console.ReadLine());
    Console.Write("nhap luong thang:");
    m_salary = float.Parse(Console.ReadLine());
  }
}
```

Phương thức khởi tạo không tham số

```
class Person
{ string id, name;
  bool male;

  public Person()
  { id = “”; name =“”; male = true; }
}

class Employee: Person
{ byte days;
  float m_salary;

  public Employee():base()
  { days = 26; m_salary = 4.5; }
}
```

Phương thức khởi tạo có tham số

```
class Person
{ string id, name;
  bool male;

  public Person(string i, string n, bool m)
  { id = i ; name = n ; male = m;}
}

class Employee: Person
{ byte days;
  float m_salary;

  public Employee(string i, string n, bool m, byte d, float s):base(i, n, m)
  { days = d;  m_salary = s;}
}
```

Phương thức khởi tạo có tham số

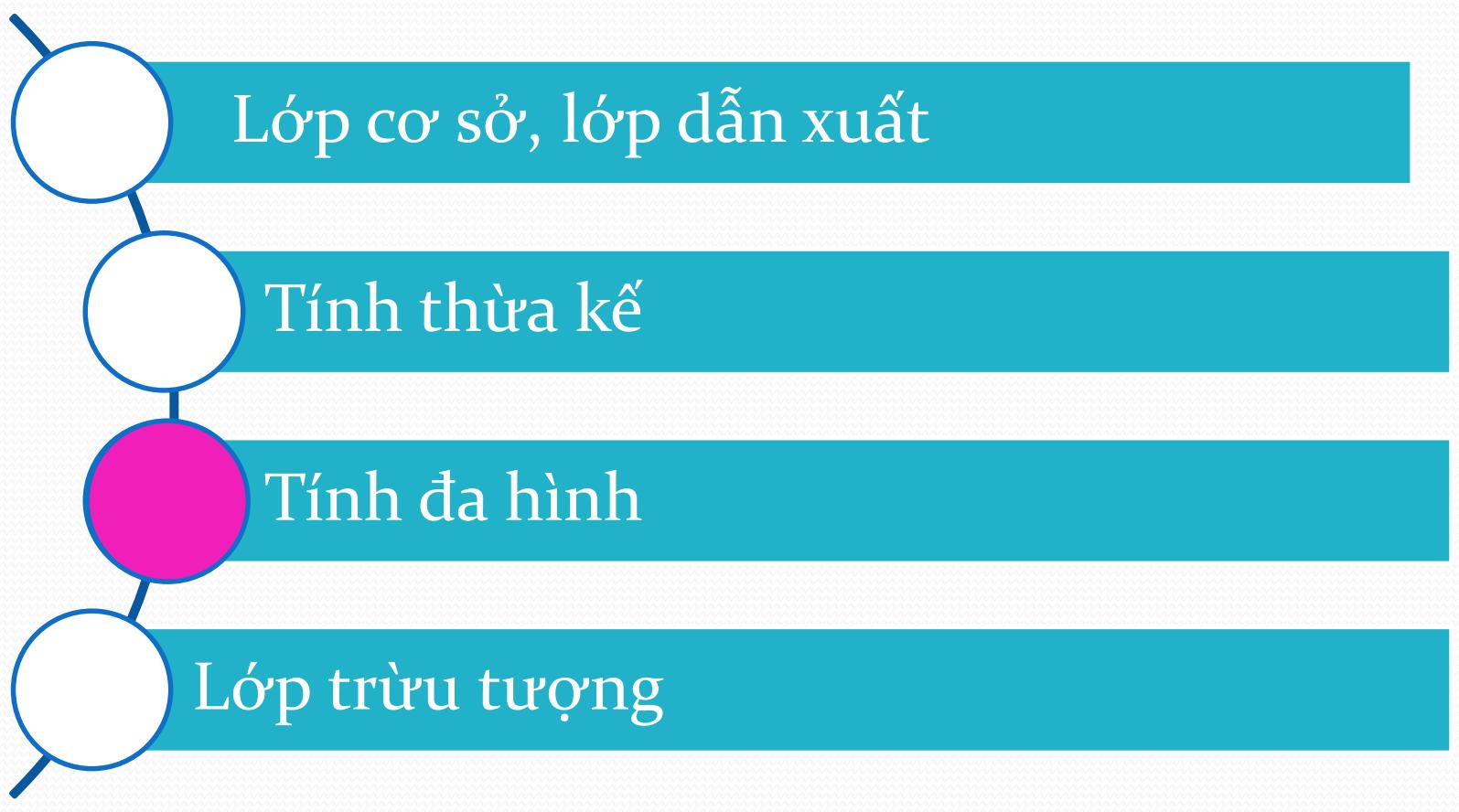
```
class Person
{ string id, name;
  bool male;

  public Person(Person p)
  { id = p.id ; name = p.name ; male = p.male; }
}

class Employee: Person
{ byte days;
  float m_salary;

  public Employee(Employee e):base((Person) e)
  { days = e.days; m_salary = e.m_salary; }
}
```

Nội dung



Tính đa hình

- **Đa hình** là ý tưởng “sử dụng một giao diện chung cho nhiều phương thức khác nhau”, dựa trên phương thức ảo (virtual method) và cơ chế liên kết muộn (late binding).
- **Tính đa hình** là cơ chế các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.
- Phân loại:
 - **Đa hình tĩnh**: Hàm cụ thể được gọi là phần tử nhận thông điệp lúc **khai báo**.
 - **Đa hình động**: Hàm cụ thể được gọi là phần tử nhận thông điệp lúc **chạy chương trình**.

Tính đa hình tĩnh

➤ Để thể hiện được tính đa hình tĩnh:

- Cho phép khai báo các phương thức trùng tên nhau nhưng có tham số khác nhau trong class.
- Dùng nạp chồng phương thức (overloading).

➤ Các điểm cần lưu ý khi thực hiện nạp chồng phương thức:

- Tên của các phương thức phải trùng nhau.
- Số lượng tham số phải khác nhau.
- Kiểu dữ liệu của các tham số và thứ tự các tham số phải khác nhau.

Tính đa hình tĩnh

```
class Person
{ string id, name;
  bool male;

  public Person()
  { id = “” ; name =“” ; male = true; }

  public Person(string i, string n, bool m)
  { id = i ; name = n ; male = m; }
}
```

Tính đa hình tĩnh

```
class Person
{ string id, name;
  bool male;

  public void Output()
  {     ...   }

}

class Employee: Person
{ byte days;
  float m_salary;

  public void Output(float s) //xuat luong > s
  {     ...   }

}
```

Tính đa hình động

➤ Để thể hiện được tính đa hình động:

- Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó.
- Phương thức đa hình phải được ghi đè (override) ở các lớp con.

Tính đa hình động

- **Ghi đè (overriding):** được dùng để định nghĩa lại phương thức của lớp cơ sở (lớp cha) trong lớp dẫn xuất (lớp con kế thừa)
- Các điểm cần lưu ý khi thực hiện ghi đè:
 - Phương thức ở lớp cơ sở và lớp dẫn xuất phải có cùng dạng hàm và kiểu dữ liệu trả về
 - Phương thức lớp cơ sở phải được khai báo với từ khóa **virtual**
 - Phương thức lớp dẫn xuất phải được khai báo với từ khóa **override**

Tính đa hình động

```
class Person
{ string id, name;
  bool male;

  public virtual float Salary()
  { return 0; }

}

class Employee: Person
{ byte days;
  float m_salary;

  public override float Salary() //tính lương
  { return days*m_salary/26; }

}
```

Tính đa hình động

```
class Person
{ string id, name;
  bool male;

  public virtual float Salary()
  { return 0; }

}

class Teacher: Person
{ byte hours;
  float h_salary;

  public override float Salary() //tính lương
  { return hours*h_salary; }

}
```

Tính đa hình động

```
class Person
{ string id, name;
  bool male;

  public virtual float Salary()
  { return 0; }

  public void Output()
  {
    Console.WriteLine("{0} {1} {2}", id, name, Salary());
  }
}
```

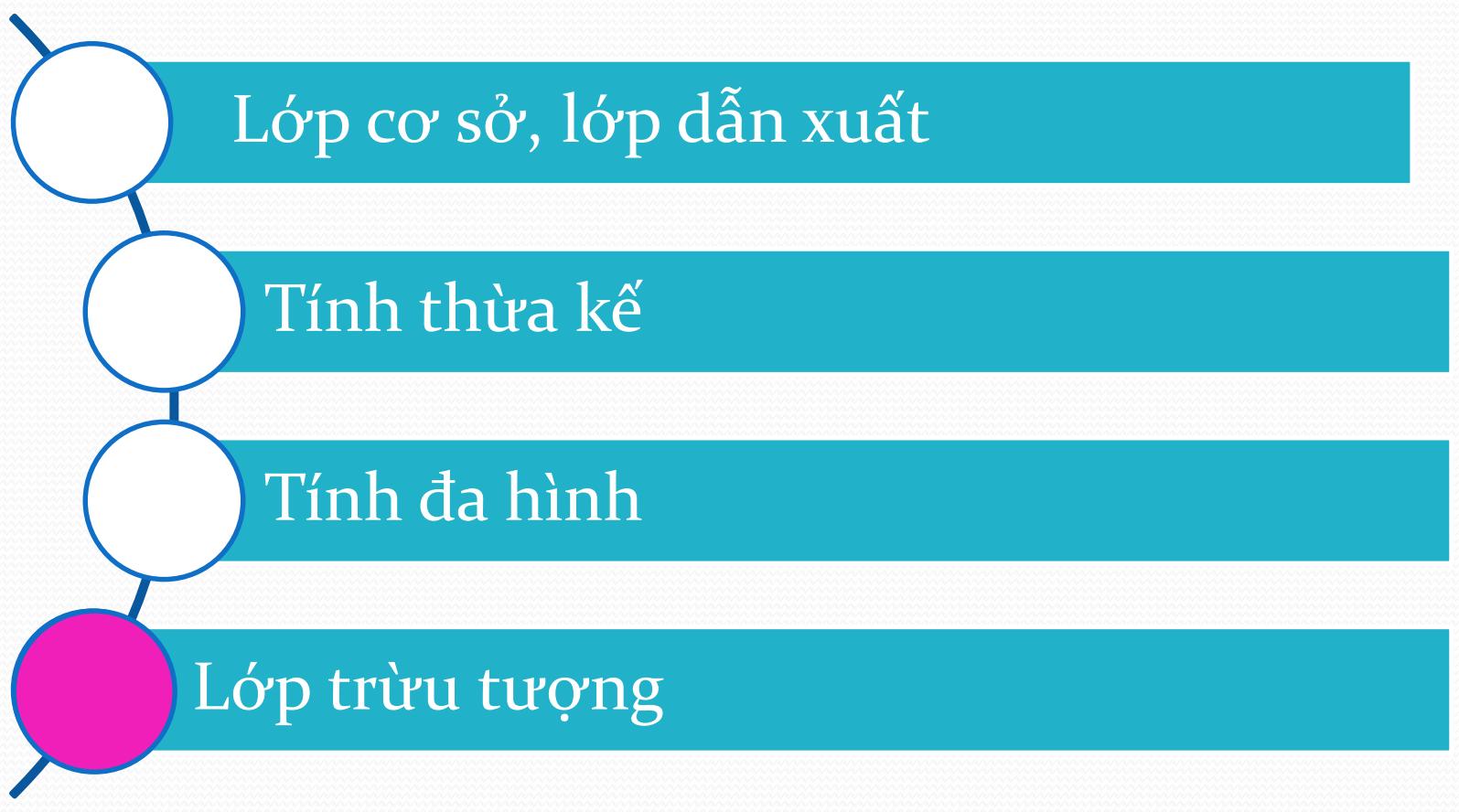
Tính đa hình động

```
class program
{
    static void Main()
    {
        Person p1 = new Person("T01","N V A" true);
        p1.Output();

        Person p2 = new Teacher("T01","N V C" true, h, s);
        p2.Output();

        Person p3 = new Employee("NV1", "T T B", false, d,s);
        p3.Output();
    }
}
```

Nội dung



Lớp trừu tượng (abstract class)

- **class trừu tượng** là class cơ sở (base class) mà các class khác có thể được dẫn xuất từ nó
- class trừu tượng có thể có cả hai loại phương thức: phương thức trừu tượng và phương thức cụ thể
- Khi nào thì sử dụng class trừu tượng?
 - Nếu muốn tạo các class mà các class này sẽ chỉ là các class cơ sở, và **không** muốn bất cứ ai **tạo các đối tượng** của các kiểu class này.
 - class trừu tượng thường được dùng để biểu thị rằng nó là class không đầy đủ và rằng nó được dự định sẽ chỉ được dùng như là một class cơ sở.

Lớp trừu tượng

- class trừu tượng có chứa phương thức trừu tượng
- Phương thức trừu tượng: phương thức **không có phần thân**.

Lớp trừu tượng

➤ Cú pháp

```
<access> abstract class class_name  
{  
    attributes, methods  
    public abstract <Data type> <Method_name>([Parameters]);  
}
```

Lớp trừu tượng

➤ Ví dụ

```
abstract class Person
{
    string id, name;
    bool male;

    public abstract float Salary();

    public void Output()
    {
        Console.WriteLine("{0} {1} {2}", id, name, Salary());
    }
}
```

Lớp trừu tượng

- Trong lớp dẫn xuất sẽ phải **override** các phương thức trừu tượng ở lớp cơ sở.

```
class Employee: Person
{ byte days;
  float m_salary;

  public override float Salary() //tính lương
  { return days*m_salary/26; }
}

class Teacher: Person
{ byte hours;
  float h_salary;

  public override float Salary() //tính lương
  { return hours*h_salary; }
}
```

Lớp trừu tượng

- Không được tạo ra đối tượng từ lớp trừu tượng

```
abstract class Person
{
    string id, name;
    bool male;

    public abstract float Salary();
    ...
}

class program
{
    static void Main()
    {
        Person p1 = new Person("T01", "N V A" true);
    }
}
```

Lớp niêm phong (sealed class)

- Từ khóa **sealed** được sử dụng để biểu thị khi khai báo một class nhằm ngăn ngừa sự dẫn xuất từ một class, điều này cũng giống như việc ngăn cấm một class nào đó có class con.
- Một **class sealed** cũng không thể là một class trừu tượng.
- Các structs trong C# được ngầm định sealed. Do vậy, chúng không thể được thừa kế

Lớp niêm phong

➤ Cú pháp

```
<access> sealed class class_name
{
    attributes, methods
}
```

➤ Ví dụ

```
sealed class Person
{ string id, name;
  bool male;
  ...
  public void Output()
  {
      Console.WriteLine("{0} {1} {2}", id, name, Salary());
  }
}
class Employee: Person
{ ... }
```

Bài tập 2

Xây dựng chương trình quản lý cán bộ nhân viên đơn giản cho một trung tâm ngoại ngữ. Trung tâm có 2 loại cán bộ:

- Nhân viên văn phòng có các thông tin: mã số, họ tên, giới tính, số ngày công, lương tháng; tiền lương được tính: số ngày công * lương tháng/26
- Giáo viên có các thông tin: mã số, họ tên, giới tính, số tiết giảng, thù lao 1 tiết giảng, tiền lương được tính: số tiết * tiền thù lao 1 tiết.

➤ Chương trình thực hiện các chức năng sau:

- Nhập vào thông tin n cán bộ nhân viên cho một trường học ($5 \leq n \leq 30$)
- In bảng lương cán bộ nhân viên (mã số, họ tên, tiền lương)

Kết thúc chủ đề 5

