*Chương 3*

# HƯỚNG ĐỐI TƯỢNG VỚI NGÔN NGỮ C#

Lập trình hướng đối tượng là học phần bắt buộc trong hầu hết các ngành đào tạo về công nghệ thông tin. Các kiến thức về lập trình hướng đối tượng được trình bày khá chi tiết trong học phần đó. Các kiến thức trọng tâm của học phần này tập trung vào xây dựng ứng dụng dạng Windows Forms có truy vấn, thao tác với cơ sở dữ liệu (sẽ được trình bày kỹ lưỡng trong chương 4 và chương 5). Do đó, chương này chủ yếu sẽ nhắc lại những kiến thức cơ bản cần thiết về lập trình hướng đối tượng để làm cơ sở cho người học có thể dễ dàng tiếp cận các kiến thức mới trình bày chương 4 và chương 5. Các kiến thức về lập trình hướng đối tượng được nhắc lại gồm các khái niệm cơ bản, các đặc điểm của phương pháp lập trình hướng đối tượng. Cách tạo và sử dụng lớp, các tính chất thừa kế, nạp chồng phương thức, ghi đè phương thức, … được minh họa bằng ngôn ngữ C#.

## 3.1 CÁC KHÁI NIỆM CƠ BẢN TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

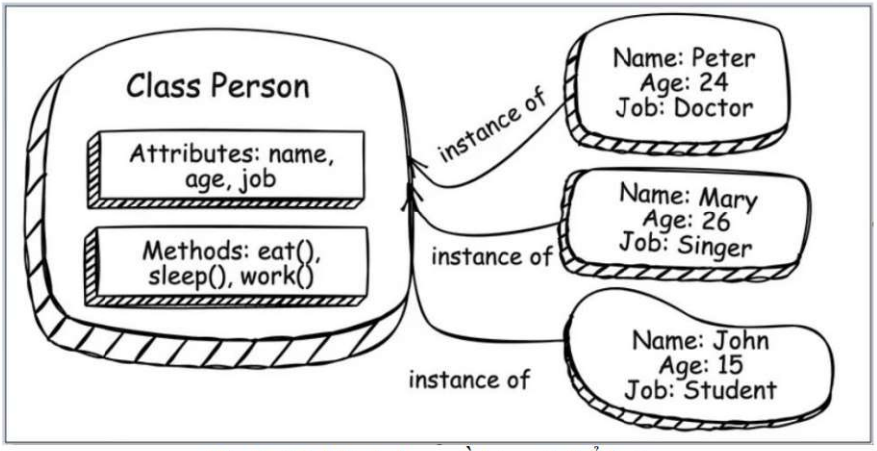
Lập trình hướng đối tượng là một phương pháp lập trình dựa trên kiến trúc lớp và đối tượng.

Đối tượng (object) là một thực thể ở ngoài thế giới thực. Mỗi đối tượng có các đặc tính và hành động của nó. Ví dụ như con người, tài khoản ngân hàng, …

Lớp (class) là phần mã lệnh được viết bằng một ngôn ngữ lập trình nào đó nhằm mô tả một tập các đối tượng có chung các đặc tính và hành động ở ngoài thế giới thực. Đặc tính được gọi là các thuộc tính (Attributes) và hành động được gọi là các phương thức (Methods) của lớp.

Lớp được xem như khuôn mẫu để tạo ra các đối tượng ngoài thế giới thực, mỗi đối tượng được tạo ra từ một lớp được gọi là một thể hiện (instance) của lớp đó.

Trong hình 3.1 sau, lớp con người (Class Person) có ba thuộc tính: name, age, job và các phương thức eat(), sleep(), work(). Các thể hiện của lớp là (Name: Peter, Age: 24, Job: Doctor), (Name: Mary, Age: 26, Job: Singer) và (Name: John, Age: 15, Job: Student). Có thể thấy khi các thuộc tính của lớp nhận giá trị thì ta có một thể hiện của lớp đó.




**Hình 3.1** Minh họa về lớp và thể hiện.

## 3.2 CÁC ĐẶC ĐIỂM CỦA LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lập trình hướng đối tượng có bốn đặc điểm chính:

- **Tính trừu tượng (Abstraction)** là kỹ thuật đơn giản hoá việc lập trình cho những thành phần phức tạp. Xét một ví dụ về tính trừu tượng như sau.

+ Lớp UIElement đại diện cho tất cả các thành phần là giao diện người dùng (UI - User Interface) và mỗi thành phần giao diện có phương thức Render() để hiển thị.
+ Vấn đề là phương thức Render() được viết như thế nào trong lớp UIElement thì lại không viết được vì ta không biết đó là thành phần giao diện nào (TextBox hay Button hay PictureBox, …)? Vì thế Render() trong lớp UIElement là phương thức trừu tượng (abstract method), các thành phần giao diện sẽ kế thừa lớp UIElement và phương thức Render() được viết chi tiết để hiển thị chúng. Lúc này, lớp UIElement là trở thành một lớp trừu tượng (abstract class).

– **Tính đóng gói (Encapsulation)** cho phép che giấu thông tin của một đối tượng. Một đối tượng không được phép truy cập trực tiếp dữ liệu của các đối tượng khác và cũng không cho các đối tượng khác truy cập trực tiếp dữ liệu của mình. Tất cả mọi thao tác truy xuất dữ liệu từ đối tượng này qua đối tượng khác phải được thực hiện bởi các phương thức của chính đối tượng chứa dữ liệu. Tính đóng gói thể hiện rất rõ ở các thuộc tính, phương thức, … có phạm vi truy cập là privte trong một lớp.
– **Tính thừa kế (Inheritance)** là sự cho phép tạo một lớp mới dựa trên lớp đã có trước đó. Lớp đã có trước đó gọi là lớp cha (hay lớp cơ sở) và lớp mới tạo gọi là lớp con (hay lớp dẫn xuất). Tất cả các thành phần của lớp cha sẽ được lớp con kế thừa và lớp con có thể mở rộng các thành phần này hoặc bổ sung thêm các thành phần mới. Ngôn ngữ C# chỉ hỗ trợ thừa kế đơn, nghĩa là một lớp chỉ có thể có duy nhất một lớp cha, còn lớp cha thì có thể có nhiều lớp con.
– **Tính đa hình (Polymorphism)** là kỹ thuật xử lý các đối tượng theo cùng một cách nhưng kết quả khác nhau. Tính đa hình thể hiện ở các kỹ thuật.
  + Nạp chồng phương thức (Overloading): Kỹ thuật này cho phép một lớp có nhiều thuộc tính, phương thức cùng tên nhưng các tham số sẽ khác nhau về kiểu và số lượng. Khi được sử dụng, dựa vào tham số truyền vào, thuộc tính hay phương thức tương ứng sẽ được thực hiện.
  + Ghi đè phương thức (Overriding) áp dụng trong kế thừa, lớp con được phép có thuộc tính, phương thức cùng tên, cùng số lượng tham số có kiểu dữ liệu như thuộc tính, phương thức của lớp cha nhưng cài đặt khác đi. Lúc chạy chương trình thì thuộc tính, phương thức cùng tên của lớp con được gọi.

## 3.3 TẠO LỚP VÀ SỬ DỤNG LỚP

### 3.3.1 Tạo lớp

Cú pháp tạo lớp:

```
<từ khóa truy cập> class <tên lớp>
{
    <các thành phần của lớp>
}
```

Giải thích cú pháp trên:

– <từ khóa truy cập> quyết định phạm vi sử dụng của lớp được tạo. Gồm các từ khóa:

+ public: cho phép sử dụng lớp ở mọi nơi trong chương trình.
+ private: chỉ được phép sử lớp ở nơi nó được tạo.
+ protected: tương tự private ngoài ra còn cho phép truy cập từ lớp con.
+ internal: chỉ được truy cập trong cùng một project (mặc định).

– Ngoài ra còn một vài từ khóa trong tạo lớp nhằm ngăn chặn việc sử dụng và tạo thể hiện của lớp như:
  + abstract: lớp không cho phép khai báo thể hiện của lớp, lớp này cho làm cơ sở cho các lớp con.
  + sealed: lớp không cho phép thừa kế.
– class: từ khóa dùng để tạo lớp.
– <tên lớp>: tên của lớp được tạo, đặt theo quy tắc đặt tên biến.
– <các thành phần của lớp>: là các thành phần của lớp gồm các biến thành viên, thuộc tính, phương thức được đặt bên trong cặp dấu ngoặc nhọn ({}) của lớp.

Trong ví dụ 3.1 sau, lớp nhân viên (NhanVien) được tạo gồm các biến thành viên mã nhân viên (maNhanVien), họ tên nhân viên (hoTenNhanVien), hệ số lương (heSoLuong), mức lương cơ bản (mucLuongCoBan) và một phương thức hiển thị thông tin nhân viên (hienThongTinNhanVien()).

Ví dụ 3.1: Tạo lớp nhân viên (NhanVien)

```
//Tạo lớp nhân viên
internal class NhanVien
{
    //Khai báo các biến thành viên của lớp NhanVien
    public string maNhanVien;
    public string hoTenNhanVien;
    double heSoLuong;
    public static double mucLuongCoBan;

    //Phương thức hiển thị thông tin nhân viên
    public void hienThongTinNhanVien()
    {
        Console.WriteLine(string.Format("{0} - {1} - {2} -
                          {3}", maNhanVien, hoTenNhanVien,
                          heSoLuong, mucLuongCoBan));
    }
}
```

### 3.3.2 Cách sử dụng lớp

Lớp được dùng để tạo ra các đối tượng là chính, mỗi một đối tượng còn gọi là một thể hiện của lớp. Cú pháp tạo đối tượng như sau.

```
<tên lớp> <tên đối tượng>;
```

Hoặc vừa tạo đối tượng vừa khởi tạo giá trị cho các biến thành viên của đối tượng đó theo cú pháp sau.

```
<tên lớp> <tên đối tượng> = new <tên lớp>;
```

Ví dụ 3.2: Tạo đối tượng nv của lớp nhân viên (NhanVien) trong ví dụ trên.

```
NhanVien nv = new NhanVien();
```

## 3.4 CÁC THÀNH PHẦN CƠ BẢN CỦA LỚP

Như đề cập bên trên, các thành phần cơ bản của một lớp gồm các biến thành viên, các thuộc tính và các phương thức.

### 3.4.1 Biến thành viên của lớp

Biến thành viên của lớp là thành phần dữ liệu của lớp. Khi tạo đối tượng từ lớp, các biến thành viên này sẽ nhận các giá trị cụ thể, các giá trị này có thể là mặc định tùy theo kiểu của biến hoặc các giá trị do người dùng gán thông qua phương thức khởi tạo của lớp.

Các biến thành viên được khai báo trong lớp theo cú pháp:

```
<từ khóa truy cập> <tên biến thành viên> [= <giá trị>];
```

Các từ khóa truy cập quyết định phạm vi sử dụng biến thành viên gồm private, protected, public.

- private (mặc định): biến thành viên chỉ được phép truy cập trong lớp mà nó được khai báo.
- protected: tương tự như private ngoài ra nó còn được truy cập ở lớp con.
- public: được phép truy cập từ bên ngoài lớp.

Nếu trong khai báo biến thành viên có thêm từ khóa static thì biến thành viên này là chung cho các đối tượng và được truy cập thông qua tên lớp thay vì thông qua tên đối tượng.

Cú pháp truy cập biến thành viên chung:

```
<tên lớp>.<tên biến thành viên chung>;
```

Cú pháp truy cập biến thành viên thông thường:

```
<tên đối tượng>.<tên biến thành viên thường>;
```

Ví dụ 3.3: Truy cập biến thành viên maNhanVien, hoTenNhanVien thông qua đối tượng nv của lớp NhanVien trên.

```
Console.WriteLine(nv.maNhanVien);
Console.WriteLine(nv.hoTenNhanVien);
```

Truy cập biến thành viên chung mucLuongCoBan.

```
NhanVien.mucLuongCoBan = 4000000;
```

### 3.4.2 Thuộc tính của lớp

Ta không thể truy cập trực tiếp các biến thành viên là private từ bên ngoài lớp. Tuy nhiên, các biến thành viên này có thể được truy cập gián tiếp thông qua thuộc tính của nó. Để tạo thuộc tính cho biến thành viên, ta sử dụng câu lệnh get và set theo cú pháp sau.

```
public <kiểu dữ liệu biến thành viên> <tên thuộc tính>
{
    //Lấy giá trị biến thành viên
    get{ return <tên biến thành viên>; }

    //Gán giá trị <value> cho biến thành viên
    set{ <tên biến thành viên> = <value>; }
}
```

Khi tạo thuộc tính mà chỉ có câu lệnh get thì đó là thuộc tính chỉ đọc (read only), còn chỉ có câu lệnh set thì đó là thuộc tính chỉ ghi.

Ví dụ 3.4: Tạo thuộc tính hsl cho biến thành viên private heSoLuong của lớp nhân viên.

```
//Tạo lớp nhân viên
internal class NhanVien
{
    //Khai báo các biến thành viên của lớp NhanVien
    public string maNhanVien;
    public string hoTenNhanVien;
    double heSoLuong;
    public static double mucLuongCoBan = 4000000;
    /*Tạo thuộc tính hsl cho biến thành viên heSoLuong*/
    public double hsl
    {
        get { return heSoLuong; }
        set { heSoLuong = value; }
    }
    //Phương thức hiển thị thông tin nhân viên
    public void hienThongTinNhanVien()
    {
        Console.WriteLine(string.Format("{0} - {1} - {2} -
                          {3}", maNhanVien, hoTenNhanVien,
                          heSoLuong, mucLuongCoBan));
    }
}

/*Lớp Progam này được tạo sẵn khi tạo dự án (project) dạng
Console. Trong project chỉ duy nhất lớp này chứa phương thức
Main(), nơi mà chương sẽ thực thi đầu tiên.*/
internal class Program
{
    static void Main(string[] args)
    {
        NhanVien nv = new NhanVien();
        nv.hsl = 2.34;
        Console.WriteLine("He so luong: " + nv.hsl);
        Console.Read();
    }
}
```

Trong ví dụ 3.4 trên, khi viết nv.hsl = 2.34; thì câu lệnh set của thuộc tính hsl được gọi thực hiện, lúc này <value> là giá trị 2.34. Còn khi viết nv.hsl thì câu lệnh set của thuộc tính hsl sẽ được thực hiện. Kết quả hiển thị ra màn hình là 2.34.

### 3.4.3 Phương thức của lớp

Một phương thức của lớp là một cài đặt cho một hành động của đối tượng. Cú pháp tạo phương thức của lớp như sau.

```
<từ khóa truy cập> <kiểu kết quả trả về>
                          <tên phương thức>([<các tham số>])
{
    <các câu lệnh của phương thức>
}
```

Giải thích:

- <từ khóa truy cập>: quyết định phạm vi sử dụng của phương thức, các từ khóa truy cập và ý nghĩa của chúng có thể tham khảo ở phần trình bày biến thành viên của lớp.
- <kiểu kết quả trả về>: kiểu kết quả trả về của phương thức. Phương thức không trả về kết quả gì thì ghi void.
- <tên phương thức>: đặt theo quy tắc đặt tên biến.
- <các tham số>: có thể có hoặc không, các tham số cách nhau bởi dấu phẩy, được khai báo như khai báo biến.

#### *3.4.3.1 Phương thức khởi tạo*

Phương thức khởi tạo (Constructor) được dùng để gán giá trị ban đầu cho các biến thành viên của đối tượng khi nó được tạo. Một lớp có thể có nhiều phương thức khởi tạo.

Nếu chúng ta không định nghĩa phương thức khởi tạo khi tạo lớp thì C#.Net sẽ sử dụng phương thức khởi tạo mặc định để gán các giá trị cho các biến thành viên tương ứng với kiểu của chúng. Đối tượng nv được tạo trong ví dụ 3.4 trên sử dụng phương thức khởi tạo mặc định (câu lệnh NhanVien nv = new NhanVien();). Do đó, giá trị ban đầu của các biến thành viên của đối tượng nv là: maNhanVien = "", hoTenNhanVien = "", heSoLuong = 0, mucLuongCoBan = 0.

Phương thức khởi tạo có cùng tên với tên lớp, không có kiểu trả về, nếu cho phép tạo đối tượng ngoài lớp thì phải có phạm vi truy cập là public.

Ví dụ 3.5: tạo phương thức khởi tạo cho lớp nhân viên và sử dụng nó để tạo đối tượng của lớp này.

```
//Tạo lớp nhân viên
internal class NhanVien
{
    //Khai báo các biến thành viên của lớp NhanVien
    public string maNhanVien;
    public string hoTenNhanVien;
    double heSoLuong;
    public static double mucLuongCoBan = 4000000;

    /*Tạo thuộc tính hsl cho biến thành viên heSoLuong*/
    public double hsl
    {
        get { return heSoLuong; }
        set { heSoLuong = value; }
    }

    //Tạo phương thức khởi tạo cho lớp nhân viên
    public NhanVien(string ma, string hoten, double hs)
    {
        maNhanVien = ma;
        hoTenNhanVien = hoten;
        heSoLuong = hs;
    }
```

```
    //Phương thức hiển thị thông tin nhân viên
    public void hienThongTinNhanVien()
    {
        Console.WriteLine(string.Format("{0} - {1} - {2} -
                          {3}", maNhanVien, hoTenNhanVien,
                          heSoLuong, mucLuongCoBan));
        }
    }

/*Lớp Progam này được tạo sẵn khi tạo dự án (project) dạng
Console. Trong project chỉ duy nhất lớp này chứa phương thức
Main(), nơi mà chương sẽ thực thi đầu tiên.*/

internal class Program
{
    static void Main(string[] args)
    {
        NhanVien nv = new NhanVien("100", "Nguyen Van An",
                                                    2.34);
        Console.Read();
    }
}
```

Kết quả hiển thị trên màn hình là: 100 – Nguyen Van An – 2.34 – 4000000

#### *3.4.3.2 Phương thức hủy (Destructor)*

Thực hiện giải phóng đối tượng để thu hồi vùng nhớ đã cấp phát. C#.Net cung cấp sẵn bộ thu dọn (Garbage Collection) nên ta không cần khai báo tường minh các phương thức hủy.

Đối với các mã tự quản thì cần phải khai báo tường minh các phương thức hủy để giải phóng các tài nguyên. Phương thức Finalize() được C# cung cấp ngầm định để thực hiện công việc này. Ngoài ra, ta cũng có thể sử dụng phương thức Dispose() thuộc giao diện Idisposable.

### 3.5 NẠP CHỒNG PHƯƠNG THỨC

Nạp chồng phương thức (Method Overloading) là một kỹ thuật cho phép tạo nhiều phương thức cùng tên trong một lớp. Tuy nhiên, các phương thức cùng tên này phải khác nhau về số lượng và kiểu của tham số, không xét kiểu trả về của phương thức (nghĩa là các tham số giống nhau nhưng kiểu trả về khác nhau thì vẫn sẽ báo lỗi).

Ví dụ 3.6: nạp chồng phương thức hien_TT_NhanVien() của lớp nhân viên.

```
//Tạo lớp nhân viên
internal class NhanVien
{
    //Khai báo các biến thành viên của lớp NhanVien
    public string maNhanVien;
    public string hoTenNhanVien;
    double heSoLuong;
    public static double mucLuongCoBan = 4000000;

    /*Tạo thuộc tính hsl cho biến thành viên heSoLuong*/
```

```
    public double hsl
    {
        get { return heSoLuong; }
        set { heSoLuong = value; }
    }

    //Tạo phương thức khởi tạo cho lớp nhân viên
    public NhanVien(string ma, string hoten, double hs)
    {
        maNhanVien = ma;
        hoTenNhanVien = hoten;
        heSoLuong = hs;
    }

    //Phương thức hiển thị thông tin nhân viên
    public void hien_TT_NhanVien()
    {
        Console.WriteLine(string.Format("{0} - {1} - {2} -
                          {3}", maNhanVien, hoTenNhanVien,
                          heSoLuong, mucLuongCoBan));
    }

    public NhanVien hien_TT_NhanVien(NhanVien nv)
    {
        return nv;
    }
}
```

### 3.6 THỪA KẾ

Thừa kế (Inheritance) là sự cho phép tạo một lớp mới dựa trên lớp đã có trước đó. Lớp đã có trước đó gọi là lớp cha (hay lớp cơ sở) và lớp mới tạo gọi là lớp con (hay lớp dẫn xuất). Tất cả các thành phần của lớp cha sẽ được lớp con kế thừa và lớp con có thể mở rộng các thành phần này hoặc bổ sung thêm các thành phần mới. Ngôn ngữ C# chỉ hỗ trợ thừa kế đơn, nghĩa là một lớp chỉ có thể có duy nhất một lớp cha, còn lớp cha thì có thể có nhiều lớp con.

Ví dụ 3.7: Tạo lớp nhân viên văn phòng thừa kế lớp nhân viên ở ví dụ 3.6. Trong ví dụ, base(ma, hoten, hs) là gọi phương thức khởi tạo có ba tham số của lớp nhân viên (lớp cha).

```
internal class NhanVienVanPhong : NhanVien
{
    int soNgayVang;

    public NhanVienVanPhong(string ma, string hoten, double hs,
                            int ngayVang) : base(ma, hoten, hs)
    {
        soNgayVang = ngayVang;
    }
}
```

## 3.7 GHI ĐÈ PHƯƠNG THỨC

Ghi đè phương thức (Method Overriding) được áp dụng trong thừa kế. Phương thức của lớp cha được tạo lại trong lớp con nhưng cài đặt khác đi.

Một vài lưu ý khi thực hiện ghi đè phương thức:

- Phương thức cho phép ghi đè trong lớp cha phải là phương thức ảo, nghĩa là phải có khai báo từ khóa virtual trong cài đặt phương thức. Đồng thời, nó cũng phải có phạm vị truy cập là public hoặc protected.
- Phương thức ghi đè trong lớp con phải có khai báo từ khóa override khi cài đặt.
- Phương thức cho phép ghi đè trong lớp cha và phương thức ghi đè trong lớp con phải giống nhau về kiểu trả về, tên phương thức, các tham số và phạm vi truy cập.
- Không được ghi đè phương thức tĩnh và phương thức khởi tạo trong lớp cha. Phương thức tĩnh là phương thức có khai báo từ khóa static trong khi cài đặt.

Ví dụ 3.8: Ghi đè phương thức tính tiền lương trong lớp nhân viên.

```
internal class NhanVien
{
    public string maNhanVien;
    public string hoTenNhanVien;
    double heSoLuong;
    public static double mucLuongCoBan;

    //Tạo thuộc tính cho biến thành viên heSoLuong
    public double hsl
    {
        get { return heSoLuong; }
        set { heSoLuong = value; }
    }

    //Tạo phương thức khởi tạo cho lớp nhân viên
    public NhanVien(string ma, string hoten, double hs)
    {
        maNhanVien = ma;
        hoTenNhanVien = hoten;
        heSoLuong = hs;
    }

    public void hienThongTinNhanVien()
    {
        Console.WriteLine(string.Format("{0} - {1} - {2} -
                          {3}", maNhanVien, hoTenNhanVien,
                          heSoLuong, mucLuongCoBan));
    }
```

```
public virtual double tinhLuong()
    {
        return Convert.ToDouble(heSoLuong * mucLuongCoBan);
    }
}

internal class NhanVienVanPhong : NhanVien
{
    int soNgayVang;
    public NhanVienVanPhong(string ma, string hoten, double hs,
                            int ngayVang) : base(ma, hoten, hs)
    {
        soNgayVang = ngayVang;
    }

    public override double tinhLuong()
    {
        return (base.tinhLuong() - (soNgayVang * 500000));
    }
}
```

## BÀI TẬP CHƯƠNG 3

**Bài 3.1:** Hãy tạo lớp tài khoản ngân hàng (TaiKhoanNganHang) gồm các thuộc tính: số tài khoản (soTK, kiểu số nguyên), tên tài khoản (tenTK, kiểu chuỗi) và số dư tài khoản (soDuTK, kiểu số thực) và các phương thức cho phép gửi tiền, rút tiền và chuyển khoản. Các phương thức của lớp được mô tả như sau:

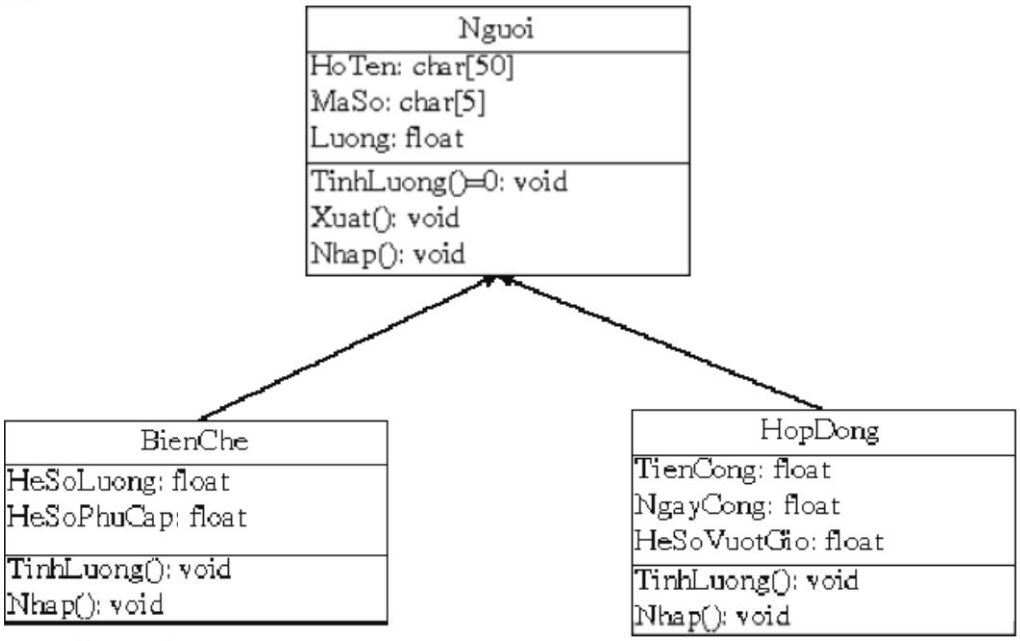
- Phương thức guiTien(): nhận tham số là số tiền cần gửi, khi phương thức được gọi thực hiện sẽ tăng số dư tài khoản thêm số tiền cần gửi.
- Phương thức rutTien(): nhận tham số là số tiền cần rút, khi phương thức được gọi thực hiện thì số dư tài khoản sẽ giảm một lượng tiền tương ứng với số tiền rút.
- Phương thức chuyenKhoan(): nhận tham số là tài khoản nhận tiền và số tiền chuyển, khi phương thức được gọi sẽ thực hiện tăng số dư của tài khoản nhận thêm lượng tiền chuyển đồng thời giảm số dư tài khoản chuyển một lượng tiền chuyển.

**Hãy thực hiện các yêu cầu sau:**

1. Hãy tạo project QuanLyTaiKhoanNganHang (quản lý tài khoản ngân hàng) dạng Console trong Solution BaiTapChuong3.
2. Xây dựng lớp TaiKhoanNganHang như mô tả bên trên.
3. Viết chương trình kiểm tra tính đúng đắn của lớp đã xây dựng.

**Bài 3.2:** Cơ quan X có những người lao động nhận lương từ ngân sách nhà nước (biên chế) và người lao động nhận lương từ ngân sách của cơ quan (hợp đồng). Giữa biên chế và hợp đồng có các thông tin chung cần quản lý gồm mã số, họ tên, lương.

Mặt khác, người lao động trong biên chế thì có hệ số lượng và phụ cấp chức vụ còn người lao động hợp đồng thì chỉ có tiền công lao động, số ngày làm việc trong tháng và hệ số vượt giờ. Ngoài các thông tin chung thì mỗi đối tượng lao động có các thông tin riêng cần quản lý, mà các thông tin riêng này có ảnh hưởng đến việc tính lương nên cách tính lương cũng khác nhau. Có thể sơ đồ hóa việc quản lý tính lương trên bằng sơ đồ sau.




Sơ đồ trên trình bày ba lớp (Nguoi, BienChe, HopDong), lớp BienChe và lớp HopDong là hai lớp con của lớp Nguoi. Chi tiết mỗi lớp được mô tả như sau.

**Lớp Nguoi:**

- Các thuộc tính mã số (MaSo), họ tên (HoTen), lương.
- Phương thức TinhLuong(): là một phương thức ảo (virtual), dùng tính lương cho người lao động, phương thức này sẽ được ghi đè ở các lớp con vì cách tính lương giữa chúng là khác nhau.
- Phương thức Xuat(): dùng để xuất các thông tin mã số, họ tên và lương của người lao động.
- Phương thức Nhap(): là một phương thức ảo dùng để nhập thông tin của người lao động và được ghi đè ở các lớp con vì giữa chúng có các thông tin riêng.

**Lớp BienChe:**

- Ngoài các thuộc tính kế thừa từ lớp cha (lớp Nguoi) thì còn các thuộc tính riêng là hệ số lượng (HeSoLuong) và hệ số phụ cấp (HeSoPhuCap).
- Phương thức TinhLuong() sẽ ghi đè phương thức TinhLuong() của lớp cha, với cách tính lương theo công thức "Lương = Mức lương cơ bản * (1.0 + Hệ số lương + Hệ số phụ cấp)".
- Phương thức Nhap() sẽ ghi đè phương thức Nhap() của lớp cha. Ngoài nhập các thông tin mã số, họ tên còn phải nhập thêm hệ số lương và hệ số phụ cấp.

**Lớp HopDong:**

- Tương tự lớp BienChe, lớp này cũng kế thừa lớp Nguoi. Ngoài ra, lớp này có các thuộc tính riêng là tiền công (TienCong), ngày công (NgayCong) và hệ số vượt giờ (HeSoVuotGio).
- Phương thức TinhLuong() cũng sẽ ghi đè phương thức TinhLuong() của lớp cha với cách tính lương theo công thức "Lương = Tiền công * Số ngày công * (1 + Hệ số vượt giờ)".
- Phương thức Nhap() sẽ ghi đè phương thức Nhap() của lớp cha. Ngoài nhập các thông tin mã số, họ tên còn phải nhập thêm tiền công, số ngày công và hệ số vượt giờ.

**Hãy thực hiện các yêu cầu sau:**

1. Trong Solution BaiTapChuong3 đã tạo ở bài 3.1, hãy tạo project QuanLyTinhLuong dạng Console.
2. Xây dựng các lớp như mô tả bên trên.
3. Viết chương trình kiểm tra tính đúng đắn của các lớp đã xây dựng.