CHƯƠNG 2

Đối Tượng Và Lớp

Nội Dung

- Đối Tượng Và Lớp
- Quan hệ giữa các lớp
- Cài đặt lớp
- Lớp
- Các thành phần của lớp
- Thuộc tính truy xuất
- Khởi động và dọn dẹp
- Phương thức thiết lập và huỷ bỏ

Nội Dung

- Một số đặc điểm khác của lớp
- Thiết lập và huỷ bỏ đối tượng
- Phương thức thiết lập bản sao
- Giao diện và chi tiết cài đặt
- Một số nguyên tắc xây dựng lớp

Đối Tượng và Lớp

- Ta định nghĩa một đối tượng là một "cái gì đó" có ý nghĩa cho vấn đề ta quan tâm. Đối tượng phục vụ hai mục đích: Giúp hiểu rõ thế giới thực và cung cấp cơ sở cho việc cài đặt trong máy.
- Mỗi đối tượng có một *nét nhận dạng* để phân biệt nó với các đối tượng khác. Nét nhận dạng mang ý nghĩa các đối tượng được phân biệt với nhau do sự tồn tại vốn có của chúng chứ không phải các tính chất mà chúng có.

Đối Tượng và Lớp

- Các đối tượng có các đặc tính tương tự nhau được gom chung lại thành lớp đối tượng. Ví dụ *Người* là một lớp đối tượng. Một lớp đối tượng được đặc trưng bằng các *thuộc tính*, và các *hoạt động* (hành vi).
- Một *thuộc tính* (attribute) là một sự trừu tượng hoá các giá trị dữ liệu cho mỗi đối tượng trong lớp. *Tên, Tuổi, Cân nặng* là các thuộc tính của *Người.*

Đối Tượng và Lớp – Phương Thức

- Một *thao tác* (operation) là một hàm hay một phép biến đổi có thể áp dụng vào hay áp dụng bởi các đối tượng trong lớp.
- Cùng một thao tác có thể được áp dụng cho nhiều lớp đối tượng khác nhau, một thao tác như vậy được gọi là có tính *đa dạng* (polymorphism).

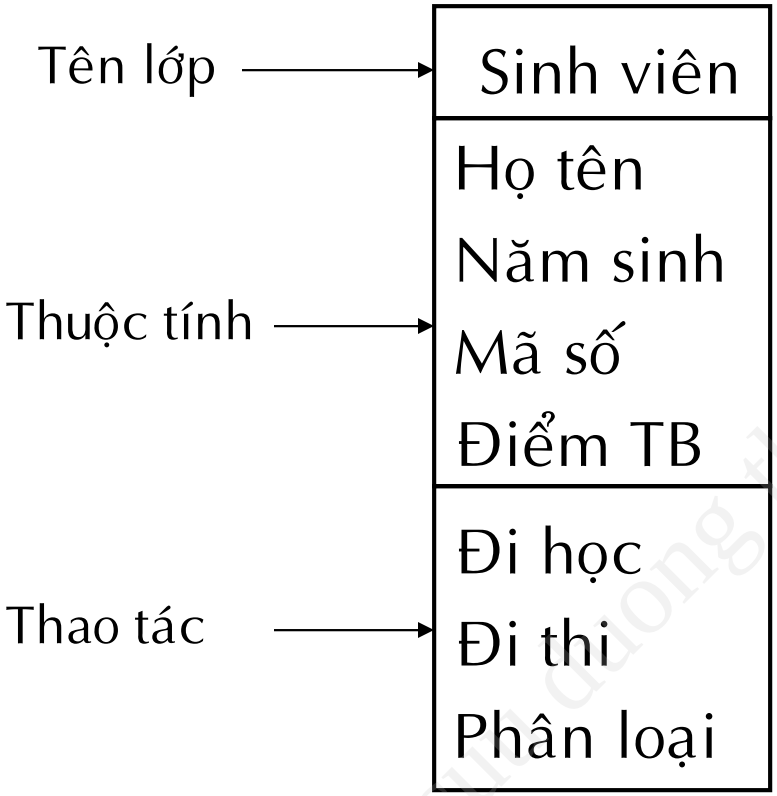
Đối Tượng và Lớp – Phương thức

- Mỗi thao tác trên mỗi lớp đối tượng cụ thể tương ứng với một cài đặt cụ thể khác nhau. Một cài đặt như vậy được gọi là một *phương thức* (method).
- Một đối tượng cụ thể thuộc một lớp được gọi là một *thể hiện* (instance) của lớp đó. Joe Smith, 25 tuổi, nặng 58kg, là một thể hiện của lớp người.

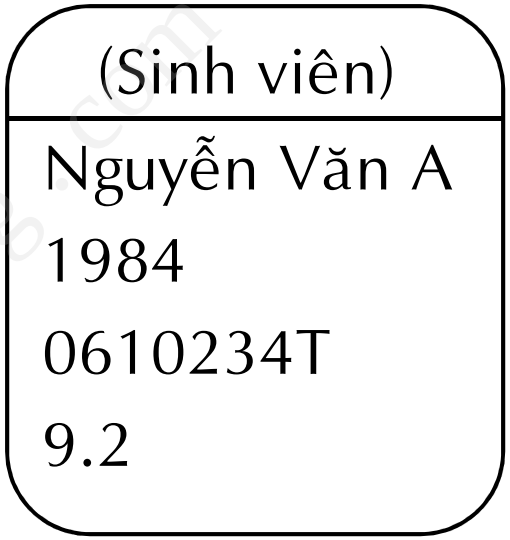
Sơ Đồ Đối Tượng

- Ta dùng *sơ đồ đối tượng* để mô tả các lớp đối tượng. Sơ đồ đối tượng bao gồm *sơ đồ lớp* và *sơ đồ thể hiện.*
- Sơ đồ lớp mô tả các lớp đối tượng trong hệ thống, một lớp đối tượng được diễn tả bằng một hình chữ nhật có 3 phần: phần đầu chỉ tên lớp, phần thứ hai mô tả các thuộc tính và phần thứ ba mô tả các thao tác của các đối tượng trong lớp đó.

Sơ đồ lớp và sơ đồ thể hiện

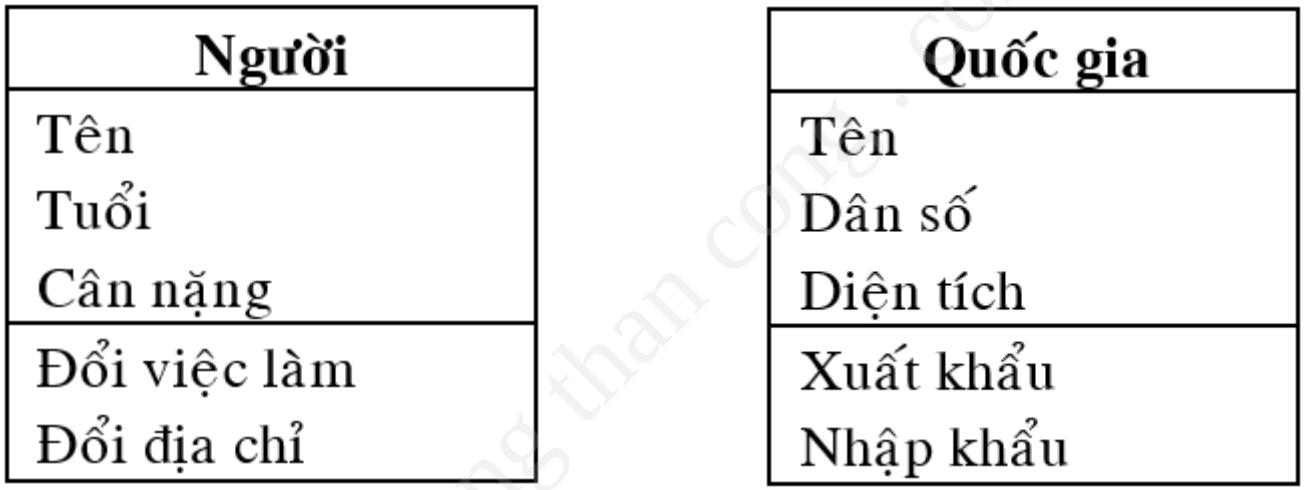


Sơ đồ lớp



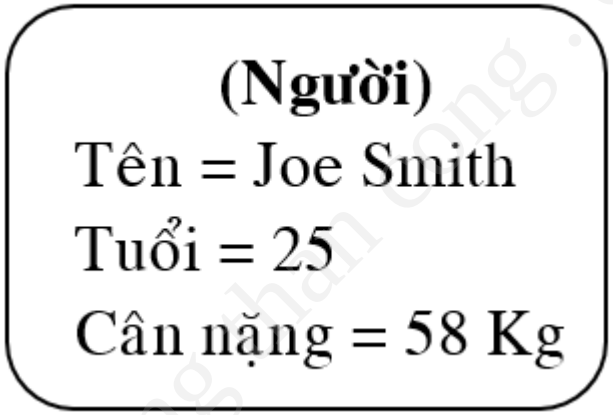
Sơ đồ thể hiện

Sơ đồ lớp và sơ đồ thể hiện



Sơ đồ lớp

Sơ đồ lớp và sơ đồ thể hiện



Sơ đồ thể hiện

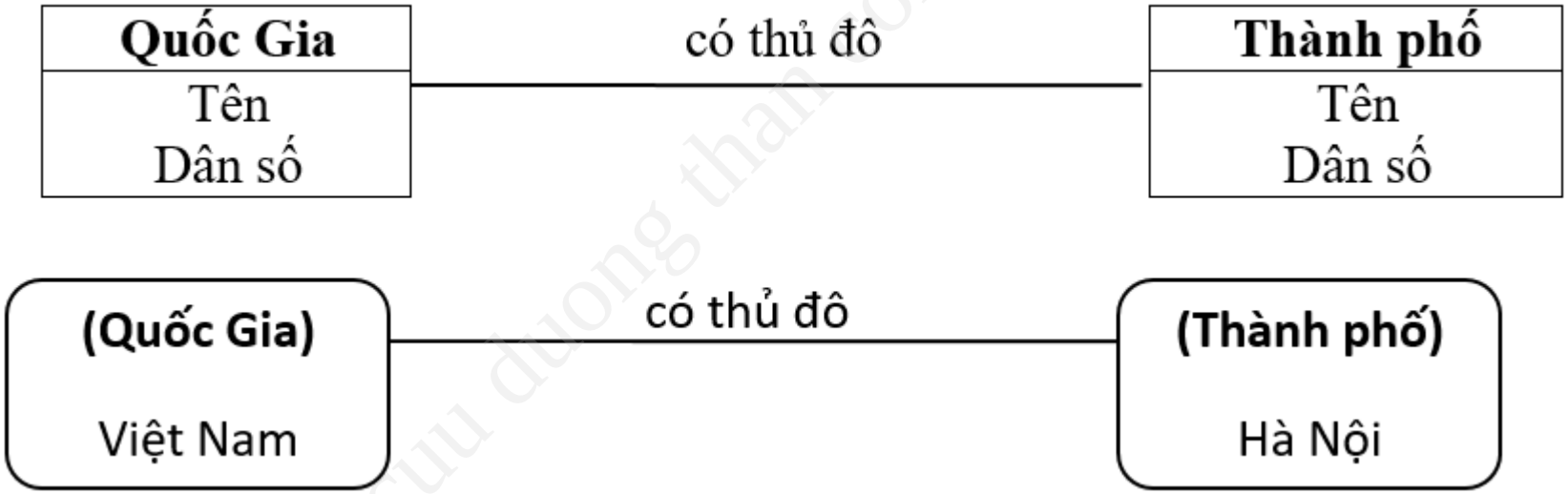
Quan hệ giữa các lớp

- Một mối *liên kết* (link) là một sự kết nối về mặt khái niệm giữa các đối tượng với nhau. Ví dụ Nguyễn Văn A *làm việc cho* Công ty X. Về mặt toán học, một mối liên kết là một bộ có thứ tự các đối tượng. Một mối liên kết là một thể hiện của một quan hệ.
- Một *quan hệ* (association) mô tả một nhóm các mối liên kết tương tự nhau về cấu trúc và về ngữ nghĩa. Ví dụ: Một người *làm việc cho* một Công ty.

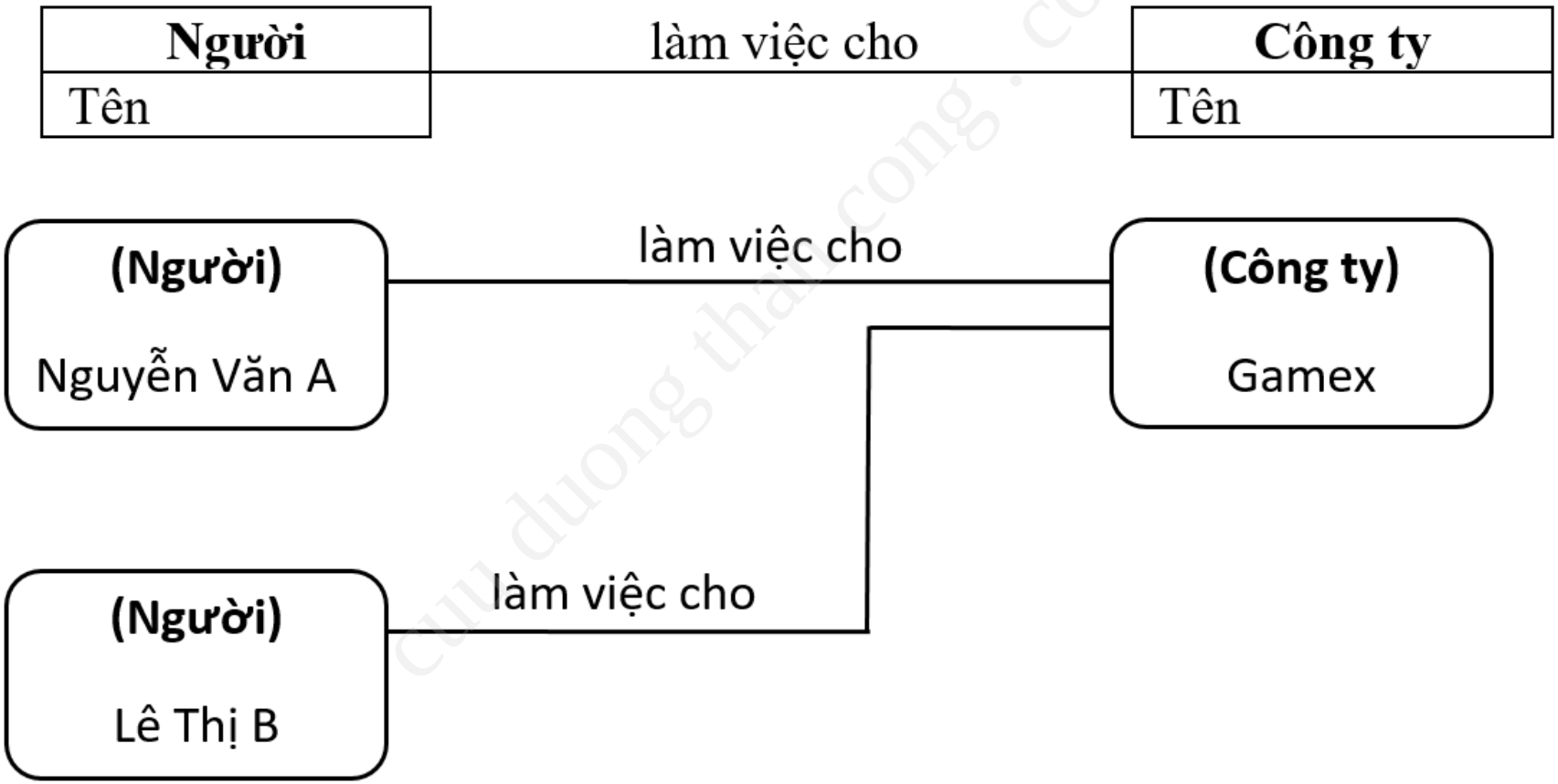
Quan hệ giữa các lớp

- Quan hệ có tính hai chiều. Một người làm việc cho một Công ty bao hàm ý nghĩa một công ty thuê một người.
- Quan hệ được mô hình bằng một *đường thẳng* nối hai lớp, trên đường thẳng có ghi nhãn là tên mối quan hệ. Một mối liên kết được vẽ bằng nối đường thẳng giữa hai thể hiện của các lớp đối tượng.

Quan hệ giữa các lớp



Quan hệ giữa các lớp

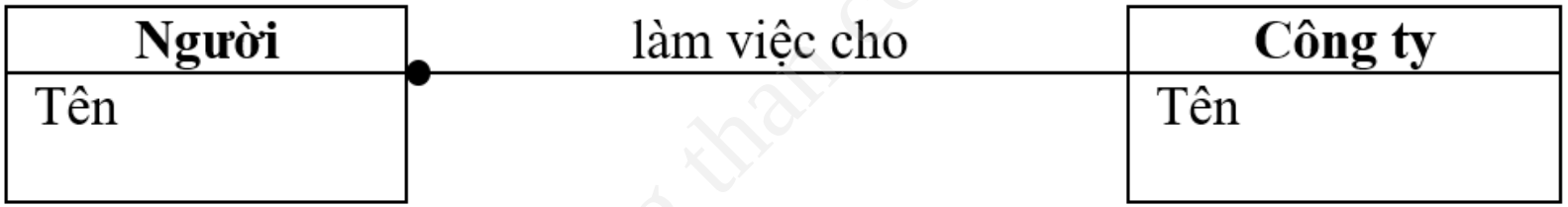


Quan hệ giữa các lớp – Bản số

- *Bản số:* Bản số qui định một thể hiện của một lớp có thể có quan hệ với bao nhiêu thể hiện của lớp khác. Dấu chấm tròn đặc để chỉ đối tượng ở bên phía nhiều của quan hệ, có thể kèm theo số.
  - ● : Từ 0 trở lên
  - ° : 0 hoặc 1
  - 2+ : Từ 2 trở lên
  - 3-5 : Từ 3 đến 5
  - 2,3,5 : 2 hoặc 3 hoặc 5

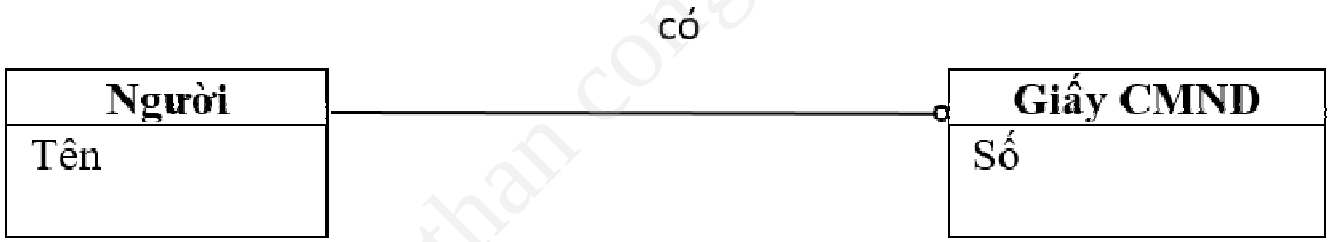
Bản số

- Quan hệ 1-nhiều:

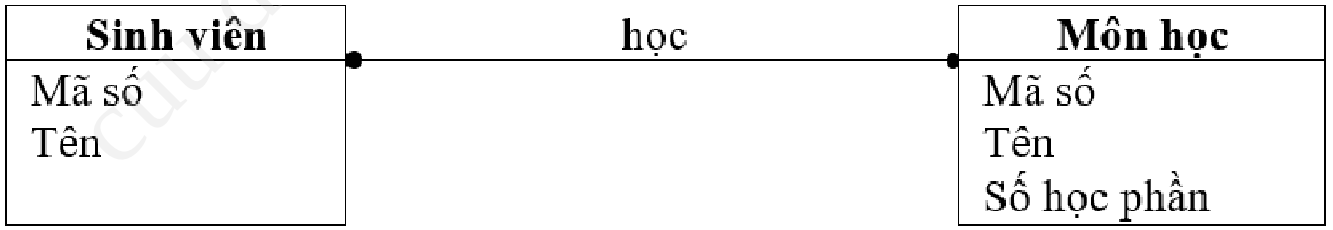


Bản số

Quan hệ 1-1(hoặc 0) :



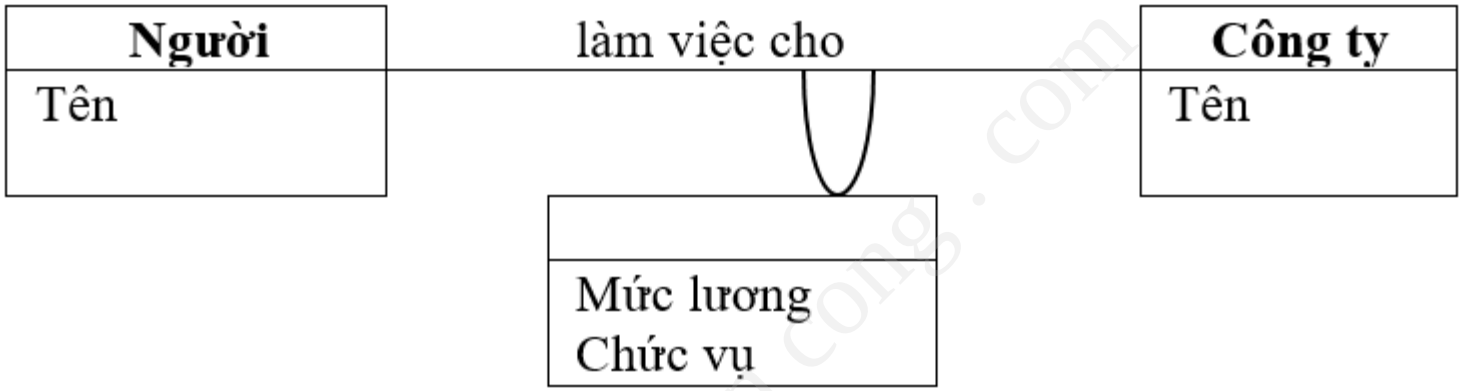
Quan hệ nhiều- nhiều



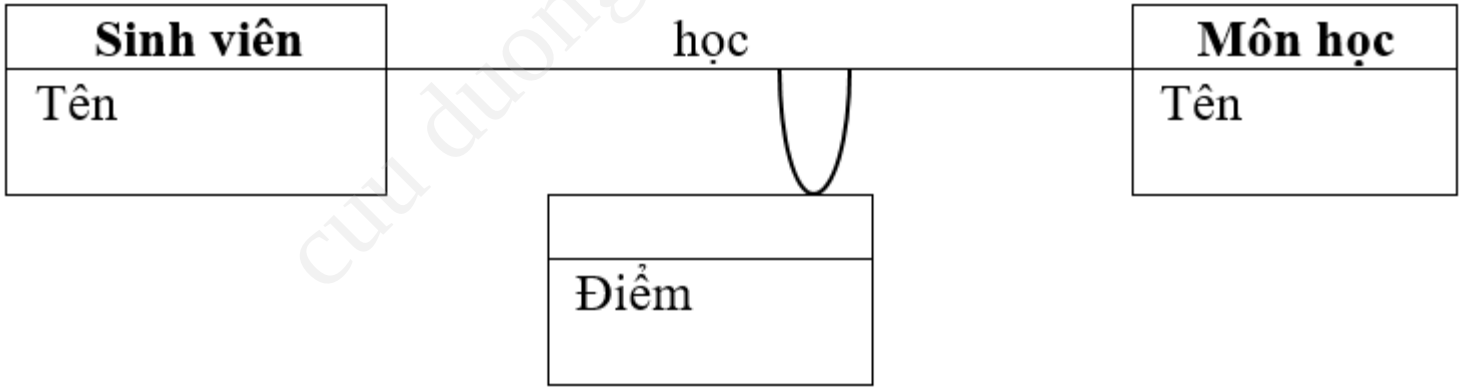
Quan hệ giữa các lớp

- *Thuộc tính liên kết:* Một thuộc tính liên kết (link attribute) là một tính chất của một mối quan hệ, thuộc tính liên kết được mô tả bằng một hình chữ nhật nối với quan hệ bằng một đường cong.

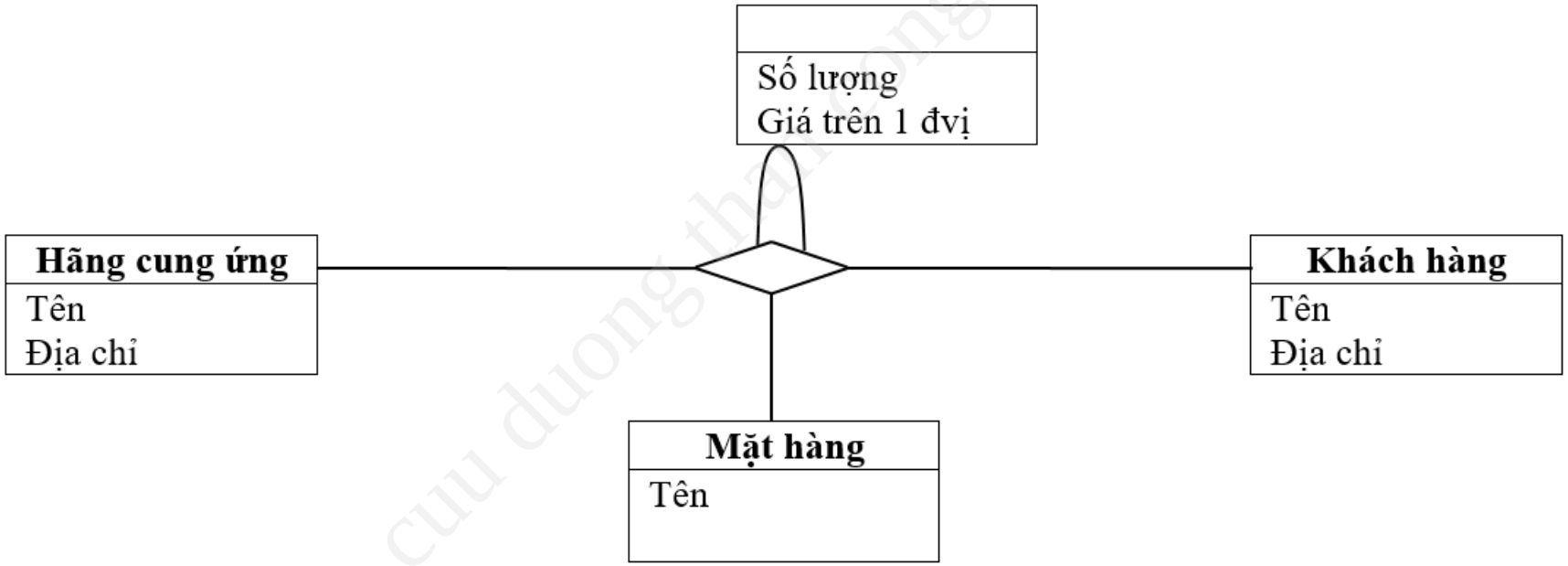
Thuộc tính liên kết



Quan hệ một người làm việc cho một công ty có mức lương và chức vụ



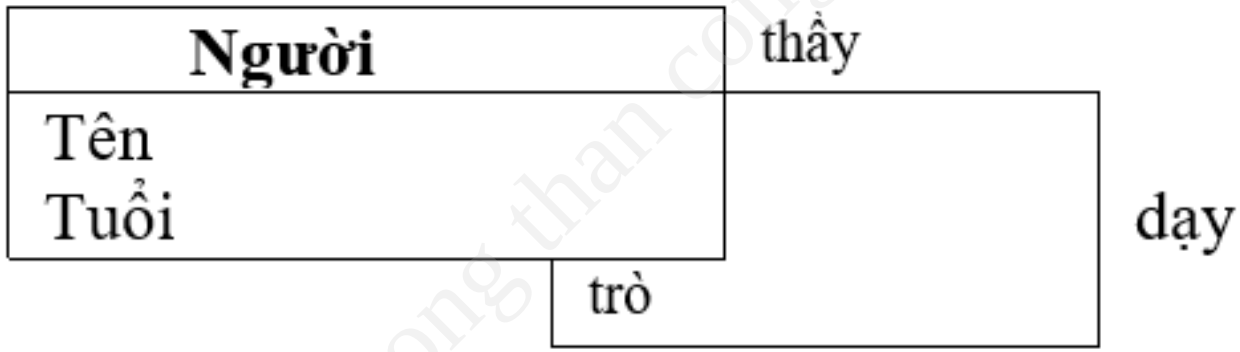
Thuộc tính liên kết



Quan hệ giữa các lớp

- *Tên vai trò:* Tên vai trò là danh hiệu được gắn vào một đầu của đường thẳng biểu diễn quan hệ để chỉ rõ vai trò của đối tượng tham gia với mối quan hệ.
- Thông thường tên vai trò không cần thiết khi mối quan hệ là giữa các lớp khác nhau. Trong trường hợp đó tên lớp đã chỉ rõ vai trò của đối tượng tham gia vào mối quan hệ.
- Tên vai trò có ý nghĩa khi mối quan hệ là giữa các đối tượng thuộc cùng một lớp.

Tên vai trò

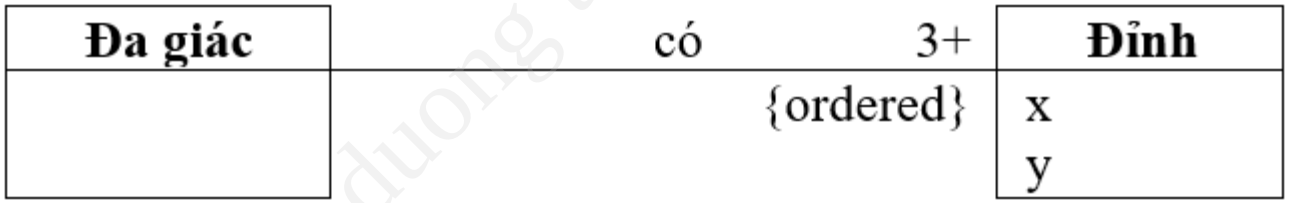


Quan hệ giữa các lớp

- *Sắp thứ tự:* Thông thường đối tượng ở phía nhiều của mối quan hệ không có thứ tự, và có thể quan điểm như một tập hợp.
- Đôi khi, các đối tượng có thứ tự rõ ràng, trong trường hợp này dùng từ khoá 'ordered' giữa hai dấu móc để biểu diễn tính thứ tự của các đối tượng.
- Sơ đồ sau chỉ mối quan hệ ‘đa giác có đỉnh’ với ràng buộc có tối thiểu 3 đỉnh và các đỉnh được sắp thứ tự.

Quan hệ giữa các lớp

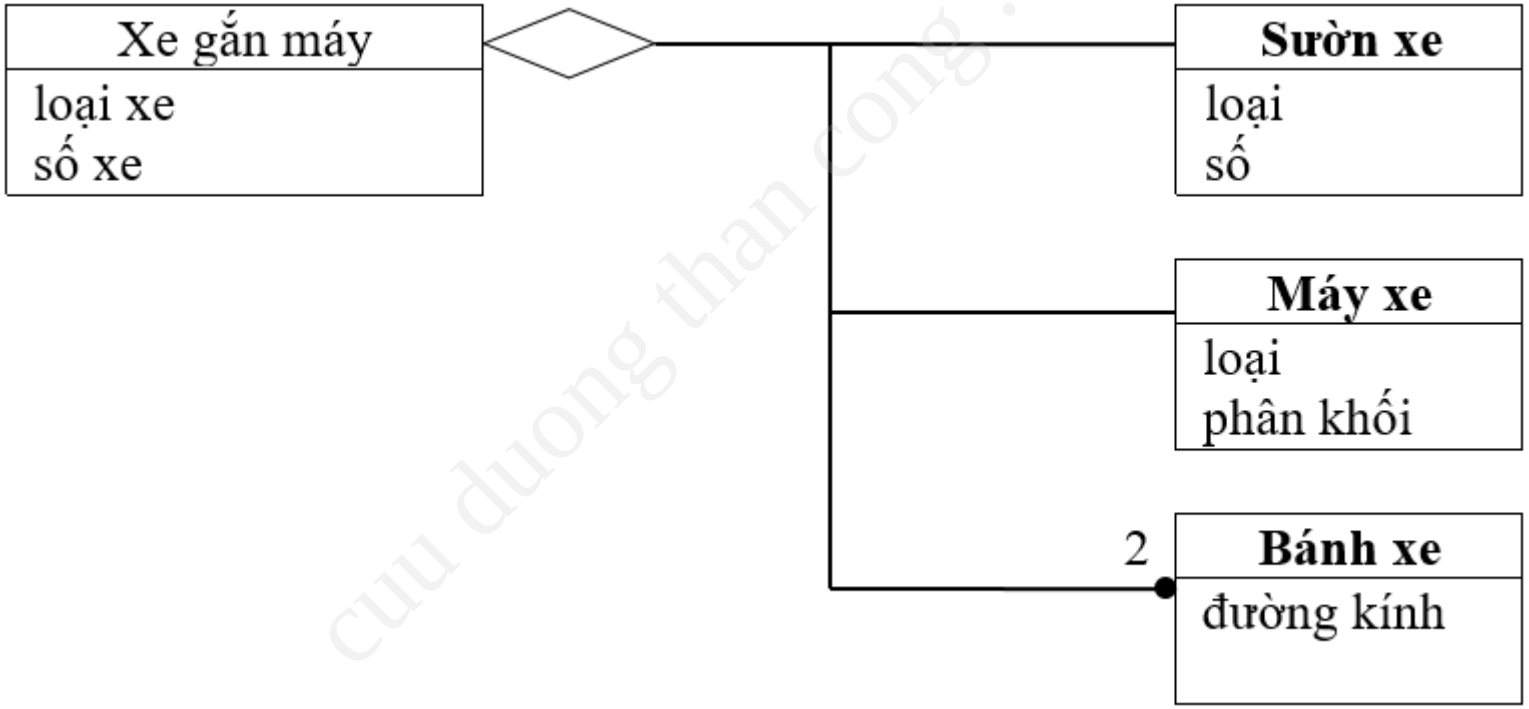
- Sơ đồ sau chỉ mối quan hệ ‘đa giác có đỉnh’ với ràng buộc có tối thiểu 3 đỉnh và các đỉnh được sắp thứ tự.



Quan hệ giữa các lớp

- *Quan hệ kết hợp:* Quan hệ kết hớp là một quan hệ đặc biệt, mô tả mối quan hệ "là sự kết hợp của" hoặc "là thành phần của". Mối quan hệ kết hợp được biểu diễn bằng một hình thoi nhỏ.

Quan hệ kết hợp



Cài đặt lớp trong NNLT

- Lớp có thể được cài đặt trong một ngôn ngữ lập trình.
- Các ngôn ngữ lập trình hướng đối tượng hỗ trợ tốt cho việc cài đặt lớp nhờ có hỗ trợ cài đặt thuộc tính, phương thức và kế thừa.

Cài đặt lớp trong một NNLT

- Lớp có thể được cài đặt trong một ngôn ngữ lập trình.
- Các ngôn ngữ lập trình hướng đối tượng hỗ trợ tốt cho việc cài đặt lớp nhờ có hỗ trợ cài đặt thuộc tính, phương thức và kế thừa.
- Ta sẽ minh hoạ cài đặt lớp bằng hai NNLT hướng đối tượng là C++ và/hoặc Objective C.

Cài đặt lớp trong một NNLT

- Nhắc lại: Một *kiểu dữ liệu* (trong một ngôn ngữ lập trình) là một biểu diễn cụ thể của một khái niệm có trong thực tế.
- Các ngôn ngữ lập trình cấp cao đều hỗ trợ các kiểu dữ liệu cơ bản, gọi là *kiểu có sẵn* (built-in data types), ví dụ trong C: int, long, float, double, char..., cho phép kiểm tra lúc biên dịch và phát sinh mã chương trình tối ưu. Các kiểu dữ liệu này cung cấp một giao diện tự nhiên độc lập với phần cài đặt.

Cài đặt lớp trong một NNLT

- Lớp là kiểu dữ liệu trừu tượng do *người sử dụng định nghĩa*, cho phép kết hợp dữ liệu, các phép toán, các hàm liên quan để tạo ra một đơn vị chương trình duy nhất. Các lớp này có đầy đủ ưu điểm và tiện lợi như các kiểu dữ liệu có sẵn.
- Lớp tách rời phần giao diện (chỉ liên quan với người sử dụng) và phần cài đặt lớp.

Cài đặt lớp trong một NNLT

- Lớp trong C/C++, Objective C có thể được cài đặt sử dụng từ khoá struct.
- Tuy nhiên, trong C++ ta dùng từ khoá class để cài đặt lớp sẽ tận dụng được các tính năng hướng đối tượng của C++.
- Objective C dùng cú pháp gần giống với Smalltalk để cài đặt lớp.

Cài đặt lớp - Ví dụ so sánh

- *Bài toán 1:* Xây dựng kiểu dữ liệu *phân số*, viết một ứng dụng cho phép thực hiện các phép toán số học trên hai phân số.
- *Bài toán 2:* Xây dựng kiểu dữ liệu *Stack* các số nguyên, viết hàm sử dụng Stack cho phép xuất một số nguyên trong hệ 16 (dưới dạng hexa). Viết ứng dụng minh hoạ.

Bài toán 1 - Cách tiếp cận cổ điển

```
struct PhanSo
{
   int tu, mau;
};

int uscln(int a, int b)
{
   if (b == 0) return abs(a);
      return uscln(b, a%b);
}
```

```
void PS_RutGon(PhanSo *a)
{
   int u = uscln(a->tu, a->mau);
   a->tu /= u;
   a->mau /= u;
   if (a->mau < 0)
   {
      a->tu = -a->tu;
      a->mau = -a->mau;
   }
}
```

```
void PS_Xuat(PhanSo a)
{
   printf("%d", a.tu);
   if (a.tu || a.mau != 1)
      printf("/%d", a.mau);
}

void PS_Nhap(PhanSo *a)
{
   scanf("%d%d", &a->tu, &a->mau);
}
```

```
PhanSo PS_Cong(PhanSo a, PhanSo b)
{
    PhanSo c;
    c.tu = a.tu*b.mau + a.mau*b.tu;
    c.mau = a.mau*b.mau;
    PS_RutGon(&c);
    return c;
}

PhanSo PS_Tru(PhanSo a, PhanSo b)
{
    PhanSo c;
    c.tu = a.tu*b.mau - a.mau*b.tu;
    c.mau = a.mau*b.mau;
    PS_RutGon(&c);
    return c;
}
```

```
PhanSo PS_Nhan(PhanSo a, PhanSo b)
{
    PhanSo c;
    c.tu = a.tu*b.tu;
    c.mau = a.mau*b.mau;
    PS_RutGon(&c);
    return c;
}

PhanSo PS_Chia(PhanSo a, PhanSo b)
{
    PhanSo c;
    c.tu = a.tu*b.mau;
    c.mau = a.mau*b.tu;
    PS_RutGon(&c);
    return c;
}
```

```
void main()
{
    PhanSo a = { 1,4 }, b = { 1,2 }, c;
    c = PS_Cong(a, b);
    PS_Xuat(c);
    puts("");
    c = PS_Tru(a, b);
    PS_Xuat(c);
    puts("");
    c = PS_Nhan(a, b);
    PS_Xuat(c);
    puts("");
    c = PS_Chia(a, b);
    PS_Xuat(c);
    puts("");
}
```

```
struct Fraction
{
   int numerator, denominator;
};

int gcd(int a, int b)
{
   if (b == 0) return abs(a);
      return gcd(b, a%b);
}
```

```
void fracReduce(Fraction *a)
{
   int u = gcd(a->numerator, a->denominator);
   a->numerator /= u;
   a->denominator /= u;
   if (a->denominator < 0)
   {
      a->numerator = -a->numerator;
      a->denominator = -a->denominator;
   }
}
```

```
void fracPrint(Fraction a)
{
   printf("%d", a.numerator);
   if (a.numerator || a.denominator != 1)
      printf("/%d", a.denominator);
}

void fracRead(Fraction *a)
{
   scanf("%d%d", &a->numerator, &a-
>denominator);
}
```

```
Fraction fracAdd(Fraction a, Fraction b)
{
    Fraction c;
    c.numerator = a.numerator*b.denominator +
a.denominator*b.numerator;
    c.denominator = a.denominator*b.denominator;
    fracReduce(&c);
    return c;
}

Fraction fracSub(Fraction a, Fraction b)
{
    Fraction c;
    c.numerator = a.numerator*b.denominator -
a.denominator*b.numerator;
    c.denominator = a.denominator*b.denominator;
    fracReduce(&c);
    return c;
}
```

```
Fraction fracMul(Fraction a, Fraction b)
{
    Fraction c;
    c.numerator = a.numerator*b.numerator;
    c.denominator = a.denominator*b.denominator;
    fracReduce(&c);
    return c;
}

Fraction fracDiv(Fraction a, Fraction b)
{
    Fraction c;
    c.numerator = a.numerator*b.denominator;
    c.denominator = a.denominator*b.numerator;
    fracReduce(&c);
    return c;
}
```

```
int main()
{
    Fraction a = { 1,4 }, b = { 1,2 }, c;
    c = fracAdd(a, b);
    fracPrint(c);
    puts("");
    c = fracSub(a, b);
    fracPrint(c);
    puts("");
    c = fracMul(a, b);
    fracPrint(c);
    puts("");
    c = fracDiv(a, b);
    fracPrint(c);
    puts("");
    return 0;
}
```

Kiểu Stack - Cách tiếp cận cổ điển

```
// stack.cpp
#pragma once

typedef int Item;

struct Stack
{
   Item *st, *top;
   int size;
};

void stackInit(Stack *ps, int sz);
void stackCleanUp(Stack *ps);
bool stackEmpty(Stack *ps);
bool stackFull(Stack *ps);
bool stackPush(Stack *ps, Item x);
bool stackPop(Stack *ps, Item *px);
```

Kiểu Stack - Cách tiếp cận cổ điển

```
// stack.cpp
#include "Stack.h"

void stackInit(Stack *ps, int sz)
{
   ps->top = ps->st = new Item[ps->size = sz];
}

void stackCleanUp(Stack *ps)
{
   delete[] ps->st;
}

bool stackEmpty(Stack *ps)
{
   return ps->top <= ps->st;
}
```

Kiểu Stack - Cách tiếp cận cổ điển

```
bool stackFull(Stack *ps)
{
    return ps->top - ps->st >= ps->size;
}

bool stackPush(Stack *ps, Item x)
{
    if (stackFull(ps)) return false;
    *ps->top++ = x;
    return true;
}

bool stackPop(Stack *ps, Item *px)
{
    if (stackEmpty(ps)) return false;
    *px = *--ps->top;
    return true;
}
```

Kiểu Stack - Cách tiếp cận cổ điển

```
#include <iostream>
using namespace std;
#include "stack.h"

void printHex(long n)
{
   static char hTab[] = "0123456789ABCDEF";
   Stack s;
   stackInit(&s, 8);
   int x;
   do {
      stackPush(&s, n % 16);
      n /= 16;
   } while (n);
   while (stackPop(&s, &x))
   cout << hTab[x];
}
```

```
int main()
{
   int a;
   cout << “Input a: ";
   cin >> a;
   printHex(a);
   cout << "\n";
   return 0;
}
```

Cài đặt lớp – Ví dụ so sánh

Nhận xét:

- Giải quyết được vấn đề.
- Khai báo cấu trúc dữ liệu nằm riêng, các hàm xử lý dữ liệu nằm riêng ở một nơi khác. Do đó *khó theo dõi quản lý khi hệ thống lớn*. Vì vậy khó bảo trì.
- Mọi thao tác đều có tham số đầu tiên là con trỏ đến đối tượng cần thao tác. Tư tưởng thể hiện ở đây là *hàm hay thủ tục đóng vai trò trọng tâm*. Đối tượng được gởi đến cho hàm xử lý.
- Trình tự sử dụng qua các bước: Khởi động, sử dụng thực sự, dọn dẹp.

Kiểu Stack - Cách tiếp cận dùng hàm thành phần

```
// stack.h
#pragma once
typedef int Item;

struct Stack
{
    Item *st, *top;
    int size;
    void init(int sz){ top = st = new Item[size =
    sz];}
    void cleanUp(){ delete[] st; }
    bool empty() { return top <= st; }
    bool full() { return top - st >= size; }
    bool push(Item x);
    bool pop(Item *px);
};
```

Kiểu Stack - Cách tiếp cận dùng hàm thành phần

```
#include "Stack.h"

bool Stack::push(Item x)
{
   if (full()) return false;
   *top++ = x;
   return true;
}

bool Stack::pop(Item *px)
{
   if (empty()) return false;
   *px = *--top;
   return true;
}
```

Kiểu Stack - Cách tiếp cận dùng hàm thành phần

```
// stack.cpp, other include(s) ...
#include "stack.h"

void printHex(long n)
{
    static char hTab[] = "0123456789ABCDEF";
    Stack s;
    s.init(8);
    int x;
    do {
        s.push(n % 16);
        n /= 16;
    } while (n);
    while (s.pop(&x))
        cout << hTab[x];
    s.cleanUp();
}
```

```
int main()
{
   int a;
   cout << "Nhap a: ";
   cin >> a;
   printHex(a);
   cout << "\n";
   return 0;
}
```

Cài đặt lớp - Ví dụ so sánh

- *Nhận xét:*
- Giải quyết được vấn đề.
- Dữ liệu và các hàm xử lý dữ liệu được gom vào một chỗ bên trong cấu trúc. Do đó dễ theo dõi quản lý, dễ bảo trì nâng cấp.
- Các thao tác đều bớt đi một tham số so với cách tiếp cận cổ điển. Vì vậy việc lập trình gọn hơn. Tư tưởng thể hiện ở đây là *đối tượng đóng vai trò trọng tâm*. Đối tượng thực hiện thao tác trên chính nó.
- Trình tự sử dụng qua các bước: Khởi động, sử dụng thực sự, dọn dẹp.