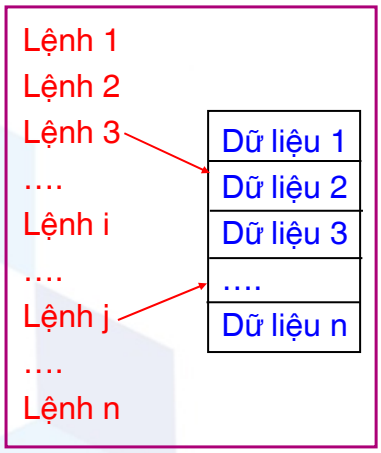
Chương 2

Tổ chức cấu trúc phần mềm hướng đối tượng

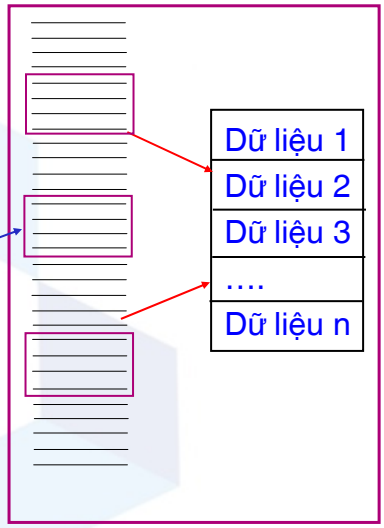
2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

- Ban đầu, chương trình thường rất ngắn, chỉ giải quyết 1 vấn đề nhỏ, rõ ràng, đơn giản.
- Lúc này, chương trình là 1 danh sách ngắn các lệnh, các lệnh này sẽ xử lý tập các dữ liệu (số lượng cũng rất ít).



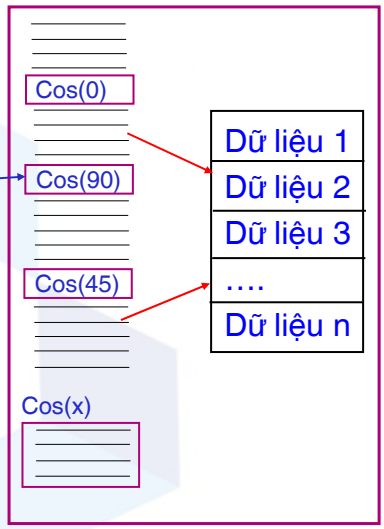
2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

- Nếu phân tích kỹ hơn, ta thấy trong danh sách các lệnh của chương trình có hiện tượng sau : đoạn lệnh giải quyết vấn về nhỏ hơn nào đó xuất hiện nhiều lần vì chương trình cần thực hiện nó nhiều lần.
- Ta viết đoạn lệnh này 1 lần, gán cho nó 1 tên nhận dạng. Ta gọi nó là chương trình con. Trong họ ngôn ngữ C, ta dùng thuật ngữ function.
- Function giúp ta tổ chức chương trình nhất quán hơn, gọn nhẹ hơn, dễ bảo trì và phát triển hơn.



2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

- Trong chương trình bên phải, ta thấy chức năng tính cos(x) cần 3 lần trong chương trình, ta định nghĩa hàm cos(x) 1 lần với tham số x.
- Mỗi khi cần tính cos(x) trong chương trình, ta chỉ cần viết 1 lệnh gọi hàm đơn giản.

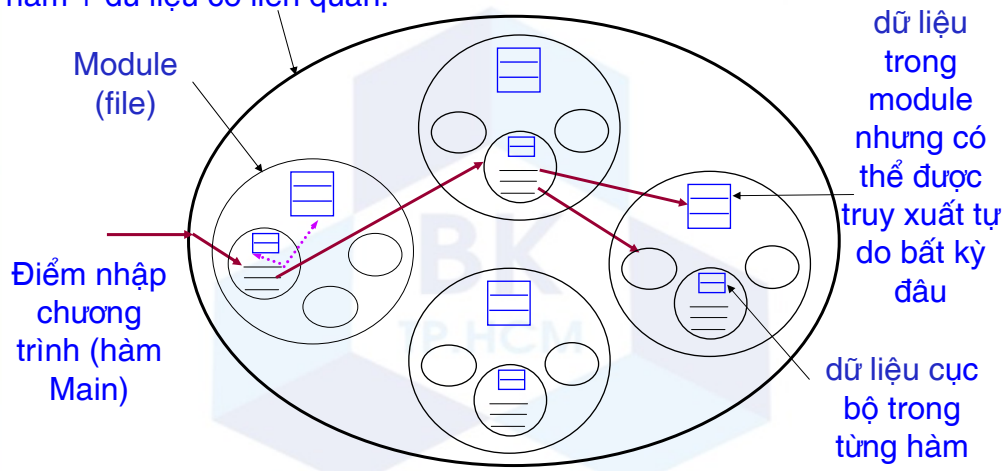


2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

- Trong thực tế hiện nay, chương trình thường giải quyết nhiều vấn đề lớn, phức tạp, nó tương ứng với số lượng rất lớn các hàm và dữ liệu. Lúc này để chúng trong 1 module (file) rất bất tiện, khó duy trì...
- Ta phải tìm cách khác tổ chức phần mềm : phân rã module rất lớn và phức tạp ban đầu thành nhiều module nhỏ : mỗi module chỉ chứa 1 ít hàm chức năng và dữ liệu có mối quan hệ mật thiết nào đó.
- Đây là cách tổ chức phần mềm hướng cấu trúc cổ điển trước đây.

2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

Chương trình = tập các module chức năng, mỗi module chứa 1 số hàm + dữ liệu có liên quan.



2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

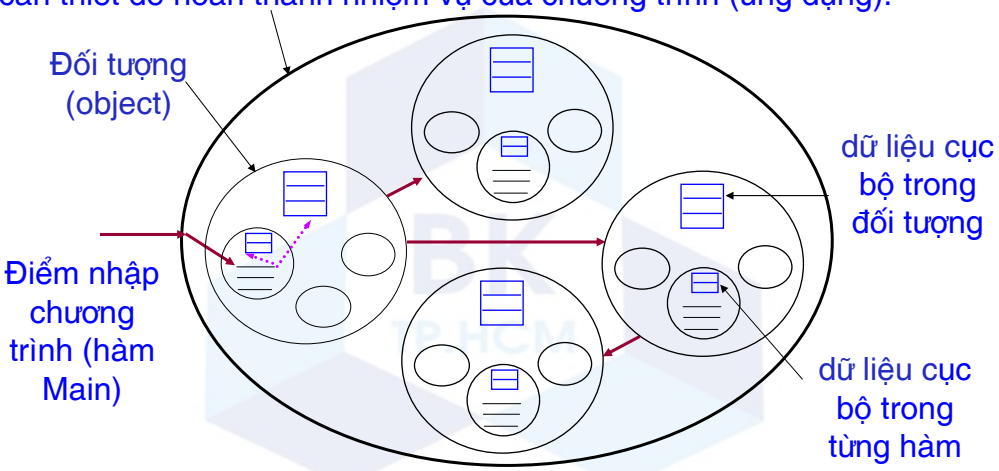
- Mặc dù mỗi hàm hay dữ liệu được đặt trong 1 module xác định, nhưng mặc định chúng được truy xuất tự do bởi bất kỳ đâu trong chương trình. Đây là 1 khuyết điểm lớn, ta khắc phục bằng cách miêu tả tầm vực riêng cho từng phần tử : mặc định trong các ngôn ngữ C, C++, mỗi phần tử đều có tầm vực công cộng, nghĩa là bất kỳ đâu trong chương trình đều truy xuất được nó.
- Nếu muốn hạn chế việc truy xuất từ ngoài module, ta dùng từ khóa static kết hợp với phần tử cần bao đóng, phần tử sẽ có tầm vực cục bộ, bên ngoài module không truy xuất được nó nữa.
- Một phương pháp khác để hạn chế tầm vực truy xuất từng phần tử là định nghĩa nó theo cấu trúc lồng nhau dạng phân cấp : nếu A chức B thì B chỉ được truy xuất bởi A, các nơi khác bên ngoài A sẽ không thấy và truy xuất được B. Pascal là ngôn ngữ điển hình về hỗ trợ khả năng này.

2.1 Sự phát triển trong cấu trúc tổ chức phần mềm

- Cho dù đã dùng 1 số biện pháp ở slide trước thì cấu trúc tổ chức phần mềm hướng cấu trúc vẫn còn 1 số khuyết điểm khác.
- Thí dụ, trong phần mềm ta cần nhiều module có cùng chức năng A, số lượng là động chưa biết trước. Cấu trúc hướng cấu trúc chỉ cho phép nhân bản module A với số lượng tĩnh biết trước khi phần mềm chạy. Do đó ta cần tìm 1 cấu trúc tổ chức phần mềm khác : đó là cấu trúc tổ chức phần mềm hướng đối tượng mà ta dùng chủ yếu hiện nay.

2.2 Cấu trúc tổ chức chương trình OOP

Chương trình = tập các đối tượng sống độc lập, tương tác nhau khi cần thiết để hoàn thành nhiệm vụ của chương trình (ứng dụng).



2.2 Cấu trúc tổ chức chương trình OOP

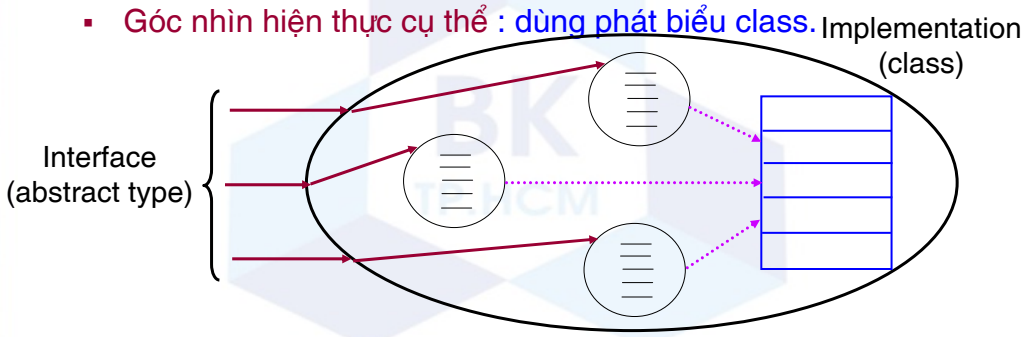
- Cấu trúc chương trình hướng đối tượng rất thuần nhất, chỉ chứa 1 loại thành phần : đối tượng.
- Các đối tượng có tính độc lập rất cao ⇒ quản lý, kiểm soát chương trình rất dễ (cho dù chương trình có thể rất lớn) ⇒ dễ nâng cấp, bảo trì.
- Không thể tạo ra dữ liệu toàn cục của chương trình ⇒ điểm yếu nhất của chương trình cấu trúc không tồn tại nữa.

2.3 Đối tượng (Object)

- Đối tượng là nguyên tử cấu thành ứng dụng.
- Đối tượng bao gồm 2 loại thành phần chính yếu :
  - Tập các tác vụ (operation) : mỗi tác vụ thực hiện 1 chức năng rõ ràng đơn giản nào đó mà bên ngoài cần dùng.
  - Tập các thuộc tính dữ liệu (attribute) : mỗi thuộc tính có kiểu dữ liệu cụ thể, và chứa 1 giá trị cụ thể thuộc kiểu tương ứng tại từng thời điểm. Các thuộc tính phục vụ cho các tác vụ và là đối tượng xử lý bởi các tác vụ, do đó nên bao đóng và ẩn dấu chúng trong đối tượng, không cho bên ngoài thấy và truy xuất trực tiếp.

2.3 Đối tượng (Object)

- Viết phần mềm hướng đối tượng là qui trình đặc tả các loại đối tượng cấu thành ứng dụng.
- Đặc tả một loại đối tượng là đặc tả 2 góc nhìn khác nhau về đối tượng :
  - Góc nhìn sử dụng : dùng phát biểu interface.
  - Góc nhìn hiện thực cụ thể : dùng phát biểu class.



2.4 Kiểu trừu tượng (Abstract type) hay interface

- Phát biểu interface định nghĩa thông tin sử dụng đối tượng mà bên ngoài thấy, kết hợp các thông tin này với 1 tên gọi, tên này được gọi là tên kiểu trừu tượng (Abstract type) hay ngắn gọn là type.
- Interface là tập hợp các điểm nhập (entry) mà bên ngoài có thể giao tiếp với đối tượng. C# cho phép định nghĩa nhiều loại điểm nhập, nhưng phổ biến nhất là tác vụ chức năng (operation).
- Ta dùng chữ ký (signature) để định nghĩa và phân biệt mỗi điểm nhập. Chữ ký của 1 tác vụ gồm :
  1. tên tác vụ (operation)
  2. danh sách tham số hình thức, mỗi tham số được đặc tả bởi 3 thuộc tính : tên, type và chiều di chuyển (IN, OUT, INOUT).
  3. đặc tả chức năng của tác vụ (thường ở dạng chú thích).

2.4 Kiểu trừu tượng (Abstract type) hay interface

- Muốn làm việc với 1 đối tượng nào đó, ta thường dùng biến đối tượng. Biến đối tượng nên được đặc tả kiểu bằng tên interface, hạn chế dùng tên class cụ thể.
- Biến đối tượng là biến tham khảo, nó không chứa trực tiếp đối tượng, nó chỉ chứa các thông tin để truy xuất được đối tượng, bất chấp đối tượng đang nằm ở đâu.
- Biến đối tượng thuộc kiểu interface có thể tham khảo đến nhiều đối tượng thuộc các class cụ thể khác nhau miễn sao các đối tượng này hỗ trợ được interface tương ứng.
- Như vậy, nếu ta dùng đối tượng thông qua biến thuộc kiểu interface thì ta không cần biết bất kỳ thông tin hiện thực chi tiết nào về đối tượng mà mình đang dùng, nhờ vậy code ứng dụng sẽ độc lập hoàn toàn với class hiện thực của đối tượng được sử dụng trong ứng dụng.

Thí dụ interface

- Thí dụ sau đây miêu tả 1 interface của đối tượng mà hỗ trợ 2 tác vụ chuẩn hóa chuỗi tiếng Việt về dạng tổ hợp và dựng sẵn. Thông qua interface, người dùng không hề thấy và biết chi tiết về hiện thực của các tác vụ, nhưng điều này không hề ngăn cản họ trong việc dùng đối tượng nào đó có interface IVietLib.

```
interface IVietLib {
    //tác vụ chuẩn hóa chuỗi tiếng Việt về dạng tổ hợp
    int VnPre2Comp(String src, int len, ref String dst);
    //tác vụ chuẩn hóa chuỗi tiếng Việt về dạng dựng sẵn
    int VnComp2Pre(String src, int len, ref String dst);
}
```


2.5 Class (Implementation)

- Phát biểu class định nghĩa chi tiết hiện thực đối tượng :
  - định nghĩa các thuộc tính, mỗi thuộc tính được đặc tả bởi các thông tin về nó như tên nhận dạng, kiểu dữ liệu, tầm vực truy xuất,... Kiểu của thuộc tính có thể là type cổ điển (kiểu giá trị : số nguyên, thực, ký tự, chuỗi ký tự,...) hay kiểu đối tượng (kiểu tham khảo), trong trường hợp sau thuộc tính sẽ là tham khảo đến đối tượng khác. Trạng thái của đối tượng là tập giá trị của tất cả thuộc tính của đối tượng tại thời điểm tương ứng.
  - 'coding' các tác vụ (miêu tả giải thuật chi tiết về hoạt động của tác vụ), các hàm nội bộ trong class và các thành phần khác.
- Ngoài các thành phần chức năng, ta còn phải định nghĩa các tác vụ quản lý đối tượng như : khởi tạo trạng thái ban đầu (constructor), dọn dẹp các phần tử liên quan đến đối tượng khi đối tượng bị xóa (destructor).

Thí dụ về class

- Thí dụ sau đây miêu tả 1 class hiện thực interface IVietLib :

```
class MyVietLib : IVietLib {
    //định nghĩa các thuộc tính cần dùng cho 2 tác vụ
    ...
    //định nghĩa 2 tác vụ quản lý đối tượng
    MyVietLib() {…}
    ~MyVietLib() {…}
    //định nghĩa thuật giải tác vụ chuẩn hóa chuỗi tiếng Việt về dạng tổ hợp
    int VnPre2Comp(String src, int len, ref String dst) {….}
    //định nghĩa thuật giải tác vụ chuẩn hóa chuỗi tiếng Việt về dạng dựng sẵn
    int VnComp2Pre(String src, int len, ref String dst) {…}
}
```


2.6 Tính bao đóng (encapsulation)

- Bao đóng : che dấu mọi chi tiết hiện thực của đối tượng, không cho bên ngoài thấy và truy xuất ⇒ tạo độ độc lập cao giữa các đối tượng (tính nối ghép – coupling – hay phụ thuộc giữa các đối tượng rất thấp), nhờ vậy việc quản lý, hiệu chỉnh và nâng cấp từng thành phần phần mềm dễ dàng, không ảnh hưởng đến các thành phần khác.
  - che dấu các thuộc tính dữ liệu : nếu cần cho phép bên ngoài truy xuất 1 thuộc tính vật lý, ta tạo 1 thuộc tính luận lý (2 tác vụ get/set tương ứng) để giám sát và kiểm soát việc truy xuất.
  - che dấu chi tiết hiện thực các tác vụ.
  - che dấu các local function và sự hiện thực của chúng.
- C# cung cấp các từ khóa private, protected, internal, public (slide 5, chương 3) để xác định tầm vực truy xuất từng thành phần của class.

2.7 Tính thừa kế (inheritance)

- Tính thừa kế cho phép giảm nhẹ công sức định nghĩa interface/class : ta có thể định nghĩa các interface/class không phải từ đầu mà bằng cách kế thừa interface/class có sẵn nhưng gần giống với mình :
  - Miêu tả tên cha : mọi thành phần của cha trở thành của mình.
  - override 1 số method của class cha, kết quả override chỉ tác dụng trên đối tượng của class con.
  - định nghĩa thêm các chi tiết mới (thường khá ít).
- Đa thừa kế hay đơn thừa kế. C# cho phép đa thừa kế interface (đa hiện thực), nhưng chỉ hỗ trợ đơn thừa kế class.
- Thừa kế tạo ra mối quan hệ cha/con : phần tử đã có là cha, phần tử thừa kế cha được gọi là con. Cha/con có thể là trực tiếp hay gián tiếp.

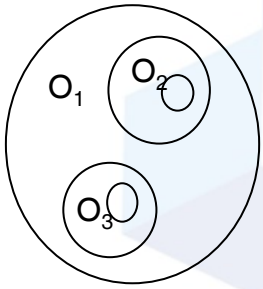
2.7 Tính thừa kế (inheritance)

- Với các tính chất về thừa kế như slide trước, ta rút ra được 1 số kết luận :
  - Đối tượng của class con luôn lớn hay hay ít nhất bằng đối tượng class cha.
  - Và như thế, đối tượng class con hoàn toàn có thể đóng vai trò của đối tượng class cha và thay thế đối tượng class cha khi cần thiết, nhưng ngược lại thường không được.

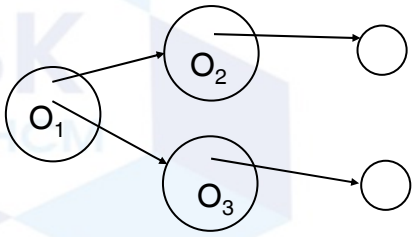
2.8 Tính bao gộp (aggregation)

- 1 đối tượng có thể chứa nhiều đối tượng khác ⇒ tạo nên mối quan hệ bao gộp 1 cách đệ quy giữa các đối tượng. Thí dụ đối tượng quốc gia chứa nhiều đối tượng tỉnh, đối tượng tỉnh chứa nhiều đối tượng quận/huyện,…
- Có 2 góc nhìn về tính bao gộp : ngữ nghĩa & hiện thực.

Góc nhìn ngữ nghĩa



Góc nhìn hiện thực



Ví dụ về bao gộp

```
//định nghĩa class miêu tả đối tượng đồ họa cơ bản
abstract class Geometry {          // abstract base class
   public abstract void Draw (Graphics g); // abstract operation
   protected int xPos, yPos;
   protected COLORREF color;
};

//định nghĩa class đồ họa phức hợp = tập các đối tượng đồ họa đã có
class GeoGroup : Geometry {
  public override void Draw (Graphics g) {...} ; // override
  private Geometry [] objList; //danh sách các đối tượng thành phần;
  int count; //số lượng các đối tượng thành phần
};
```

2.9 Thông điệp (Message), đa xạ (Polymorphism)

- Thông điệp là phương tiện giao tiếp (hay tương tác) duy nhất giữa các đối tượng, nó cho phép gọi 1 tác vụ chức năng của 1 đối tượng từ 1 tham khảo đến đối tượng.
- Thông điệp bao gồm 3 thành phần :
  - tham khảo đến đối tượng cần nhờ.
  - tên tác vụ muốn gọi.
  - danh sách tham số thực cần truyền cho (hay nhận về từ) tác vụ.

```
public override void Draw (Graphics g) {
  for (int i=0; i < count; i++)
    objList[i].Draw(g);       //gởi thông điệp nhờ đối tượng objList[i]
                              // tự hiển thị mình lên đối tượng vẽ g
}
```


2.9 Thông điệp (Message), đa xạ (Polymorphism)

- Xét đoạn lệnh sau :

```
C1 obj = new C1();
obj.func();  //lần 1
obj = new C2();
obj.func(); //lần 2
```

- Lệnh gởi thông điệp obj.func() kích hoạt tác vụ func() của class C1 hay tác vụ func() của class C2 ?

1. **Dùng kỹ thuật xác định hàm và liên kết tĩnh** : Dựa vào thông tin tại thời điểm dịch, biến obj thuộc kiểu C1 và máy dịch lời gởi thông điệp obj.func() thành lời gọi hàm C1_func(). Như vậy mỗi khi máy chạy lệnh này, hàm C1_func() sẽ chạy, bất chấp tại thời điểm chạy, obj đang tham khảo đối tượng của class khác (C2). Điều này không đúng với ý muốn người lập trình.

2.9 Thông điệp (Message), đa xạ (Polymorphism)

2. **Dùng kỹ thuật xác định hàm và liên kết động** : Lệnh gởi thông điệp obj.func() không được dịch ra 1 lời gọi hàm nào cả mà được dịch thành đoạn lệnh máy với chức năng sau : xác định biến obj đang tham khảo đến đối tượng nào, thuộc class nào, rồi gọi hàm func() của class đó chạy. Như vậy, nếu obj đang tham khảo đối tượng thuộc class C1 thì hàm C1_func() sẽ được gọi, còn nếu obj đang tham khảo đối tượng thuộc class C2 thì hàm C2_func() sẽ được gọi. Ta nói lời gởi thông điệp obj.func() có tính đa xạ. Điều này giải quyết đúng ý muốn người lập trình.

- **Tính đa xạ** : cùng 1 lệnh gởi thông điệp đến đối tượng thông qua cùng 1 tham khảo nhưng ở vị trí/thời điểm khác nhau có thể kích hoạt việc thực thi tác vụ khác nhau của các đối tượng khác nhau.

Kiểm tra kiểu (type check)

- Khi lập trình, ta thường phạm nhiều lỗi : lỗi về từ vựng, cú pháp, lỗi về thuật giải... Trong các lỗi thì lỗi về việc gán dữ liệu khác kiểu thường xảy ra nhất.
- Để phát hiện triệt để và sớm nhất các lỗi sai về kiểu, máy sẽ dùng cơ chế kiểm tra kiểu chặt và sớm tại thời điểm dịch.
- Trong lúc dịch, bất kỳ hoạt động gán dữ liệu nào (lệnh gán, truyền tham số) đều được kiểm tra kỹ lưỡng, nếu dữ liệu và biến lưu trữ không "tương thích" thì báo sai.
- Tiêu chí không "tương thích" là gì ?
  - dùng kỹ thuật so trùng tên kiểu : tên kiểu không trùng nhau là không tương thích.

Kiểm tra kiểu (type check)

- dùng mối quan hệ 'conformity' (tương thích tổng quát). Kiểu A 'conformity' với kiểu B nếu A cung cấp mọi tác vụ mà B có, từng tác vụ của A cung cấp tương thích với tác vụ tương ứng của B. Nói nôm na A lớn hơn hay bằng B thì A 'conformity' với B.

- Như vậy, quan hệ so trùng hay quan hệ con/cha (sub/super) là trường hợp đặc biệt của quan hệ tương thích tổng quát.
- Nhờ dùng mối quan hệ 'conformity', một biến obj thuộc kiểu C1 có thể chứa tham khảo đến nhiều đối tượng thuộc nhiều class khác nhau, miễn sao các class này tương thích với class được dùng để định nghĩa biến obj.

2.10 Tính tổng quát hóa (Generalization)

- Viết phần mềm hướng đối tượng là quá trình lặp : viết phát biểu interface/class để đặc tả từng loại đối tượng cấu thành phần mềm.
- Nếu số lượng class cấu thành ứng dụng quá lớn thì việc viết phần mềm sẽ khó khăn, tốn nhiều thời gian công sức hơn.
- Làm sao giảm nhẹ thời gian, công sức lập trình các ứng dụng lớn ?
  1. sử dụng cơ chế thừa kế trong định nghĩa interface/class.
  2. thay vì trực tiếp viết các class cụ thể đặc tả cho các đối tượng trong phần mềm, ta chỉ viết 1 class tổng quát hóa, rồi nhờ class này sinh tự động mã nguồn các class cụ thể.

2.10 Tính tổng quát hóa (Generalization)

- Thí dụ, thay vì phải viết n class gần giống nhau như danh sách các số nguyên, danh sách các số thực, danh sách các chuỗi, danh sách các record Sinhvien, danh sách các đối tượng đồ họa,... ta chỉ cần viết 1 class tổng quát hóa : danh sách các phần tử có kiểu hình thức T. Khi cần tạo 1 class danh sách các phần tử thuộc kiểu cụ thể nào đó, ta chỉ viết lệnh gọi class tổng quát hóa và truyền tên kiểu cụ thể của phần tử trong danh sách.
- Mỗi ngôn ngữ hướng đối tượng (C++, Java, C#) có khả năng, tính chất, mức độ hỗ trợ tổng quát hóa khác nhau.

2.11 Kết chương

- Chương này đã giới thiệu cấu trúc của chương trình hướng đối tượng, các phương tiện đặc tả đối tượng như interface, class.
- Chương này cũng đã giới thiệu các tính chất liên quan đến việc đặc tả và sử dụng đối tượng như thừa kế, bao đóng, bao gộp, tổng quát hóa.
- Chương này cũng đã giới thiệu phương tiện giao tiếp duy nhất giữa các đối tượng là thông điệp, nhu cầu cần phải có tính đa xạ trong việc thực hiện lệnh gởi thông điệp.