Chương 3

Đối tượng : đặc tả, tương tác, vòng đời

3.0 Dẫn nhập

- Chương này giới thiệu cú pháp của phát biểu interface và class của Java để đặc tả thông tin giao tiếp sử dụng và chi tiết hiện thực 1 loại đối tượng được dùng trong chương trình.
- Chương này cũng giới thiệu cú pháp các phát biểu định nghĩa các thành phần cấu thành đối tượng như thuộc tính vật lý, thuộc tính giao tiếp, tác vụ chức năng... Chương này cũng phân biệt 2 loại thành phần được đặc tả trong 1 class : thành phần dùng chung (static) và thành phần nhân bản theo từng đối tượng.
- Chương này cũng giới thiệu vòng đời của từng đối tượng trong chương trình, cách thức quản lý đời sống của đối tượng, các thời điểm quan trọng nhất như lúc tạo mới đối tượng, lúc xóa đối tượng và các hoạt động xảy ra tại các thời điểm này.
- Chương này cũng giới thiệu sự tương tác giữa các đối tượng trong lúc chúng đang sống để hoàn thành nhiệm vụ của chương trình.

3.1 Tổng quát về phát biểu class của Java

Ngôn ngữ Java (hay bất kỳ ngôn ngữ lập trình nào khác) cung cấp cho người lập trình nhiều phát biểu (statement) khác nhau, trong đó phát biểu class để đặc tả chi tiết hiện thực từng loại đối tượng cấu thành phần mềm là phát biểu quan trọng nhất. Sau đây là 1 template của 1 class Java :

```
class MyClass [extends BaseClass] [implements TypeList] {
  //định nghĩa các thuộc tính vật lý của đối tượng
  //định nghĩa các tác vụ chức năng
  //định nghĩa các tác vụ quản lý đời sống đối tượng
}
```

3.1 Tổng quát về phát biểu class của Java

- Khi định nghĩa 1 class mới, ta có thể thừa kế tối đa 1 class đã có (đơn thừa kế), tên class này nếu có, phải nằm sau từ khóa extends.
- Khi định nghĩa 1 class, ta có thể hiện thực nhiều interface khác nhau (đa hiện thực), danh sách này nếu có, phải nằm sau từ khóa implements. Trong trường hợp nhiều interface có cùng 1 tác vụ (phân biệt bằng chữ ký), chỉ có 1 hiện thực tác vụ này trong class. Việc truy xuất tác vụ này thông qua interface nào cũng đều kích hoạt được đúng tác vụ duy nhất được hiện thực trong class :

```
class MyClass extends BaseClass implements I1, I2, I3 {
  //hiện thực các tác vụ cùng chữ ký trong các interface khác
nhau
  void func1() {}
  ...
}
```

3.2 Định nghĩa thuộc tính vật lý

- Mỗi thuộc tính vật lý của đối tượng là 1 biến dữ liệu cụ thể. Phát biểu định nghĩa 1 thuộc tính vật lý sẽ đặc tả các thông tin sau về thuộc tính tương ứng :
  - Tên nhận dạng.
  - Kiểu dữ liệu.
  - Giá trị ban đầu.
  - Tầm vực truy xuất
- Cú pháp đơn giản để định nghĩa 1 thuộc tính vật lý như sau :

  [scope | attribute] *type name* [= *value*];

3.2 Định nghĩa thuộc tính vật lý

- thành phần scope miêu tả tầm vực truy xuất của thuộc tính. Sau đây là danh sách các từ khóa có thể được dùng (cho cả thuộc tính và tác vụ chức năng) :
  - public : thuộc tính có thể được truy xuất bất kỳ đâu.
  - protected : thuộc tính có thể được truy xuất bởi class hiện hành và các class con, cháu.
  - private : thuộc tính chỉ có thể được truy xuất nội bộ trong class hiện hành.
  - nếu thành phần scope không được miêu tả tường minh, thuộc tính sẽ có tầm vực friendly (chỉ cho các class trong cùng package truy xuất).

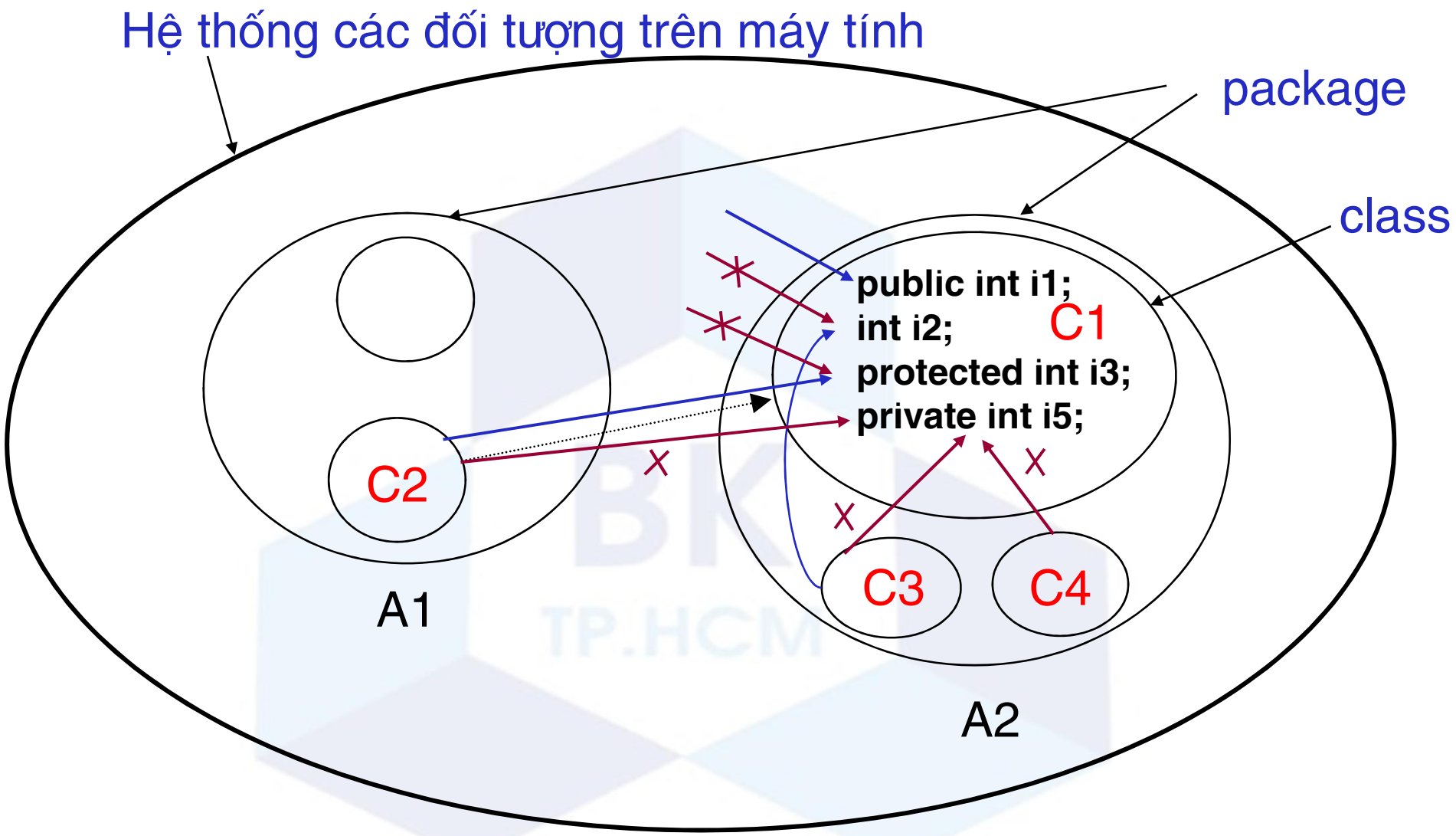
3.2 Định nghĩa thuộc tính vật lý

- thành phần attribute miêu tả tính chất của thuộc tính. Sau đây là danh sách các từ khóa có thể được dùng (cho cả thuộc tính và tác vụ chức năng) :
  - static : thuộc tính dùng chung, chỉ có 1 instance kết hợp với class, không được nhân bản cho từng đối tượng.
  - abstract : chỉ dùng cho tác vụ để miêu tả tác vụ tương ứng là trừu tượng, chưa được hiện thực cụ thể.
  - final : thuộc tính có giá trị cố định, không thể thay đổi sau này. Đây là phương tiện định nghĩa hằng gợi nhớ trong Java hay để định nghĩa tác vụ mà không cho phép class con override.
  - native : chỉ dùng cho tác vụ, miêu tả tác vụ tương ứng được hiện thực bằng ngôn ngữ khác, chứ không được hiện thực trong class Java này.

3.2 Định nghĩa thuộc tính vật lý

- synchronized : chỉ dùng cho tác vụ, miêu tả tác vụ tương ứng được thực thi theo kiểu loại trừ tương hỗ.
- transcient : thuộc tính này không cần đọc/ghi ra đĩa.
- volatile : các threads đồng thời truy xuất trực tiếp thuộc tính này, chứ không cần đệm cục bộ nó cho từng thread.
- strictfp : tác vụ sẽ xử lý số thực theo version cũ, mặc định tác vụ xử lý số thực theo version Java 2.

3.2 Định nghĩa thuộc tính vật lý



3.2 Định nghĩa thuộc tính vật lý

- thành phần *type* thường là tên kiểu dữ liệu của thuộc tính tương ứng, nó có thể là tên kiểu giá trị hay tên kiểu tham khảo.
- thành phần *name* là tên nhận dạng thuộc tính.
- thành phần [= *value*] miêu tả biểu thức xác định trị ban đầu của thuộc tính.
- thành phần nào nằm trong [] là nhiệm ý (optional), có thể có hoặc không. Các thành phần khác bắt buộc phải có.
- Thí dụ :

```
private int dorong = 10;
private int docao = 10;
```

3.3 Định nghĩa tác vụ chức năng

- Mỗi tác vụ (operation) thực hiện 1 chức năng xác định, rõ ràng nào đó mà bên ngoài đối tượng (client) cần dùng. Định nghĩa tác vụ gồm 2 phần : định nghĩa interface sử dụng và định nghĩa thuật giải chi tiết mà tác vụ thực hiện (method).
- Lệnh định nghĩa 1 tác vụ thường gồm 5 phần sau :

  [scope | attribute] *return_type name* **(**arg_list**)** *body*
  - scope miêu tả tầm vực truy xuất của tác vụ. attribute miêu tả tính chất của tác vụ. Thông tin này cũng giống như trong thuộc tính vật lý.
  - return_type là tên kiểu của giá trị mà tác vụ sẽ trả về.
  - name là tên tác vụ, arg_list là danh sách từ 0 tới n tham số hình thức cách nhau bởi dấu ',', định nghĩa mỗi tham số hình thức gần giống như định nghĩa thuộc tính vật lý.

3.4 Thành phần static và thành phần không static

- Phát biểu class được dùng để đặc tả các đối tượng cùng loại mà phần mềm dùng. Về nguyên tắc, khi đối tượng được tạo ra (bằng lệnh new), nó sẽ chứa tất cả các thành phần được đặc tả trong class tương ứng. Tuy nhiên, nếu xét chi li thì Java cho phép đặc tả 2 loại thành phần trong 1 class như sau :
  1. Thành phần static : là thành phần có từ khóa static trong lệnh định nghĩa nó. Đây là thành phần kết hợp với class, nó không được nhân bản cho từng đối tượng và như thế đối tượng không thể truy xuất nó. Cách duy nhất để truy xuất thành phần static là thông qua tên class.

```
//System.out là tên class chứa các hàm truy xuất
//các thiết bị xuất chuẩn
System.out.println("Nội dung cần hiển thị");
```

3.4 Thành phần static và thành phần không static

2. Thành phần không static : là thành phần không dùng từ khóa static trong lệnh định nghĩa nó. Đây là thành phần kết hợp với từng đối tượng, nó sẽ được nhân bản cho từng đối tượng. Ta truy xuất thành phần không static thông qua tham khảo đối tượng.

```
class Rectangle {
    public int Cao;  //thuộc tính vật lý
    ...
}
Rectangle r = new Rectangle();
r.Cao = 10;
```

3.5 Thí dụ về định nghĩa 1 đối tượng

```
public class C_IntStack implements T_IntStack {
   // khai báo các hằng cần dùng cho class
   private final int GROWBY = 4;
   private int data[];        //danh sách đặc các phần tử trong stack
   private int top;           // chỉ số phần tử của đỉnh stack
   private int max; // số lượng max hiện hành của stack
   //hàm constructor
   public C_IntStack() {
       top = 0;
        max = 0;
    }
```

3.5 Thí dụ về định nghĩa 1 đối tượng

```
//hiện thực hàm push của stack
public boolean push(int newVal) {
   if (top==max) try { //hết stack
      //phân phối lại vùng nhớ mới lớn hơn GROWBY phần tử
      int newdata[] = new int[GROWBY+max];
      //di chuyển stack cũ về stack mới
      for (int i = 0; i<max; i++)
         newdata[i] =data[i];
      //không cần delete vùng nhớ của stack cũ
      //ghi nhớ stack mới
      data = newdata;
      max += GROWBY;
   }
   catch (Exception e) { return false; }
```

3.5 Thí dụ về định nghĩa 1 đối tượng

```
        //chứa giá trị mới vào đỉnh stack
        data[top++] =newVal;
        return true;
    }
    //hiện thực hàm pop của stack
    public int pop() throws Exception {
        if (top == 0) //cạn stack
            throw new Exception ("Cạn stack");
        else //lấy giá trị từ đỉnh stack ra retVal
            return data[--top];
    }
}
```

3.5 Thí dụ về định nghĩa 1 đối tượng

```
public interface T_IntStack {
  boolean push(int newVal);    // cất giá trị vào đỉnh stack
  int pop() throws Exception ;  // lấy giá trị từ đỉnh stack
}
```

3.6 Quản lý đời sống đối tượng - Hàm Constructor

- Class mô hình các đối tượng cùng loại mà phần mềm dùng. Lúc lập trình, ta chỉ đặc tả class, đối tượng chưa có. Khi ứng dụng chạy, tại thời điểm cần thiết, phần mềm sẽ phải tạo tường minh đối tượng bằng lệnh new :

```
Rectangle objRec = new Rectangle(); //tạo đối tượng
```

- Trạng thái của đối tượng là tập giá trị cụ thể của các thuộc tính. Ngay sau đối tượng được tạo ra, nó cần có trạng thái ban đầu xác lập nào đó. Hàm constructor cho phép người lập trình miêu tả hoạt động xác lập trạng thái ban đầu của đối tượng.
- Cũng giống như nhiều tác vụ khác, hàm contructor có thể có nhiều "overloaded" khác nhau (với số lượng tham số khác nhau hay tính chất của 1 tham số nào đó khác nhau).

3.6 Quản lý đời sống đối tượng - Hàm Constructor

- Mỗi lần đối tượng được tạo ra (bởi lệnh new), máy sẽ gọi tự động contructor của class tương ứng. Tùy theo tham số của lệnh new mà contructor nào tương thích sẽ được kích hoạt chạy.
- Trong nội bộ 1 class, các tác vụ chỉ có thể truy xuất các thuộc tính của mình và các thuộc tính thừa kế từ cha có tầm vực protected, public, chứ không thể truy xuất trực tiếp các thuộc tính thừa kế từ cha có thuộc tính private. Do đó nếu chỉ chạy constructor của class cần tạo đối tượng thì không thể khởi tạo hết các thuộc tính của đối tượng, cần kích hoạt hết các contructor của các class cha (gián tiếp hay trực tiếp).
- Mặc định, khi cần gọi constructor của class cha chạy, máy sẽ gọi contructor không tham số. Nếu người lập trình muốn khác thì phải khai báo lại tường minh "overloaded" nào cần chạy thông qua lệnh super() nằm ở đầu thân contructor.

3.6 Quản lý đời sống đối tượng - Hàm Constructor

```
//class A  có 2 hàm contructor
class A {
   A() {...}
   A(int i) {...}
   ...
};
//class B thừa kế A, có 2 hàm contructor
class B extends A {
   B() { super(); ...}
   B(int i) {super(i);...}
   ...
};
```

3.6 Quản lý đời sống đối tượng - Hàm Constructor

```
//class C thừa kế B, có 2 hàm contructor
class C : B {
   C() { super();...}
   C(int i) { super(i);...}
   ... };
C c1 = new C();  //kích hoạt A() → B() → C()
C c2 = new C(5);   //kích hoạt A(5) → B(5) → C(5)
```

- Việc xác định contructor nào được kích hoạt phải theo chiều từ dưới lên bắt đầu từ class được new, nhưng các contructor được chạy thực sự sẽ theo chiều từ trên xuống bắt đầu từ class tổ tiên đời đầu.

3.7 Quản lý đời sống đối tượng - Hàm Destructor

- Đối tượng là 1 thực thể, nó có đời sống như bao thực thể khác. Như ta đã biết, khi ta gọi lệnh new, 1 đối tượng mới thuộc class tương ứng sẽ được tạo ra (trong không gian hệ thống), trạng thái ban đầu sẽ được xác lập thông qua việc kích hoạt dây chuyền các contructor của các class thừa kế. Chương trình sẽ lưu giữ tham khảo đến đối tượng trong biến tham khảo để khi cần, gởi thông điệp nhờ đối tượng thực thi dùm 1 tác vụ nào đó.
- Java không cung cấp tác vụ nào để xóa đối tượng khi không cần dùng nó nữa. Thật vậy, đánh giá 1 đối tượng nào đó có cần dùng nữa hay không là việc không dễ dàng, dễ nhầm lẫn nếu để chương trình tự làm.
- Tóm lại, trong Java, chương trình chỉ tạo tường minh đối tượng khi cần dùng nó, chương trình không cần quan tâm việc xóa đối tượng.

3.7 Quản lý đời sống đối tượng - Hàm Destructor

- Như vậy, đối tượng sẽ bị xóa lúc nào, bởi ai ? Hệ thống có 1 module đặc biệt tên là "Garbage collection" (trình dọn rác), module này sẽ theo dõi việc dùng các đối tượng, khi thấy đối tượng nào mà không còn ai dùng nữa thì nó sẽ xóa dùm tự động.
- Trình dọn rác không biết trạng thái đối tượng tại thời điểm bị xóa nên nó không làm gì ngoài việc thu hồi vùng nhớ mà đối tượng chiếm. Như vậy rất nguy hiểm, thí dụ như đối tượng bị xóa đã mở, khóa file và đang truy xuất file dỡ dang.
- Để giải quyết vấn đề xóa đối tượng được triệt để, trình dọn rác sẽ gọi tác vụ destructor của đối tượng sắp bị xóa, nhiệm vụ của người đặc tả class là hiện thực tác vụ này.
- Tác vụ destructor không có kiểu trả về, không có tham số hình thức → không có overloaded, chỉ có 1 destructor/class mà thôi.

3.7 Quản lý đời sống đối tượng - Hàm Destructor

- Mặc dù người đặc tả class sẽ hiện thực tác vụ destructor nếu thấy cần thiết, nhưng code của chương trình không được gọi trực tiếp destructor của đối tượng. Chỉ có trình dọn rác của hệ thống mới gọi destructor của đối tượng ngay trước khi xóa đối tượng đó.
- Destructor của 1 class cũng chỉ xử lý trạng thái đối tượng do các thuộc tính của class đó qui định, nó cần gọi destructor của class cha để xử lý tiếp trạng thái đối tượng do các thuộc tính private của class cha qui định, và cứ thế tiếp tục.
- Tóm lại trước khi xóa một đối tượng, trình dọn rác sẽ gọi các destructor theo chiều từ dưới lên, bắt đầu từ class hiện hành của đối tượng, sau đó tới class cha, ... và cuối cùng là class tổ tiên đời đầu (root).

3.8 Tương tác giữa các đối tượng trong Java

- Muốn tương tác với đối tượng nào đó, ta phải có tham khảo đến đối tượng đó. Thường ta lưu giữ tham khảo đối tượng cần truy xuất trong biến đối tượng (biến tham khảo). Thông qua tham khảo đến đối tượng, ta có thể thực hiện 1 trong các hành động tương tác sau đây :
  1. truy xuất 1 thuộc tính vật lý của đối tượng có tầm vực cho phép (public hay friendly hay protected).
  2. truy xuất 1 thuộc tính luận lý của đối tượng.
  3. gọi 1 tác vụ có tầm vực cho phép.
- Chúng ta sẽ làm rõ chi tiết lệnh gởi thông điệp để nhờ 1 tác vụ của đối tượng xác định ở các slide sau.

3.9 Liên kết tĩnh trong việc gởi thông điệp

- Xét đoạn lệnh sau :

```
class C1 {
    private void func1() {} //dịch ra hàm mã máy có tên là C1_func1
    public void func2() {} //dịch ra hàm mã máy có tên là C1_func2
}
class C2 extends C1 {
    private void func1() {}      //dịch ra hàm mã máy có tên là
    C2_func1
    public void func2() {}  //dịch ra hàm mã máy có tên là C2_func2
}
C1 obj = new C1();
obj.func1();  //lần 1   gọi hàm mã máy nào ?
//đoạn code có thể làm obj chỉ về đối tượng của class C2, C3,...
obj.func1(); //lần 2   gọi hàm mã máy nào ?
```

3.9 Liên kết tĩnh trong việc gởi thông điệp

Hai lệnh gởi thông điệp obj.func1() trong slide trước sẽ kích hoạt tác vụ func1() của class C1 hay tác vụ func1() của class C2 ?

1. **Dùng kỹ thuật xác định hàm và liên kết tĩnh** : Tại thời điểm dịch, chương trình dịch chỉ biết biến obj thuộc kiểu C1 và nó dịch cả 2 lời gởi thông điệp obj.func1() thành lời gọi hàm C1_func1(). Như vậy mỗi khi máy chạy lệnh obj.func1() lần 1, hàm C1_func1() sẽ được gọi, điều này đúng theo yêu cầu của phần mềm. Nhưng khi máy chạy lệnh obj.func1() lần 2, hàm C1_func1() cũng sẽ được gọi, điều này không đúng theo yêu cầu của phần mềm vì lúc này obj đang tham khảo đối tượng của class C2.

   Mặc định, Java dùng kỹ thuật xác định hàm và liên kết tĩnh khi dịch lệnh gọi hàm privated cục bộ trong cùng class để đạt hiệu quả cao nhưng không làm sai ngữ nghĩa gọi hàm.

3.10 Liên kết động để đảm bảo tính đa xạ

Bây giờ nếu ta hiệu chỉnh 2 lệnh gởi thông điệp obj.func1() trong slide trước thành obj.func2() thì máy sẽ kích hoạt tác vụ func2() của class C1 hay tác vụ func2() của class C2 ?

2. **Dùng kỹ thuật xác định hàm và liên kết động** : Lệnh gởi thông điệp obj.func2() 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 func2() của class đó chạy. Như vậy, lần gởi thông điệp 1, biến obj đang tham khảo đối tượng thuộc class C1 nên máy sẽ gọi hàm C1_func2(), điều này đúng theo yêu cầu của phần mềm. Khi máy chạy lệnh obj.func2() lần 2, đoạn code xác định hàm và liên kết động sẽ gọi được hàm C2_func2(), điều này cũng đúng theo yêu cầu của phần mềm. Ta nói lời gởi thông điệp obj.func2() có tính đa xạ.

3.10 Liên kết động để đảm bảo tính đa xạ

- Trong Java, lệnh gởi thông điệp (hay gọi hàm) đến tác vụ có tầm vực khác private luôn được xử lý theo cơ chế liên kết động và sẽ đảm bảo được tính đa xạ, tức đảm bảo tính đúng đắn trong lời gởi thông điệp.

3.11 Kết chương

- Chương này đã giới thiệu vòng đời của từng đối tượng trong chương trình, cách thức quản lý đời sống của đối tượng, các thời điểm quan trọng nhất như lúc tạo mới đối tượng, lúc xóa đối tượng cũng như cách miêu tả các hoạt động xảy ra tại các thời điểm này.
- Chương này cũng đã giới thiệu sự tương tác giữa các đối tượng trong lúc chúng đang sống để hoàn thành nhiệm vụ của chương trình. Gởi thông điệp là sự tương tác chính yếu giữa các đối tượng, và cần phải có tính đa xạ.

3.11 Kết chương

- Chương này đã giới thiệu cú pháp của phát biểu interface và class của Java để đặc tả thông tin giao tiếp sử dụng và chi tiết hiện thực 1 loại đối tượng được dùng trong chương trình.
- Chương này cũng đã giới thiệu cú pháp các phát biểu định nghĩa các thành phần cấu thành đối tượng như thuộc tính vật lý, thuộc tính giao tiếp, tác vụ chức năng... Chương này cũng phân biệt 2 loại thành phần được đặc tả trong 1 class : thành phần dùng chung (static) và thành phần nhân bản theo từng đối tượng.
- Chương này cũng đã giới thiệu vòng đời của từng đối tượng trong chương trình, cách thức quản lý đời sống của đối tượng, các thời điểm quan trọng nhất như lúc tạo mới đối tượng, lúc xóa đối tượng và các hoạt động xảy ra tại các thời điểm này.
- Chương này cũng đã giới thiệu sự tương tác giữa các đối tượng để hoàn thành nhiệm vụ của chương trình.