Kế thừa và đa hình

(Dựa trên slide của PGS. TS. Nguyễn Việt Hà)

Nội dung

- Đa hình
  - Khái niệm
  - Chuyển kiểu: upcasting / downcasting
  - Hành vi: liên kết động
- Lớp và phương thức trừu tượng
  - lớp/phương thức trừu tượng
  - template method
- Giao diện và đa kế thừa
- Mẫu thiết kế
  - Template method, Prototype, Composition

Tài liệu tham khảo

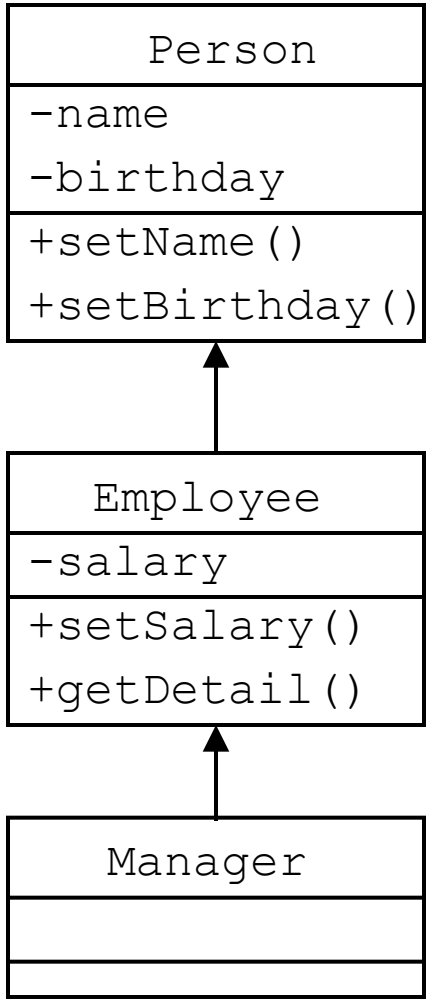
- *Giáo trình lập trình HĐT*, chương 7, 8

Polymorphism (đa hình) là gì

- Polymorphism: nhiều hình thức, nhiều kiểu tồn tại
- Đa hình trong lập trình
  - đa hình hàm: hàm trùng tên, phân biệt bởi danh sách tham số
  - đa hình đối tượng
    - nhìn nhận đối tượng theo nhiều kiểu khác nhau
    - các đối tượng khác nhau giải nghĩa thông điệp theo cách thức khác nhau

Chuyển kiểu lên: Up casting

- Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở
  - Dùng đối tượng của lớp dẫn xuất để truyền tham số
  - Dùng đối tượng của lớp dẫn xuất làm thuộc tính
  - *Quản lý thống nhất, đảm bảo tính mở cho các lớp dẫn xuất mới*




```
Person p;
Employee e = new Employee();
p = (Person) e;
p.setName(...);
p.setSalary(...);  // compile error
e.setSalary(…)
```

Giả sử lớp Person có phương thức getDetail(). Nếu gọi p.getDetail() thì phương thức của lớp nào được gọi?

```
String teamInfo(Person p1, Person p2) {
    return "Leader: " + p1.getName() +
           "; member: " + p2.getName();
}
...
Employee e1, e2;
Manager  m1, m2;
…
System.out.println(teamInfo(e1, e2));
teamInfo(m1, m2);
teamInfo(m1,e2);
```

```
class Manager extends Employee {
    Employee assistant;
...
    public void setAssistant(Employee e) {
        assistant = e;
    }
...
}
...
Manager junior, senior;
...
senior.setAssistant(junior);
```

Đa hình và liên kết động

- Khả năng giải nghĩa các thông điệp theo các cách thức khác nhau

```
Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
...
System.out.println(p1.getDetail());
System.out.println(p2.getDetail());
System.out.println(p3.getDetail());
```

```
class EmployeeList {
    Employee list[];
...
    public void add(Employee e) {...}
    public void print() {
        for (int i=0; i<list.length; i++) {
            System.out.println(list[i].getDetail());
...
}
...
EmployeeList list = new EmployeeList();
Employee e1; Manager m1;
...
list.add(e1); list.add(m1);
list.print();
```

Liên kết tĩnh và liên kết động

Static and dynamic binding

- Liên kết tĩnh: lời gọi hàm (phương thức) được quyết định khi biên dịch, do đó chỉ có một phiên bản của chương trình con được thực hiện
  - ưu điểm về tốc độ
- Liên kết động: lời gọi phương thức được quyết định khi thực hiện, phiên bản của phương thức phù hợp với đối tượng được gọi
  - Java mặc định sử dụng liên kết động
  - liên kết tĩnh: final / private method

Đa hình: Gọi phương thức trong constructor

```
class Shape {
   public Shape() {
            draw();
   }
   public void draw() {}
}

class Point extends Shape {
    protected int x, y;
    public Point(int xx, int yy) {
            x = xx;  y = yy;
    }
    public void draw() {
            System.out.println("(" + x + "," + y + ")");
    }
}
--
Point p = new Point(10, 10);
p.draw();
```

Đa hình: private method

```
class Base {
  private void f() { System.out.println(”base f()”); }
  public void show() { f(); }
}

public class Derived extends Base {
  public void f() {
    System.out.println(”derived f()”);
  }

  public static void main(String args[]) {
    Derived d = new Derived();
    Base b = d;
    b.show();
   d.show();
  }
}
```

Chuyển kiểu xuống: down casting

```
Employee e = new Employee();
Person p = (Person)e;  // up casting
Employee ee = (Employee)p;  // down casting
Manager m = (Manager)ee;  //  run-time error

Person p2 = new Manager();
Employee e2 = (Employee) p2;
```

Toán tử `instanceof`

```
public class Employee extends Person {}
public class Student extends Person {}
---
public doSomething(Person e) {
    if (e instanceof Employee) {...
    } else if (e instanceof Student) {...
    } else {...}
}
```

Đa hình: Tổng kết

- Là một trong ba đặc điểm quan trọng của lập trình hướng đối tượng
- Cho phép nhìn nhận đối tượng theo các cách khác nhau, giải nghĩa thông điệp theo các cách thức khác nhau
- Dựa trên cơ chế chuyển kiểu và liên kết động
- Tạo sự mềm dẻo và tính mở cho thiết kế phần mềm
- Tăng khả năng sử dụng lại

Lớp trừu tượng

- Chúng ta có thể tạo ra các lớp cơ sở để tái sử dụng mà không muốn tạo ra đối tượng thực của lớp
  - các lớp Point, Circle, Rectangle chung nhau khái niệm cùng là hình vẽ *Shape*
- Giải pháp là khai báo lớp trừu tượng
  - không thể tạo đối tượng

```
abstract class Shape {
 protected int x, y;
 Shape(int _x, int _y) {
     x = _x;
     y = _y;
 }
}

class Circle extends Shape {...}

Shape s = new Shape(10, 10) //  compile error
Shape s1 = new Circle();
```

```
class Circle extends Shape {
  int r;

  public Circle(int _x, int _y, int _r) {
      super(_x, _y);
      r = _r;
  }
...
}
```

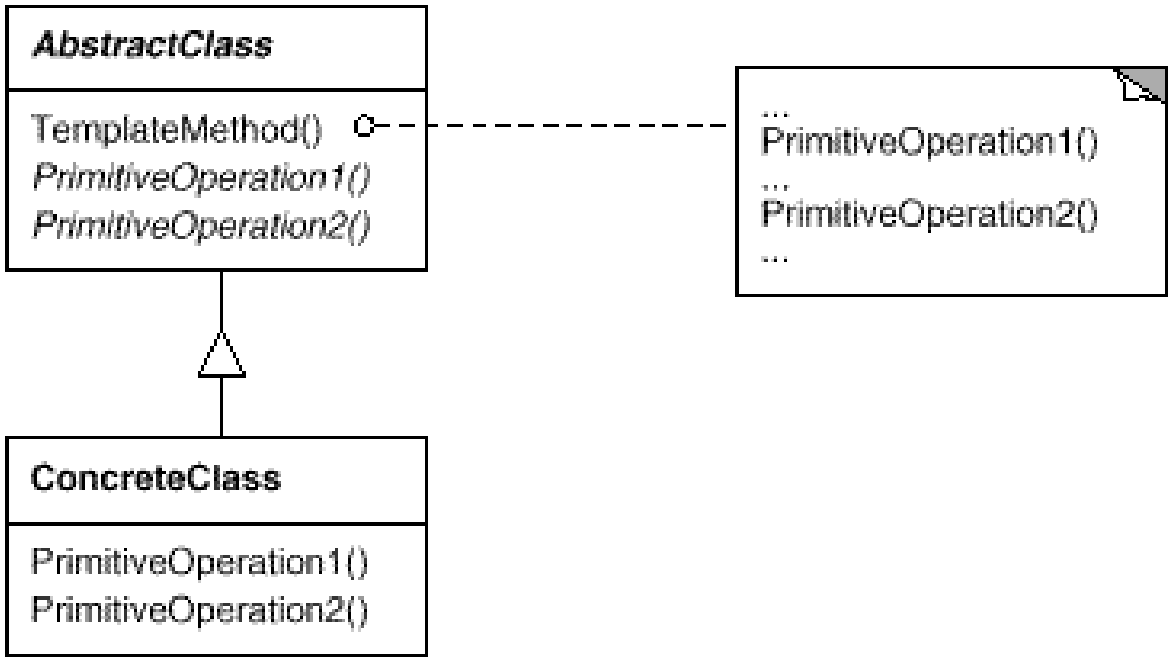
Phương thức trừu tượng

- Để thống nhất giao diện, có thể khai báo các phương thức tại lớp cơ sở nhưng được cài đặt thực tế tại lớp dẫn xuất
  - Các lớp dẫn xuất khác nhau có cách cài đặt khác nhau
- Phương thức trừu tượng
  - Là phương thức bắt buộc phải định nghĩa lại (chuyên biệt hóa) tại lớp dẫn xuất

```
abstract class Shape {
 protected int x, y;

 abstract public void erase();
 abstract public void draw();

  public void moveTo(int x1, int y1) {
     x=x1;
      y=y1;
      erase();
      draw();
   }
}
```

```
class Circle extends Shape {
  int r;

  public Circle(int _x, int _y, int _r) {
        super(_x, _y);
        r = _r;
        draw();
  }
  public void erase() {
        System.out.println("Erase at (" + x + "," + y + ")");
  }

  public void draw() {
        System.out.println("Draw at (" + x + "," + y + ")");
  }
}
```

Template method




```
                                    l, int y1) {
            x = x1;
            y = y1;
            draw();
        }

        abstract public void erase();
        abstract public void draw();
    }
```

Giao diện (Interface)

- Interface là mức trừu tượng cao hơn lớp trừu tượng
- Chỉ bao gồm
  - phương thức trừu tượng
  - hằng số (static final)
  - mặc định là public
- Cú pháp:
  - từ khóa `interface` và `implements`

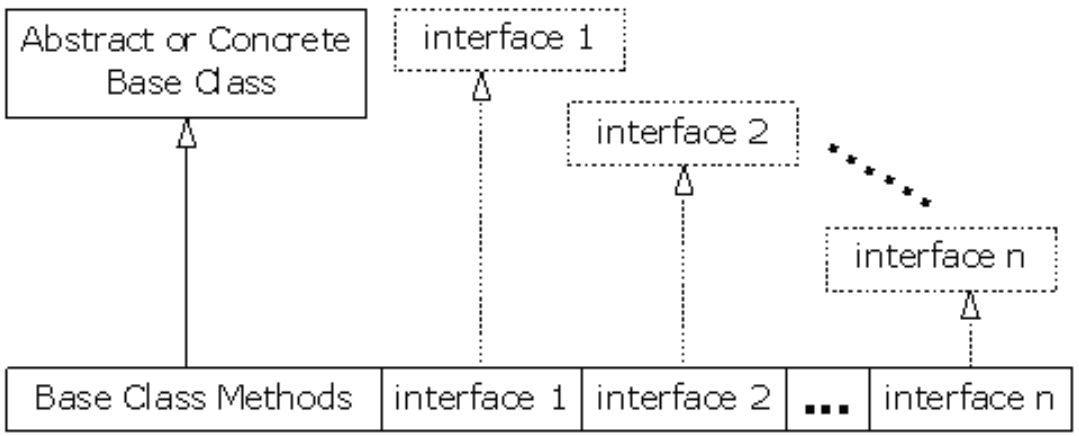
```
interface Action {
  void moveTo(int x, int y);
  void erase();
  void draw();
}

class Circle1 implements Action {
  int x, y, r;
  Circle1(int _x, int _y, int _r) { ... }

  public void erase() {...}
  public void draw() {...}
  public void moveTo(int x1, int y1) {...}
}
```

Lớp trừu tượng cài đặt giao diện

```
abstract class Shape implements Action {
  protected int x, y;

  public Shape() {...}
  public Shape(int _x, int _y) {...}

  public void moveTo(int x1, int y1) {
        erase();
        x = x1;
        y = y1;
        draw();
  }
}
```

Đa kế thừa

- Java không cho phép đa kế thừa từ nhiều lớp cơ sở
  - đảm bảo tính dễ hiểu
  - hạn chế xung đột
- Có thể cài đặt đồng thời nhiều giao diện



```
class ImageBuffer {
...
}

class Animation extends ImageBuffer
 implements Action {
...
    public void erase() {...}
    public void draw() {...}
    public void moveTo() {...}
}
```

```
interface CanFight {
    void fight();
}
interface CanSwim {
    void swim();
}
interface CanFly {
    void fly();
}
class ActionCharacter {
    public void fight() {}
}
```

```
class Hero extends ActionCharacter implements CanFight, CanSwim,
  CanFly {
    public void swim() {}
    public void fly() {}
}

public class Adventure {
    public static void t(CanFight x) { x.fight(); }
    public static void u(CanSwim x) { x.swim(); }
    public static void v(CanFly x) { x.fly(); }
    public static void w(ActionCharacter x) { x.fight(); }
    public static void main(String[] args) {
        Hero mario = new Hero();
        t(mario); // Treat it as a CanFight
        u(mario); // Treat it as a CanSwim
        v(mario); // Treat it as a CanFly
        w(mario); // Treat it as an ActionCharacter
    }
}
```

Mở rộng lớp trừu tượng và giao diện

```
interface I1 {...}

interface I2 {...}

interface I3 extends I1, I2 {...}

abstract class A1 {...}

abstract class A2 extends A1 implements I1, I2 {...}
```

Xung đột (1)

```
interface I1 { void f(); }
interface I2 { int f(int i); }
interface I3 { int f(); }
class C {
    public int f() { return 1; }
}
class C2 implements I1, I2 {
    public void f() {}
    public int f(int i) { return 1; } // overloaded
}
class C3 extends C implements I2 {
    public int f(int i) { return 1; } // overloaded
}
```

Xung đột (2)

```
class C4 extends C implements I3 {
    // Identical, no problem:
    public int f() { return 1; }
}

class C5 extends C implements I1 {}

interface I4 extends I1, I3 {}
```

Abstract class vs. Interface

- Lớp trừu tượng có thể có phương thức thực và thuộc tính
- Interface hỗ trợ đa kế thừa

Sao chép đối tượng

- Có nhu cầu sao chép các đối tượng
  - Sao chép khi truyền tham số để tránh sửa đổi đối tượng gốc
- Làm thế nào để sao chép đối tượng mà không biết rõ kiểu (lớp) thực sự của nó?
  - Sử dụng copy constructor?
  - Sử dụng phương thức copy?
    - `Interface Cloneable` và phương thức `clone()`

Copy constructor

```
class Base {
  int m;
  public Base() { m = 0; }
  public Base(Base b) { m = b.m; }
  public String print() { return "base class"; }
}

class Derived extends Base {
  int n;
  public Derived() { n = 1; }
  public Derived(Derived d) {
        super(d);
        n = d.n;
  }
  public String print() { return "derived class"; }
}
```

Sao chép sử dụng copy constructor

```
public class TestCopy {
  static void copy(Derived d) {
        Derived d1 = new Derived(d);
        System.out.println(d1.print());
  }
  static void copy2(Base b) {
        Base b1 = new Base(b);
        System.out.println(b1.print());
  }
  public static void main(String args[]) {

      Derived d = new Derived();
      copy(d);
      copy2(d);
  }
}
```

Phương thức clone()

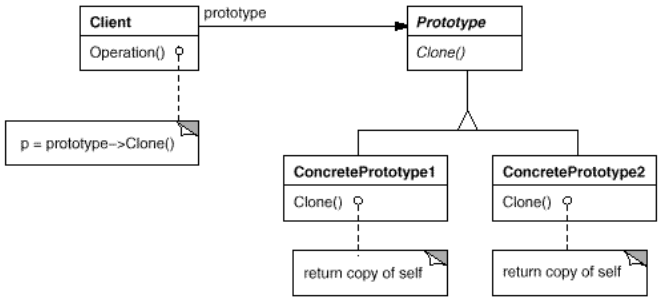
```
class Base {
  int m;
 public Base() { m = 0; }
 public Base(Base b) { m = b.m; }
 public String print() {
    return "base class";
  }
  public Base clone() {
    return new Base(this);
  }
}
```

```
class Derived extends Base {
  int n;
  public Derived() { n = 1; }
  public Derived(Derived d) {
    super(d);//cần hay không?
    n = d.n;
  }
  public String print() {
     return "derived class";
   }
   public Derived clone() {
     return new Derived(this);
   }
}
```

Local copy sử dụng clone()

```
public class TestCopy {
  static void copy(Derived d) {
        Derived d1 = d.clone();
        System.out.println(d1.print());
  }
  static void copy2(Base b) {
        Base b1 = b.clone();
        System.out.println(b1.print());
  }
  public static void main(String args[]) {
      Derived d = new Derived();
      copy(d);
      copy2(d);
  }
}
```

Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Kiểm tra

Lập trình cho ứng dụng đồ họa sau: Một sơ đồ chứa danh sách hình vẽ. Hình vẽ có thể là hình chữ nhật, tam giác, hoặc hình tròn. Các hình vẽ có thể được vẽ, di chuyển, hay xóa khỏi sơ đồ.

Tính năng phức tạp: xóa các hình trùng nhau trong sơ đồ

Bài tập: Stack làm việc với mọi đối tượng

- Hãy cài đặt lớp MyStack làm việc được với mọi đối tượng.