



ĐA HÌNH (Polymorphism)

Trường ĐH Công nghệ, ĐHQG Hà Nội

- Đa hình
 - Khái niệm đa hình – polymorphism
 - Chuyển kiểu
 - Liên kết tĩnh, liên kết động
- Lớp trừu tượng, phương thức trừu tượng
- Giao diện (interface)

Tài liệu tham khảo



- *Giáo trình Lập trình HĐT, Chương 7,8*
- *Java How to Program, Chương 9*
- *Thinking in Java, Chương 7, 8*

Khái niệm đa hình - polymorphism



- Kiểu dữ liệu tham chiếu
 - Tham chiếu trỏ đến nhiều loại đối tượng khác nhau
 - Tham chiếu có thể *trỏ đến đối tượng là dẫn xuất* của kiểu dữ liệu được khai báo cho tham chiếu đó
- Thông điệp được gọi
 - Liên kết động – *dynamic binding*: thông điệp được gọi xác định “*động*”, tùy thuộc vào loại đối tượng được tạo ra.
 - Cơ chế giải nghĩa thông điệp: liên kết động, không phải luôn luôn thực hiện phương thức theo kiểu dữ liệu đã khai báo

Đa hình – kiểu đối tượng tham chiếu trở tới

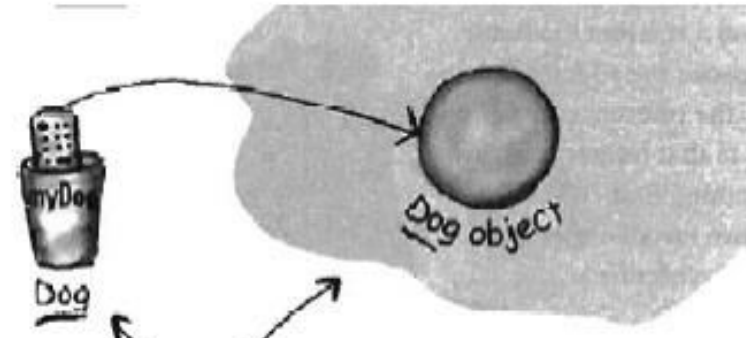
- Thông thường

`Dog d = new Dog();`

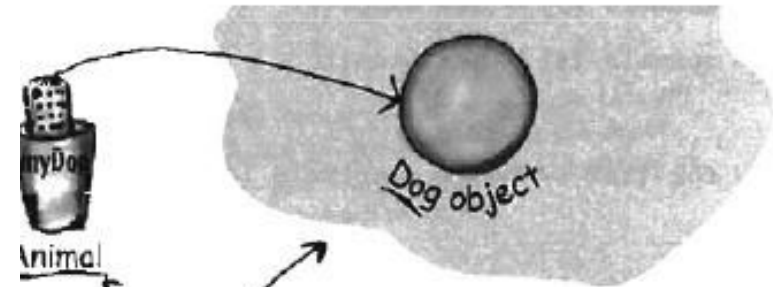
- Đa hình cho phép

`Animal d = new Dog();`

→ Tham chiếu có thể trở đến đối tượng là dẫn xuất



These two are the same type. The reference variable type is declared as `Dog`, and the object is created as `new Dog()`.



These two are **NOT** the same type. The reference variable type is declared as `Animal`, but the object is created as `new Dog()`.

Đa hình – mảng các đối tượng khác nhau



- Một mảng khai báo kiểu Animal nhưng có thể *chứa các đối tượng là dẫn xuất* của Animal

```
Animal [] animals = new Animal[5];

Animals[0] = new Dog();
Animals[1] = new Cat();
Animals[2] = new Lion();
Animals[3] = new Hippo();
Animals[4] = new Wolf();

for (int i=0; i< animals.length; i++)
    animals[i].makeNoise();

//Dynamic binding: makeNoise() of each object Dog, Cat,
//Lion, Hippo, Wolf is executed.
```

Không cần biết trong mảng có những đối tượng loại nào...

Đa hình – tham số của hàm

- Tham số kiểu Animal nhưng có thể *truyền vào tham chiếu đối tượng là dẫn xuất* của Animal

```
class Vet {  
    ...  
    public void giveShot (Animal a){  
        a.makeNoise();  
    }  
}  
...  
  
Vet v = new Vet ();  
Dog d = new Dog();  
Cat c = new Cat();  
v.giveShot (d);    //Dog makeNoise()  
v.giveShot (c);    //Cat makeNoise()
```

Đa hình – giải nghĩa thông điệp động



```
class Animal {  
    String name;  
    ...  
    public void makeNoise () {  
        System.out.println("Animal noise...");  
    }  
    public void introduce () {  
        makeNoise();  
        System.out.println("I am "+ name);  
    }  
}  
class Cat extends Animal {  
    public void makeNoise () {  
        System.out.println("Meo meo...");  
    }  
}  
class Dog extends Animal {  
    public void makeNoise () {  
        System.out.println("Gau gau...");  
    }  
}
```

```
Animal pet1, pet2;  
pet1 = new Cat("Tom cat");  
pet2 = new Dog("Lulu dog");  
pet1.introduce();  
pet2.introduce();
```

```
//Meo meo... I am Tom cat  
//Gau gau... I am Lulu dog
```

`private void makeNoise()`?
→ Liên kết tĩnh – static binding

```
//Animal noise... I am Tom cat  
//Animal noise... I am Lulu dog
```


Đa hình – ưu điểm



- Viết mã chương trình một lần
 - Áp dụng cho nhiều kiểu đối tượng

Liên kết động – dynamic binding

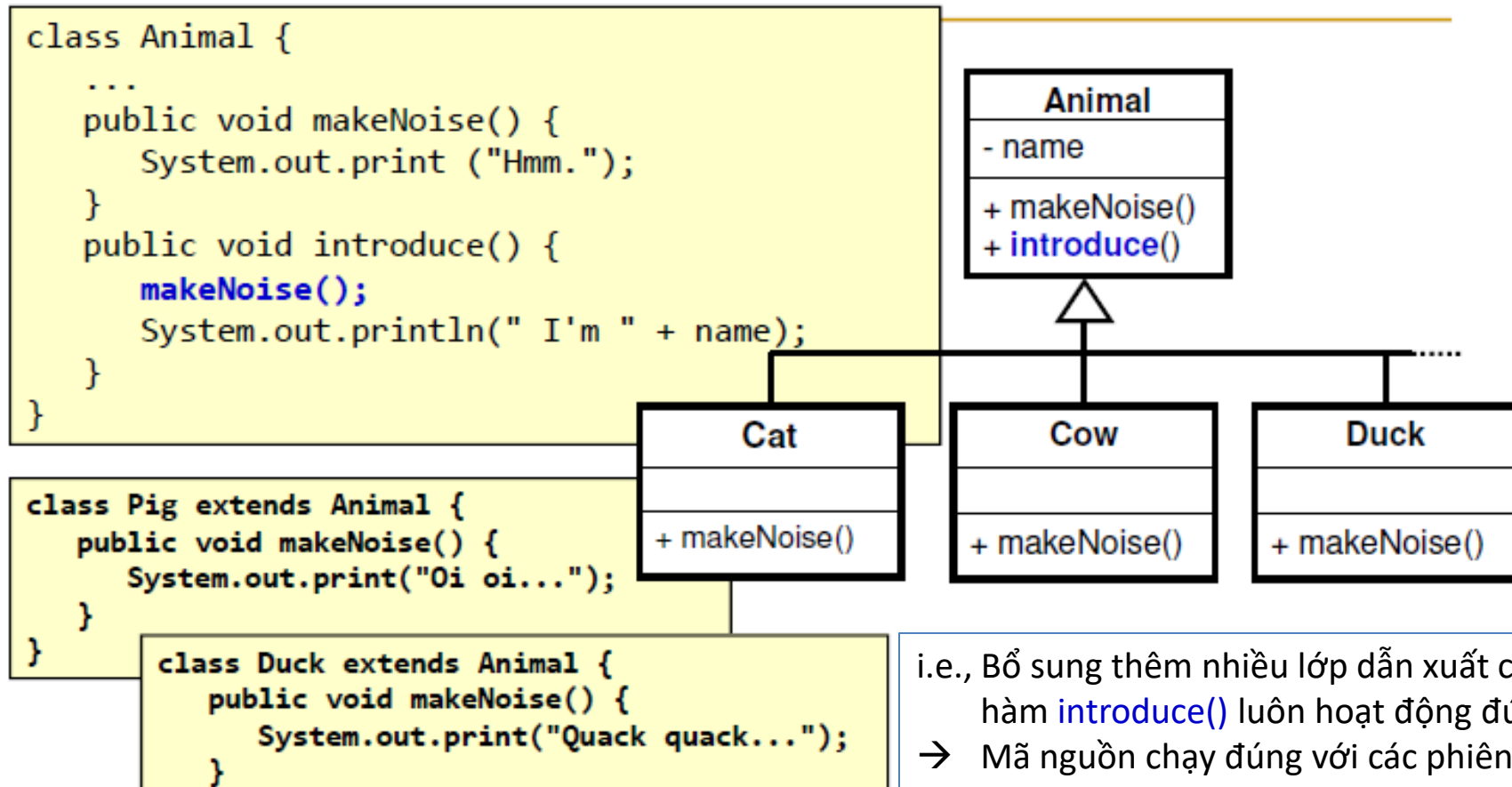
- Không phải thay đổi mã nguồn khi tạo ra các lớp dẫn xuất mới

Nâng cấp phiên bản, kiến trúc phần mềm linh hoạt,...

```
class Vet {  
    ...  
    public void giveShot (Animal a){  
        a.makeNoise();  
    }  
}  
Vet v = new Vet ();  
Dog d = new Dog();  
Cat c = new Cat();  
v.giveShot (d);    //Dog makeNoise()  
v.giveShot (c);    //Cat makeNoise()
```

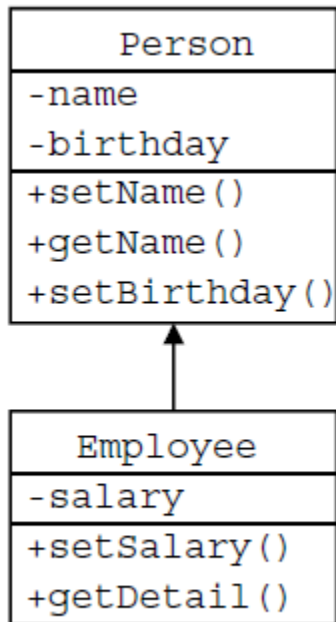
```
Animal [] animals = new Animal[5];  
...  
for (int i=0; i< animals.length; i++)  
    animals[i].makeNoise();
```

Đa hình – ưu điểm



- i.e., Bổ sung thêm nhiều lớp dẫn xuất của Animal mà hàm `introduce()` luôn hoạt động đúng
- Mã nguồn chạy đúng với các phiên bản nâng cấp trong tương lai.
 - Cho phép xây dựng các hàm có tính chất khuôn mẫu.

Chuyển kiểu lên - upCasting

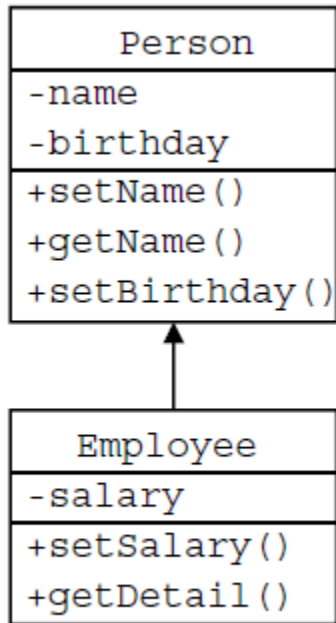


```
Person p;
Employee e = new Employee();
p = (Person) e;           //explicit up casting
p.setName (...);
p.setSalary(...);        //compile error
```

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

→ Sự khác nhau giữa explicit up casting và implicit up casting?

Chuyển kiểu xuống - downCasting



```
Person p;  
Employee e = new Employee();  
p = (Person) e;           //explicit up casting  
p.setName (...);  
p.setSalary(...); //compile error  
Employee ee = (Employee) p; //down casting  
Manager m = (Manager) ee;  //compile error
```

```
Person p = new Manager();  
Employee e = (Employee) p; //up or down casting?
```

→ Chỉ down casting trên những tham chiếu đối tượng đã up casting

Liên kết tĩnh, liên kết động

- Liên kết tĩnh – **static binding**: lời gọi phương thức được quyết định khi biên dịch
 - Ưu điểm về **tốc độ**
 - **final**, **private** method
- Liên kết động – **dynamic binding**: lời gọi phương thức được quyết định khi thực hiện (executed), phiên bản của phương thức phù hợp với đối tượng được gọi
 - Java mặc định **áp dụng liên kết động** (ngoại trừ **final/private method**)

không phụ thuộc vào kiểu dữ liệu khi khai báo,
phương thức được định nghĩa ở đâu,...

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);  
-----  
(10,10)
```

private void draw() {}

Đa hình – private method

```
class Base {  
    private void f() {  
        System.out.println("base f()");  
    }  
    public void show() { f(); }  
}  
public class Derived extends Base {  
    private void f() {  
        System.out.println("derived f()");  
    }  
    public static void main (String args[]) {  
        Derived d = new Derived();  
        Base b = d;  
        b.show();  
    }  
}
```

Liên kết tĩnh → "base f()"

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 {...}  
    }  
}
```


Lớp trừu tượng – abstract class

- Có thể tạo ra các lớp cơ sở (khuôn mẫu) để các lớp dẫn xuất kế thừa mà **không tạo ra các đối tượng** thực của lớp
 - i.e., lớp Dog, Cat, Cow,... kế thừa từ lớp **Animal**; Rectangle, Circle,... kế thừa từ lớp **Shape**
- Khai báo lớp trừu tượng – abstract class
 - Không thể tạo ra đối tượng
 - Xây dựng lớp khuôn mẫu với các thuộc tính và hành vi mà các lớp dẫn xuất bắt buộc phải có

Thực tế không có đối tượng nào là Animal, Shape,...

```
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();
```

Phương thức trừu tượng

- Để thống nhất giao diện, có thể khai báo các phương thức trừu tượng (**abstract method**) tại các lớp cơ sở và cài đặt chi tiết tại các lớp dẫn xuất.
 - Các lớp dẫn xuất cài đặt các phiên bản khác nhau của phương thức trừu tượng được kế thừa.
- Phương thức trừu tượng
 - Bắt buộc phải định nghĩa lại 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) {...}  
}
```

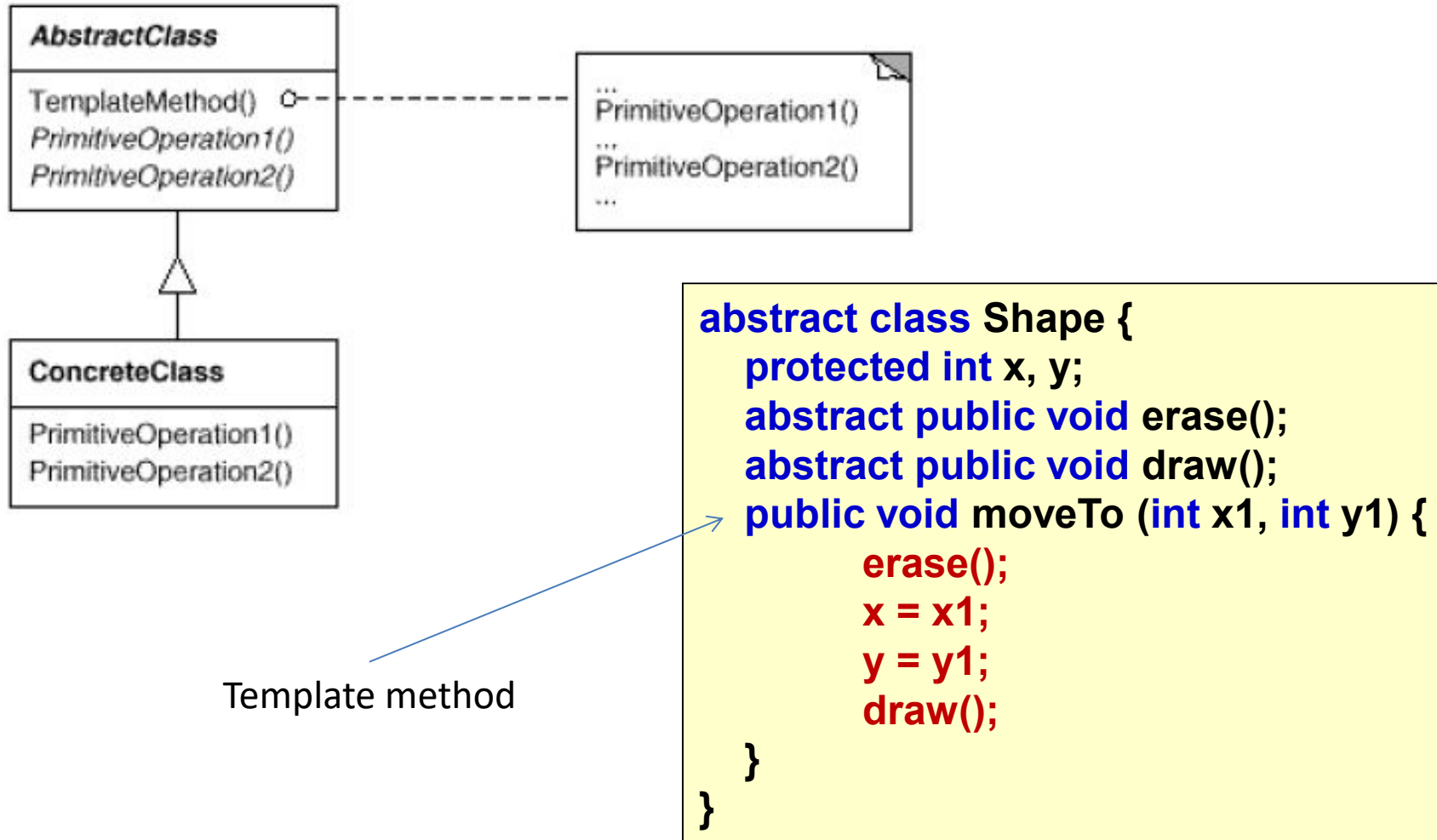
Phương thức trừu tượng

```
abstract class Shape {  
    protected int x, y;  
    abstract public void erase();  
    abstract public void draw();  
    public void moveTo (int x1, int  
y1) {...}  
}
```

```
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 + ")");  
    }  
}
```

- Phương thức trừu tượng khai báo trong lớp **trừu tượng**.
- Lớp trừu tượng có **phương thức thường** và phương thức trừu tượng.
- Phải định nghĩa phương thức trừu tượng trong lớp dẫn xuất.

Phương thức khuôn mẫu - template method



Giao diện - interface

- **interface** là mức trừu tượng cao
- **interface** bao gồm:
 - Phương thức trừu tượng – abstract method
 - Hằng số: mặc định là **public static final**
 - Mặc định là public
- Từ khóa: **interface** và **implements**

```
interface Config {  
    int MAX_CONN = 20;  
    void display(); //it will be public  
}  
...  
class Control implements Config  
{...}
```

Lớp cài đặt giao diện



Lớp thường cài đặt giao diện

```
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();  
    }  
    //public void erase() {...}  
    //public void draw() {...}  
}
```

Các giao diện kế thừa



```
interface I1 {...}
interface I2 {...}
interface I3 extends I1, I2 {...}
//Giao diện kế thừa giao diện

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

Xung đột khi cài đặt giao diện

```
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
}
class C4 extends C implements I3 {
    public int f() { return 1; }
    // Identical, no problem:
}
class C5 extends C implements I1 {}
interface I4 extends I1, I3 {}
```

Giao diện khai báo như 1 kiểu dữ liệu



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

Truyền vào hàm các tham
chiếu đối tượng có cài đặt
giao diện khai báo

```
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 h = new Hero();  
        t(h); // Treat it as a CanFight  
        u(h); // Treat it as a CanSwim  
        v(h); // Treat it as a CanFly  
        w(h); //Treat it as an ActionCharacter  
    }  
}
```


- Đa hình
 - Khái niệm đa hình – polymorphism
 - Chuyển kiểu
 - Liên kết tĩnh, liên kết động
- Lớp trừu tượng, phương thức trừu tượng
- Giao diện (interface)