

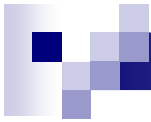


Lập trình tổng quát



Nội dung

- Khái niệm
- Lớp tổng quát
- Phương thức tổng quát



Tài liệu tham khảo

- *Giáo trình Lập trình HĐT*, chương 13
- *Java how to program*, chapter 18

Vấn đề

- Nhiều giải thuật về cơ bản không phụ thuộc vào kiểu dữ liệu cụ thể, ví dụ: sắp xếp, tìm kiếm,...
- Nhiều cấu trúc dữ liệu cũng không phụ thuộc vào kiểu dữ liệu thành viên cụ thể, ví dụ Ngăn xếp, danh sách liên kết,...
- Xuất hiện nhu cầu sử dụng lại “mã chương trình” cho nhiều kiểu dữ liệu khác nhau

➡ Tổng quát hóa

Giải pháp sử dụng kế thừa

- Một giải pháp là sử dụng kế thừa
 - Các lớp đều kế thừa từ lớp Object
 - Đối tượng được chuyển kiểu lên thành kiểu Object

```
public class MyList { // items could be objects of any classes
    public void add(Object o) {...}
    public Object getFirst() {...}
    ...
}
```



Hạn chế

■ Luôn phải chuyển kiểu

```
MyList myPets = new MyList();  
...  
Animal a = (Animal) myPets.getFirst();
```


■ Không có cơ chế kiểm tra lỗi

```
myPets.add(new Integer(3));  
...  
Animal a = (Animal) myPets.getFirst();
```

Giải pháp: lớp tổng quát

- Từ Java 6 cung cấp cơ chế lớp tổng quát
Generic class
 - Các kiểu đối tượng được tham số hóa
 - Hầu hết các thư viện của Java được tổng quát hóa

```
ArrayList<Cat> myPets = new ArrayList<Cat>();  
myPets.add(new Cat());  
myPets.add(new Cat());  
myPets.add(new Dog()); // compile-time error! type mismatch  
Cat cat = myPets.get(0); // no cast required! . . .
```



Ví dụ (tiếp)

```
List myList = new LinkedList();  
myList.add(new Integer(0));  
Integer x = (Integer) myList.iterator().next();
```

```
List<Integer> myList = new LinkedList<Integer>();  
myList.add(new Integer(0));  
Integer x = myList.iterator().next();
```

```
myList.add(new Long(0));
```




Tự tạo lớp tổng quát

```
public class Pair<T>
{
    public Pair() { first = null; second = null; }
    public Pair(T first, T second)
        { this.first = first; this.second = second; }

    public T getFirst() { return first; }
    public T getSecond() { return second; }

    public void setFirst(T newValue) { first = newValue; }
    public void setSecond(T newValue) { second = newValue; }

    private T first;
    private T second;
}
```

```
...
Pair<String> mm = new Pair<String> ("1st", "2nd");
System.out.println(mm.getFirst() + "," + mm.getSecond());
```

Thêm kiểu

```
public class Pair<T, U>
{
    public Pair() { first = null; second = null; }
    public Pair(T first, U second)
        { this.first = first; this.second = second; }

    public T getFirst() { return first; }
    public U getSecond() { return second; }

    public void setFirst(T newValue) { first = newValue; }
    public void setSecond(U newValue) { second = newValue; }

    private T first;
    private U second;
}
```

```
...
Pair<String, Integer> mm =
    new Pair<String, Integer> ("1st", 1);
System.out.println(mm.getFirst() + "," + mm.getSecond());
```

Phương thức tổng quát

- Có thể tham số hóa kiểu ở mức phương thức
 - Khai báo trong lớp tổng quát hoặc ngay trong lớp thông thường

```
class ArrayAlg {  
    public static <T> T getMiddle(T[] a)  
        { return a[a.length / 2]; }  
}
```

```
String[] names = { "John", "Q.", "Public" };  
String middle = ArrayAlg.<String>getMiddle(names);
```


Ràng buộc về kiểu

```
class ArrayAlg {  
    public static <T> T min(T[] a) // almost correct  
    {  
        if (a == null || a.length == 0) return null;  
        T smallest = a[0];  
        for (int i = 1; i < a.length; i++)  
            if (smallest.compareTo(a[i]) > 0)  
                smallest = a[i];  
        return smallest;  
    }  
}
```

Nếu lớp **T** không cài đặt **compareTo()** thì sao?

➡ Cần ràng buộc T có phương thức compareTo

```
...  
public static <T extends Comparable> T min(T[] a)  
//the rest is the same as before
```



Ràng buộc về kiểu

Syntax: `<T extends BoundingType>`

- T và BoundingType có thể là giao diện (interface) hoặc lớp
 - T là kiểu con (*subtype*) của BoundingType
- Có thể có nhiều ràng buộc (BoundingType)
`<T extends Comparable & Serializable>`

```
class ArrayAlg
```

```
{
```

```
    /**
```

```
        Gets the minimum and maximum of an array of objects of type T.
```

```
        @param a an array of objects of type T
```

```
        @return a pair with the min and max value,  
        or null if a is null or empty
```

```
    */
```

```
    public static <T extends Comparable> Pair<T> minmax(T[] a)
```

```
    {
```

```
        if (a == null || a.length == 0) return null;
```

```
        T min = a[0];
```

```
        T max = a[0];
```

```
        for (int i = 1; i < a.length; i++)
```

```
        {
```

```
            if (min.compareTo(a[i]) > 0) min = a[i];
```

```
            if (max.compareTo(a[i]) < 0) max = a[i];
```

```
        }
```

```
        return new Pair<T>(min, max);
```

```
    }
```

```
}
```

```
...
```

```
String[] words = { "Mary", "had", "a", "little", "lamb" };
```

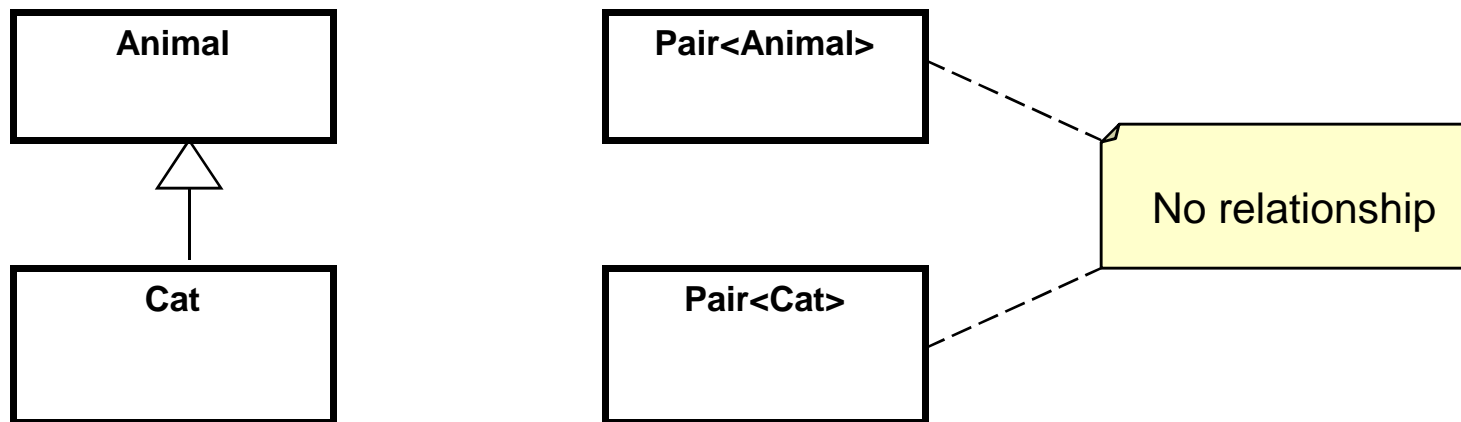
```
Pair<String> mm = ArrayAlg.minmax(words);
```

```
System.out.println("min = " + mm.getFirst());
```

```
System.out.println("max = " + mm.getSecond());
```

Kế thừa và tổng quát hóa

- Không có mối quan hệ kế thừa trong tổng quát hóa!



Sử dụng ký tự đại diện (wildcards)

- Tạo một kiểu Pair có thể làm việc với một số kiểu con (subtype) của Animal như sau:
 - `Pair<? extends Animal> aPair = ...;`

```
...  
Pair<Cat> catBuddies = new Pair<Cat>(tomCat, timCat);  
Pair<? extends Animal> wildcardBuddies = catBuddies; // OK  
Animal someAnimal = new Cow();  
wildcardBuddies.setFirst(someAnimal); // compile-time error
```