Lập trình tổng quát

Bộ môn công nghệ phần mềm
Khoa công nghệ thông tin
Trường ĐHCN, ĐHQG Hà Nội

Nội dung

- Định nghĩa
- Lớp tổng quát
- Giao diện tổng quát
- Phương thức tổng quát
- Khởi tạo kiểu tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của lập trình tổng quát

Tài liệu tham khảo

- *Giáo trình Lập trình HĐT*, chương 13
- *Java how to program*, chapter 18
- Java documentation https://docs.oracle.com/javase/tutorial/java/generics/types.html

Định nghĩa

Định nghĩa

- Lập trình tổng quát (generic programming) là một dạng lập trình máy tính trong đó:
  - (i) Khi định nghĩa lớp/giao diện/phương thức, kiểu dữ liệu không cần xác định tường minh
  - (ii) Khi sử dụng lớp/giao diện/phương thức, những kiểu dữ liệu không tường minh sẽ phải xác định rõ kiểu

Định nghĩa

- Để cho đơn giản
  - Lớp/giao diện/phương thức được lập trình tổng quát gọi tắt là lớp/giao diện/phương thức tổng quát
  - Kiểu dữ liệu không tường minh trong lớp/giao diện/phương thức tổng quát gọi tắt là kiểu tổng quát
- Java hỗ trợ 3 loại tổng quát
  - Lớp tổng quát (generic class)
  - Giao diện tổng quát (generic interface)
  - Phương thức tổng quát (generic method)

Lớp tổng quát

Cấu trúc

- Lớp tổng quát được viết với cấu trúc như sau:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

, trong đó

  - `T1, T2, ..., Tn` là các kiểu tổng quát
  - Kí hiệu <> được sử dụng để bọc lấy các kiểu tổng quát
- Khai báo/Khởi tạo một đối tượng thuộc lớp tổng quát tương tự như các lớp không tổng quát, điểm khác biệt duy nhất là phải xác định tường minh kiểu tổng quát

Quy ước đặt tên kiểu tổng quát

- E: Element
- K, V: Key, Value
- N: Number
- T: Type
- S, U, V v..v: kiểu thứ 2, thứ 3, thứ 4, v.v.

Ví dụ

Ví dụ 1: Tạo một lớp tổng quát `Box`, lớp này có một kiểu tổng quát `T`

```
public class Box<T> {
   private T t;
   public void set(T t) { this.t = t; }
   public T get() { return t; }
}
```

Khai báo và khởi tạo một đối tượng thuộc lớp tổng quát `Box`, trong đó kiểu tổng quát `T` là lớp `Integer`:

```
Box<Integer> integerBox = new
        Box<Integer>();
```

Ví dụ

Ví dụ 2: Tạo lớp tổng quát `Pair` lưu `key` và `value`, trong đó `key` và `value` là hai kiểu tổng quát

```
public class Pair<K, V> {
   private K key;
   private V value;
   public void setKey(K key){ this.key = key;}
   public K getKey(){ return key;}
}
```

Khai báo và khởi tạo, trong đó `key` có kiểu `String`, `value` có kiểu `Integer`

```
Pair <String, Integer> p1 =
new Pair <String, Integer>("Even", 8);
```

Thừa kế lớp tổng quát

Các cách để thừa kế lớp tổng quát

- Chỉ định rõ kiểu tổng quát của lớp tổng quát

  Ví dụ: Xác định `K` là `String`, `V` là `Integer` khi định nghĩa lớp `ContactEntry`

```
public class ContactEntry extends Pair <String,
Integer>{ ...}
```

- Chỉ định rõ một phần trong các kiểu tổng quát

  Ví dụ: Xác định `K` là `String` khi định nghĩa lớp `ContactEntry`

```
public class ContactEntry<V> extends Pair
<String, V>{ ...}
```

Thừa kế lớp tổng quát

- Giữ nguyên tất cả các kiểu tổng quát

  Ví dụ:

```
public class ContactEntry<K, V> extends
Pair <K, V>{ ...}
```

- Thêm tham số tổng quát

  Ví dụ: Thêm tham số tổng quát `T`

```
public class ContactEntry<K, V, T>
extends Pair <K, V>{ ...}
```

Giao diện tổng quát

Cấu trúc

- Tương tự như lớp tổng quát, giao diện tổng quát được viết với cấu trúc như sau:

```
interface name<T1, T2, ..., Tn> {
 /* ... */
}
```

, trong đó

  - `T1, T2, ..., Tn` là các kiểu tổng quát
  - Kí hiệu <> được sử dụng để bọc lấy các kiểu tổng quát
- Implement một giao diện tổng quát tương tự như thừa kế một lớp tổng quát

Ví dụ

- Tạo một interface `GenericDao` có kiểu tổng quát `T`

```
public interface GenericDao<T>{
   void insert(T obj);
   void update(T obj);
}
```

- Một class cài đặt từ giao diện trên

```
public abstract class
AbstractGenericDao<T> implements
GenericDao<T>{}
```

Phương thức tổng quát

Cấu trúc

- Phương thức tổng quát được viết với cấu trúc như sau:

```
<T1, T2, ..., Tn> return_type name_func(...){
/* ... */
}
```

, trong đó

  - `T1, T2, ..., Tn` là các kiểu tổng quát
  - Kí hiệu <> được sử dụng để bọc lấy các kiểu tổng quát
  - `return_type` là kiểu trả về của phương thức tổng quát
  - `name_func` là tên của phương thức tổng quát

Tạo phương thức tổng quát

- Phương thức tổng quát có các đặc điểm sau:
  - sử dụng ít nhất một kiểu tổng quát
  - có thể nằm trong lớp/giao diện tổng quát hoặc không
  - có thể là phương thức tĩnh hoặc không tĩnh

Ví dụ

Viết phương thức tổng quát `compare` để so sánh hai đối tượng `p1` và `p2`. Trong đó, `p1` và `p2` đều thuộc lớp tổng quát `Pair`.

Hai đối tượng kiểu `Pair` bằng nhau khi giá trị `value` và khóa `key` đều bằng nhau.

```
// Pair.java
public class Pair<K, V> { private K key; private V value; }
// Utils.java
public class Util {
    public static <K, V> boolean compare(
                        Pair<K, V> p1, Pair<K, V> p2) {
    return p1.getKey().equals(p2.getKey()) &&
    p1.getValue().equals(p2.getValue());
    }
}
```

Ví dụ (tiếp)

```
// UtilTest.java
public static void main(String[] args){
   Pair<Integer, String> p1 = new Pair<>(1, "apple");
   Pair<Integer, String> p2 = new Pair<>(2, "pear");
   boolean same = Util.compare(p1, p2); // False
}
```

Khởi tạo kiểu tổng quát

Khởi tạo kiểu tổng quát

- Hai nhu cầu thường gặp khi khởi tạo kiểu tổng quát
  - Khởi tạo đối tượng có kiểu tổng quát
  - Khởi tạo mảng dữ liệu (trong đó các phần tử có kiểu tổng quát) – gọi là mảng tổng quát

Vấn đề khi khởi tạo đối tượng có kiểu tổng quát

- Không thể tạo trực tiếp đối tượng có kiểu tổng quát bằng cách dùng từ khóa `new`
- Trình biên dịch của Java cần biết rõ kiểu tổng quát T là cái gì mới có thể biên dịch

Ví dụ: Xét lớp tổng quát `Box`

```
public class Box<T> {
    private T t = new T(); //
    lỗi
}
```

Lớp này không thể biên dịch thành công do cố tình khởi tạo một đối tượng có kiểu tổng quát T.

```
$javac Box.java
Box.java:2: error: unexpected type
    private T t = new T();
                      ^
  required: class
  found:    type parameter T
  where T is a type-variable:
    T extends Object declared in class Box
1 error
```

Vấn đề khi khởi tạo mảng tổng quát

- Không thể tạo trực tiếp mảng tổng quát bằng cách dùng từ khóa `new`
- Trình biên dịch của Java cần biết rõ kiểu tổng quát T của các phần tử mảng là cái gì mới có thể biên dịch

Ví dụ: Xét lớp tổng quát Box

```
public class BoxArray<T> {
    private T[] t = new T[5];
    // lỗi
}
```

Lớp này không thể biên dịch thành công do cố tình khởi tạo một mảng tổng quát.

```
$javac BoxArray.java
BoxArray.java:2: error: generic array creation
    private T[] t = new T[5];
                    ^
1 error
```

Giải pháp

- Java hỗ trợ cơ chế reflection để khởi tạo:
  - đối tượng có kiểu tổng quát
  - mảng tổng quát
- Sử dụng phương thức `newInstance()` của lớp `Class` để tạo một đối tượng từ tên đối tượng
  - Tên đối tượng được thể hiện dưới dạng xâu

Ví dụ khởi tạo đối tượng có kiểu tổng quát

```
public class GenericInstance<T> {
    //Khai báo biến obj kiểu T
    private T obj;

    //Sử dụng đối tượng Class<T>, khai báo biến aClazz
    public GenericInstance(Class<T> aClazz)
            throws InstantiationException, IllegalAccessException {
        //Tạo đối tượng thông qua hàm newInstance()
        this.obj = (T) aClazz.newInstance();
    }

    public T getObj() {
        return obj;
    }
}
```

Ví dụ khởi tạo mảng tổng quát

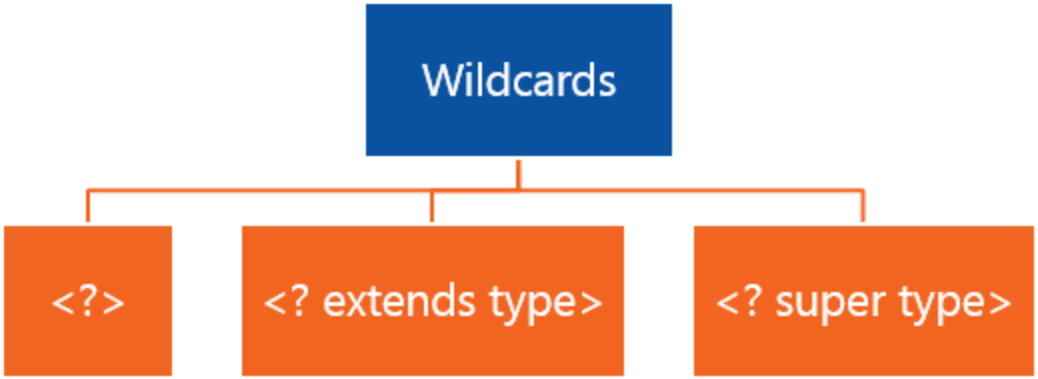
```
public class GenericArrayContructor<T> {
    private final int size = 10;
    private Class<T> aClazz;

    private T[] myArray;

    public GenericArrayContructor(Class<T> aClazz) {
        this.aClazz = aClazz;
        myArray = (T[]) Array.newInstance(aClazz, size);
    }

    public T[] getMyArray() {
        return this.myArray;
    }
}
```

Các ký tự đại diện (Wildcard)

Ký tự đại diện (Wildcard)

- Ký tự (?): đại diện cho một loại (type) chưa xác định
- Kiểu tham số đại diện (wildcard parameterized type)
  - Ít nhất một kiểu tham số là wildcard
    - `Collection<?>`
    - `List<? extends Number>`
    - `Comparator<? super String>`
    - `Pair<String,?>`

Ký tự đại diện (Wildcard)



- `<?>`: chấp nhận tất cả các loại đối số
- `<? extends type>`: chấp nhận các đối tượng kế thừa từ type hoặc chính type
- `<? super type>`: chấp nhận các đối tượng là cha của type hoặc chính type

Ký tự đại diện (Wildcard)

- Không thể khởi tạo một đối tượng thuộc kiểu đại diện
  - Kiểu đại diện không phải là một kiểu dữ liệu cụ thể
  - Không thể sử dụng toán tử `new` để khởi tạo đối tượng

Ký tự đại diện (Wildcard)

- Khai báo hợp lệ

```
Collection<?> coll = new ArrayList<String>();

// Một tập hợp chỉ chứa kiểu Number hoặc kiểu con của Number
List<? extends Number> list = new ArrayList<Long>();

// Một đối tượng có kiểu tham số đại diện.
// (A wildcard parameterized type)
Pair<String,?> pair = new Pair<String,Integer>();
```

- Khai báo không hợp lệ

```
// String không phải là kiểu con của Number, vì vậy lỗi.
List<? extends Number> list = new ArrayList<String>();

// String không phải là kiểu cha của Integer vì vậy lỗi
ArrayList<? super String> cmp = new ArrayList<Integer>();
```

Ví dụ kiểu đại diện

```
public class WildCardExample1 {

    public static void main(String[] args) {

        // Một danh sách chứa các phần tử kiểu String.
        ArrayList<String> listString = new ArrayList<String>();

        listString.add("Tom");
        listString.add("Jerry");

        // Một danh sách chứa các phần tử kiểu Integer
        ArrayList<Integer> listInteger = new ArrayList<Integer>();

        listInteger.add(100);

        // Bạn không thể khai báo:
        // ArrayList<Object> list1 = listString; // ==> Error!

        // Một đối tượng kiểu tham số đại diện.
        // (wildcard parameterized object).
        ArrayList<? extends Object> list2;

        // Bạn có thể khai báo:
        list2 = listString;

        // Hoặc
        list2 = listInteger;

    }

}
```

Ví dụ kiểu đại diện

```
public class WildCardExample2 {

    public static void main(String[] args) {

        List<String> names = new ArrayList<String>();
        names.add("Tom");
        names.add("Jerry");
        names.add("Donald");

        List<Integer> values = new ArrayList<Integer>();
        values.add(100);
        values.add(120);

        System.out.println("--- Names --");
        printElement(names);

        System.out.println("-- Values --");
        printElement(values);

    }

    public static void printElement(List<?> list) {
        for (Object e : list) {
            System.out.println(e);
        }
    }

}
```

Ưu, nhược điểm của lập trình tổng quát

Ưu điểm

- Ưu điểm 1: Giảm thiểu chi phí kiểm thử và bảo trì mã nguồn do không cần viết nhiều chương trình có có thuật toán giống nhau cho các kiểu dữ liệu khác nhau
  - Ví dụ: Viết thuật toán sắp xếp chèn (selection sort) cho mảng lưu các số nguyên `arr1` và mảng lưu các kí tự `arr2`
  - Cách 1 (lập trình không tổng quát): viết hai phương thức sắp xếp riêng biệt cho `arr1` và `arr2`
    - Thu được hai phương thức giống hệt nhau về thuật toán nhưng khác nhau về kiểu dữ liệu
    - Tốn thêm chi phí kiểm thử và bảo trì

Ưu điểm

- Cách 2 (lập trình tổng quát): Nhận thấy thuật toán sắp xếp chèn cho `arr1` và `arr2` là giống hệt nhau
  - Do đó, thay vì viết hai phương thức giống hệt nhau về thuật toán nhưng khác nhau về kiểu dữ liệu, chỉ cần viết một phương thức tổng quát để định nghĩa thuật toán sắp xếp chèn
  - Khi sử dụng thuật toán sắp xếp chèn, kiểu tổng quát sẽ được định nghĩa tường minh
  - Chỉ cần kiểm thử và bảo trì một phương thức tổng quát là đủ

Ưu điểm

- Ưu điểm 2: Trình biên dịch có thể kiểm tra tính đúng đắn của dữ liệu ngay khi biên dịch, từ đó giảm thiểu các lỗi liên quan đến sử dụng sai kiểu dữ liệu khi thực thi
- Ưu điểm 3: Hạn chế việc ép kiểu (cast) thủ công không an toàn

Nhược điểm

- Tham số tổng quát phải là lớp đối tượng (không phải kiểu dữ liệu nguyên thủy)
- Không thể khởi tạo kiểu tổng quát, thay vào đó sử dụng reflection
- Không thể ép kiểu một đối tượng tổng quát
- Không thể sử dụng `instanceof` cho đối tượng tổng quát

```
public static <E> void rtti(List<E> list) {
     if (list instanceof ArrayList<Integer>) {  // compile-time error
     // ...
     }
}

List<Integer> li = new ArrayList<Integer>();
List<Number>  ln = (List<Number>) li;  // compile-time error
```

Kết luận

- Ba thành phần có thể lập trình tổng quát trong Java: lớp, giao diện, và phương thức
- Thừa kế một lớp tổng quát
- Thực thi (implements) một giao diện tổng quát
- Để khởi tạo đối tượng của lớp tổng quát hoặc mảng tổng quát, phải dùng reflection
- Ký tự đại diện
- Ưu điểm, nhược điểm của lập trình tổng quát