Lập trình tổng quát (Generic)

Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

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. VD: 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

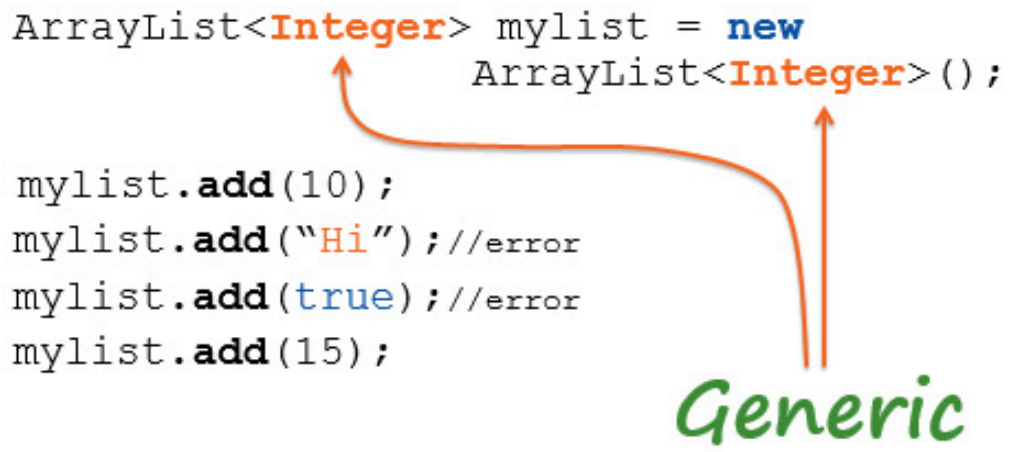
```
MyList   myPets = new MyList();
 . . .
Animal a = (Animal) myPets.getFirst();
```

- 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)
  - Cho phép một object hoạt động với nhiều kiểu dữ liệu khác nhau
  - Hầu hết các thư viện của Java được tổng quát hóa



Một số quy ước

- **Đặt tên kiểu tham số**
  - E: Element
  - K, V: Key, Value
  - N: Number
  - T: Type
- **Ký tự Diamond <>**
  - Gọi hàm khởi tạo của một lớp Generic

```
// Trước Java 7
List<Integer> integerBox = new ArrayList<Integer>();

// Khai báo sử dụng cặp dấu <> từ phiên bản Java 7
List<Integer> integerBox = new ArrayList<>();
```

Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

Tự tạo lớp tổng quát

- Khai báo lớp tổng quát chứa cặp Key-Value

```
public class KeyValuePair<K, V> {
    private K key;
    private V value;

    public KeyValuePair(K key, V value) {
        this.key = key;
        this.value = value;
    }
}
```

- Khi sử dụng, cần xác định kiểu tham số cụ thể cho K, V

```
public class KeyValuePairExample {
    public static void main(String[] args) {
        KeyValuePair<String, Integer> entry =
            new KeyValuePair<String, Integer>("Test Key-Value", 123456789);
        String name = entry.getKey();
        Integer id = entry.getValue();
        // Name = Test Key-Value, Id = 123456789
        System.out.println("Name = " + name + ", Id = " + id);
    }
}
```

Thừa kế lớp tổng quát (1)

- Có thể chỉ định rõ kiểu tham số lớp tổng quát

```
public class ContactEntry extends KeyValuePair<String, Integer> {

    public ContactEntry(String key, Integer value) {
        super(key, value);
    }

}
```

Thừa kế lớp tổng quát (2)

- Chỉ định rõ một phần trong các tham số

```
public class ContactEntry2<V> extends KeyValuePair<String, V> {

    public ContactEntry2(String key, V value) {
        super(key, value);
    }

}
```

Thừa kế lớp tổng quát (3)

- Giữ nguyên tham số lớp tổng quát

```
public class ContactEntry3<K, V> extends KeyValuePair<K, V> {

    public ContactEntry3(K key, V value) {
        super(key, value);
    }

}
```

Thừa kế lớp tổng quát (4)

- Thêm tham số từ lớp tổng quát

```
public class ContactEntry4<K, V, T> extends KeyValuePair<K, V> {

    private T obj;

    public ContactEntry4(K key, V value, T obj) {
        super(key, value);
        this.obj = obj;
    }

    public T getObj() {
        return obj;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }

}
```

Tạo giao diện tổng quát (1)

- Tạo một interface có tham số tổng quát

```
public interface GenericDao<T> {

    void insert(T obj);

    void update(T obj);

}
```

- Một class cài đặt từ giao diện trên

```
public class GenericDaoImpl<T> implements GenericDao<T> {

    @Override
    public void insert(T obj) {
        // do something
    }

    @Override
    public void update(T obj) {
        // do something
    }

}
```

Tạo giao diện tổng quát (2)

- Giả sử có 2 lớp Student và Teacher
- Cài đặt lớp **StudentDao**

```
public class StudentDao extends GenericDaoImpl<Student> {

}
```

- Cài đặt lớp **TeacherDao**

```
public class TeacherDao extends GenericDaoImpl<Teacher> {

}
```

```
public class GenericDaoExample {

    public static void main(String[] args) {
        Student student = new Student(1, "TrinhLK", 28);
        StudentDao dao = new StudentDao();
        dao.insert(student);
    }
}
```

Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

Tạo phương thức tổng quát (1)

- Một phương thức trong class hoặc Interface có thể sử dụng generic.

```
public class MyUtils {
    //Khai bao phuong thuc count su dung kieu du lieu tong quat <T>
    public static <T> int count(Collection<T> collection, T itemToCount) {
        int count = 0;
        for (T item : collection) {
            if (itemToCount.equals(item)) {
                count++;
            }
        }
        return count;
    }
}
```

Tạo phương thức tổng quát (2)

- Sử dụng phương thức *count* với <*T*> là *String*

```
import java.util.ArrayList;
import java.util.List;

public class MyUtilsExample {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        list.add("A");
        list.add("B");
        list.add("C");
        list.add("A");
        list.add("C");
        System.out.println(MyUtils.count(list, "A")); // 2
    }
}
```

Tạo phương thức tổng quát (3)

- Khai báo trong lớp tổng quát hoặc ngay trong lớp 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 (khi khai báo tổng quát)

```
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
```

Syntax: **<T extends BoundingType1 & BoundingType2 &…>**

Ràng buộc về kiểu: so sánh các phần tử

```
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());
```

Nguyễ

Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

Tạo đối tượng tổng quát (1)

- Không thể tạo trực tiếp đối tượng Generic

```
T obj = new T(); // Error
```

- **Lý do:**
  - Generic chỉ có tác dụng với trình biên dịch để kiểm soát code của người lập trình.
  - Java sẽ xác định T là kiểu dữ liệu gì trong quá trình biên dịch. Nếu không xác định được, T sẽ được mặc định là **Object**.

Tạo đối tượng tổng quát (2)

- Cần cung cấp đối tượng Class<T>
- → Java tạo đối tượng thông qua **Java Reflection**

```
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;
    }
}
```

Tạo đối tượng tổng quát (3)

- Tương tự với mảng Generic
  - Có thể khai báo nhưng không thể khởi tạo mảng Generic

```
T[] arr; // Ok

T[] arr2 = new T[5]; // Error
```

- **Lý do là gì?**
- Vì kiểu generic không hề tồn tại tại thời điểm chạy
- Generic chỉ có tác dụng với trình biên dịch để kiểm soát code của người lập trình
- Trình biên dịch cần biết **T** là gì mới có thể **new T[5]**

Tạo đối tượng tổng quát (4)

- **Làm thế nào để tạo một mảng Generic?**
- Tương tự như tạo một đối tượng Generic
  - Truyền cho Java đối tượng Class<T>
  - Class<T> giúp Java khởi tạo mảng generic tại runtime bằng **Java Reflection**

```
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;
    }
}
```

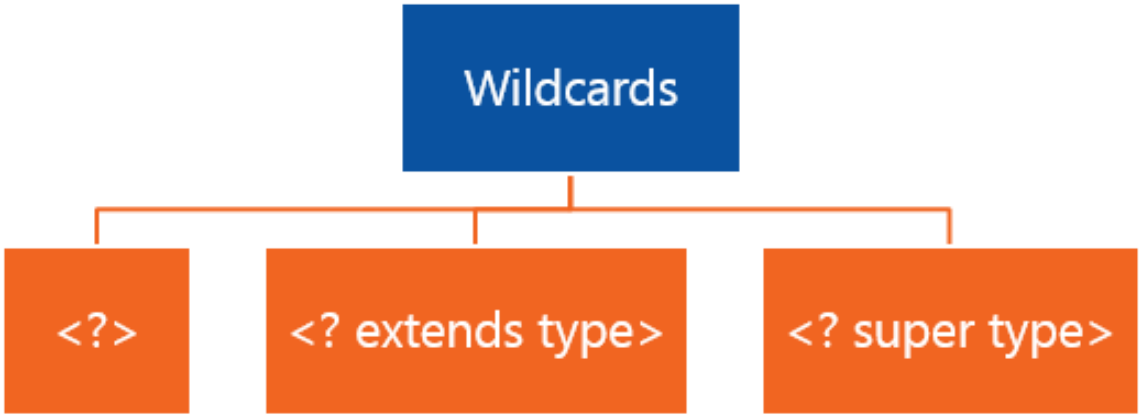
Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

Ký tự đại diện (Wildcard) (1)

- 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) (2)



- **<?>**: 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) (3)

- 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 (1)

```
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 (2)

```
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);
        }
    }

}
```

Hỏi đáp

- **Có thể new một đối tượng thuộc kiểu đại diện không?**
  - Không phải là một type cụ thể
  - → Không thể sử dụng toán tử **new**

```
// Tham số Wildcard không thể tham gia trong toán tử new.
List<? extends Object> list= new ArrayList<? extends Object>();
```

Nội dung

- Giới thiệu lập trình tổng quát
- Một số quy ước
- Lớp và giao diện tổng quát
- Phương thức tổng quát
- Đối tượng tổng quát
- Các ký tự đại diện (Wildcard)
- Ưu, nhược điểm của Generics

Ưu điểm

- Kiểu dữ liệu an toàn: Chỉ thao tác với một loại đối tượng trong generics
- Dễ kiểm soát lỗi: Kiểm tra dữ liệu tại compile time
- Hạn chế việc ép kiểu (cast) thủ công không an toàn.
- Hỗ trợ cho việc viết các thư viện phần mềm

Hạn chế

- Không thể gọi Generics bằng kiểu dữ liệu nguyên thủy
- Không thể tạo **instances** của kiểu dữ liệu Generics, thay vào đó sử dụng **reflection** từ class
- Không thể sử dụng *static* cho Generics.
- Không thể ép kiểu hoặc sử dụng **instanceof**

```
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
```

Java vs. C++

- Lập trình tổng quát trong Java không sinh ra các lớp mới
- Kiểm tra sự thống nhất về kiểu khi biên dịch
  - các đối tượng về bản chất vẫn là kiểu Object