Generics

Tham khảo: https://docs.oracle.com/javase/tutorial/java/generics/index.html

Introduction

Trong lập trình, **generics** là khái niệm giúp bạn viết cùng một mã code nhưng có thể hoạt động với nhiều kiểu dữ liệu **types**. Types (classes và interfaces) có thể là tham số khi định nghĩa classes, interfaces và methods. Ví dụ:

- ArrayList<T> là kiểu dữ liệu generic: bạn có thể tạo một danh sách của bất kỳ kiểu nào.
- public static final <T> List<T> emptyList() là hàm generic: bạn có thể sử dụng hàm để sinh ra một danh sach rỗng của bất kỳ kiểu nào.

Generic Types

Classes và Interfaces được tham số theo types.

Ví dụ sau tạo một class **Pair** biểu diễn kiểu dữ liệu có hai giá trị theo thứ tự.

```
/**
 * Pair represents data with two values in the order: F first, S second
 * @param <F>
 * @param <S>
 */
public class Pair<F, S> {
    private F fst;
    private S snd;

    public Pair(F fst, S snd) {
        this.fst = fst;
        this.snd = snd;
    }

    public F getFst() {
        return fst;
    }

    public S getSnd() {
        return snd;
    }
}
```

F và S là tham số kiểu cho class **Pair** và được sử dụng trong các hàm của class.

Ví dụ tiếp theo là tham số hóa một interface:

```java
public interface Comparable<T> {
    /**
    * Compares this object with the specified object for order. Returns a negative
    * integer, zero, or a positive integer as this object is less than, equal to, or
    * greater than the specified object.
    */
    int compareTo(T o)
}
```

Và cách implement một **generic interface** như sau:

```java
public class Integer implements Comparable<Integer> {
    int value;

    @Override
    public int compareTo(Integer o) {
        return compare(this.value, o.value);
    }

    public static int compare(int x, int y) {
        return (x < y) ? -1 : ((x == y) ? 0 : 1);
    }
}
```

Generic methods

Các tham số và kiểu trả về của hàm có thể tham số hóa. Ví dụ:

```java
public static final <F, S> pairWith(F first, S second) {
    return new Pair(first, second);
}
```

Bounded Type Parameters

Nhiều khi bạn muốn hạn định kiểu (types) được sử dụng là tham số generic, ví dụ:

```java
public static void <T> sort(List<T> a) {
    // TODO
}
```

Hàm *sort* của chúng ta được tham số hóa, nhưng khi thực hiện sort *List<T>* thì ta cần so sánh các phần tử của danh sách, do đó cần quy định các phần tử phải so sánh được, nghĩa là implements *Comparable* interface. Các quy định như vậy gọi là **bounded type parameters**. Sử dụng từ khóa **extends** để hạn định kiểu tham số.

```java
public static void <T extends Comparable> sort(List<T> a) {
    // TODO
}
```

Bạn có thể hạn định tham số cần thỏa mãn (subtypes - class, sublass hay implements interface) nhiều types (classes và interfaces) một lúc như sau:

```java
<T extends B1 & B2 & B3>
```

Wildcards

Trong **generics**, dấu hỏi **?** được dùng để thay thế cho kiểu generic (T, U, V) khi nào bạn không sử dụng chúng để khai báo (khai báo biến, khai báo hàm, ...) hay return. Ví dụ sau:

```java
public static final <T> void reverse(List<T> list) {
    int size = list.size();
    for (int i=0, mid=size>>1, j=size-1; i<mid; i++, j--)
        swap(list, i, j);
}
```

Hàm trên có thể dùng để reverse một danh sách của kiểu bất kỳ, nhưng tham số generic T không được sử dụng để khai báo hay return gì cả nên ta có thể đổi thành **?**

```java
public static final void reverse(List<?> list) {
    int size = list.size();
    for (int i=0, mid=size>>1, j=size-1; i<mid; i++, j--)
        swap(list, i, j);
}
```