Java Collections – List Set Map

Các lớp collection của Java giúp lưu trữ và xử lý các bộ dữ liệu. Bạn nên làm quen với các class này để tận dụng nhiều tính năng có sẵn của chúng trong chương trình của bạn.
Ba kiểu collection quan trọng nhất là List, Set và Map. List (danh sách) hoạt động như một cái mảng, ngoại trừ việc nó tự động thay đổi kích thước theo nhu cầu. Set (tập hợp) tương tự list, nhưng nó tự động loại bỏ các phần tử trùng. Map là các cặp ánh xạ key/value cho phép lưu trữ dữ liệu và lấy dữ liệu theo khóa (key) một cách hiệu quả.
Chi tiết xem tại tại tài liệu Java tại

Interface Collection là interface tổng quát, nó có các interface con List và Set. Nghĩa là, nếu một hàm có một tham số kiểu Collection, chẳng hạn hàm addAll(Collection col) dưới đây, ta có thể truyền đối số là một List hay một Set.
List cũng là một interface, và ArrayList là một class thông dụng tuân theo interface List. Tương tự, Set là một interface, còn HashSet và TreeSet là hai class tuân theo interface Set.
Map là một interface độc lập với Collection. HashMap là class thông dụng nhất tuân theo interface Map. Xem hình

List

Dưới đây là code tạo một list mới chứa String

```
List<String> words = new ArrayList<>();
```

Biến words được khai báo thuộc kiểu List<String>, trong đó List là interface tổng quát cho tất cả các loại list, còn <String> là cú pháp generic quy định rằng đây là list chứa các phần tử thuộc kiểu String. Ở bên phải phép gán, biểu thức new ArrayList<> tạo một đối tượng thuộc class ArrayList chứa String, không cần lặp lại tên kiểu “String” giữa hai dấu ngoặc vì thông tin này đã được quy định tại khai báo kiểu List<String> của words. Kết quả là ta lưu con trỏ tới một đối tượng ArrayList tại một biến kiểu List. Đây là cách thông dụng dùng List. Bạn có thể thay class ArrayList bằng class LinkedList chẳng hạn, nếu cần.

List add()

Đối tượng ArrayList mới có nội dung rỗng. Hàm add() thêm một phần tử vào cuối list. Ví dụ:

```
words.add("this");
words.add("and");
words.add("that");
// words is now: {"this", "and", "that"}
words.size(); // returns 3
```

Hàm size() trả về kích thước của danh sách (hay bất cứ loại collection nào)

Với các class collection, constructor mặc định tạo cho ta một collection rỗng. Nếu muốn, ta có thể dùng constructor nhận tham số là một collection để tạo ra bản sao.

Lưu ý là lệnh trên sao chép các phần tử của list, là các con trỏ, vào word2. Nó chỉ đơn giản là thực hiện một phép gán = cho mỗi phần tử trong collection, ở đây là sao chép các con trỏ từ list này sang list khác.

```
// Create words2 which is a copy of words
       List<String> words2 = new ArrayList<>(words);
```

Lưu ý: ở trên chỉ là việc sao chép các phần tử (con trỏ) từ words vào words2, các phép gán = được thực hiện cho từng phần tử của collection và các con trỏ được sao chép. Các đối tượng String được trỏ tới không được sao chép.

List get()

Các phần tử của list được đánh chỉ số từ 0 đến size -1, tương tự mảng. Hàm get(int index) trả về phần tử có chỉ số index.

```
// suppose words is {"this", "and", "that"}
       words.get(0); // returns "this"
       words.get(2); // returns "that"
       words.get(3); // ERROR index out of bounds
```

Vòng for

Foreach - duyệt từng phần tử

Đoạn code sau duyệt từng phần tử trong danh sách

```
int lengthSum = 0;
for (String str : words) {
   lengthSum += str.length();
}
```

Mỗi lần thân vòng lặp chạy, biến str nhận giá trị của phần tử String tiếp theo trong danh sách words. Không được thêm vào hoặc bớt phần tử khỏi một list trong khi foreach đang duyệt nó. Có thể “break” để thoát ra giữa chừng.

Cách khác là dùng vòng for kèm với chỉ số chạy và dùng get(index) để lấy từng phần tử của list

```
// iterate through the words backwards, skipping every other one
for (int i = words.size() - 1; i >= 0; i = i - 2) {
   String str = words.get(i);
   // do something with str
}
```

Các hàm cơ bản của List

- get(int index) -- trả về phần tử tại chỉ số index.
- set(int index, Object obj) -- gán trị cho phần tử tại chỉ số index. Hàm set() chỉ ghi đè phần tử chỉ số index chứ không thay đổi độ dài của list.
- add(Object obj) -- thêm một phần tử vào cuối danh sách.

- add(int index, Object obj) -- thêm một phần tử vào vị trí có chỉ số index trong danh sách, dịch các phần tử có chỉ số từ index sang phải để lấy chỗ cho phần tử mới.
- remove(int index) -- xóa bỏ phần tử tại chỉ số index. Dịch các phần từ chỉ số từ index trở lên sang trái để lấp chỗ trống.
- int indexOf(Object obj) trả về chỉ số của lần xuất hiện đầu tiên của obj trong list, -1 nếu không thấy.
- List sublist(int fromIndex, int toIndex) trả về một list mới là đại diện đoạn con từ fromIndex đến (toIndex-1).Các sửa đổi đối với list con có hiệu quả trực tiếp lên list ban đầu.

Các hàm tiện ích Collection

Dưới đây là các hàm dùng được cho bất cứ kiểu collection nào

- int size() -- number of elements in the collection
- boolean isEmpty() -- true if the collection is empty
- boolean contains(Object target) -- true if the collection contains the given target element anywhere, using equals() for comparisions. For a list, this is a "linear" O(n) cost operation. For a Set, this may be a fast, constant time operation.
- boolean containsAll(Collection coll) -- true if the collection contains all of the elements in the given collection (order is not significant).
- void clear() -- removes all the elements in the collection, setting it back to an empty state.
- boolean remove(Object target) -- searches for and removes the first instance of the target if found. Returns true if an element is found and removed. Uses equals() for comparisons. This is an O(n) cost operation for an ArrayList. Note that this is different from the List method remove(int index) which is given the index, so it does not need to search before removing the element.
- boolean removeAll(Collection coll) -- removes from the receiver collection all of the elements which appear in the given collection (returns true if changed).
- boolean addAll(Collection coll) -- adds to the receiver collection all of the elements in the given collection (returns true if changed).
- boolean retainAll(Collection coll) -- retains in the receiver collection only the elements which also appear in the given collection. In other words, removes all the elements which are not in the given collection. After this operation, the receiver will represent effectively a subset of the given collection.
- Object[] toArray() -- builds and returns an Object[] array containing the elements from the collection. There is a variant which takes an "example" array argument and uses that example array to know what type of array to return -- so call toArray(new String[0]), and it will return a String[] array.

Code ví dụ

```
// Có thể dùng Arrays.asList() để biến một mảng thành 1 List.
       String[] array = new String[]{"red", "green", "blue", "ochre"};
       List<String> colors1 = Arrays.asList(array);
```

```java
// colors1 không phải list thực sự, nó chỉ là một cái array được bọc lại
// để trông giống một list.  (The "adapter" pattern.)
// Do đó không thể dùng add/remove cho colors1

// Tạo một ArrayList từ các phần tử của colors1 --
// Hầu hết các collection đều có constructor lấy tham số dạng này.
// (hoặc tạo ArrayList rỗng rồi dùng addAll() để lấy hết nội dung của colors1).
        List<String> colors2 = new ArrayList<>(colors1);
        colors2.add("purple"); // Add "purple" on the end.
// colors 2 is {"red", "green", "blue", "ochre", "purple"

        String str = colors2.toString(); // "[red, green, blue, ochre, purple]"
        System.out.println(str);
        int size = colors2.size(); // 5
        boolean hasOchre = colors2.contains("ochre"); // true
        int indexOcher = colors2.indexOf("ochre"); // 3

// Search out and remove the first "ochre".
        colors2.remove("ochre");
// colors2 is {"red", "green", "blue", "purple"};

// Make up some more colors and add them.
        List<String> colors3 = Arrays.asList("yellow", "pink", "purple");
        colors2.addAll(colors3);
// colors2 is {"red", "green", "blue", "purple", "yellow", "pink", "purple"}
// Remove all "purple" and "yellow".
        colors2.removeAll(Arrays.asList("purple", "yellow"));
// colors2 is {"red", "green", "blue", "pink"}
        System.out.println("remove all " + colors2);
// Use subList() to make a "front" List showing just elements 0, 1, 2 of colors2.
        List<String> front = colors2.subList(0, 3);
        System.out.println("front " + front); // [red, green, blue]
// front is {"red", "green", "blue"}
// Can make changees to front, and they go through to underlying colors2,
// but do not make changes to colors2 while using front.
        front.contains("green"); // true
        front.remove("green");
        front.add("orange");
// front is {"red", "blue", "orange"}
// colors2 is {"red", "blue", "orange", "pink"}
```

Set

Set là collection với ràng buộc là mỗi phần tử chỉ có mặt một lần. Do đó nếu hàm add() được gọi với một phần tử đã có sẵn trong set thì nó sẽ im lặng lờ đi không làm gì. Các hàm thông dụng khác của Collection vẫn hoạt động bình thường. Set không có chỉ số 0...size-1 như List.

Để tạo union (hợp) của hai set (tập hợp) x và y, dùng hàm x.addAll(y). Để lấy giao của x và y, gọi hàm x.retainAll(y). Để xem x có phải tập con của y không, dùng hàm y.containsAll(x).

HashSet là loại Set thông dụng nhất. HashSet chỉ chạy đúng với các kiểu dữ liệu có hàm hashCode đã được định nghĩa, ví dụ String và Integer đã có sẵn hàm hashCode đúng. TreeSet là lựa chọn khác, chạy châm hơn một chút nhưng có giữ thứ tự các phần tử trong tập hợp, và cho phép duyệt theo thứ tự.

```
        // Create a Set of Integer objects
        Set<Integer> nums = new HashSet<>();
// Add the numbers 1..10
        for (int i = 1; i <= 10; i++) {
            nums.add(i);
        }
// Add 1, 4, 9, 16, 25
// Since it's a Set, only the add() of 16 and 25 do anything.
        for (int i = 1; i <= 5; i++) {
            nums.add(i * i);
        }
// Foreach works on a Set.
        for (int num : nums) {
            System.out.println(num);
        }
// Iterator works on a set.
// (the values will appear in some random order for a HashSet
// as we have here)
        Iterator<Integer> it = nums.iterator();
        while (it.hasNext()) {
            int val = it.next();
        }
// Other Collection utilties work
        nums.contains(9); // true
        nums.containsAll(Arrays.asList(1, 2, 3)); // true
// addAll() is essentially a mathematical union operation.
// Change nums to the union with the set {16, 17}
        nums.addAll(Arrays.asList(16, 17));
// Accessing by index number DOES NOT work
// (index numbers are List feature only)
// int val2 = nums.get(0); //NO does not compile
```

Map

Map là cấu trúc dictionary (bảng tra cứu). Có sẵn hai loại Map là HashMap dùng cơ chế băm và TreeMap dùng cây nhị phân. HashMap thông dụng hơn.
Các hàm thông dụng của Map là put(key, value) để thêm cặp key-value vào map và get(key) để tìm value tương ứng. Hàm containsKey(key) kiểm tra xem trong map có key đó hay không.

```
        Map<String, String> states = new HashMap<>();
        states.put("ca", "California");
        states.put("az", "Arizona");
        states.put("mn", "Minnesota");
        states.put("nj", "New Jersey");
        System.out.println(states.get("ca")); // California
        System.out.println(states.get("zz")); // null (no entry for key "zz")
        states.put("ca", "The Golden State"); // Overwrite the old entry

// Pull out live Collection of all the values.
        Collection<String> values = states.values();
        System.out.println(values); // [Minnesota, New Jersey, The Golden State,
Arizona]

// Pull out live set of the keys -- use to print key->value for
// the whole map. The order of the keys is random for a HashSet.
        Set<String> keys = states.keySet();
        for (String key : keys) {
            System.out.println(key + "->" + states.get(key));
        }
// mn->Minnesota
// nj->New Jersey
// ca->The Golden State
// az->Arizona

// More complex -- create a map of Strings to Lists of Integers
        Map<String, List<Integer>> map = new HashMap<>();
        List<Integer> nums = new ArrayList<>();
        nums.addAll(Arrays.asList(1, 2, 3, 4, 5));
        map.put("1-5", nums);
```

Collection values() là hàm trả về một collection chứa tất cả các value đang nằm trong map. Collection này chỉ được đọc (read-only) chứ không được sửa. Collection này thay đổi theo khi map bị sửa đổi
Set keys() là hàm trả về một collection chứa tất cả các key. Collection này cũng thay đổi theo khi map bị sửa đổi

Cách duyệt từng cặp key-value trong map.

```
// Each Map.Entry object contains one String key + one String value.
// entrySet() returns a set of all the entries.
        Set<Map.Entry<String, String>> entries = states.entrySet();
        for (Map.Entry<String, String> entry : entries) {
            System.out.println(entry.getKey() + "->" + entry.getValue());
        }
```

Nên dùng hàm entrySet để duyệt khi cần để ý hiệu năng chương trình.