
Collections

Collection là gì?

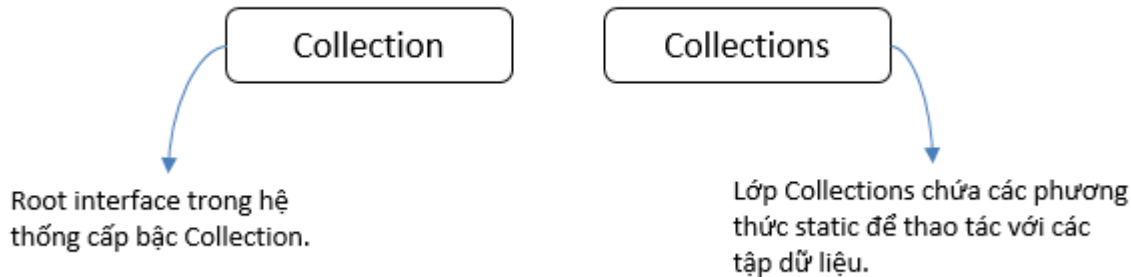


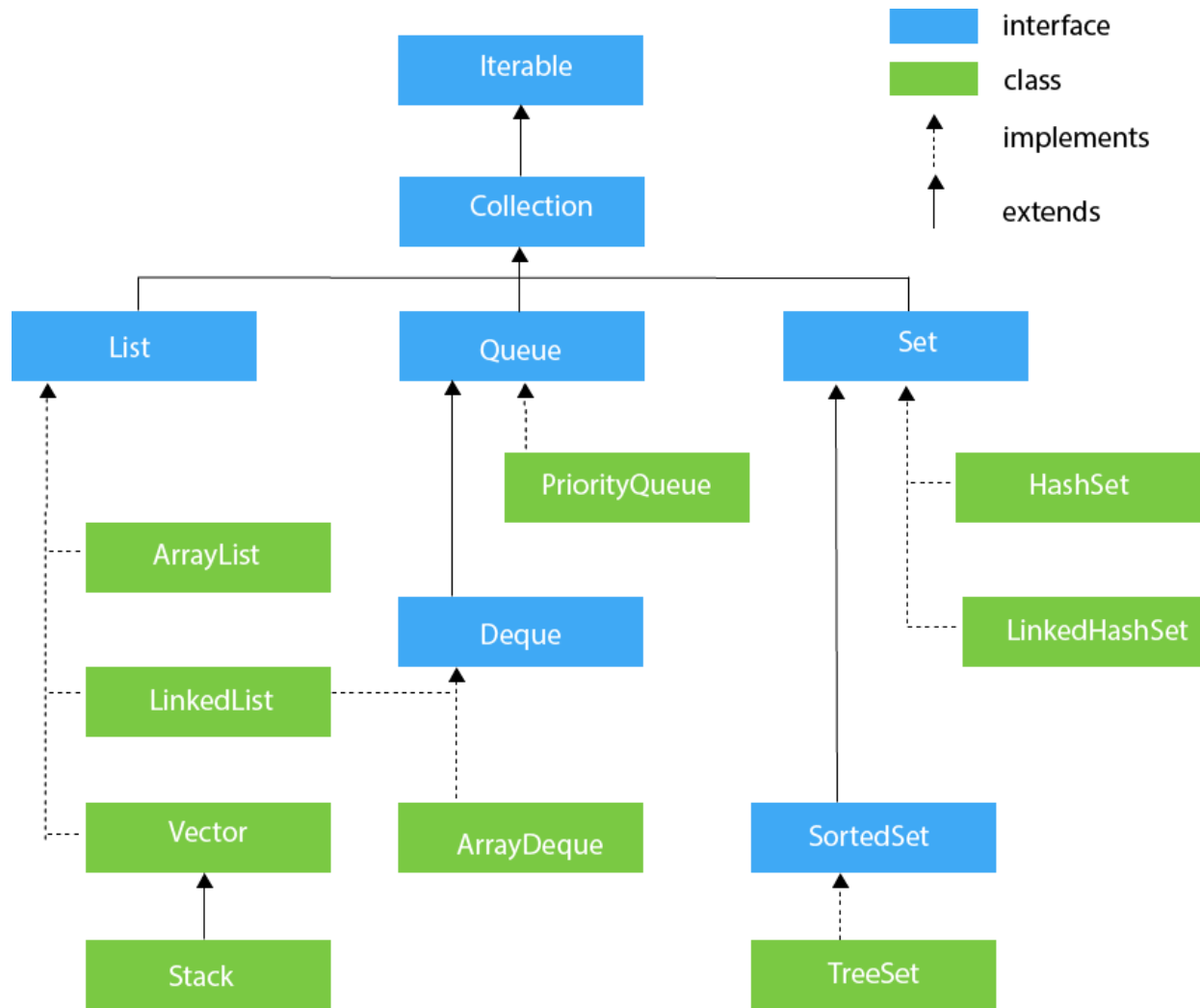
Collections là một framework cung cấp một kiến trúc để lưu trữ và thao tác với nhóm các đối tượng

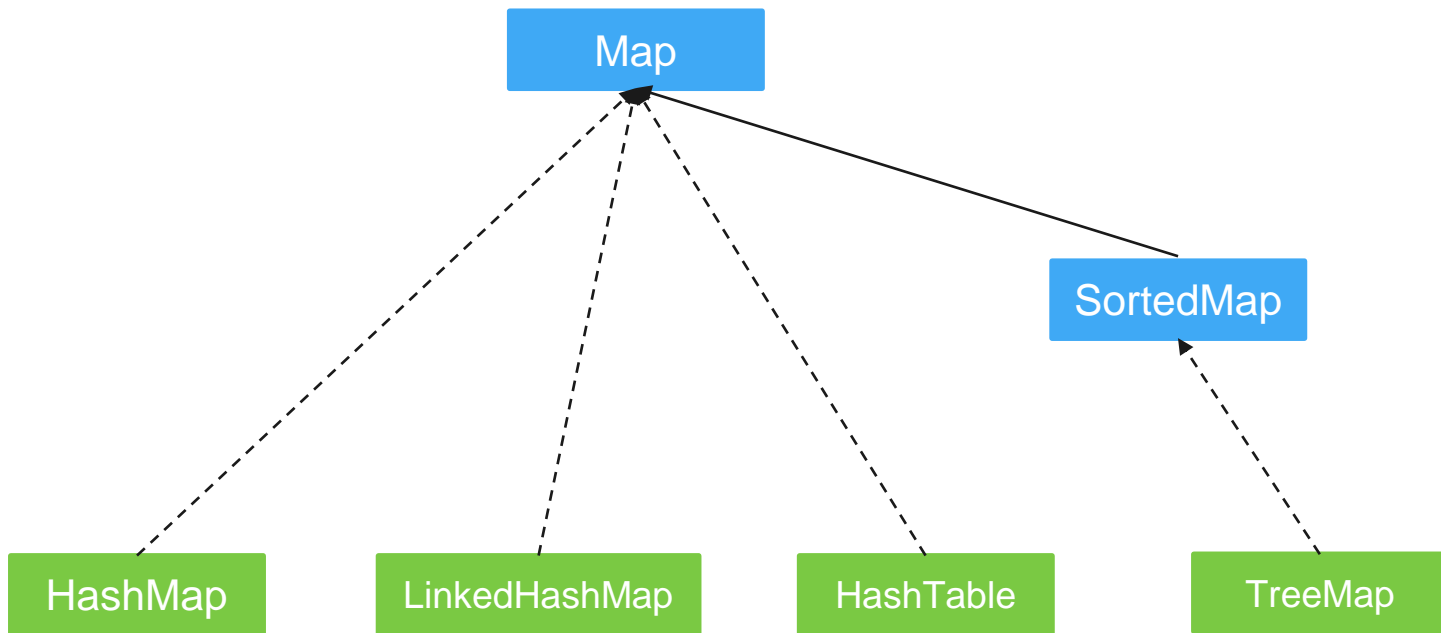
Java collections có thể đạt được tất cả các thao tác mà bạn thực hiện trên dữ liệu như tìm kiếm, sắp xếp, chèn, xóa

Collection là gì?

Java collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet,...)







List Interface

List Interface là giao diện con của Collection Interface. Nó ngăn cách cấu trúc dữ liệu kiểu danh sách trong đó chúng ta có thể lưu trữ tập hợp các đối tượng có thứ tự.

List Interface được thực hiện bởi các lớp ArrayList, LinkedList, Vector và Stack

Để khởi tạo List interface chúng ta sử dụng:

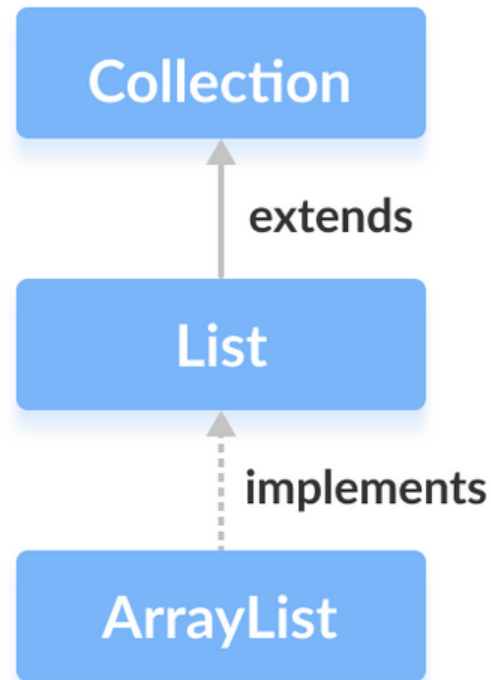
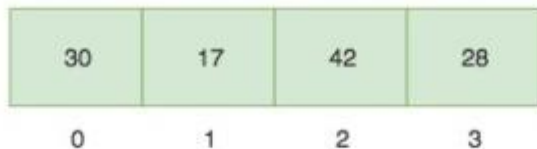
```
List <Kiểu dữ liệu> <Tên>= new ArrayList();  
List <Kiểu dữ liệu> <Tên> = new LinkedList();  
List <Kiểu dữ liệu> <Tên> = new Vector();  
List <Kiểu dữ liệu> <Tên> = new Stack();
```

ArrayList

Lớp ArrayList trong java được sử dụng như một mảng động để lưu trữ các phần tử. Nó kế thừa lớp `AbstractList` và implements giao tiếp `List`.

ArrayList có thể thay đổi kích thước

Java ArrayList
Representation



ArrayList

Khởi tạo ArrayList:

```
ArrayList<Kiểu dữ liệu> <Tên> = new ArrayList<Kiểu dữ liệu>();
```

```
ArrayList<Kiểu dữ liệu> <Tên> = new ArrayList<>();
```

Ví dụ:

```
ArrayList<String> cars = new ArrayList<String>();
```

```
ArrayList<Integer> myNumbers = new ArrayList<Integer>();
```


Một số phương thức của ArrayList

Thêm phần tử vào mảng bằng hàm add():

```
cars.add("Mazda");  
cars.add(3, "Vinfast");
```

Diagram illustrating the `add()` method:

- The first call `cars.add("Mazda");` shows the string `"Mazda"` being added to the end of the list.
- The second call `cars.add(3, "Vinfast");` shows the string `"Vinfast"` being added at the specific `index` 3.

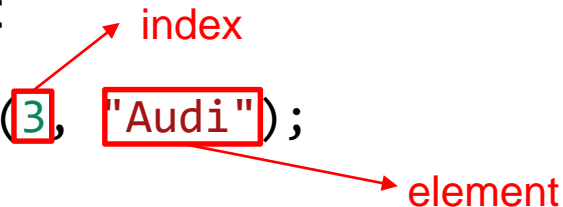
Truy cập phần tử trong ArrayList bằng hàm get():

```
System.out.println(cars.get(1));
```

Một số phương thức của ArrayList

Cập nhật giá trị của phần tử trong ArrayList bằng hàm set(index, element):

```
cars.set(3, "Audi");
```



Xóa tất cả phần tử trong ArrayList bằng hàm clear():

```
cars.clear();
```

Xóa phần tử trong ArrayList bằng hàm remove():

```
cars.remove("Mazda");
```



```
cars.remove(0);
```

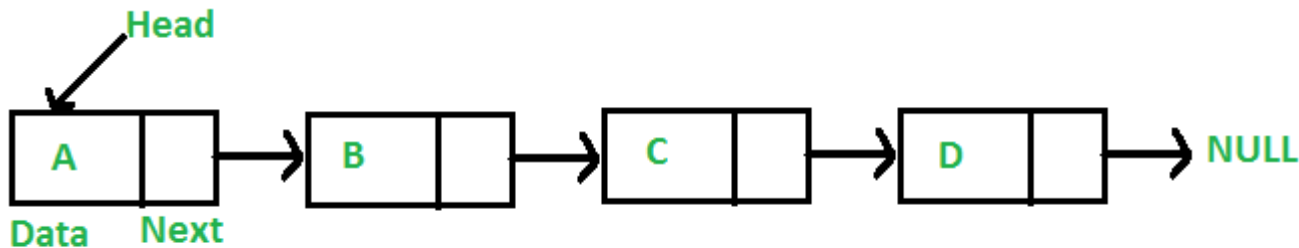


Phương thức	Mô tả
add(Object o)	Thêm phần tử được chỉ định vào cuối một danh sách.
add(int index, Object element)	Chèn phần tử element tại vị trí index vào danh sách.
addAll(Collection c)	Thêm tất cả các phần tử trong collection c vào cuối của danh sách, theo thứ tự chúng được trả về bởi bộ lặp iterator.
addAll(int index, Collection c)	Chèn tất cả các phần tử trong collection c vào danh sách, bắt đầu từ vị trí index.
retainAll(Collection c)	Xóa những phần tử không thuộc collection c ra khỏi danh sách.
removeAll(Collection c)	Xóa những phần tử thuộc collection c ra khỏi danh sách.
indexOf(Object o)	Trả về chỉ mục trong danh sách với sự xuất hiện đầu tiên của phần tử được chỉ định, hoặc -1 nếu danh sách không chứa phần tử này.

Phương thức	Mô tả
<code>lastIndexOf(Object o)</code>	Trả về chỉ mục trong danh sách với sự xuất hiện cuối cùng của phần tử được chỉ định, hoặc -1 nếu danh sách không chứa phần tử này.
<code>toArray()</code>	Trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự.
<code>toArray(Object[] a)</code>	Trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự.
<code>clone()</code>	Trả về một bản sao của ArrayList.
<code>clear()</code>	Xóa tất cả các phần tử từ danh sách này.
<code>trimToSize()</code>	Cắt dung lượng của thể hiện ArrayList này là kích thước danh sách hiện tại.
<code>contains(element)</code>	Kết quả trả về là true nếu tìm thấy element trong danh sách, ngược lại trả về false.

LinkedList

LinkedList là một cấu trúc dữ liệu tuyến tính. Các phần tử trong LinkedList không được lưu trữ liền kề nhau giống như arrays. Mỗi phần tử trong LinkedList liên kết với nhau bằng một con trỏ, nghĩa là mỗi phần tử sẽ tham chiếu đến địa chỉ của phần tử tiếp theo.



Một số phương thức với LinkedList

Khởi tạo

```
LinkedList<String> cars = new LinkedList<>();
```

Thêm phần tử

```
cars.add("Toyota");  
  
cars.add(2, "Audi");  
  
cars.addFirst("Hyundai");  
cars.addLast("BMW");
```

Lấy kích thước:

```
cars.size();
```

Lấy một phần tử:

```
String car = cars.get(1);
```

```
//Lấy phần tử đầu tiên:
```

```
String firstCar = cars.getFirst();
```

```
//Lấy phần tử vị trí cuối cùng
```

```
String lastCar = cars.getLast();
```

Cập nhật

```
cars.set(4, "Chevrolet");
```

Trả về phần tử đầu tiên và xóa nó ra khỏi danh sách

```
String first = cars.poll();  
//Hoặc sử dụng pollFirst()  
System.out.println("Phần tử đầu tiên: " + first);  
System.out.println("Danh sách sau khi xóa "+first+" :");
```


Trả về phần tử cuối cùng và xóa nó ra khỏi danh sách

```
String last = cars.pollLast();  
System.out.println("Phần tử cuối cùng: " + last);  
System.out.println("Danh sách sau khi xóa "+last+" :");  
show(cars);
```

Xóa phần tử với remove()

//Xóa phần tử đầu tiên

```
cars.remove();
```

//Hoặc cars.removeFirst();

cars.remove(3); //Xóa tại vị trí index

cars.remove("Ford"); //Xóa dựa vào element

//Xóa phần tử cuối cùng

```
cars.removeLast();
```

//Xóa hết

```
cars.clear();
```

Set Interface

Set là kiểu dữ liệu mà bên trong nó mỗi phần tử chỉ xuất hiện duy nhất một lần và Set interface cung cấp các phương thức để thao tác với set

Set interface được kế thừa từ Collection Interface nên nó được cung cấp đầy đủ các phương thức của Collection Interface



Set Interface

Một số class thực thi Set Interface thường gặp:

- **TreeSet**: là 1 class thực thi giao diện Set Interface, trong đó các phần tử trong set đã được sắp xếp.
- **HashSet**: là 1 class implement Set Interface, mà các phần tử được lưu trữ dưới dạng bảng băm (hash table).
- **EnumSet**: là 1 class dạng set như 2 class ở trên, tuy nhiên khác với 2 class trên là các phần tử trong set là các enum chứ không phải object.

Queue Interface

Queue(Hàng đợi) là kiểu dữ liệu nổi tiếng với kiểu vào ra FIFO, tuy nhiên với Queue Interface thì queue không chỉ còn dừng lại ở mức đơn giản như vậy mà nó cung cấp cho bạn các phương thức để xây dựng các queue phức tạp hơn nhiều như priority queue, deque. Queue Interface cũng kế thừa và mang đầy đủ các phương thức từ Collection Interface.

- **LinkedList**: chính là LinkedList mình đã nói ở phần List
- **PriorityQueue**: là 1 dạng queue mà trong đó các phần tử trong queue sẽ được sắp xếp.
- **ArrayDeque**: là 1 dạng deque (queue 2 chiều) được implement dựa trên mảng

Comparable Interface

Comparable interface được sử dụng để sắp xếp các đối tượng của lớp do người dùng định nghĩa. Interface này thuộc package java.lang và chỉ chứa một phương thức có tên compareTo(Object)

public int compareTo (Object obj): được sử dụng để so sánh đối tượng hiện tại với đối tượng được chỉ định.

Comparator Interface

Comparator interface được sử dụng để sắp xếp các đối tượng của lớp do người dùng định nghĩa. Interface này thuộc package `java.util` và chứa hai phương thức `compare(Object obj1, Object obj2)` và `equals(Object element)`.

`public int compare(Object obj1, Object obj2)`: so sánh đối tượng đầu tiên với đối tượng thứ hai.

Map Interface

Map (đồ thị/ánh xạ) là kiểu dữ liệu cho phép ta quản lý dữ liệu theo dạng cặp key-value, trong đó key là duy nhất và tương ứng với 1 key là một giá trị value. Không giống như các interface ở trên, Map Interface không kế thừa từ Collection Interface mà đây là 1 interface độc lập với các phương thức của riêng mình.

Class về Map

TreeMap

EnumMap

Map

HashMap

WeakHashMap

