Tìm hiểu về Map, Set, List trong java

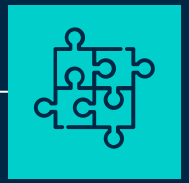
Team

—Nguyễn Thanh Tùng

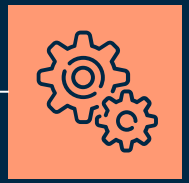
—Nguyễn Tuấn Anh

—Nguyễn Trung Dức
(Nhóm trưởng)

TABLE OF CONTENTS



01 Map



02 Set

03 List

Map

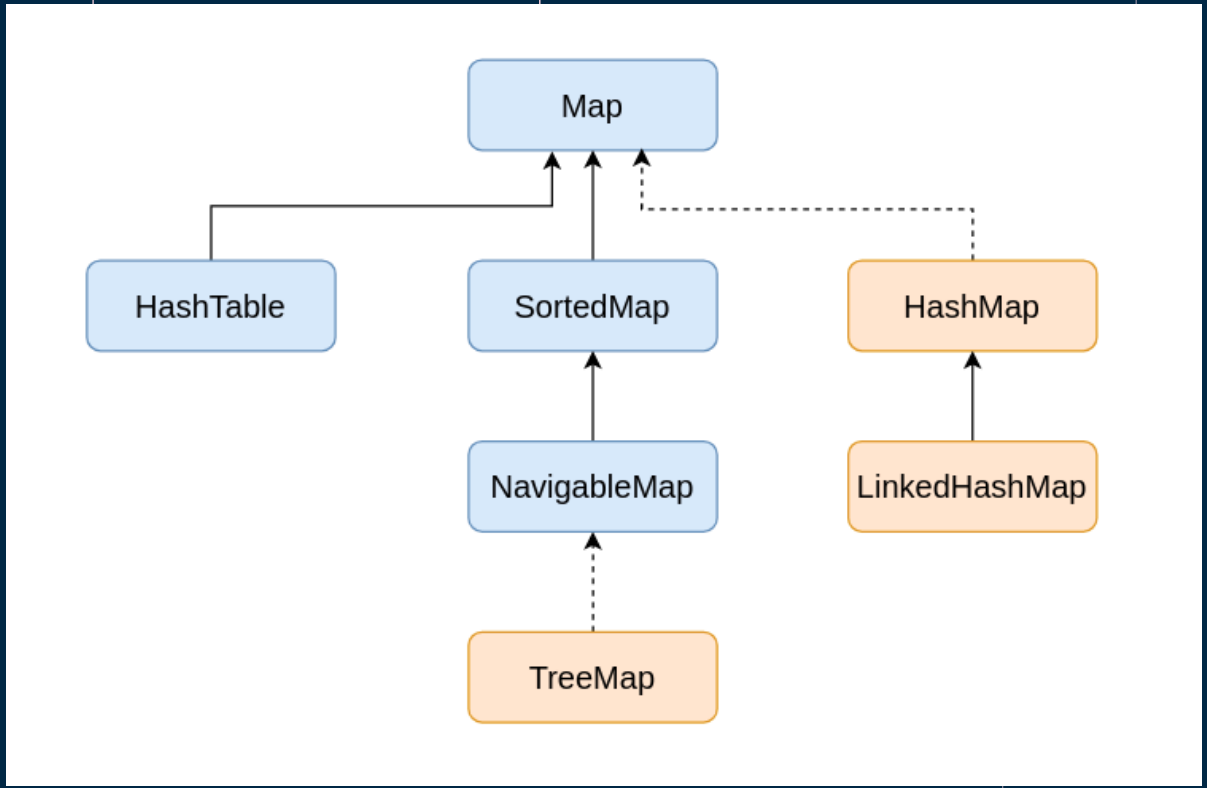
01

Map interface là gì ?

Map interface nằm trong gói java.util

Nó được sử dụng để lưu một cặp giá trị key-value

Key là giá trị duy nhất (không được trùng lặp) tương ứng với một value


Map
HashTable
SortedMap
HashMap
NavigableMap
LinkedHashMap
TreeMap

Các phương thức trong Map interface

- Object put(Object key, Object value): Thêm đối tượng được chỉ định vào map
- void putAll(Map map): Thêm một map được chỉ định vào map hiện tại
- Object remove(Object key): Xóa đối tượng được chỉ định dựa vào key

Các phương thức trong Map interface

- Object get(Object key) : Truy xuất đối tượng được chỉ định dựa vào key
- Boolean containsKey(Object key): Kiểm tra key được chỉ định có tồn tại trong map hay không
- Set keySet(): Trả về đối tượng Set có chứa tất cả các key
- Entry entrySet(): Trả về tập hợp các entry của map, mỗi entry chứa key và value

Map.Entry interface

- Entry là một interface con của Map. Vì vậy, chúng ta có thể truy cập nó bằng Map.Entry. Nó cung cấp các phương thức để truy xuất key và value.

Các phương thức trong Map.Entry

- getKey(): Trả về giá trị Key trong entry
- getValue(): Trả về giá trị Value trong entry

HashMap

HashMap là một phần trong Java collections.

Nó cung cấp các phương thức cơ bản của Map interface.

Được sử dụng để lưu trữ dữ liệu theo cặp key-value.

Để truy cập một giá trị ta phải biết key của nó.

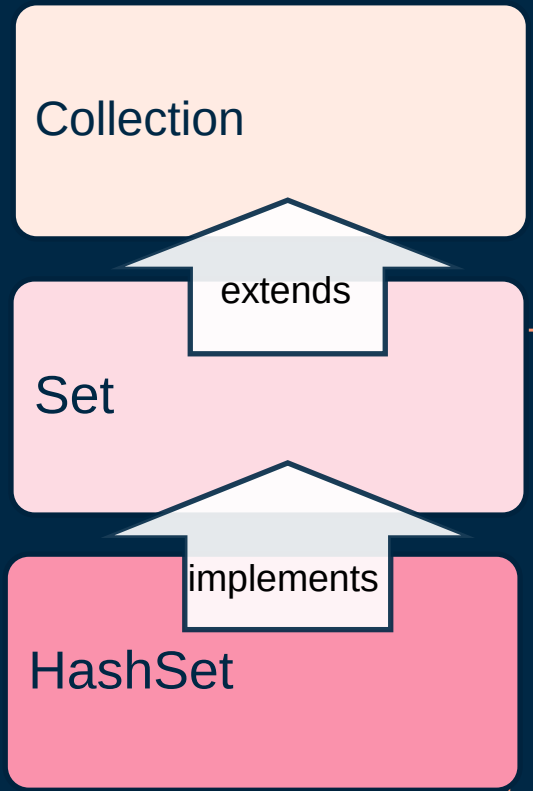
Các đặc điểm của HashMap :

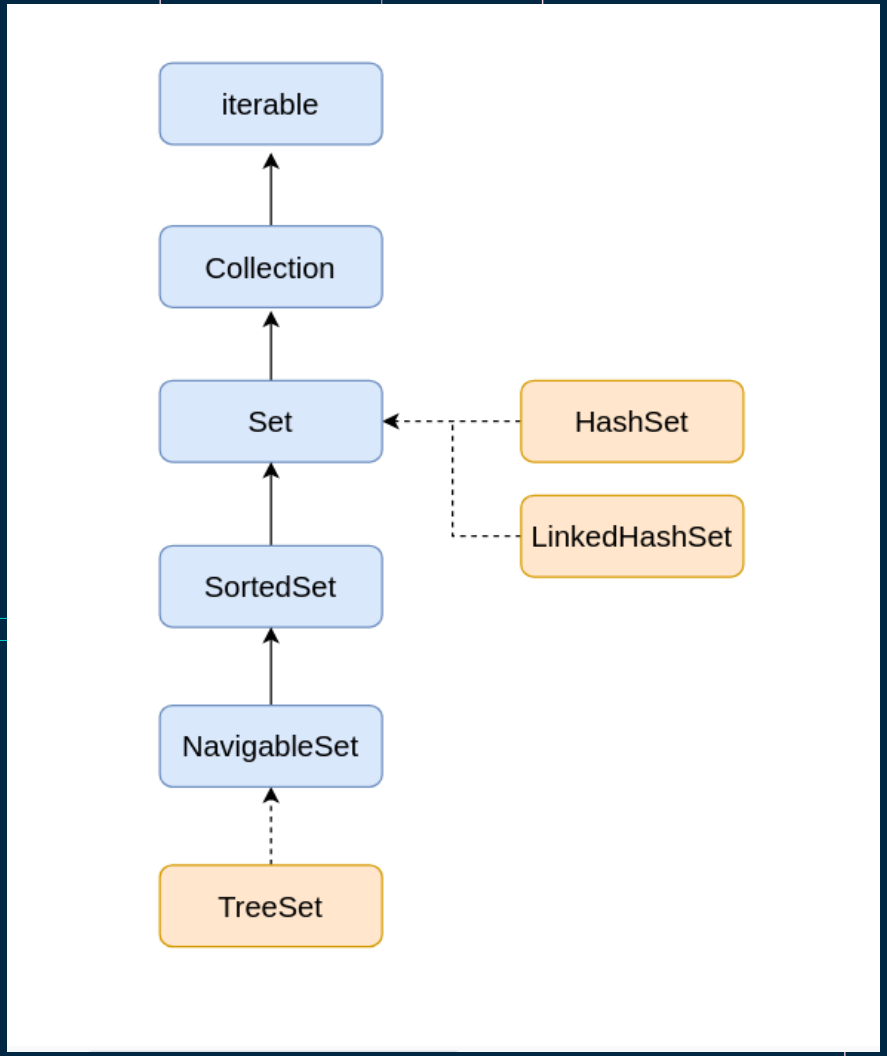
- Hashmap không thể chứa các Key(K) trùng lặp.
- Cho phép lưu các giá trị null (kể cả key và value).
- Giá trị key của nó là duy nhất.
- Khai báo HashMap:
  - HashMap<k,v> hashMap = new HashMap<>();
  - Map<k,v> hashMap = new HashMap<>();

Set 02

Set

Set interface nằm trong package java.util. Khác với Queue và List, Set không cho phép lưu trữ các phần tử trùng lặp. Set interface chứa các phương thức kế thừa từ Collection interface và thêm một tính năng hạn chế việc chèn các phần tử trùng lặp.




iterable
Collection
Set
HashSet
LinkedHashSet
SortedSet
NavigableSet
TreeSet

HashSet

HashSet là một class được sử dụng rộng rãi trong Set interface.
HashSet được dùng để tạo collection sử dụng hash table (bảng băm) để lưu trữ.

HashSet cho phép thêm phần tử “NULL”

Khai báo HashSet:

```
Set<String> setA = new HashSet<String>();
Set<String> setA = new HashSet<>();
```

Một số hàm phổ biến trong Set

- `add(Object o)`: Thêm một phần tử
- `addAll(Collection c)`: Chèn tất cả các phần tử của collection c vào set
- `clear()`: Trả Xóa tất cả các phần tử khỏi set

Một số hàm phổ biến trong Set

- contains(Object element): Trả về true nếu set chứa phần tử đã chỉ đinh

- isEmty(): Kiểm tra trong set có phần tử nào hay không (nếu không có phần tử nào thì trả về true)

- removeAll(Collection c): Xóa khỏi set tất cả các phần tử nằm trong collection được chỉ định

List

03

List

- List interface kế thừa Collection interface.
- Các phần tử có thể được chèn hoặc truy cập từ vị trí của chúng trong List. List lưu vị trí của các phần tử theo chỉ số (index), chỉ số của phần tử đầu tiên là 0.
- List được phép chứa các phần tử trùng lặp.

Một số phương thức của List

void add(int index, Object obj): Chèn đối tượng obj vào vị trí index trong List

boolean addAll(int index, Collection c): Chèn toàn bộ phần tử của Collection vào vị trí index trong List

Object get(int index): Trả về đối tượng ở vị trí index

int indexOf(Object obj) : Trả về index của phần tử obj xuất hiện đầu tiên. Nếu obj không tồn tại trong danh sách thì trả về -1

int lastIndexOf(Object obj): Trả về index của phần tử obj xuất hiện cuối cùng. Nếu obj không tồn tại trong danh sách thì trả về -1

Một số phương thức của List

ListInterator listIterator(): Trả về một iterator bắt đầu từ phần tử đầu tiên của list

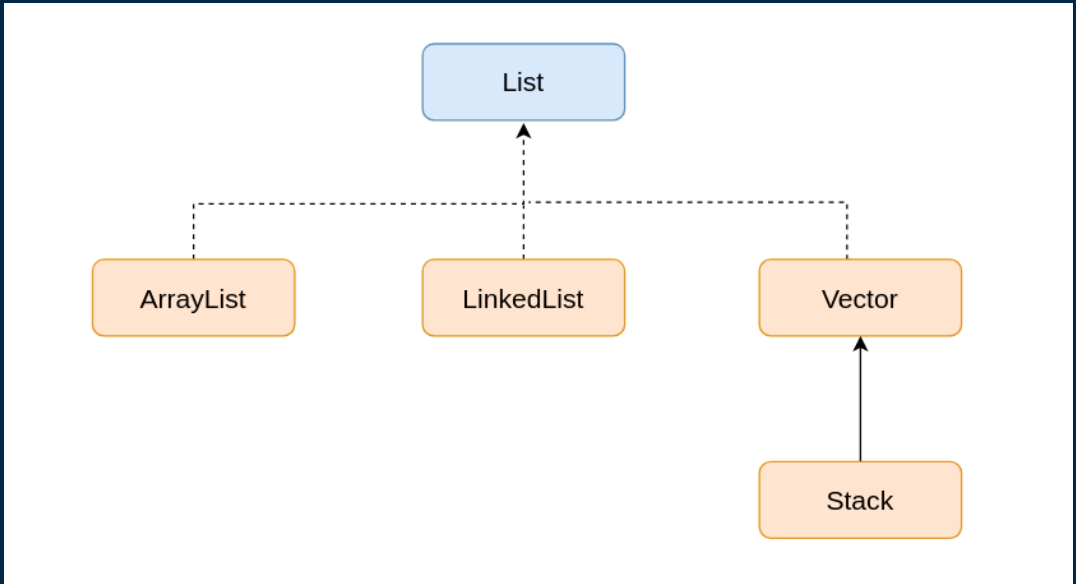
ListInterator listIterator(int index): Trả về một iterator bắt đầu từ phần tử tại index được chỉ định

Object remove(int index): Xóa phần tử ở vị trí index

Object set(int index, Object obj): Thay đổi phần tử ở vị trí index bằng obj

List subList(int start, int end): Trả về một List bao gồm các phần tử được bắt đầu từ vị trí index *start* đến vị trí index *end* trong List đang gọi.

Vì List là một interface nên không thể tạo đối tượng từ List mà phải thông qua các class implements nó như ArrayList, LinkedList, Vector, Stack.



ArrayList

ArrayList trong Java là một class kế thừa class AbtractList và implements List interface trong Collection framework. Nó có một vài đặc điểm và phương thức tương đồng với List.

ArrayList được sử dụng như một mảng động để lưu trữ các phần tử.

Mặc dù thao tác với ArrayList có thể chậm hơn Array nhưng lại hữu ích trong các chương trình cần nhiều thao tác như: thêm, sửa, xóa phần tử,…

Khai báo ArrayList:

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

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

Một số phương thức của ArrayList

Method	Description
boolean add(Collection c)	Thêm collection được chỉ định vào cuối danh sách
void add(int index, Object element)	Chèn phần tử được chỉ định vào vị trí index
void clear()	Xóa tất cả phần tử trong danh sách
int lastIndexOf(Object obj)	Trả về index của phần tử obj xuất hiện cuối cùng, nếu obj không có trong danh sách thì trả về -1
Object clone()	Tạo một ArrayList copy từ ArrayList ban đầu
Object[] toArray()	Trả về một mảng chứa toàn bộ phần tử của danh sách
void trimToSize()	Cắt giảm dung lượng của ArrayList theo kích thước thực (số lượng phần tử) của danh sách. Đọc thêm: https://www.programiz.com/java-programming/library/arraylist/trimtosize

LinkedList

LinkedList trong Java là một class kế thừa class AbstractSequentialList và implements List interface, Queue interface trong Collection Framework. Nó có một vài đặc điểm và phương thức tương đồng với List, Queue.

LinkedList có thể được sử dụng như list, stack, queue.

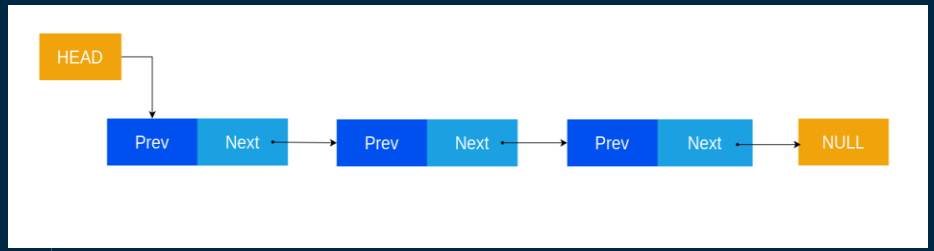
Khai báo LinkedList:

```
List<Kiểu dữ liệu> <Tên> = new LinkedList<>();
LinkedList<Kiểu dữ liệu> <Tên> = new LinkedList<>();
Queue<Kiểu dữ liệu> <Tên> = new LinkedList<>();
Deque<Kiểu dữ liệu> <Tên> = new LinkedList<>();
```

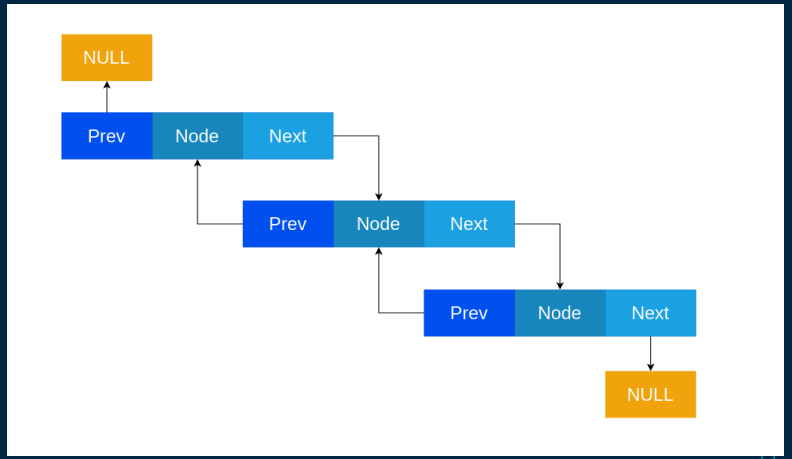
LinkedList

Trong Java, có 2 loại LinkedList được sử dụng để lưu trữ các phần tử:

Singly Linked List: mỗi node trong danh sách này lưu trữ dữ liệu của node và con trỏ trỏ tới node tiếp theo.



Doubly Linked List: mỗi node có hai tham chiếu - một tham chiếu đến node tiếp theo và một tham chiếu đến node trước đó.



Một số phương thức của LinkedList

Method	Description
boolean add(Object o)	Thêm phần tử được chỉ định vào cuối danh sách
boolean contains(Object o)	Trả về true nếu danh chứa phần tử được chỉ định và trả về false nếu ngược lại
void add (int index, Object element)	Chèn phần tử đã được chỉ định vào vị trí index
int size()	Trả về kích thước của LinkedList
boolean remove(Object o)	Xóa phần tử được chỉ định khỏi danh sách
int indexOf(Object element)	Trả về index xuất hiện đầu tiên của phần tử được chỉ định, nếu trong danh sách không có phần tử đó thì trả về -1
int lastIndexOf(Object element)	Trả về index xuất hiện cuối cùng của phần tử được chỉ định, nếu trong danh sách không có phần tử đó thì trả về -1

Vector

Vector tương tự như ArrayList, có thể chứa các phần tử trùng lặp và được truy cập thông qua index. Tuy nhiên Vector khác với ArrayList ở chỗ nó được đồng bộ hóa còn ArrayList thì không. Do cơ chế đồng bộ hóa khiến hiệu suất của Vector kém hơn nên nó ít được sử dụng hơn so với ArrayList.

Khai báo Vector:

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

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

Một số phương thức trong Vector

Method	Description
boolean add(Object o)	Thêm phần tử được chỉ định vào cuối Vector
void clear()	Xóa tất cả phần tử khỏi Vector
void add(int index, Object element)	Chèn phần tử đã được chỉ định vào vị trí index
boolean remove(Object o)	Xóa phần tử được chỉ định khỏi Vector
boolean contains(Object element)	Trả về true nếu Vector chứa phần tử được chỉ định và trả về false nếu ngược lại
int size()	Trả về kích thước của Vector
int indexOf(Object element)	Trả về index xuất hiện đầu tiên của phần tử được chỉ định, nếu trong Vector không có phần tử đó thì trả về -1
int lastIndexOf(Object element)	Trả về index xuất hiện cuối cùng của phần tử được chỉ định, nếu trong Vector không có phần tử đó thì trả về -1

Cảm ơn mọi người đã lắng nghe