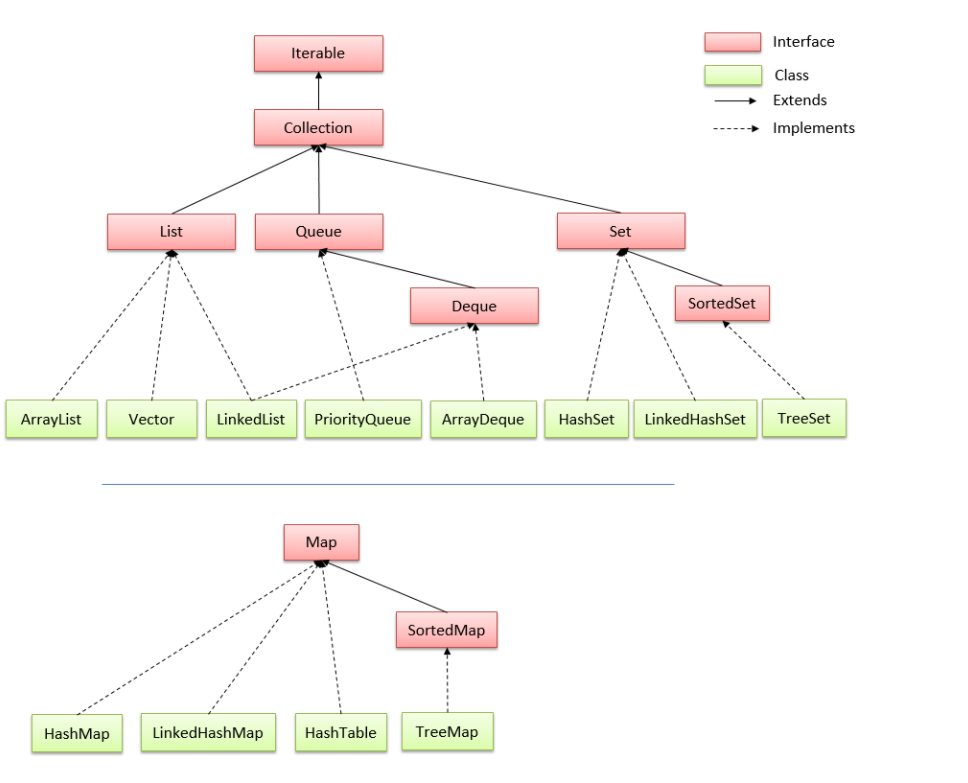
COLLECTION(S)

1. **Collection vs Collections**

"Collection" và "Collections" trong java là hai khái niệm khác nhau.

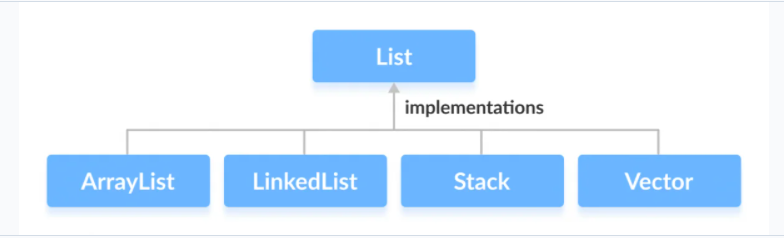
**Collections trong java** là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa,... có thể được thực hiện bởi Java Collections.

**Collection trong java** là một root interface trong hệ thống cấp bậc Collection. Java Collection cung cấp nhiều interface (Set, List, Queue, Deque vv) và các lớp (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet vv).



II. Collection phổ biến

a)List:

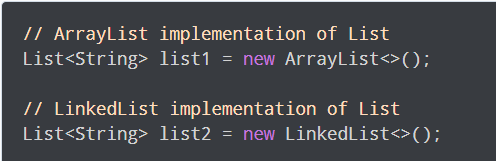


**+) List** là một interface trong java. Nó chứa các phương thức để chèn và xóa các phần tử dựa trên chỉ số index.

### +) Khai báo interface List trong java

|  | public interface List<E> extends Collection<E> |
| --- | --- |

**+) Syntax**

****

Các phương thức của interface List trong java

| **Method** | **Description** |
| --- | --- |
| void add(int index,Object element) | Nó được sử dụng để chèn các phần tử vào list tại chỉ số index. |
| boolean addAll(int index,Collection c) | Nó được sử dụng để chèn tất cả các yếu tố của c vào danh sách tại chỉ số index. |
| object get(int index) | Nó được sử dụng để trả về đối tượng được lưu trữ tại chỉ số index trong list. |
| object set(int index,Object element) | Nó được sử dụng để gán phần tử cho vị trí được chỉ định index trong list. |
| object remove(int index) | Nó được sử dụng để xóa các phần tử tại vị trí có chỉ số index và trả về phần tử đã xóa. |
| ListIterator listIterator() | Nó được sử dụng để trả về một Iterator mà bắt đầu từ phần tử đầu tiên của list. |
| ListIterator listIterator(int index) | Nó được sử dụng để trả về một Iterator mà phần tử bắt đầu từ chỉ số index chỉ định. |

b)ArrayList

**Lớp ArrayList trong java** là một lớp kế thừa lớp AbstractList và triển khai của List Interface trong Collections Framework nên nó sẽ 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.

Những điểm cần ghi nhớ về ArrayList:

* Lớp ArrayList trong java có thể chứa các phần tử trùng lặp.
* Lớp ArrayList duy trì thứ tự của phần tử được thêm vào.
* Lớp ArrayList là không đồng bộ (non-synchronized).
* Lớp ArrayList cho phép truy cập ngẫu nhiên vì nó lưu dữ liệu theo chỉ mục.
* Lớp ArrayList trong java, thao tác chậm vì cần nhiều sự dịch chuyển nếu bất kỳ phần tử nào bị xoá khỏi danh sách.

Lớp java.util.ArrayList được khai báo như sau:

public class ArrayList<E> extends AbstractList<E>

        implements List<E>, RandomAccess, Cloneable, java.io.Serializable

## **Khởi tạo ArrayList trong java**

Có 2 kiểu khởi tạo ArrayList là non-generic và generic

|  | ArrayList list = new ArrayList(); // non-generic - kiểu cũ  ArrayList<String> list = new ArrayList<String>(); // generic - kiểu mới |
| --- | --- |

## **Constructor của lớp ArrayList**

| **Constructor** | **Mô tả** |
| --- | --- |
| ArrayList() | Nó được sử dụng để khởi tạo một danh sách mảng trống. |
| ArrayList(Collection c) | Nó được sử dụng để xây dựng một danh sách mảng được khởi tạo với các phần tử của collection c. |
| ArrayList(int capacity) | Nó được sử dụng để xây dựng một danh sách mảng mà có dung lượng ban đầu được chỉ định. |

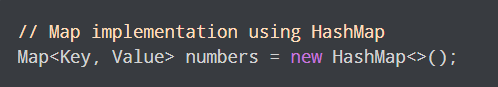
## **Phương thức của lớp ArrayList**

| **Phương thức** | **Mô tả** |
| --- | --- |
| boolean add(Object o) | Nó được sử dụng để nối thêm phần tử được chỉ định vào cuối một danh sách. |
| void add(int index, Object element) | Nó được sử dụng để chèn phần tử element tại vị trí index vào danh sách. |
| boolean addAll(Collection c) | Nó được sử dụng để nối tất cả các phần tử trong collection c vào cuối của danh sách, theo thứ tự chúng được trả về bởi bộ lặp iterator. |
| boolean addAll(int index, Collection c) | Nó được sử dụng để chèn tất cả các phần tử trong collection c vào danh sách, bắt đầu từ vị trí index. |
| void retainAll(Collection c) | Nó được sử dụng để xóa những phần tử không thuộc collection c ra khỏi danh sách. |
| void removeAll(Collection c) | Nó được sử dụng để xóa những phần tử thuộc collection c ra khỏi danh sách. |
| int indexOf(Object o) | Nó được sử dụng để 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. |
| int lastIndexOf(Object o) | Nó được sử dụng để 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. |
| Object[] toArray() | Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. |
| Object[] toArray(Object[] a) | Nó được sử dụng để trả về một mảng chứa tất cả các phần tử trong danh sách này theo đúng thứ tự. |
| Object clone() | Nó được sử dụng để trả về một bản sao của ArrayList. |
| void clear() | Nó được sử dụng để xóa tất cả các phần tử từ danh sách này. |
| void trimToSize() | Nó được sử dụng để cắt dung lượng của thể hiện ArrayList này là kích thước danh sách hiện tại. |
| boolean contains(element) | Kết quả trả về là true nếu tìm thấy element trong danh sách, ngược lại trả về false. |

c)Map

Trong java, map được sử dụng để lưu trữ và truy xuất dữ liệu theo cặp key và value. Mỗi cặp key và value được gọi là mục nhập (entry). Map trong java chỉ chứa các giá trị key duy nhất. Map rất hữu ích nếu bạn phải tìm kiếm, cập nhật hoặc xóa các phần tử trên dựa vào các key.

* Syntax



## **Các phương thức hữu ích của Map interface**

| **Phương thức** | **Mô tả** |
| --- | --- |
| Object put(Object key, Object value) | Nó được sử dụng để chèn một mục nhập trong map hiện tại. |
| void putAll(Map map) | Nó được sử dụng để chèn map chỉ định vào map hiện tại. |
| Object remove(Object key) | Nó được sử dụng để xóa một mục nhập của key được chỉ định. |
| Object get(Object key) | Nó được sử dụng để trả lại giá trị cho khoá được chỉ định. |
| boolean containsKey(Object key) | Nó được sử dụng để tìm kiếm key được chỉ định từ map hiện tại. |
| Set keySet() | Nó được sử dụng để trả đối tượng Set có chứa tất cả các keys. |
| Set entrySet() | Nó được sử dụng để trả lại đối tượng Set có chứa tất cả các keys và values. |

## **Map.Entry Interface**

Entry là một interface con của Map. Vì vậy, chúng ta được truy cập nó bằng tên Map.Entry. Nó cung cấp các phương pháp để truy xuất các key và value.

#### **Các phương thức của Map.Entry interface**

| **Phương thức** | **Mô tả** |
| --- | --- |
| Object getKey() | Nó được dùng để lấy key. |
| Object getValue() | Nó được sử dụng để lấy value. |

III. List vs ArrayList

List và ArrayList là các thành viên của khung Collection. List là một tập hợp các phần tử trong một chuỗi trong đó mỗi phần tử là một đối tượng và các phần tử được truy cập bởi vị trí (chỉ mục). ArrayList tạo ra một mảng động của các đối tượng tăng hoặc giảm kích thước bất cứ khi nào cần thiết. Sự khác biệt chính giữa List và ArrayList là **List** là một giao diện và **ArrayList** là một lớp.

Biểu đồ so sánh

| **Cơ sở để so sánh** | **List** | ArrayList |
| --- | --- | --- |
| Căn bản | List là một giao diện | ArrayList là một Class Collection tiêu chuẩn. |
| Cú pháp | Danh sách giao diện | lớp ArrayList |
| Mở rộng / Thực hiện | Giao diện danh sách mở rộng Bộ sưu tập Khung. | ArrayList mở rộng AbstractList và thực hiện Giao diện danh sách. |
| Không gian tên | System.Collections.Generic. | Hệ thống. Bộ sưu tập. |
| Công việc | Nó được sử dụng để tạo một danh sách các yếu tố (đối tượng) được liên kết với số chỉ mục của chúng. | ArrayList được sử dụng để tạo một mảng động có chứa các đối tượng. |

+ Thông thường, ArrayList là lựa chọn mặc định cho nhiều nhà phát triển khi họ cần List implementation. Thực tế thì nó sẽ là một lựa chọn tốt khi các hoạt động đọc dữ liệu diễn ra nhiều hơn số lần ghi.

Đôi khi chúng ta cần đọc và ghi thường xuyên như nhau. Nếu chúng ta có ước tính về số lượng phần tử tối đa có thể, thì việc sử dụng ArrayList vẫn có ý nghĩa. Nếu đúng như vậy, chúng ta có thể khởi tạo ArrayList với dung lượng ban đầu:

int possibleUpperBound = 10\_000;

List**<**String**>** items = **new** ArrayList**<>(**possibleUpperBound**)**;

Việc này sẽ giúp chúng ta giảm thiểu tối đa việc phải phân bổ lại array khi kích thước của nó không còn đủ để lưu trữ.

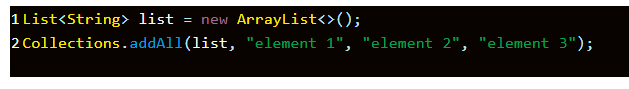
Ngoài ra, array được đánh chỉ mục với giá trị kiểu int trong Java, vì vậy số lượng phần tử tối đa được lưu trữ bên trong ArrayList không vượt quá 2^32.

+LinkedList phù hợp hơn ArrayList khi các hoạt động thêm phần tử xảy ra thường xuyên hơn là đọc. Ngoài ra, nó có thể được sử dụng trong các trường hợp việc đọc dữ liệu đa số là đọc phần tử đầu tiên và cuối cùng.

**IV. Method thường dùng trong Collections**

## 1. Phương thức addAll

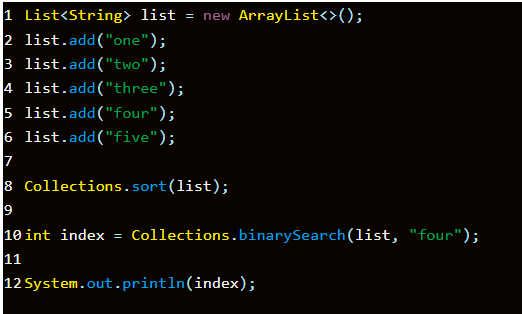
Chúng ta sử dụng phương thức addAll để add một hoặc nhiều phần tử vào trong một list có sẵn như sau



## 2. Phương thức BinarySearch

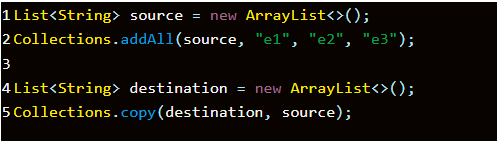
Chúng ta sử dụng binary search dùng để tìm kiếm phần tử trong List và sử dụng thuật toán Binary Search để tìm phần tử. Khi chúng ta áp dụng binary search cho List thì bắt buộc List phải được sắp xếp trước khi sử dụng Binary Search.

3. Chúng ta sử dụng method Collections.sort(list) để sắp xếp các phần tử trong List.

****

## 4. Phương thức Copy

Chúng ta sử dụng phương thức Copy để copy tất cả các giá trị từ một list vào một list khác.

****

## 5. Phương thức Reverse

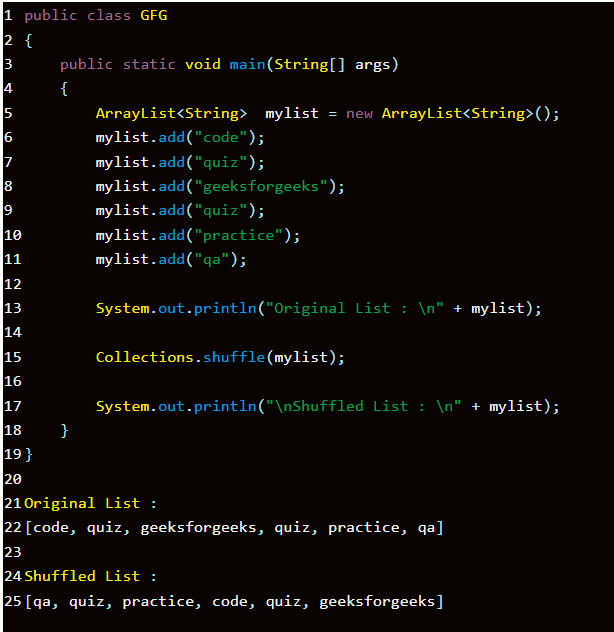
Chúng ta muốn thực hiện đảo ngược thứ tự vị trí trong các phần tử trong List thì sử dụng phương thức reverse() như sau

****

* Kết quả in ra sẽ là 3,2,1

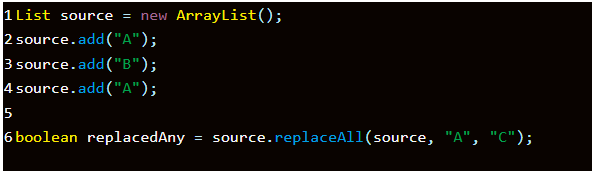
## 6. Phương thức shuffle

Chúng ta muốn xáo trộn các phần tử trong List thì sử dụng phương thức shuffle như sau

****

## 7. Phương thức ReplaceAll

Chúng ta sử dụng ReplaceAll để thay thế các phần tử có sẵn trong List bằng một phần tử khác.



* Kết quả ta nhận được sẽ là C, B, C