MỤC LỤC

[I. Java.util 2](#_Toc443387294)

[1. List 2](#_Toc443387295)

[1.1. ArrayList Trong Java 3](#_Toc443387296)

[1.2. LinkedList trong Java 8](#_Toc443387297)

[1.3. Vector trong Java 12](#_Toc443387298)

[1.4. Calendar trong Java 14](#_Toc443387299)

# I. Java.util

# 1. List

List là một interface con của Collection nó có đầy đủ các tính năng của Collection đồng thời có thêm một số tính chất đặc biệt:

Cho phép phần tử trùng lặp

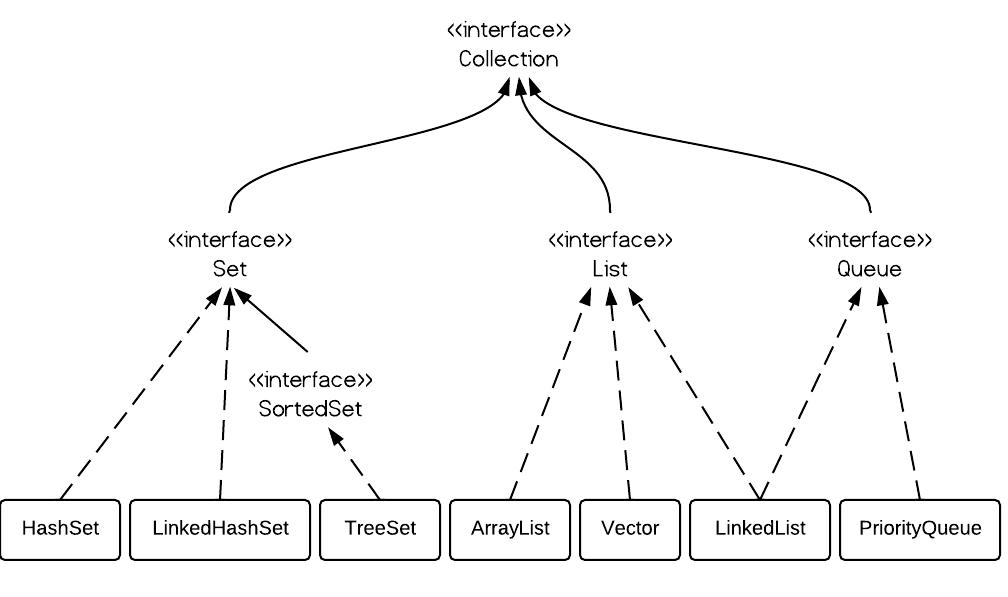
Cho phép 0 hoặc nhiều phần tử null.

Là một tập hợp có tuần tự.

Các phần tử có thể được chèn hoặc truy cập thông qua vị trí của chúng trong danh sách.

Các thư viện đi kèm List: ArrayList, LinkedList, Stack, ArrayDeque, Vector,…

Để hiểu và sử dụng được Java Collections API một cách hiểu quả, ta hãy nhìn mô hình dưới đây để có cái nhìn tổng quát



|  |  |
| --- | --- |
| http://tutorials.jenkov.com/images/java-collections/collection-overview.png | http://tutorials.jenkov.com/images/java-collections/map-overview.png |

## 1.1. ArrayList Trong Java

Việc sử dụng mảng (array) có một vài nhược điểm mà các lập trình viên thường gặp phải như sau:

Nếu khai báo kích thước mảng quá nhỏ thì sẽ dẫn đến thiếu bộ nhớ khi lưu trữ. Nếu khai báo lớn thì thừa bộ nhớ (quá lãng phí). Các phần tử trong mảng lại có cùng kiểu dữ liệu.

ArrayList đã tránh những nhược điểm này. ArrayList là một mảng động.

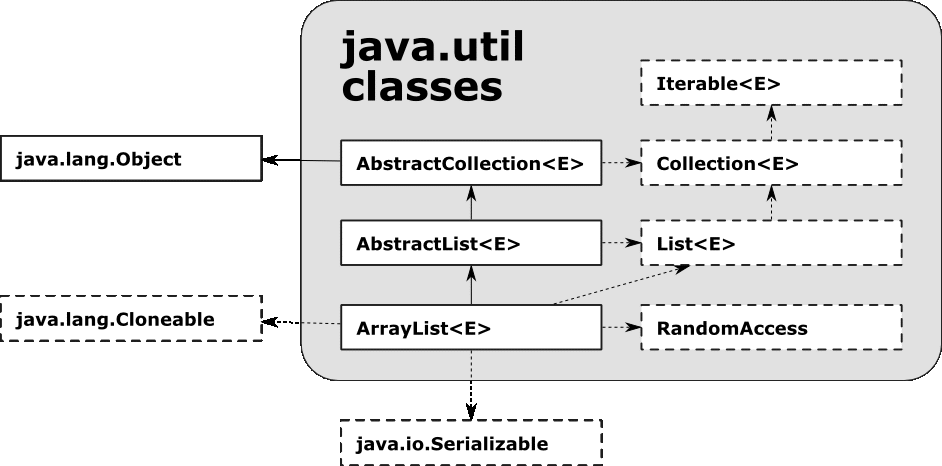
ArrayList trong Java kế thừa AbstractList và thực thi List Interface.

ArrayList được ưa chuộng trong Java vì chức năng và tính linh hoạt mà nó cung cấp. Hầu hết các lập trình viên đều chọn ArrayList thay thế cho cách dùng Array truyền thống.

Kiểu dữ liệu mảng truyền thống Array bị giới hạn với kích thước xác định. Nếu một số phần tử trong mảng bị xóa đi thì bộ nhớ chứa các phần tử đó không được thu hồi. Ngược lại kích thước của ArrayList có thể tăng hoặc giảm tùy thích.

Các mảng truyền thống có độ dài cố định. Sau khi các mảng được tạo, chúng không thể tăng hoặc giảm kích cỡ.

ArrayList được tạo với một kích cỡ ban đầu. Khi kích cỡ này bị vượt, collection tự động được tăng. Khi các đối tượng bị gỡ bỏ, ArrayList có thể bị giảm kích cỡ.



Một số phương thức thường được sử dụng trong ArrayList:

1. add( Object o): Phương thức thêm phần tử vào cuối mảng
2. add(int index, Object o): Phương thức thêm phần tử vào chỉ mục định trước
3. remove(Object o): Xóa phần tử trong mảng
4. remove(int index): Xóa phần tử theo chỉ mục
5. set(int index, Object o): Phương thức cập nhật phần tử trong mảng, nó thay thế phần tử ở chỉ mục đã cho bằng phần tử mới.
6. int indexOf(Object o): Phương thức lấy chỉ mục của một phần tử, nếu kết quả trả về -1 thì phần tử không tồn tại trong mảng.
7. Object get(int index): Phương thức lấy phần tử ở chỉ mục đã cho trong mảng.
8. int size(): Phương thức xác định kích thước mảng.
9. boolean contains(Object o): Phương thức kiểm tra phần tử có tồn tại trong mảng hay không.
10. clear(): Phương thức xóa tất cả các phần tử trong mảng.

Ví dụ:

**package** arrayListExample;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**public** **class** ArrayListExamples {

**public** **static** **void** main(String[] args) {

// Creating an empty array list

ArrayList<String> list = **new** ArrayList<String>();

// Adding items to array list

list.add("Student A");

list.add("Student B");

list.add("Student C");

list.add(3, "Student D");

list.add("Student E");

// Show the contents of the array list

System.***out***.println("The arraylist contains: " + list);

// checking index of an item

**int** position = list.indexOf("Student D");

System.***out***.println("The index of Student D is: " + position);

// checking list isEmpty

**boolean** check = list.isEmpty();

System.***out***.println("Checking if the arraylist is empty: " + check);

// getting the size of the list

**int** size = list.size();

System.***out***.println("The size of the list is: " + size);

// checking if an element is included to the list

**boolean** element = list.contains("Student F");

System.***out***.println("ArrayList contains Student F: " + element);

// getting the element in a specific position

String student = list.get(0);

System.***out***.println("The student is at the index 0 is: " + student);

// loop using index and size list

System.***out***.println("Loop using index and size list");

**for** (**int** i = 0; i < list.size(); i++) {

System.***out***.println("Index: " + i + " - Student: " + list.get(i));

}

// using foreach loop

System.***out***.println("Using foreach loop");

**for** (String str : list) {

System.***out***.println("Student is: " + str);

}

// using iterator

System.***out***.println("Using iterator");

**for** (Iterator<String> iterator = list.iterator(); iterator.hasNext();) {

System.***out***.println("Student is: " + iterator.next());

}

// replacing an element

list.set(2, "New Student");

System.***out***.println("The arraylist after replacement is: " + list);

// removing students

list.remove(0);

list.remove("Student D");

System.***out***.println("Contents of arraylist: " + list);

}

}

**Kết quả:**

The arraylist contains:

[Student A, Student B, Student C, Student D, Student E]

The index of Student D is: 3

Checking if the arraylist is empty: false

The size of the list is: 5

ArrayList contains Student F: false

The student is at the index 0 is: Student A

Loop using index and size list

Index: 0 - Student: Student A

Index: 1 - Student: Student B

Index: 2 - Student: Student C

Index: 3 - Student: Student D

Index: 4 - Student: Student E

Using foreach loop

Student is: Student A

Student is: Student B

Student is: Student C

Student is: Student D

Student is: Student E

Using iterator

Student is: Student A

Student is: Student B

Student is: Student C

Student is: Student D

Student is: Student E

The arraylist after replacement is:

[Student A, Student B, New Student, Student D, Student E]

Contents of arraylist: [Student B, New Student, Student E]

## 1.2. LinkedList trong Java

Lớp LinkedList trong Java kế thừa lớp AbstractSequentialList và thực thi List Interface. Nó cung cấp một cấu trúc dữ liệu linked-list (danh sách liên kết).

****

LinkedList có cơ chế giống như 1 sợi dây xích, phần tử 0 móc vào phần tử 1, rồi 1 móc vào 2, 2 móc vào 3,… không phải khai báo thừa ra như ArrayList nên bộ nhớ tốn ít hơn.

Thời gian gọi tới 1 phần tử thứ i thì chậm dần đều từ ngoài vào trong: tức là phần tử ngoài cùng của sợi dây xích thì gọi tới rất nhanh, nhưng càng vào giữa thì càng chậm dần

Thời gian xóa thì nhanh hơn so với ArrayList, nó chỉ mất chủ yếu là thời gian tìm tới phần tử thứ i thôi, còn sau đó chỉ việc tách nó ra khỏi sợi dây xích và nối 2 đầu dây lại

Thêm phần tử mới cũng nhanh hơn ArrayList, chỉ việc móc thêm vào đuôi cái xích là xong

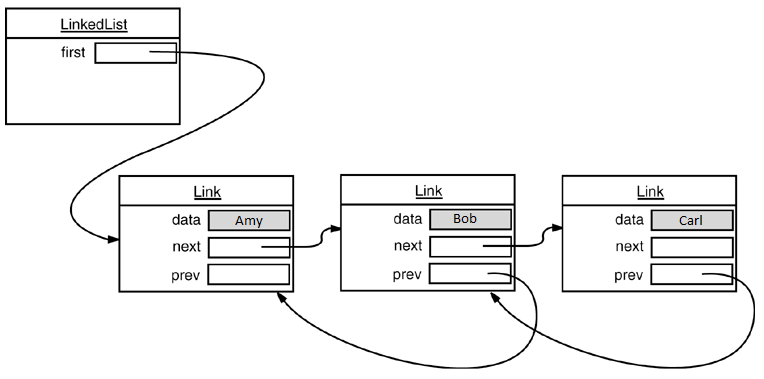
Ngoài các method như ArrayList, LinkedList còn có thêm các method:

1. addFirst()
2. addLast()
3. getFirst()
4. getLast()
5. removeFirst()
6. removeLast()
7. peek() = getFirst()
8. poll() = getFirst() + removeFirst()

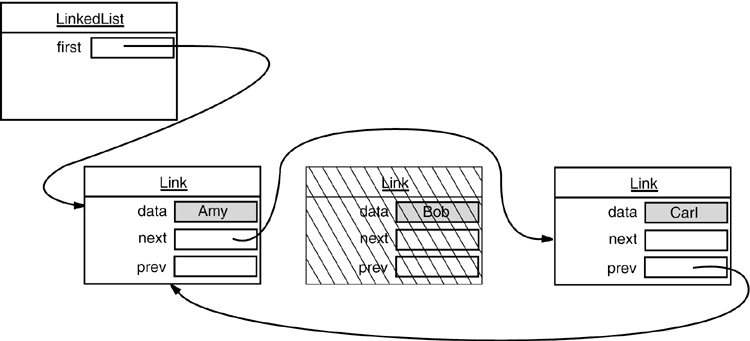
***Khi nào dùng ArrayList, khi nào dùng LinkedList?***

Nếu chương trình thường xuyên phải truy nhập trực tiếp dữ liệu trong list à sử dụng ArrayList

Nếu chương trình thường xuyên phải thêm hoặc xóa 1 phần tử trong list(đặc biệt là thêm hoặc xóa ở giữa) thì nên sử dụng LinkedList



Khi có một phần tử bất kì bị remove: Tôi remove **Bob** nhé:



Ví dụ:

**package** linkedListinJava;

**import** java.util.LinkedList;

**import** java.util.ListIterator;

**public** **class** LinkedListExample {

**public** **static** **void** main(String[] args) {

// create a LinkedList

LinkedList<String> list = **new** LinkedList<String>();

// add elements to the linked list

list.add("Tuesday");

list.add("Wednesday");

list.add("Thursday");

list.add("Friday");

// create a list iterator

ListIterator<String> listIterator = list.listIterator();

System.***out***.println("-----LinkedList-----");

**while** (listIterator.hasNext()) {

System.***out***.println(listIterator.next());

}

// add elements in the beginning and in the end of the list

list.addFirst("Monday");

list.addLast("Saturday");

System.***out***.println("-----After adding-----");

listIterator = list.listIterator();

**while** (listIterator.hasNext()) {

System.***out***.println(listIterator.next());

}

System.***out***.println("Check if list contains the Friday: " + list.contains("Friday"));

System.***out***.println("Position of Wednesday is: " + list.indexOf("Wednesday"));

System.***out***.println("Get element in 6th position: " + list.get(5));

**int** size = list.size();

System.***out***.println("The size of list is: " + size);

System.***out***.println("-----Reversing list-----");

listIterator = list.listIterator(size);

**while**(listIterator.hasPrevious()) {

System.***out***.println(listIterator.previous());

}

// remove element from list

list.remove("Thursday");

list.removeFirst();

list.removeLast();

System.***out***.println("---LinkedList after removing---");

listIterator = list.listIterator();

**while** (listIterator.hasNext()) {

System.***out***.println(listIterator.next());

}

}

}

Kết quả:

-----LinkedList-----

Tuesday

Wednesday

Thursday

Friday

-----After adding-----

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Check if list contains the Friday: true

Position of Wednesday is: 2

Get element in 6th position: Saturday

The size of list is: 6

-----Reversing list-----

Saturday

Friday

Thursday

Wednesday

Tuesday

Monday

-----LinkedList after removing-----

Tuesday

Wednesday

Friday

## 1.3. Vector trong Java

Lớp vector tương tự như ArrayList và nó cũng thực thi các mảng động. Lớp vector lưu mảng các đối tượng và kích thước của mảng này có thể tăng hoặc giảm. Chúng ta có thể truy cập các phần tử của Vector bằng cách sử dụng vị trí của phần tử đó.

Sự khác nhau giữa Vector và ArrayList là các phương thức của Vector dùng cơ chế đồng bộ (synchronised) và thread-safe. Cho nên Vector thường chạy chậm, các phương thức của ArrayList chạy nhanh hơn.

**Các phương thức cơ bản của lớp này:**

**addElement(Object obj)** phương thức này thêm phần tử vào vị trí cuối cùng của Vector và kích thước của Vector sẽ tăng lên 1.

**insertElementAt(Object obj, int index)** chèn một phần tử tại vị trí xác định.

**setElementAt(Object obj, int index)**đặt giá trị cho một phần tử tại một vị trí

**toArray()**trả về một mảng chứa tất cả các phần tử trong Vector.

**elementAt(int pos)**  lấy một đối tượng được lưu trữ xác định bởi vị trí.

**removeElement(Object obj)** xóa một phần tử xuất hiện đầu tiên trong Vector có giá trị bằng với Object.

**setSize(int newSize)** thiết lập kích cỡ của Vector này

**int size()** trả về số phần tử trong Vector này

**int lastIndexOf(Object elem, int index)** tìm kiếm ngược về sau cho đối tượng đã cho, bắt đầu từ index đã xác định, và trả về một chỉ mục

Ví dụ:

**package** vecTorExample;

**import** java.util.Enumeration;

**import** java.util.Vector;

**public** **class** VectorExample {

**public** **static** **void** main(String[] args) {

// Create a Vector and populate it with elements

Vector<Object> vector = **new** Vector<Object>();

vector.addElement(**new** String("Task 1"));

vector.addElement(**new** String("Task 2"));

vector.addElement(**new** String("Task 3"));

vector.addElement("Task 4");

vector.add("Task 5");

System.***out***.println(vector);

System.***out***.println("Capacity: " + vector.capacity());

System.***out***.println("Vector Size: " + vector.size());

// Clone another vector

System.***out***.println(vector.clone());

// Replacing an element at the specified index of Vector

vector.set(1, "Finish");

System.***out***.println(vector);

// checking if an element is included to the list

**boolean** check = vector.contains("Task 3");

System.***out***.println("Vector contains Task 3: " + check);

Enumeration<Object> enumeration = vector.elements();

System.***out***.println("-----Vector Elements-----");

**while** (enumeration.hasMoreElements()) {

System.***out***.println(enumeration.nextElement());

}

// Remove the specific object from the Vector

vector.remove("Task 4");

System.***out***.println("After removing: " + vector);

Object object = vector.elementAt(3);

System.***out***.println("Position 3 is: " + object);

}

}

Kết quả:

[Task 1, Task 2, Task 3, Task 4, Task 5]

Capacity: 10

Vector Size: 5

[Task 1, Task 2, Task 3, Task 4, Task 5]

[Task 1, Finish, Task 3, Task 4, Task 5]

Vector contains Task 3: true

-----Vector Elements-----

Task 1

Finish

Task 3

Task 4

Task 5

After removing: [Task 1, Finish, Task 3, Task 5]

Position 3 is: Task 5

## 1.4. Calendar trong Java

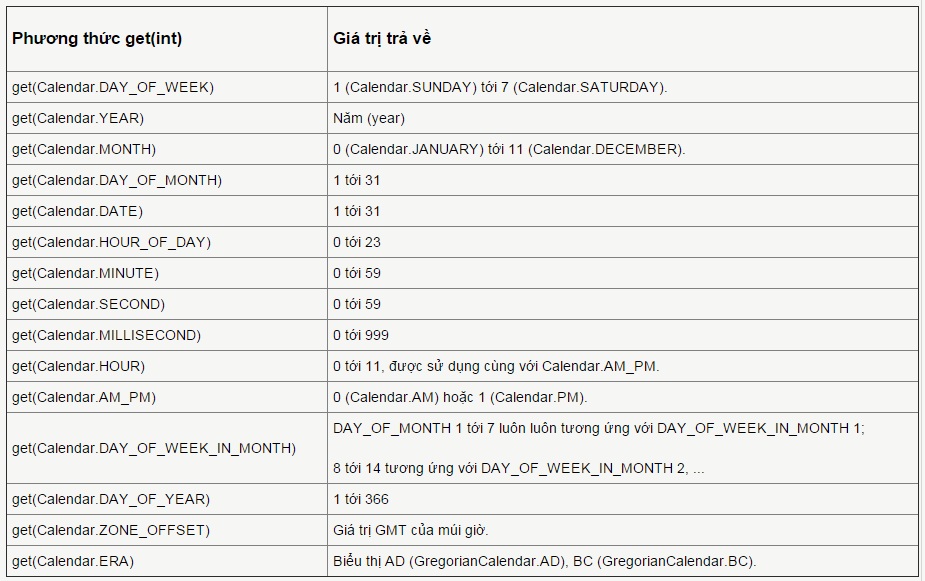
Sơ lược về các bộ lịch:

**Calendar** là class mô phỏng một hệ thống Lịch.

**Gregorian Calendar**:Đây chính là Dương Lịch, còn gọi lịch Thiên chúa giáo, là lịch quốc tế. Nó được sử dụng rộng rãi nhất được đặt tên theo Đức Giáo Hoàng Gregory XIII, người đã giới thiệu nó vào năm 1582.

**Buddhist Calendar**: Đây là một bộ lịch phật giáo, thường được sử dụng tại một số nước Đông Nam Á trước kia như Thái Lan, Lào, Campuchia, cũng như Sri Lanka. Hiện nay lịch này được sử dụng trong các lễ hội phật giáo. Và không còn quốc gia nào sử dụng lịch này một cách chính thức, các quốc gia này đã đổi sang sử dụng **Gregorian Calendar**.

**Japanese Imperial Calendar**: Đây là bộ lịch truyền thống của Nhật bản, hiện nay Nhật bản đã chuyển sang sử dụng dương lịch (**Gregorian Calendar**), tuy nhiên bộ lịch truyền thống vẫn được sử dụng một cách không chính thức.



Ví dụ:

**package** calendarExample;

**import** java.util.Calendar;

**import** java.util.TimeZone;

**public** **class** CalendarExample {

**public** **static** **void** main(String[] args) {

// Create calendar instance

Calendar calendar = Calendar.*getInstance*();

**int** year = calendar.get(Calendar.***YEAR***);

// get time date

**int** month = calendar.get(Calendar.***MONTH***);

**int** day = calendar.get(Calendar.***DAY\_OF\_MONTH***);

**int** hour = calendar.get(Calendar.***HOUR\_OF\_DAY***);

**int** minute = calendar.get(Calendar.***MINUTE***);

**int** second = calendar.get(Calendar.***SECOND***);

**int** millis = calendar.get(Calendar.***MILLISECOND***);

**int** hour12 = calendar.get(Calendar.***AM\_PM***);

**int** GMT = calendar.get(Calendar.***ZONE\_OFFSET***);

// get current TimeZone

TimeZone timeZone = calendar.getTimeZone();

System.***out***.println("Year: " + year);

System.***out***.println("Month: " + (month+1));

System.***out***.println("Day: " + day);

System.***out***.println("Hour: " + hour);

System.***out***.println("Minute: " + minute);

System.***out***.println("Second: " + second);

System.***out***.println("Milli Second: " + millis);

System.***out***.println("AM/ PM: " + hour12);

System.***out***.println("GMT: " + GMT);

System.***out***.println("Current week of this month: " + calendar.get(Calendar.***WEEK\_OF\_MONTH***));

System.***out***.println("Current week of this year: " + calendar.get(Calendar.***WEEK\_OF\_YEAR***));

System.***out***.println("Current TimeZone is : " + timeZone.getDisplayName());

System.***out***.println(hour + ":" + minute + ":" + second + ":" + millis + "," + day + "/" + (month+1) + "/" + year);

}

}

Kết quả:

Year: 2016

Month: 2

Day: 16

Hour: 17

Minute: 6

Second: 8

Milli Second: 817

AM/ PM: 1

GMT: 25200000

Current week of this month: 3

Current week of this year: 8

Current TimeZone is : Indo china Time

17:6:8:817,16/2/2016