1. Collections

## [Khái niệm về Collections (tập hợp)](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h2-0)

**Collections là một tập các lớp dùng để lưu trữ danh sách và có khả năng tự co dãn khi danh sách đó thay đổi**, ví dụ như khi chúng ta thêm, sửa, xóa, chèn phần tử trong danh sách đó. Ngoài ra, Collections còn được dùng để lưu trữ, truy xuất, tương tác với dữ liệu và truyền dữ liệu giữa các phương thức với nhau

Một đặc điểm rất quan trong là khi sử dụng Collections đó là chúng ta không cần phải khai báo trước số lượng phần tử. Chính đặc điểm này đã khắc phục được hạn chế về kích thước khi khai báo mảng trong Java.

## [Collections Framework](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h2-1)

“Framework là một tập hợp các thư viện (Library) đã được đóng gói để hỗ trợ phát triển ứng dụng dựa trên Framework đó. Đồng thời, Framework cung cấp các nguyên tắc, cấu trúc của ứng dụng mà chúng ta phải tuân thủ theo nó."

**Collections Framework** được hiểu như sau: "**Một Java Collections Framework là một tập hợp các lớp (class) và các interface dùng để hỗ trợ việc thao tác trên tập các đối tượng**"

Collection framework được thiết kế với mục đích như sau:

* Framework phải là hiệu năng cao. Sự triển khai cho các tập hợp cơ bản (các mảng động, linked list, tree và hashtable) được sử dụng với hiệu quả cao.
* Framework phải cho phép các kiểu tập hợp khác nhau để làm việc theo một cách tương tự như nhau với độ phân hóa ở mức cao.
* Kế thừa và/hoặc tìm hiểu với các tập hợp phải là dễ dàng.

| **Loại thành phần** | **Mô tả** |
| --- | --- |
| Interfaces | Kiểu dữ liệu trừu tượng (*abstract*) biểu diễn Collections . |
| Implementations | Là sự triển khai các Interface, ví dụ như các Class. |
| Algorithms (*các thuật toán*) | Là các phương thức dùng để thực thi các phép toán như tìm kiếm và sắp xếp trên các đối tượng mà triển khai các Interface. |

## [Interface Collections và Class Collections](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h2-2) :

### [Interface Collections](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h3-0)

Collection Interface định nghĩa những phương thức cơ bản khi làm việc với tập hợp, đây là gốc cũng là nền móng để từ đó xây dựng lên cả bộ thư viện Java Collection Framework. Collection Interface được kế thừa từ Iterable Interface nên các bạn có thể dễ dàng duyệt qua từng phần tử thông qua việc sử dụng Iterator.

Trong Interface Collections chúng ta có các Interface chính như: List Interface, Set, SortedSet, Map và SortedMap.

| **Tên Interface** | **Đặc điểm khái quát** |
| --- | --- |
| List Interface | Các phần tử trong List Interface được sắp xếp có thứ tự và có thể có giá trị giống nhau. |
| Set Interface | Các phần tử trong Set là duy nhất (*nghĩa là giá trị của các phần tử này không được giống nhau*). |
| SortedSet | Là 1 dạng riêng của Set Interface, trong đó giá trị của các phần tử mặc định được sắp xếp tăng dần. |
| Map | Giá trị của mỗi phần tử trong Map bao gồm 2 phần đó là khóa (*ke*y) và giá trị tương ứng của key đó (*value*) và khóa của các phần tử này là duy nhất. |
| SortedMap | Là 1 dạng riêng của Map Interface, trong đó giá trị key được sắp xếp tăng dần. |

### [Class Collections](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h3-1)

Java cung cấp một tập hợp các lớp tiêu chuẩn dùng để triển khai các Interface Collection. Trong Class Collections chúng ta có rất nhiều loại nhưng trong phạm vi của series Lập trình Java căn bản này thì chúng ta chỉ cần nắm 6 loại chính sau: LinkedList, ArrayList, HashSet, TreeSet, HashMap và TreeMap.

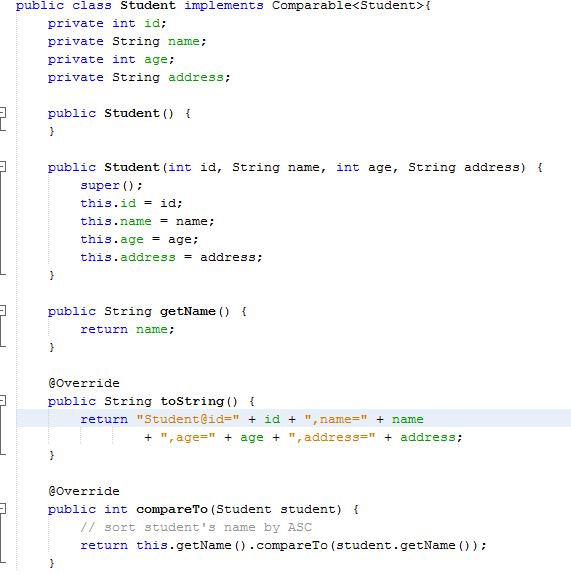
| **Tên Class** | **Đặc điểm khái quát** |
| --- | --- |
| LinkedList | Là 1 cấu trúc dữ liệu lưu trữ các phần tử dưới dạng danh sách. Các phần tử trong LinkedList được sắp xếp có thứ tự và có thể có giá trị giống nhau. |
| ArrayList | Là kiểu danh sách sử dụng cấu trúc mảng để lưu trữ phần tử. Thứ tự các phần tử dựa theo thứ tự lúc thêm vào và giá trị của các phần tử này có thể trùng nhau. |
| HashSet | Thứ tự các phần tử trong HashSet không dựa theo thứ tự lúc thêm vào và giá trị của các phần tử này là duy nhất. |
| TreeSet | Các phần tử trong TreeSet mặc định được sắp xếp tăng dần và giá trị của các phần tử này là duy nhất. |
| HashMap | Giá trị của mỗi phần tử trong HashMap bao gồm 2 phần đó là khóa (*ke*y) và giá trị tương ứng của key đó (*value*) và khóa của các phần tử này là duy nhất. HashMap cho phép truy xuất trực tiếp dữ liệu bằng khóa duy nhất của nó. |
| TreeMap | Giá trị của mỗi phần tử trong TreeMap bao gồm 2 phần đó là khóa (*key*) và giá trị tương ứng của key đó (value) và khóa của các phần tử này là duy nhất. Giá trị của các phần tử trong TreeMap được sắp xếp tăng dần. |

## [Interface Comparable và Comparator](https://freetuts.net/tong-quan-ve-collections-trong-java-1070.html#goto-h2-4)

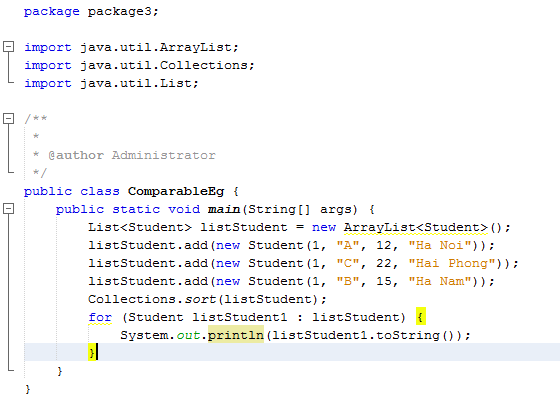
**Comparable** và **Comparator** để so sánh và sắp xếp vị trí 2 đối tượng của một Collection.

**Ví dụ sử dụng Comparable**

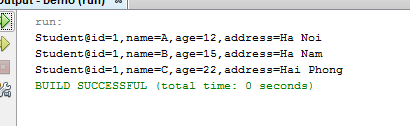
**Student.java**



ComparableEg.java

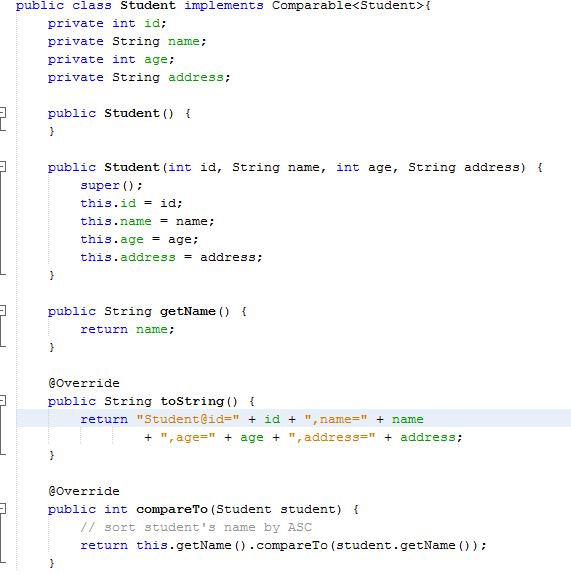


Kết quả



Ví dụ sử dụng **Comparator**

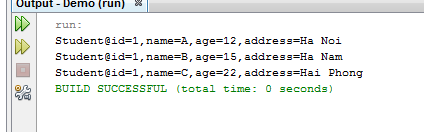
**Student.java**



ComparatorEg.java



Kết quả



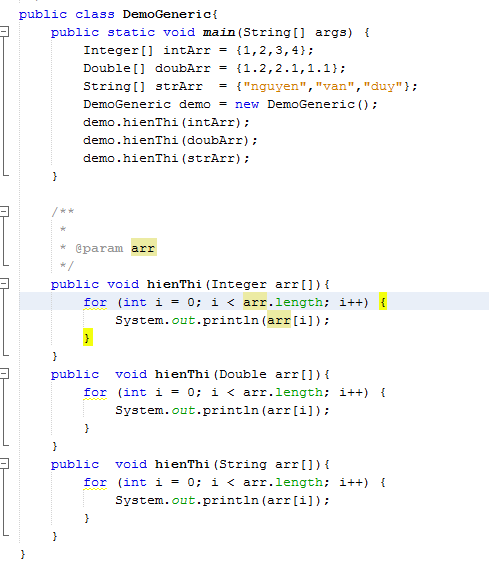
Từ 2 ví dụ trên ta thấy Comparable và Comparator cả hai đều là các giao diện và có thể được sử dụng để sắp xếp các phần tử của collection.

Nhưng vẫn có nhiều sự khác nhau :

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| Bạn phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Bạn **KHÔNG** phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. |
| Comparable cung cấp **phương thức compareTo()**để sắp xếp các phần tử. | Comparator cung cấp **phương thức compare()**để sắp xếp các phần tử. |
| Comparable thuộc về **java.lang** package. | Comparator thuộc về **java.util** package. |
| Chúng ta có thể sắp xếp các phần tử của kiểu Comparable bởi **phương thức Collections.sort(List)** . | Chúng ta có thể sắp xếp các phần tử của kiểu Comparator bởi **phương thức Collections.sort(List,Comparator)**. |

1. Generics Type

Hãy xem đoạn code dưới đây



Trọng đoạn chương trình trên có 3 phương thức được Overload là hienThi(). 3 phương thức trên đều có 1 chức năng là hiển thị các phần tử trong mảng. Tuy nhiên, do 3 mang có 3 kiểu dữ liệu khác nhau nên phải tạo ra 3 phương thức để thực hiện cùng 1 chức năng.

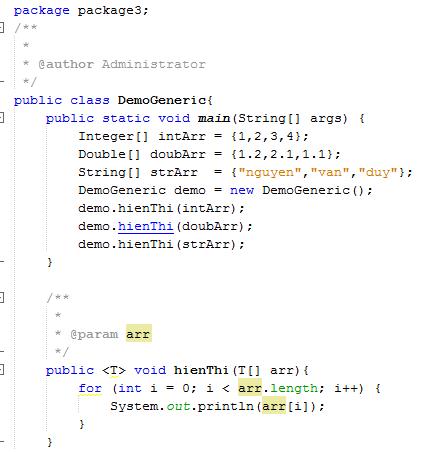
Giả sử nếu có thêm các mảng có kiểu dữ liệu khác thì lại phải tạo thêm phương thức khác cho phù hợp với kiểu dữ liệu.Và rõ ràng việc phải viết các phương thức cho phù hợp với kiểu dữ liệu mặc dù nó thực hiện cùng 1 chức năng như vậy dẫn đến việc trùng lặp code

Để giảm thiểu việc trùng lặp code thì ta có thể sử dụng kiểu Generic cho phương thức hienThi()

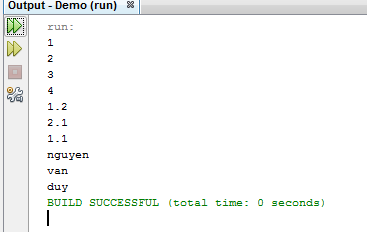
Thuật ngữ “*Generics*” nghĩa là **tham** **số** **hóa** **kiểu** **dữ** **liệu**. Tham số hóa kiểu dữ liệu rất quan trọng vì nó cho phép chúng ta tạo ra và sử dụng một class, interface, method với **nhiều** **kiểu** **dữliệu** **khác** **nhau.**

Generics là cách thức lập trình tổng quát cho phép một object hoạt động với nhiều kiểu dữ liệu khác nhau.

Cụ thể chúng ta làm như sau



Kết quả vẫn như vậy :



Ngoài đại diện tham số <T> ta còn có các tham số khác :

|  |  |
| --- | --- |
| **Ký** **tự** | **Ý** **nghĩa** |
| E | Element – phần tử |
| K | Key – khóa |
| V | Value – giá trị |
| T | Type – kiểu dữ liệu |
| N | Number – số |

Ngoài ra còn có các đại diện khác

**Ký tự đại diện <?>** chấp nhận bất kì đối tượng nào

Ví dụ

Collection<?> col = new ArrayList<String>();

Collection<?> col = new ArrayList<Integer>();

**Ký tự đại diện <? extends type>** chấp nhận bất kỳ đối tượng nào miễn là đối tượng này kế thừa từ type hoặc là đối tượng của type

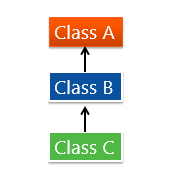
Ví dụ

public void processElement(List<? extends A> elements){

...

}

Trong đó :



Phương thức processElement() chấp nhận bất kì đối tượng nào kế thừa từ A hoặc đối tượng của A => class A , class B, class C

List<A> listA = new ArrayList<A>();

processElement(listA);

List<B> listB = new ArrayList<B>();

processElement(listB);

List<C> listC = new ArrayList<C>();

processElement(listC);

**Ký tự đại diện <? super type>** chấp nhận bất kì đối tượng nào miễn đối tượng này là cha của type hay là đối tượng của type

Ví dụ: Vẫn lấy các lớp A,B,C kế thừa như ở trên

public void processElement(List<? super B> elements){

...

}

List<B> listA = new ArrayList<B>();

processElement(listB);

List<A> listA = new ArrayList<A>();

processElement(listA);

List<Object> listO = new ArrayList<Object>();

processElement(listO);