

Lớp vector

- Vector thực hiện một mảng động. Nó tương tự như ArrayList, nhưng có hai điểm khác biệt
 - Vector được đồng bộ hóa.
 - Vector chứa nhiều phương thức kế thừa không thuộc khuôn khổ tập hợp.
- Người xây dựng:
 - **Vector ()**: Hàm tạo này tạo một vectơ mặc định, có dung lượng ban đầu là 10.
 - **Vector (kích thước int)**: Hàm tạo này chấp nhận một đối số bằng với kích thước được yêu cầu

- **Vector (kích thước int, int incr):**Hàm tạo này tạo ra một vectơ có dung lượng ban đầu được chỉ định bởi kích thước và gia số của nó được chỉ định bởi incr.
- **Vector (Bộ sưu tập c):**Hàm tạo này tạo một vector chứa các phần tử của tập hợp c.

Phương pháp

void add (int index, Object element)

Chèn phần tử được chỉ định tại vị trí được chỉ định trong Vector này.

boolean add (Đối tượng o)

Thêm phần tử được chỉ định vào cuối Vector này.

boolean addAll (Bộ sưu tập c)

Nối tất cả các phần tử trong Bộ sưu tập được chỉ định vào cuối Vector này

void clear ()

Xóa tất cả các phần tử khỏi vectơ này.

boolean chứa (Đối tượng elem)

Kiểm tra xem đối tượng được chỉ định có phải là một thành phần trong vectơ này hay không.

boolean chứaAll (Bộ sưu tập c)

Trả về true nếu vectơ này chứa tất cả các phần tử trong Bộ sưu tập được chỉ định.

Phần tử đối tượngAt (int index)

Trả về thành phần tại chỉ mục được chỉ định.

Các phần tử liệt kê ()

Trả về một bảng liệt kê các thành phần của vectơ này.

boolean bằng (Đối tượng o)

So sánh Đối tượng được chỉ định với vectơ này cho bằng nhau.

Đối tượng đầu tiênElement ()

Trả về thành phần đầu tiên (mục ở chỉ mục 0) của vectơ này.

Đối tượng get (int index)

Trả về phần tử ở vị trí đã chỉ định trong vectơ này.

int indexOf (Đối tượng elem), int indexOf (Đối tượng elem, int

index): Tìm kiếm lần xuất hiện đầu tiên của đối số đã cho (bắt đầu tìm kiếm tại chỉ mục,), kiểm tra tính bình đẳng bằng cách sử dụng phương pháp bằng.

void insertElementAt (Object obj, int index)

Chèn đối tượng được chỉ định làm thành phần trong vectơ này tại chỉ mục được chỉ định.

boolean isEmpty ()

Kiểm tra xem vectơ này không có thành phần nào.

Đối tượng lastElement ()

Trả về thành phần cuối cùng của vectơ.

Loại bỏ đối tượng (int index), loại bỏ boolean (Đối tượng o):Loại bỏ phần tử tại vị trí được chỉ định trong vectơ này. Loại bỏ sự xuất hiện đầu tiên của phần tử được chỉ định trong vectơ này,

void removeAllElements ()

Loại bỏ tất cả các thành phần khỏi vectơ này và đặt kích thước của nó thành 0.

Tập đối tượng (chỉ mục int, phần tử đối tượng)

Thay thế phần tử ở vị trí được chỉ định trong vectơ này bằng phần tử được chỉ định.

int size ()

Trả về số lượng thành phần trong vectơ này.

Liệt kê danh sách con (int fromIndex, int toIndex)

Trả về chế độ xem một phần của Danh sách này giữa fromIndex, inclusive và toIndex, only.

Đối tượng [] toArray ()

Trả về một mảng chứa tất cả các phần tử trong vectơ này theo đúng thứ tự.

Chuỗi toString ()

Trả về biểu diễn chuỗi của vectơ này, chứa biểu diễn chuỗi của mỗi phần tử.

void trimToSize ()

Cắt giảm dung lượng của vectơ này thành kích thước hiện tại của vectơ.

Ví dụ

```
lớp công khai VectorDemo {  
    public static void main (String args []) {  
        Vector v = new Vector (3, 2);  
        System.out.println ("Kích thước bắt đầu:" +  
v.size ());  
        System.out.println ("start Công suất:"  
+ v.capacity ());  
        v.addElement (new Integer (1));  
        v.addElement (new Integer (2));  
        v.addElement (new Integer (3));  
        v.addElement (new Integer (4));  
        System.out.println ("Dung lượng:" +  
v.capacity ());  
    }  
}
```

```
v.addElement (Double mới (5.45));  
System.out.println ("Dung lượng:" +  
    v.capacity ());  
    v.addElement (Double mới (6.08));  
    v.addElement (new Integer (7));  
System.out.println ("Dung lượng:" +  
v.capacity ());  
    System.out.println ("Phần tử đầu tiên:  
"+        (Số nguyên) v.firstElement ());  
    System.out.println ("Phần tử cuối cùng:  
"+        (Số nguyên) v.lastElement ());  
    if (v.contains (new Integer (3)))  
System.out.println ("Vector chứa 3");
```


Sự liệt kê vEnum =
v.elements ();

System.out.println ("\ n
Vectơ: ");

while (vEnum.hasMoreElements ())
System.out.print (vEnum.nextElement () +
""");

System.out.println ();

}}

Sử dụng lớp Vector

java.util.**Véc tơ**<E> (triển khai java.lang.Cloneable,
java.util.List <E>, java.util.RandomAccess, java.io.Serializable)

Lớp TheVector đã lỗi thời so với Java 1.6 nhưng nó vẫn được giới thiệu vì nó một tham số trong phương thức khởi tạo của lớp javax.swing.JTable, một lớp sẽ là được giới thiệu trong lập trình GUI.

```
import java.util.Vector;
class Point {
    int x,y;
    Point() { x=0; y=0; }
    Point(int xx, int yy) {
        x=xx; y=yy;
    }
    public String toString() { return "[" + x + "," + y + ""];}
}
public class UseVector {
    public static void main(String[] args) {
        Vector v = new Vector();
        v.add(15);
        v.add("Hello");
        v.add(new Point());
        v.add(new Point(5,-7));
        System.out.println(v);
        v.remove(2);
        System.out.println(v);
        for (int i=0;i<v.size();i++) System.out.print(v.get(i) + ", ");
        System.out.println();
    }
}
```

Output - Chapter08 (run)

run:

[15, Hello, [0,0], [5,-7]]

[15, Hello, [5,-7]]

15, Hello, [5,-7],

Lớp phân số

```
giai cấp công cộng Phân số{  
    tử số int riêng;  
    người điều hành int riêng tư; công cộng  
    Phân số (int n, int d) { nếu (d! = 0) {  
  
        tử số = n;  
        mệnh giá = d;  
    }  
    khác  
        System.exit (0);  
    }  
// getter / setter
```

```
int tính riêngGCD (int x, int y){
```

```
    int mod;
```

```
    nếu (x < y) {
```

```
        mod = x;
```

```
        x = y;
```

```
        y = mod;    }
```

```
    int r = x% y;
```

```
    trong khi (r != 0) {
```

```
        x = y;
```

```
        y = r;
```

```
        r = x% y; }
```

```
    trả lại y;
```

```
}
```

```
phần riêng tưgiảm (int n, int d){
```

```
    int gcd = GCD (n, d); d =
```

```
    d / gcd;
```

```
    n = n / gcd;
```

```
    trả về phân số mới (n, d); }
```

phân số cộng khai**cộng (Phân số b){**

```
    int num1 = (this.numerator *  
    b.denomirator) + (b.numerator *  
    this.denomirator);
```

```
    int num2 = this.denomirator *
```

```
    b. người điều khiển;
```

```
    trả về giảm (num1, num2); phân số cộng
```

khai**trừ (Phân số b){**

```
    int num1 = (this. tử số *  
    b.denomirator) - (b.numerator *  
    this.denomirator);
```

```
    int num2 = this.denomirator *
```

```
    b. người điều khiển;
```

```
    trả về giảm (num1, num2); }
```

phân số công khai**nhân (Phân số b){**

int num1 = this.numerator *

b. người điều tra;

int num2 = this.denomirator *

b. người điều khiển;

trả về giảm (num1, num2); phân số công

kha**chia (Phân số b){**

int num1 = this.numerator *

b. người điều khiển;

int num2 = this.denomirator * b. tử số;

trả về giảm (num1, num2); }

```
chuỗi công khaitoString() {  
    if (tử số > mẫu số & & mẫu số > 1)  
  
        return (tử số + "/" +  
        mẫu số + "hoặc" + (tử số / mẫu số) + "" +  
        (tử số % mẫu số) + "/" +  
  
        mệnh giá);  
        khác  
        return (tử số + "/" +  
        mệnh giá);  
}}
```

Bộ

- Danh sách dựa trên thứ tự của các thành viên của họ. Bộ không có khái niệm về thứ tự.
- Một Tập hợp chỉ là một cụm tham chiếu đến các đối tượng.
- Bộ có thể **không phải** Lưu trữ **bản sao** các yếu tố.
- Các tập hợp sử dụng phương thức equals (), không phải toán tử ==, để kiểm tra sự trùng lặp của các phần tử.

```
void addTwice (Set set) {  
    set.clear ();  
    Point p1 = new Point (10, 20);  
    Point p2 = new Point (10, 20);  
    set.add (p1);  
    set.add (p2);  
    System.out.println (set.size ());  
}
```

sẽ in ra 1, không phải 2.

Bộ...

- Đặt Bộ sưu tập mở rộng nhưng không thêm bất kỳ phương thức bổ sung nào.
- Hai lớp hiện thực được sử dụng phổ biến nhất là:
 - **TreeSet**
 - Đảm bảo rằng tập hợp đã sắp xếp sẽ theo thứ tự phần tử tăng dần.
 - $\log(n)$ chi phí thời gian cho các hoạt động cơ bản (thêm, bớt và chứa).
 - **HashSet**
 - Hiệu suất thời gian không đổi cho các hoạt động cơ bản (thêm, bớt, chứa và kích thước).

TreeSet và Iterator

- Cây có thứ tự - Được giới thiệu trong Toán học rời rạc
- Đặt: Nhóm các phần tử khác nhau
- TreeSet: Tập hợp + cây có thứ tự, mỗi phần tử được gọi là nút
- Trình lặp lại: Một hoạt động trong đó các tham chiếu của tất cả các nút được nhóm lại để tạo danh sách liên kết. Trình lặp lại là một cách để truy cập mọi nút của cây.
- Danh sách được liên kết: một nhóm các phần tử, mỗi phần tử chứa một tham chiếu đến phần tử tiếp theo

TreeSet = Set + Tree

Kết quả có thể là:

```
Random r = new Random ();
TreeSet myset = new TreeSet ();
for (int i = 0; i < 10; i++) {
    int number = r.nextInt
    (100); myset.add (số);
}
// sử dụng Iterator
Iterator iter = myset.iterator ();
while (iter.hasNext ()) {
    System.out.println (iter.next ());
}
```



7
27
36
41
43
46
49
57
75
83

Sử dụng lớp TreeSet & Iterator

```
import java.util.TreeSet;
import java.util.Iterator;
public class UseTreeSet {
    public static void main (String[] args){
        TreeSet t= new TreeSet();
        t.add(5); t.add(2); t.add(9);t.add(30); t.add(9);
        System.out.println(t);
        t.remove(9);
        System.out.println(t);
        Iterator it= t.iterator();
        while (it.hasNext())
            System.out.print(it.next() + ", ");
        System.out.println();
    }
}
```

Output - Chapter08 (run)

run:
[2, 5, 9, 30]
[2, 5, 30]
2, 5, 30,

Một TreeSet sẽ lưu trữ các phần tử theo thứ tự tăng dần. Thứ tự tự nhiên được áp dụng cho số và thứ tự từ vựng (từ điển) được áp dụng cho chuỗi.

Nếu bạn muốn một TreeSet chứa các đối tượng của riêng mình, bạn phải triển khai phương thức CompareTo (Đối tượng), được khai báo trong giao diện So sánh.

Bảng băm

- Trong mảng, các phần tử được lưu trữ trong một khối bộ nhớ liền kề → Tìm kiếm tuyến tính được áp dụng → chậm, tìm kiếm nhị phân là một cải tiến.
- Bảng băm: các phần tử có thể được lưu trữ trong một khối bộ nhớ khác nhau. Chỉ số của một phần tử được xác định bởi một hàm (hàm băm) → Thao tác Thêm / Tìm kiếm rất nhanh ($O(1)$).



Hàm băm f có thể là: $'S' * 10000 + 'm' * 1000 + 'i' * 100 + 't' * 10 + 'h' \% 50$



HashSet = Đặt + Bảng băm

```
Random r = new Random ();  
HashSet myset = new HashSet ();  
for (int i = 0; i < 10; i++) {  
    int number = r.nextInt  
    (100); myset.add (số);  
}  
// sử dụng Iterator  
Iterator iter = myset.iterator ();  
while (iter.hasNext ()) {  
    System.out.println (iter.next ());  
}
```

Kết quả có thể là:



84
55
7
76
77
95
94
12
91
44

HashSet hay TreeSet?

- Nếu bạn quan tâm đến thứ tự lặp lại , sử dụng Bộ cây và trả tiền phạt theo thời gian.
- Nếu thứ tự lặp lại không quan trọng, hãy sử dụng Bộ băm hiệu suất cao hơn.

Làm thế nào để TreeSet sắp xếp các phần tử?

- Bộ cây dựa vào tất cả các yếu tố của chúng để triển khai giao diện `java.lang.Comparable`. Có thể so sánh được.

`public int compareTo (Đối tượng x)`

- Trả về một số dương nếu đối tượng hiện tại là “lớn hơn” x, theo bất kỳ định nghĩa nào về “lớn hơn” mà bản thân lớp muốn sử dụng.

Làm thế nào để TreeSet sắp xếp các phần tử?

lớp Học sinh thực hiện **Có thể so sánh được**{

int không;

...

int công cộng **so với**(Đối tượng o) {

Student st = (Sinh viên) o; if (no >
st.getNo ())

trả về 1;

khác nếu (không == st.getNo ())

trả về 0;

khác

trả về -1;

}

...

}

So sánh 2 học sinh
dựa trên ID của họ
(trường không)

Làm thế nào để TreeSet sắp xếp các phần tử?

```
public static void main (String [] args) {  
    Random r = new Random (); TreeSet  
    myset = new TreeSet (); for (int i = 0; i  
    <10; i ++) {  
        int no = r.nextInt (100); Student st = new  
        Student (không, "abc"); myset.add (st);  
    }  
    // sử dụng Iterator  
    Iterator iter = myset.iterator (); while  
    (iter.hasNext ()) {  
        Student st = (Sinh viên) iter.next ();  
        System.out.println ("Không:" + st.getNo ());  
    }  
}
```

Không: 2

Không: 8

Không: 11

Không: 19

Không: 33

Không: 52

Không: 78

Không: 83

Không: 92

Không: 96

Các lớp chung

- Một khai báo lớp chung trông giống như một khai báo lớp nongeneric, ngoại trừ việc tên lớp được theo sau bởi một phần tham số kiểu.
- ```
public class Box <T> {private
 T t;
 public void set (T t) {
 this.t = t;
 }
 public T get () {
 trả lại t;
 }
}
```

# Loại tham số

**1.T - Loại**

**2.E - Phần tử**

**3.K - Chìa khóa**

**4.V - Giá trị**

**5.N - Số**

**6.Ư, S, I, G,...**

```
Từ điển lớp <K, V> {
 khóa K riêng tư; giá trị V riêng;
 public Dictionary (khóa K, giá trị V) {
 this.key = key;
 this.value = giá trị; }
 public K getKey () {
 chìa khóa quay trở lại; }
 public void setKey (K key) {
 this.key = key;
 }
 public V getValue () {
 giá trị trả về;
 }
 public void setValue (V value) {
 this.value = giá trị;
 }
}
```

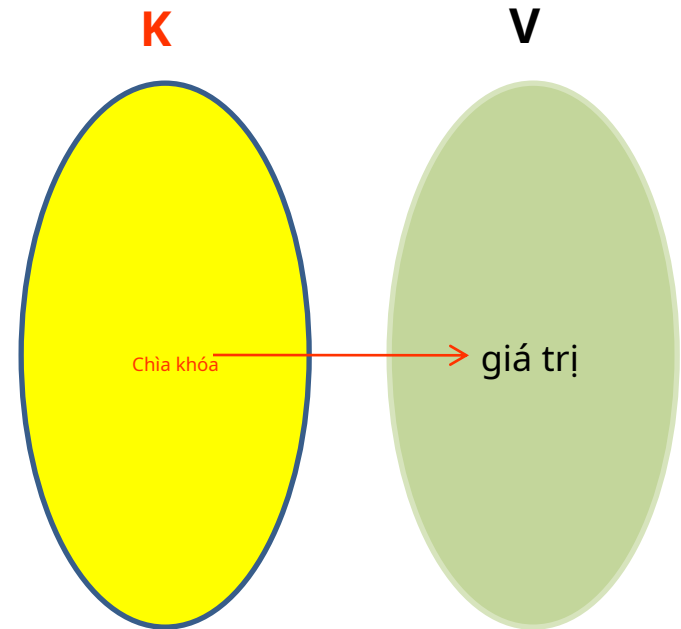
```
public class DemoGeneric {
 public static void main (String [] args) {
 Dictionary <String, String> d = new
Dictionary <String, String> ("Study", "hoc");
 Chuỗi tiếng anh = d.getKey (); String
 vietnamese = d.getValue ();
 System.out.println (tiếng anh + ":" +
tiếng việt); // Ouput: Study: hoc
 }
}
```

```
class Book mở rộng Từ điển <Chuỗi, Chuỗi> {
 Sách công cộng (Khóa chuỗi, Giá trị chuỗi) {
 siêu (khóa, giá trị);
 }
}
```

```
Demo lớp công khai {
 public static void main (String [] args) {
 Book l = new Book ("Học", "học");
 Chuỗi tiếng anh = l.getKey ();
 String vietnamese = l.getValue ();
 System.out.println (tiếng anh + ":" + tiếng việt); //
 Ouput: Study: hoc
 }
}
```

# Bản đồ

- Bản đồ không triển khai giao diện `java.util.Collection`.
- Một bản đồ kết hợp *hợp* tập, được gọi là khóa và giá trị.
- Công việc của Bản đồ là liên kết chính xác một giá trị với mỗi khóa.
- Bản đồ giống như một cuốn từ điển.
- Bản đồ kiểm tra tính duy nhất của khóa dựa trên phương thức `equals()`, không phải toán tử `==`.
- Các ID, Mã hàng, số cuộn là chìa khóa.
- Kiểu dữ liệu thông thường cho các khóa là Chuỗi.



Mỗi phần tử: `<key, value>`

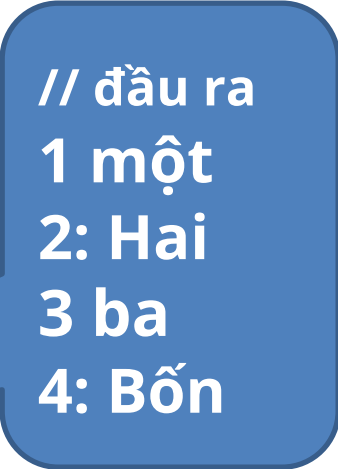


## Bản đồ ..

- Hai lớp Bản đồ quan trọng nhất của Java:
  - HashMap (các khóa ánh xạ có thứ tự không thể đoán trước - bảng băm được sử dụng, hàm băm được xác định trước trong Thư viện Java).
  - TreeMap (các khóa ánh xạ là thứ tự tự nhiên) -> tất cả các khóa phải hiện thực Comparable (một cây được sử dụng để lưu trữ các phần tử).

## Bản đồ băm

```
public static void main (String [] args) {
 HashMap mymap = new HashMap ();
 mymap.put (1, "Một");
 mymap.put (2, "Hai");
 mymap.put (3, "Ba");
 mymap.put (4, "Bốn");
 // sử dụng Iterator
 Iterator iter = mymap.keySet (). Iterator ();
 while (iter.hasNext ()) {
 Khóa đối tượng = iter.next (); System.out.println (key +
 ":" + mymap.get (key));
 }
}
```



// đầu ra  
1 một  
2: Hai  
3 ba  
4: Bốn

Khóa: số nguyên, giá trị: Chuỗi

# Sử dụng lớp HashMap & Iterator

```
1 import java.util.HashMap;
2 import java.util.Iterator;
3 public class UseHashMap {
4 public static void main(String[] args){
5 HashMap h = new HashMap();
6 h.put("Sáu Tấn", "Huỳnh Anh Tuấn");
7 h.put("Bình Gà", "Nguyễn Tấn Sầu");
8 h.put("Ba Địa", "Trần Mai Hoà");
9 System.out.println(h);
10 h.put("Sáu Tấn", "Nguyễn Văn Tuấn");
11 System.out.println(h);
12 h.remove("Bình Gà");
13 System.out.println(h);
14 Iterator it = h.keySet().iterator();
15 while (it.hasNext())
16 { String key= (String)(it.next());
17 String value = (String)(h.get(key));
18 System.out.println(key + ", " + value);
19 }
20 }
21 }
```

Khóa: Chuỗi, giá trị: Chuỗi

## Output - Chapter08 (run)

```
run:
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Huỳnh Anh Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn, Bình Gà=Nguyễn Tấn Sầu}
{Ba Địa= Trần Mai Hoà, Sáu Tấn=Nguyễn Văn Tuấn}
Ba Địa, Trần Mai Hoà
Sáu Tấn, Nguyễn Văn Tuấn
BUILD SUCCESSFUL (total time: 1 second)
```

# Nhóm bộ sưu tập luồngBy ()

- Java 8 giờ đây trực tiếp cho phép bạn thực hiện GROUP BY trong Java bằng cách sử dụng phương thức `Collectors.groupingBy ()`.

```
public class TaiLieu {
 private String ma, tenNXB;
 private int soBan;

 // contructor

 // getter và setter

 //

}
```

List danh sách <TaiLieu>;

# đếm

```
public void count () {
```

```
 Bản đồ <Chuỗi, Dài>
```

```
 count = list.stream (). collect (Collectors.groupin
gBy (TaiLieu :: getTenNXB,
```

```
 Collectors.counting ());
```

```
 System.out.println (số lượng);
```

```
}
```

# Tổng

```
public void sum () {
```

Bản đồ <Chuỗi, Số nguyên>

```
sum = list.stream (). collect (Collectors.grouping
Bởi (TaiLieu :: getTenNXB,
Collectors.summingInt (TaiLieu :: getSoBan)));
System.out.println (sum);
}
```

## tối đa

```
public void max () {
```

```
 Tùy chọn <TaiLieu> max = list.stream ().
 Collect (Collectors.maxBy (Comp
 arator.comparing (TaiLieu :: getSoBan)));
```

```
 System.out.println ("Tai Lieu co max So
 ban:" + (max.isPresent ()? Max.get (): "Không
 Áp dụng"));
}
```



# min

```
public void min () {
```

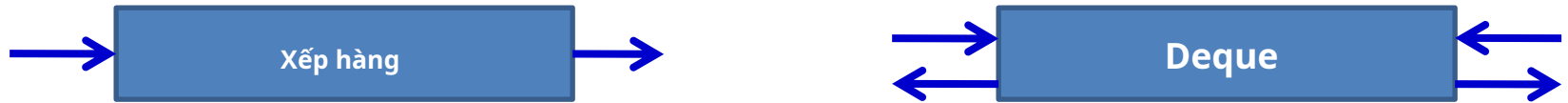
```
 Tùy chọn <TaiLieu> min = list.stream ().
 Collect (Collectors.minBy (Compa rator.comparing
 (TaiLieu :: getSoBan))));
```

```
 System.out.println ("Tai Lieu co max So
 ban:" + (min.isPresent ()? Min.get (): "Không
 Áp dụng"));
}
```

## Tối đa trong nhóm

```
public void maxGroup () {
 Bản đồ <String, TaiLieu> o = list.stream ().
 Collect (Collectors.groupingBy (TaiLieu ::
 getTenNXB,
 Collectors.collectingAndThen (
 Collectors.reducing ((TaiLieu t1, TaiLieu
t2) -> t1.getSoBan ()> t2.getSoBan ())? T1: t2),
 Optional :: get)));
 System.out.println (o);
}
```

# Giao diện **Xếp hàng** và **Deque**



- Giao diện danh sách hạn chế (thao tác hạn chế), người lập trình không thể truy cập vào một phần tử tùy ý mà chỉ truy cập vào phần tử ở đầu hoặc cuối danh sách.
- **Deque**: Một tập hợp tuyến tính hỗ trợ chèn và loại bỏ phần tử ở cả hai đầu. Tên *deque* là viết tắt của "hàng đợi kết thúc kép" và thường là **phát âm là "boong"**. Hầu hết *Deque* triển khai không đặt giới hạn cố định về số lượng phần tử mà chúng có thể chứa, nhưng giao diện này hỗ trợ các phần tử bị giới hạn dung lượng cũng như các phần tử không có giới hạn kích thước cố định.

# Hàng đợi giao diện



giao diện công cộng **Hàng đợi** <E> kéo dài Bộ sưu tập <E>

**boolean**     **cộng** (E e) Chèn phần tử được chỉ định vào hàng đợi này nếu có thể làm như vậy ngay lập tức mà không vi phạm các giới hạn dung lượng, trả về true khi thành công và ném `IllegalStateException` nếu hiện không còn dung lượng.

**E**     **yếu tố** () Truy xuất, nhưng không xóa, người đứng đầu hàng đợi này.

**boolean**     **chào hàng** (E e) Chèn phần tử được chỉ định vào hàng đợi này nếu có thể làm như vậy ngay lập tức mà không vi phạm giới hạn dung lượng.

**E**     **nhìn trộm** () Truy xuất, nhưng không loại bỏ, phần đầu của hàng đợi này hoặc trả về null nếu hàng đợi này trống.

**E**     **cuộc thăm dò ý kiến** () Lấy và loại bỏ phần đầu của hàng đợi này hoặc trả về null nếu hàng đợi này trống.

**E**     **tẩy** () Truy xuất và loại bỏ phần đầu của hàng đợi này.

## Các lớp học:

- `java.util.AbstractQueue<E>` (implements `java.util.Queue<E>`)
  - `java.util.PriorityQueue<E>` (implements `java.io.Serializable`)

# Giao diện Deque...



giao diện công cộng **Deque** <E> kéo dài [Xếp hàng](#) <E>

Ngoài các phương thức kế thừa từ Giao diện hàng đợi, một số phương thức được khai báo:

| Summary of Deque methods |                            |                            |                           |                           |
|--------------------------|----------------------------|----------------------------|---------------------------|---------------------------|
|                          | First Element (Head)       |                            | Last Element (Tail)       |                           |
|                          | <i>Throws exception</i>    | <i>Special value</i>       | <i>Throws exception</i>   | <i>Special value</i>      |
| <b>Insert</b>            | <code>addFirst(e)</code>   | <code>offerFirst(e)</code> | <code>addLast(e)</code>   | <code>offerLast(e)</code> |
| <b>Remove</b>            | <code>removeFirst()</code> | <code>pollFirst()</code>   | <code>removeLast()</code> | <code>pollLast()</code>   |
| <b>Examine</b>           | <code>getFirst()</code>    | <code>peekFirst()</code>   | <code>getLast()</code>    | <code>peekLast()</code>   |

## Các lớp học

java.util.[LinkedList](#) <E> (thực hiện java.lang.[Nhân bản](#) , java.util.[Deque](#) <E> ,  
java.util.[Danh sách](#) <E> , java.io.[Serializable](#) )

java.util.[ArrayDeque](#) <E> (thực hiện java.lang.[Nhân bản](#) ,  
java.util.[Deque](#) <E> , java.io.[Serializable](#) )

## Hàng đợi / Bản trình diễn Deque.



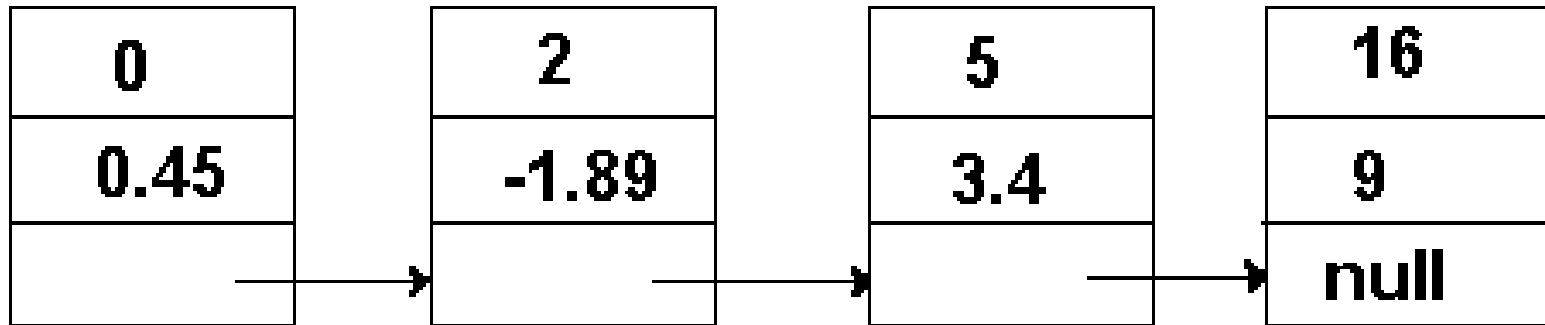
```
import java.util.LinkedList;
public class DequeDemo {
 public static void main(String args[]){
 int N=5;
 // 3 list are the same
 LinkedList list1= new LinkedList();
 LinkedList list2= new LinkedList();
 LinkedList list3= new LinkedList();
 for (int i=0; i<N; i++) {
 list1.add(i); list2.add(i); list3.add(i);
 }
 // Access list1 as a queue
 while(!list1.isEmpty()) System.out.print(list1.remove() + ",");
 System.out.println();
 // Access list2 from it's head
 while(!list2.isEmpty()) System.out.print(list2.removeFirst()+ ",");
 System.out.println();
 // Access list2 from it's tail
 while(!list3.isEmpty()) System.out.print(list3.removeLast()+ ",");
 System.out.println();
 }
}
```

run:

0,1,2,3,4,  
0,1,2,3,4,  
4,3,2,1,0,

# LinkedList

- Constructor:
  - LinkedList ()
  - LinkedList (Bộ sưu tập c)
- $0,45 - 1,89 x_2 + 3,4 x_5 + 9 x_{16}$



# Ví dụ: Đa thức

giai cấp công cộng **Đa thức**{

private Node đầu tiên = new Node (0, 0); Node  
riêng cuối cùng = đầu tiên;

lớp tính riêng **Nút**{

int coef; *// hệ số - số* int exp; *// Số*  
*mũ - số mũ* Nút tiếp theo;

**Nút (int coef, int exp){**

this.coef = coef;

this.exp = exp;

}

}



riêng **Đa thức ()**{}

//  $a * x^b - -1,89 x^2$

công cộng **Đa thức (int coef, int exp)**

{

last.next = new Node (coef, exp);

last = last.next;

}

```

// trả về c = a + b
đa thức cộng cộng (Đa thức b){
 Đa thức a = this; Polynomial c = new
 Polynomial (); Nút x = a.first.next;

 Nút y = b.first.next;
 while (x != null || y != null) {
 Nút t = null;
 if (x == null) {t = new Node (y.coef, y.exp); else if (y == null) {t = y = y.next; }
 new Node (x.coef, x.exp); x = x.next; }
 else if (x.exp > y.exp) {t = new Node (x.coef, x.exp); x = x.next; } else if (x.exp < y.exp) {t
 = new Node (y.coef, y.exp); y = y.next; }
 khác {
 int coef = x.coef + y.coef; int
 exp = x.exp;
 x = x.next;
 y = y.next;
 if (coef == 0) tiếp tục; t = new Node
 (coef, exp);}
 c.last.next = t;
 c.last = c.last.next;
 }
 trả lại c;
}

```

```

// trả về c = a * b
đa thức công cộng nhân (Đa thức b){
 Đa thức a = this; Polynomial c = new
 Polynomial ();
 for (Nút x = a.first.next; x != null; x = x.next)
 {
 Polynomial temp = new Polynomial ();
 for (Nút y = b.first.next; y != null; y =
 y.next) {
 temp.last.next = new Node (x.coef *
 y.coef, x.exp + y.exp);
 temp.last = temp.last.next;
 }
 c = c.plus (tạm thời);
 }
 trả lại c;
}

```

chuỗi công khai **toString ()**{

Chuỗi s = "";

for (Nút x = first.next; x != null; x = x.next) {

if (x.coef > 0) s = s + "+" + "x ^" + x.exp;                      x.coef +

else if (x.coef < 0) s = s + "-" + (-  
x.coef) + "x ^" + x.exp;} return s;

}

khoảng trống tĩnh công cộng **main (Chuỗi [] args)**{

Polynomial zero = new Polynomial (0, 0); //  $4x^3 +$

$3x^2 + 1$

Đa thức p1                      = new Polynomial (4, 3); = new

Đa thức p2                      Polynomial (3, 2); = new

Đa thức p3                      Polynomial (1, 0); = p1.plus

Đa thức p                      (p2) .plus (p3);

```
// 3x ^ 2 + 5
```

```
Đa thức q1 = new Polynomial (3, 2); = new
Đa thức q2 Polynomial (5, 0); = q1.plus
Đa thức q (q2);
```

```
Đa thức r = p.plus (q);
```

```
Đa thức s = p.multiply (q);
```

```
System.out.println ("zero (x) không); = " +
```

```
System.out.println ("p (x) System.out.println ("q (x) P);
```

```
System.out.println ("p (x) + q (x) =" + r); " + q);
```

```
System.out.println ("p (x) * q (x) =" + s);
```

```
}
```

```
}
```

# Tóm lược

- Khung Bộ sưu tập
  - **Các *Bộ sưu tập* Siêu giao diện và sự lặp lại**
  - Danh sách
  - **Bộ**
  - Bản đồ
  - Lớp hỗ trợ
  - Bộ sưu tập và bảo trì mã