

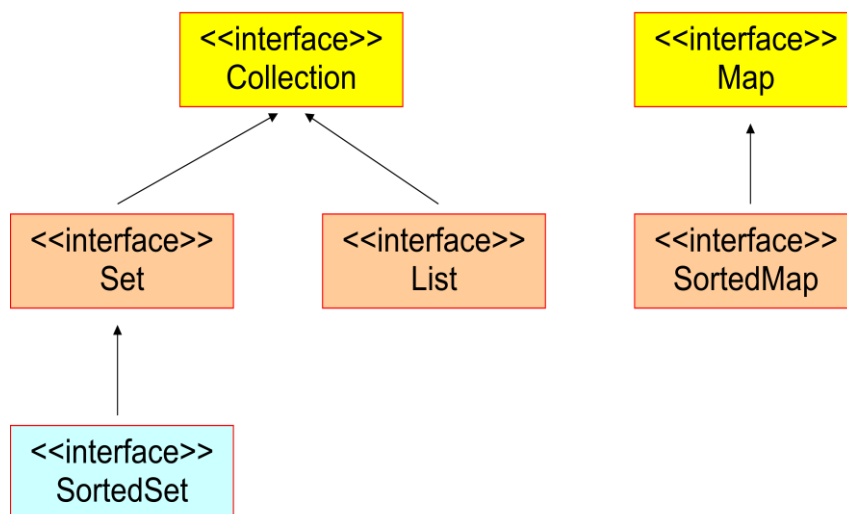
Java Collection Framework

I. Tổng quát về Java Collection Framework

Các Collection (Collection, Set, List, Map, ArrayList, Vector, Hashtable, HashSet, HashMap). Các collection được đặt trong gói java.util. JCF à một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.

1. Các thành phần của Java Collection

- Interfaces: Là các giao tiếp thể hiện tính chất của các kiểu collection khác nhau như List, Set, Map.
- Implementations: Là các lớp collection có sẵn được cài đặt các collection interfaces.
- Algorithms: Là các phương thức tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...



2. Một số lợi ích của Collections Framework

- Giảm thời gian lập trình
- Tăng cường hiệu năng chương trình
- Dễ mở rộng các collection mới
- Khuyến khích việc sử dụng lại mã chương trình

3. Các thao tác chính trên collection

Collection cung cấp các thao tác chính như thêm/xoá/tìm phần tử... boolean add(Object element);

- boolean add(Object element);
- boolean remove(Object element);
- boolean contains(Object element);
- int size();
- boolean isEmpty();
- void clear();
- Object[] toArray();

Nếu lớp cài đặt Collection không muốn hỗ trợ các thao tác làm thay đổi collection như add, remove, clear... nó có thể tung ra ngoại lệ UnsupportedOperationException.

II. Chi tiết về các cài đặt (implements) collection

1. List

List là một interface dùng chứa danh sách liên tục, nó cung cấp thêm các phương thức để xử lý collection kiểu danh sách (Danh sách là một collection với các phần tử được xếp theo chỉ số).

Sau đây là một chương trình đơn giản nhất minh họa cho việc sử dụng danh sách bằng cách dùng giao diện List, chú ý cách khai báo một biến danh sách:
List list = new ArrayList();

```
import java.util.*;
public class ListDemo {
    public static void main(String args[]) throws Exception {
        List list = new ArrayList();
        list.add("A");
        list.add("Birthday");
        list.add(88.82);
        list.add(1, "MIDDLE");
        System.out.println(list);
    }
}
```

1.1. ArrayList

Một bất lợi của việc dùng mảng là phải xác định trước số lượng phần tử trong mảng:

- Nếu khai báo kích thước mảng quá nhỏ thì sẽ dẫn đến thiếu bộ nhớ.
- Nếu khai báo quá lớn thì lại lãng phí bộ nhớ.
- Các phần tử trong mảng phải có cùng kiểu dữ liệu.

ArrayList đã khắc phục được nhược điểm này (ArrayList là một kiểu mảng động. Nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ).

Ví dụ 1: Chương trình quản lý nhân viên tại một phòng ban nào đó trong một công ty:

- Số lượng nhân viên trong phòng ban có thể thay đổi.
- Kích thước không cố định.(không sử dụng mảng trong ví dụ này).
Thay vào đó ta sẽ sử dụng ArrayList.

1.1.1. Đặc điểm:

Quản lý một dãy các đối tượng.

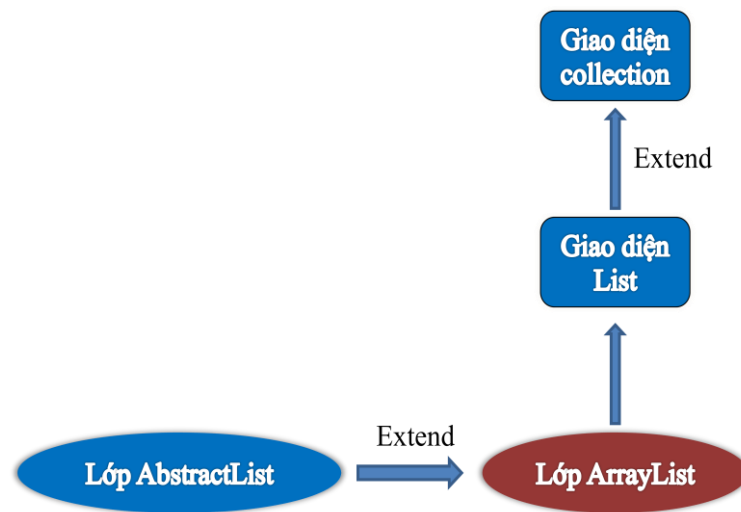
Có thể tăng, giảm kích thước theo nhu cầu.

Cung cấp các phương thức cho các thao tác thông thường: chèn, xóa phần tử,...

Từ java 5.0 trở đi ArrayList là class generic

ArrayList <T> ; // *Tập các đối tượng có kiểu T*

1.1.2. Sơ đồ lớp ArrayList trong gói java.util.*



1.1.3. Khai báo ArrayList

```
ArrayList<TypeObject> v = new ArrayList<TypeObject> ();
```

ArrayList<TypeObject> : Định kiểu cho danh sách ngay khi khai báo biến danh sách.

Lưu ý: Trên chỉ là một trong những cách khai báo thông thường hay sử dụng nhất. Thật ra lớp ArrayList còn nhiều hàm thiết lập có tham số.

// khai báo 1 danh sách mảng các string;

Ví dụ: ArrayList<String> str = new ArrayList < String >();

1.1.4. Một số phương thức trong ArrayList

int size(); *// Lấy kích thước hiện thời của ArrayList*

Ví dụ:

// Lấy kích thước hiện thời của ArrayList

```
System.out.println("Chiều dài ArrayList: " + str.size());
```

Object get(arg0); *// Lấy một phần tử trong ArrayList theo chỉ mục*

Ví dụ:

```
// Lấy phần tử đầu tiên trong ArrayList
System.out.println("Phan tu dau tien cua ArrayList la: " + str.get(0));
```

bool add(arg0); // Thêm phần tử vào ArrayList;

Ví dụ:

```
// Them mot phan tu vao ArrayList
String s = "Xuan";
if(str.add(s)) {
    System.out.println("Da them thanh cong mot phan tu gia tri "
        + s + " vao ArrayList");
}
// Insert
s = "Ha";
str.add(0, s);
```

bool remove(); // Xóa item tại một chỉ số

Ví dụ:

```
// Remove mot phan tu
if(str.remove(s)) {
    System.out.println("Da xoa thanh cong mot phan tu gia tri "
        + s + " khoi ArrayList");
}
// View ArrayList
System.out.println("Cac phan tu hien co trong mang");
for(int i = 0; i < str.size(); i++) {
    System.out.println(str.get(i));
}
// Remove tat ca
if(str.removeAll(str)) {
    System.out.println("Da xoa thanh cong tat ca phan tu" +
        " trong ArrayList");
}
```

1.2. *LinkedList*

1.2.1. *Đặc điểm*

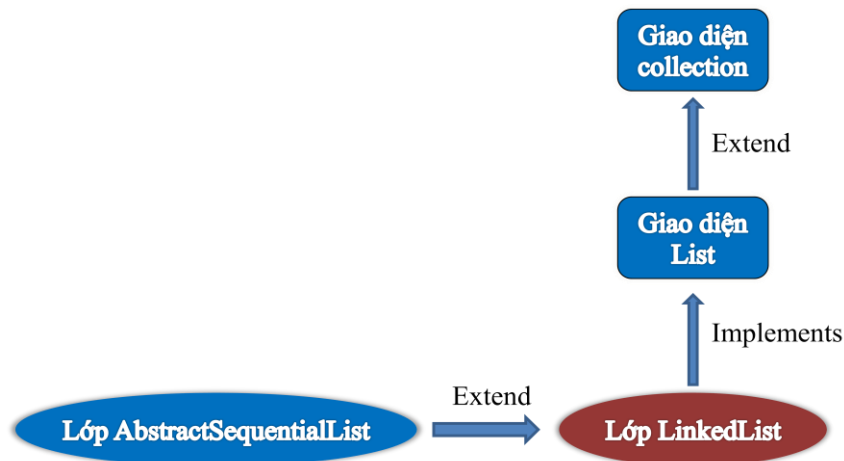
Danh sách liên kết 2 chiều. Hỗ trợ thao tác trên đầu và cuối danh sách.

LinkedList giúp tiết kiệm bộ nhớ so với mảng trong các bài toán xử lý danh sách.

Khi chèn/xoá một node trên LinkedList, không phải dẫn/dồn các phần tử như trên mảng.

Việc truy nhập trên LinkedList luôn phải tuần tự.

1.2.2. Sơ đồ lớp LinkedList trong gói java.util.*



1.2.3. Khai báo ArrayList

```
LinkedList< TypeObject> list = new LinkedList<TypeObject>();
```

LinkedList< TypeObject>: Định kiểu cho danh sách ngay khi khai báo biến danh sách.

Ví dụ:

```
// khai báo 1 danh sách mảng các string;
```

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

1.2.4. Một số phương thức trong LinkedList

Ngoài các phương pháp mà nó kế thừa, lớp **LinkedList** định nghĩa một số phương pháp hữu ích của riêng của mình để thao tác và truy cập vào danh sách. To add elements to the start of the list, use **addFirst()** ; to add elements to the end, use **addLast()** . Để thêm phần tử vào đầu danh sách, sử dụng **addFirst ()**; để thêm các yếu tố để kết thúc, sử dụng **addLast ()**...

Ví dụ:

```
// Add phan tu moi
String s = "Xuan";
if(str.add(s)) {
    System.out.println("Da them thanh cong mot phan tu gia tri'"
        + s + "' vao ArrayList");
}
// Insert
s = "Ha";
// Add first
// str.addFirst(s);
str.add(0, s);
// Add last
s = "Thu";
str.addLast(s);
// View ArrayList
System.out.println("Cac phan tu hien co trong mang");
for(int i = 0; i < str.size(); i++) {
    System.out.println(str.get(i));
}
```

Để lấy item đầu tiên, sử dụng **getFirst()**, item cuối mảng, sử dụng **getLast()**, lấy item theo chỉ mục sử dụng **get(int index)**.

Ví dụ:

```
// Phan tu dau tien cua mang
System.out.println("Phan tu dau tien cua mang: " + str.getFirst());
// Phan tu cuoi mang
System.out.println("Phan tu cuoi mang: " + str.getLast());

// Lay phan tu theo chi muc
System.out.println("Cac phan tu hien co trong mang");
for(int i = 0; i < str.size(); i++) {
    System.out.println(str.get(i));
}
```

Để xóa item đầu tiên của mảng, sử dụng **string removeFirst()**; để xóa bỏ phần tử cuối cùng, gọi **string removeLast()**, xóa theo chỉ mục dùng **string remove(int index)**.

Ví dụ:

```
// Xoa phan tu dau tien cua mang
System.out.println("Xoa phan tu dau tien:");
System.out.println("Phan tu da xoa: " + str.removeFirst());
// Xoa phan tu cuoi mang
System.out.println("Xoa phan tu cuoi mang:");
System.out.println("Phan tu da xoa: " + str.removeLast());
// Xoa phan tu tai vi tri 0
str.remove(0);
```

Để sửa giá trị cho một item, sử dụng **string set(int index, String element)**

Ví dụ:

```
// Set giá trị cho phần tử  
s = "Dong";  
System.out.println("Sua phần tử tại index 1, " +  
    "giá trị " + str.set(1, s));
```

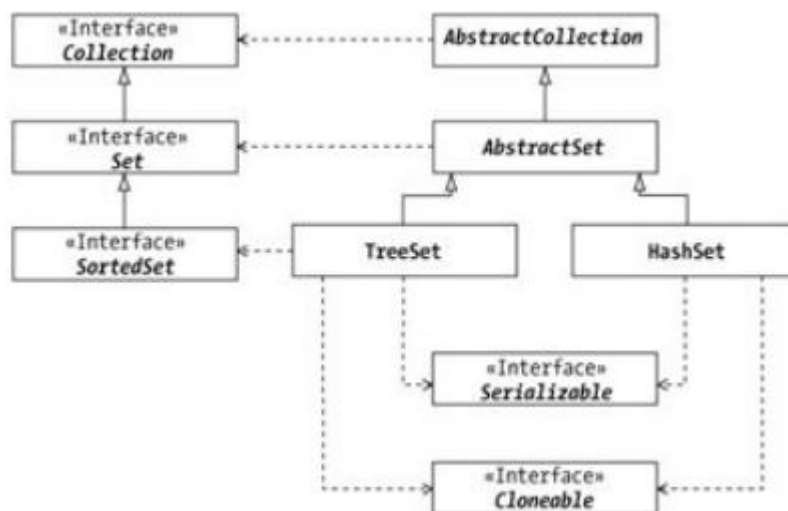
2. Set

Giao diện này đại diện cho một nhóm các phần tử không trùng lặp.

Set kế thừa từ Collection, hỗ trợ các thao tác xử lý trên collection kiểu tập hợp (Một tập hợp yêu cầu các phần tử phải không được trùng lặp).

Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.

Dưới đây là mô hình phân cấp lớp trong java.util.*:



2.1. SortedSet

SortedSet kế thừa từ Set, nó hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được. Các đối tượng đưa vào trong một SortedSet phải cài đặt giao tiếp Comparable hoặc lớp cài đặt SortedSet phải nhận một Comparator trên kiểu của đối tượng đó.

2.1.1. Một số phương thức của SortedSet

- `Object first();` // lấy phần tử đầu tiên (nhỏ nhất)
- `Object last();` // lấy phần tử cuối cùng (lớn nhất)
- `SortedSet subSet(Object e1, Object e2);` // lấy một tập các phần tử nằm trong khoảng từ `e1` tới `e2`.

TreeSet thực hiện giao diện **Set**, lưu trữ theo dạng cây. Các đối tượng được sắp xếp tăng dần.. Truy cập và thu hồi khá nhanh, làm cho **TreeSet** trở thành một lựa chọn tuyệt vời khi lưu trữ số lượng lớn thông tin được sắp xếp có khả năng tìm kiếm một cách nhanh chóng.

Sau đây là một số hàm thiết lập được định nghĩa trong **TreeSet**:

```
TreeSet<TypeObject> ()
TreeSet<TypeObject> (Collection c )
TreeSet<TypeObject> (Comparator comp )
TreeSet<TypeObject> (SortedSet ss )
```

Ví dụ: Khai báo một **TreeSet** với **TypeObject** kiểu **String**.

```
Demonstrate TreeSet.
import java.util.*;
public class TreeSetDemo {
    public static void main(String args[]) {
        // Create a tree set
        TreeSet<String> ts = new TreeSet<String>();
        // Add elements to the tree set
        ts.add("C");
        ts.add("A");
        ts.add("B");
        ts.add("E");
        ts.add("F");
        ts.add("D");
        System.out.println(ts);
    }
}
```

2.2. **HashSet**

HashSet hỗ trợ bởi một bảng băm để lưu trữ các item duy nhất, nghĩa là các item trong **HashSet** sẽ không trùng lặp. Mỗi phần tử được lưu trữ và truy xuất thông qua các mã băm của nó.

Hầu hết các function trong **HashSet** được cung cấp bởi các superclasses là **AbstractCollection** và **AbstractSet**.

2.2.1. Khởi tạo HashSet

Lớp HashSet cung cấp bốn constructor . Ba constructor đầu tiên tạo ra các thiết lập rỗng với các size khác nhau:

```
public HashSet()
```

```
public HashSet(int initialCapacity)
```

```
public HashSet(int initialCapacity, int loadFactor)
```

Nếu không xác định, kích thước thiết lập ban đầu cho các phần tử lưu trữ sẽ được kích thước mặc định của một HashMap. Khi dung lượng mặc định đã đầy thì khi thêm một item mới vào, HashSet sẽ tăng dung lượng lên gấp đôi so với lúc trước khi thêm item đó.

Khi chúng ta khai báo HashSet, thì nên dùng cấu trúc khai báo sau:

```
Set set = new HashSet();
```

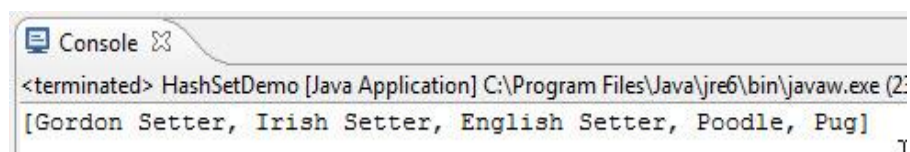
Mục đích của việc khai báo này là khi chúng ta cần chuyển *set* sang dạng khác như TreeSet, chúng ta có thể sử dụng các phương thức có sẵn trong Set interface.

Constructor thứ tư hoạt động như một hàm sao chép, nó sao chép các phần tử từ một collection khác.

```
public HashSet(Collection col)
```

Ví dụ:

```
// TODO Auto-generated method stub
String elements[] = {"Irish Setter", "Poodle", "English Setter",
                    "Gordon Setter", "Pug"};
Set<String> set = new HashSet<String>(Arrays.asList(elements));
System.out.print(set);
```



```
<terminated> HashSetDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2:
[Gordon Setter, Irish Setter, English Setter, Poodle, Pug]
```

2.2.2. Một số phương thức của HashSet

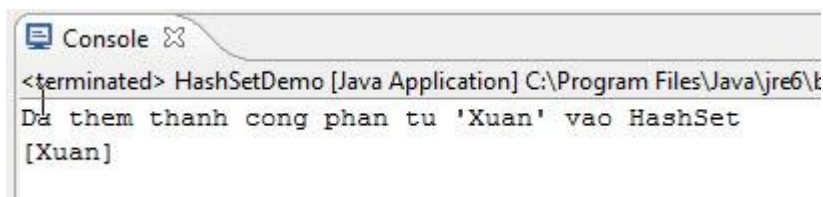
Để thêm một item vào HashSet, sử dụng hàm add():

public boolean add(Object element)

Ví dụ:

```
// Khởi tạo lại set
set = new HashSet<String>();

String str = "Xuan";
if(set.add(str)) {
    System.out.println("Đã thêm thành công phần tử '"
        + str + "' vào HashSet");
}
System.out.print(set);
```

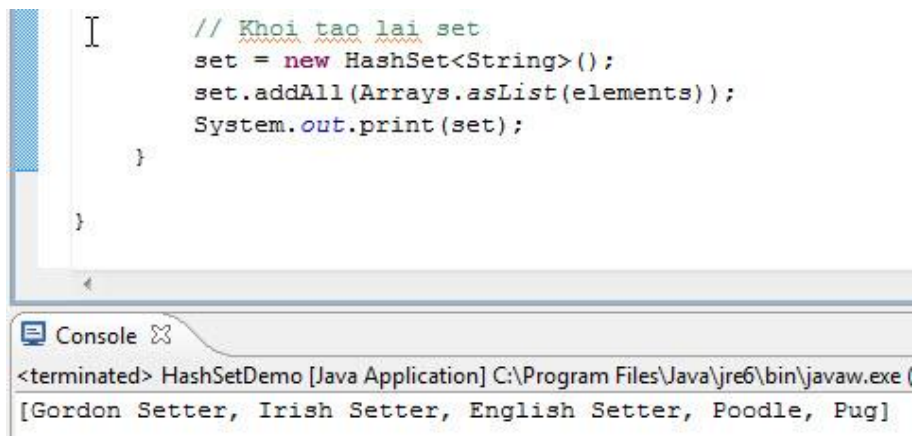


The screenshot shows a console window titled "Console" with the following output:
<terminated> HashSetDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
Đã thêm thành công phần tử 'Xuan' vào HashSet
[Xuan]

Để thêm một collection, sử dụng addAll():

public boolean addAll(Collection c)

Ví dụ:



The screenshot shows a code editor with the following code:
// Khởi tạo lại set
set = new HashSet<String>();
set.addAll(Arrays.asList(elements));
System.out.print(set);
}
}
The console window below shows the output:
<terminated> HashSetDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
[Gordon Setter, Irish Setter, English Setter, Poodle, Pug]

Để xóa một item, sử dụng remove():

public boolean remove(Object element)

Ví dụ:

```
System.out.println("HashSet trước khi xóa: " + set);
System.out.println("Đang xóa...");
// Xóa một phần tử khỏi HashSet
str = "Gordon Setter";
if(set.remove(str)) {
    System.out.println("Đã xóa thành công phần tử '"
        + str + "' khỏi HashSet");
}
else {
    System.out.print("Loi! Không thể xóa");
}
System.out.println("HashSet sau khi xóa: " + set);
}
```

Console

<terminated> HashSetDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (00:12:01 14-03-2011)
HashSet trước khi xóa: [Gordon Setter, Irish Setter, English Setter, Poodle, Pug]
Đang xóa...
Đã xóa thành công phần tử 'Gordon Setter' khỏi HashSet
HashSet sau khi xóa: [Irish Setter, English Setter, Poodle, Pug]

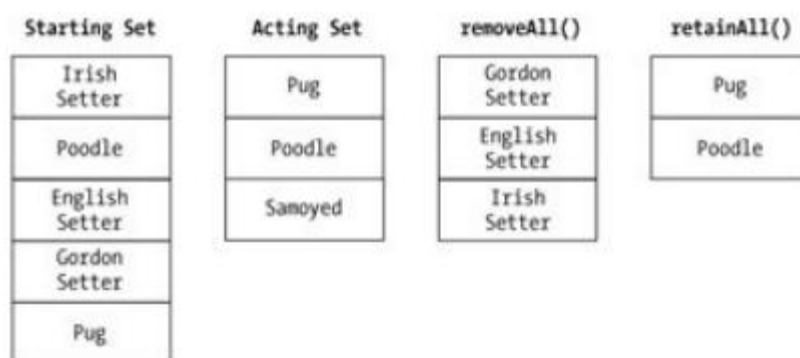
Xóa nhiều item:

public boolean removeAll(Collection c)

Xóa tất cả:

public void clear()

Phương thức *retainAll()* hoạt động giống như *removeAll()*, nhưng theo hướng ngược lại:



VARIABLE/METHOD NAME	VERSION	DESCRIPTION
HashSet()	1.2	Constructs a hash set.
add()	1.2	Adds an element to the set.
clear()	1.2	Removes all elements from the set.
clone()	1.2	Creates a clone of the set.
contains ()	1.2	Checks if an object is in the set.
isEmpty()	1.2	Checks if the set has any elements.
iterator()	1.2	Returns an object from the set that allows all of the set's elements to be visited.
remove()	1.2	Removes an element from the set.
size()	1.2	Returns the number of elements in the set.

2.3. *LinkedHashSet*

Tương tự HashSet nhưng có kèm theo danh sách liên kết

3. Map

Map là một interface dùng để thay thế cho lớp từ điển cổ. Lớp này là một lớp trừu tượng nên cần được thay thế bằng một giao diện(interface). Giao diện Map định nghĩa một sự hỗ trợ cơ bản để lưu trữ một cặp key-value sao cho mỗi key có thể ánh xạ đến một giá trị duy nhất, và các key(khóa) không được trùng nhau. Map không extend từ Collection interface mà được định nghĩa riêng, và là root cho một nhánh phân cấp khác (giống như Collection). Có 4 triển khai (implementation) cụ thể của Map đó là: HashMap, WeakHashMap, TreeMap, và Hashtable.

Sau đây là một số phương thức được định nghĩa trong Map:

VARIABLE/METHOD NAME	VERSION	DESCRIPTION
clear()	1.2	Removes all the elements from the map.
containsKey()	1.2	Checks to see if an object is a key for the map.
containsValue()	1.2	Checks to see if an object is a value within the map.
entrySet()	1.2	Returns the set of key-value pairs in the map.
equals()	1.2	Checks for equality with another object.

Map.Entry:

get()	1.2	Retrieves a value for a key in the map.
hashCode()	1.2	Computes a hash code for the map.
isEmpty()	1.2	Checks if hash map has any elements.
keySet()	1.2	Retrieves a collection of the keys of the map.
put()	1.2	Places a key-value pair into the map.
putAll()	1.2	Places a collection of key-value pairs into the map.
remove()	1.2	Removes an element from the map.
size()	1.2	Returns the number of elements in the map.
values()	1.2	Retrieves a collection of the values of the map.

Map cung cấp 3 cách view dữ liệu:

- View các khoá:
 - ✓ Set keySet(); // Trả về các khoá
- View các giá trị:
 - ✓ Collection values(); // Trả về các giá trị
- View các cặp khoá-giá trị
 - ✓ Set entrySet(); // Trả về các cặp khoá-giá trị

Sau khi nhận được kết quả là một collection, ta có thể dùng iterator để duyệt các phần tử của nó.

3.1. *HashMap*

HashMap là lớp implement giao diện Map. Các item(key-value) trong HashMap không được sắp xếp.

3.1.1. *Khởi tạo HashMap*

Có bốn constructor để tạo ra một HashMap. Ba constructor đầu tiên cho phép tạo ra một HashMap rỗng (empty HashMap):

```
public HashMap()
```

```
public HashMap(int initialCapacity)
```

```
public HashMap(int initialCapacity, float loadFactor)
```

Constructor thứ tư hoạt động như một hàm thiết lập sao chép, nó tạo ra một HashMap mới từ một Map khác:

```
public HashMap(Map map)
```

3.1.2. Các phương thức trong HashMap

VARIABLE/METHOD NAME	VERSION	DESCRIPTION
HashMap()	1.2	Constructs an empty hash map.
clear()	1.2	Removes all the elements from the hash map.
clone()	1.2	Creates a clone of the hash map.
containsKey()	1.2	Checks to see if an object is a key for the hash map.
containsValue()	1.2	Checks to see if an object is a value within the hash map.
entrySet()	1.2	Returns a set of key-value pairs in the hash map.
get()	1.2	Retrieves the value for a key in the hash map.
isEmpty()	1.2	Checks if hash map has any elements.
keySet()	1.2	Retrieves a collection of the keys of the hash map.
put()	1.2	Places a key-value pair into the hash map.
putAll()	1.2	Places a collection of key-value pairs into the hash map.
remove()	1.2	Removes an element from the hash map.
size()	1.2	Returns the number of elements in the hash map.
values()	1.2	Retrieves a collection of the values of the hash map.

Thêm một item mới vào HashMap (key-value), sử dụng put():

```
public Object put(Object key, Object value)
```

Ví dụ:

```
// TODO Auto-generated method stub
Map<Integer, String> map = new Hashtable<Integer, String>();
map.put(0, "Micheal Jackson");
System.out.println(map);
```

Thêm từ một Map khác:

```
// Create new Map
Map<Integer, String> mapTmp = new HashMap<Integer, String>();
mapTmp.put(1, "Matt Butcher");
mapTmp.put(2, "Cristiano Ronado");
map.putAll(mapTmp);
System.out.println(map);
```

Xóa một item khỏi HashMap:

```
public Object remove(Object key)
```

Ví dụ:

```
// Remove a element  
System.out.println(map.remove(1));  
System.out.println(map);
```

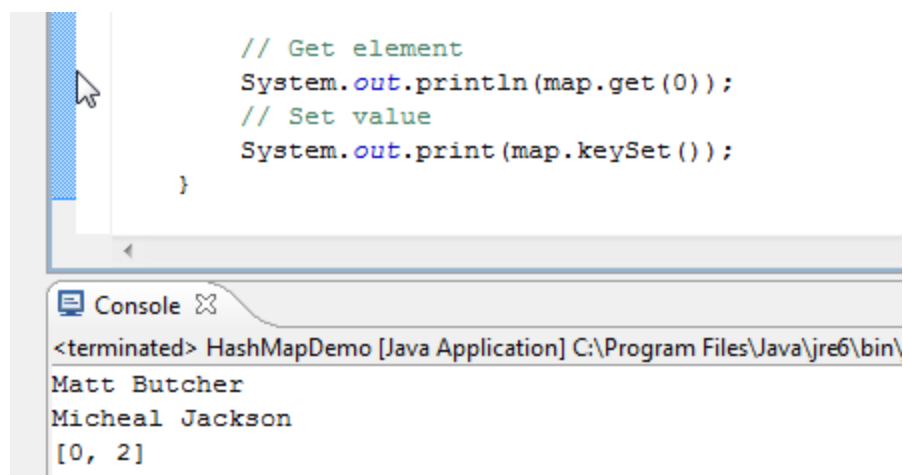
Xóa tất cả các item trong HashMap:

```
public void clear()
```

Lấy một item từ HashMap:

```
public Object get(Object key)
```

Ví dụ:



3.2. *TreeMap*

TreeMap là lớp implement Map Interface. TreeMap chứa các item với các key được sắp xếp dưới dạng một cây cân bằng, cây đỏ đen.

3.2.1. Khởi tạo *TreeMap*:

```
public TreeMap()
```

```
public TreeMap(Map map)
```


public TreeMap(Comparator comp)

public TreeMap(SortedMap map)

3.2.2. Các phương thức của TreeMap

VARIABLE/METHOD NAME	VERSION	DESCRIPTION
TreeMap()	1.2	Constructs an empty tree map.
clear()	1.2	Removes all the elements from the tree map.
clone()	1.2	Creates a clone of the tree map.
comparator()	1.2	Retrieves the comparator for the map.
containsKey()	1.2	Checks to see if an object is a key for the tree map.
containsValue()	1.2	Checks to see if an object is a value within the tree map.
entrySet()	1.2	Returns a set of key–value pairs in the tree map.
firstKey()	1.2	Retrieves the first key of the map.
get()	1.2	Retrieves a value for a key in the tree map.
headMap()	1.2	Retrieves the sub map at the beginning of the entire map.
keySet()	1.2	Retrieves a collection of keys from the tree map.
lastKey()	1.2	Retrieves the last key of the map.
put()	1.2	Places a key–value pair into the tree map.

3.3. *LinkedHashMap*

LinkedHashMap cũng tương tự như HashMap nhưng có sử dụng danh sách liên kết, các item được liên kết với nhau và có thứ tự. Tuy nhiên, LinkedHashMap truy cập item chậm hơn so với HashMap.

III. Sự khác nhau giữa các Collection

1. List, Set và Map

Giao diện này cung cấp method để chèn và xóa các item tại một điểm bất kỳ trong danh sách, các item được truy cập và tìm kiếm các phần tử trong danh sách theo chỉ mục (index).. Không giống như các bộ, danh sách có thể chứa các item trùng nhau.

Set hỗ trợ các thao tác xử lý trên collection kiểu tập hợp, các phần tử trong Set phải không được trùng nhau.

Map cung cấp các thao tác xử lý trên các bảng ánh xạ, các phần tử được lưu trữ theo khoá và không được có 2 khoá trùng nhau.

2. Các triển khai(implement) của List

Có 2 lớp implement giao diện List, đó là ArrayList và LinkedList.

Không giống như ArrayList, LinkedList là danh sách liên kết kép cung cấp các method chèn, xóa một phần tử vào đầu và kết thúc của danh sách, danh sách liên kết được sử dụng có thể là một hàng đợi hoặc ngăn xếp.

Trong việc lookup ArrayList truy cập nhanh hơn LinkedList. Đối với LinkedList, muốn truy cập vào item hay chỉ số bất kỳ yêu cầu phải đi qua nhiều nút.

Thêm và xóa các phần tử trong LinkedList thường nhanh hơn so với ArrayList. Tuy nhiên, điều này còn phụ thuộc vào kích thước của collection và vị trí các chỉ số.

3. Các triển khai(implement) của Set

3.1. HashSet và TreeSet

HashSet implement Set interface. Các item trong HashSet không được sắp xếp trật tự.

TreeSet là một tập được sắp xếp, các yếu tố sẽ được xếp theo thứ tự tăng dần. Các yếu tố được sắp xếp theo trình tự tự nhiên hoặc bằng cách so sánh bởi các quy định tại lúc khởi tạo.

3.2. LinkedHashSet và HashSet

Cũng giống như ArrayList và LinkedList, LinkedHashSet và HashSet khác ở chỗ LinkedHashSet duy trì một danh sách liên kết kép có chứa các hashCode và thứ tự ban đầu của các item.

4. Các triển khai(implement) của Map

4.1. HashMap và Hashtable

Sự khác nhau giữa 2 cái là việc truy nhập đến Hashtable là đồng bộ(Synchronized) trong khi với HashMap thì không.

Synchronized ở đây có nghĩa là chỉ có một luồng có thể modify một Hashtable tại một điểm trong cùng một thời gian. Nếu có một luồng khác nào muốn update trên Hashtable đó thì nó phải chiếm được quyền kiểm soát trên đối tượng trong khi các luồng kia sẽ phải đợi để nhả khóa.

HashTable là luồng an toàn(thread safe) bởi khi có nhiều luồng truy nhập đến một HashTable thì chỉ có một luồng thực thi update sau khi đã khóa để giữ toàn vẹn dữ liệu.

HashMap không an toàn khi có nhiều luồng truy nhập vào HashMap và một trong các luồng đó cố update dữ liệu và sau đó nó sẽ tung ra một ngoại lệ (Exception). Chúng ta sử dụng HashMap nếu bạn chắc chắn rằng HashMap sẽ không bị truy nhập bởi nhiều luồng. Tuy HashMap không phải là luồng an toàn nhưng chính vì thế nó sẽ thực thi nhanh hơn so với HashTable.

Một sự khác biệt nữa là HashMap cho phép có value là null còn HashTable thì không.

Chính vì các lý do trên, bây giờ các lập trình viên thường sử dụng HashMap.

4.2. *TreeMap* và *HashMap*

TreeMap là cây Đỏ-Đen cây dựa trên giao diện SortedMap. TreeMap đảm bảo rằng các item trong nó sẽ được xếp tăng dần theo trình tự tự nhiên của các khóa, hoặc do so sánh được cung cấp khi khởi tạo. Các item trong HashMap thì không được sắp xếp.

Tìm kiếm một mục trong TreeMap là chậm hơn so với HashMap.

4.3. *LinkedHashMap* và *HashMap*

Sự khác biệt cơ bản giữa HashMap và LinkedHashMap là LinkedHashMap duy trì trật tự chèn các phần tử. LinkedHashMap duy trì một danh sách liên kết kép có chứa các hashCode và thứ tự ban đầu của các item.