Lập trình Java

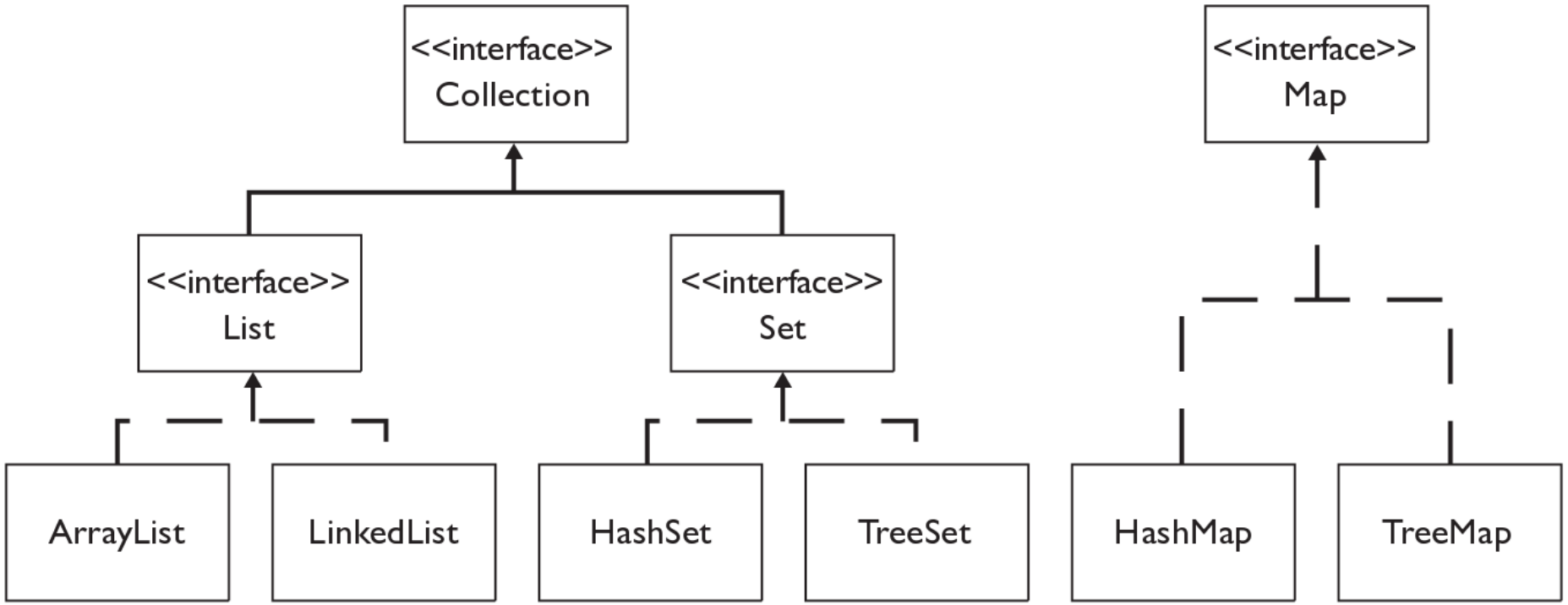
Tuần 6: Collection

Nội dung

Tìm hiểu về Collection trong java: Array, Arraylist, Map

Collection

Collection là một nền tảng tập hợp, giúp đơn giản hóa khi làm việc với tập hợp và đồ thị.
Collection 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).



Java's `Collection` interface

Java's `Collection` interface nằm trong java.until
Có nhiều phương thức:

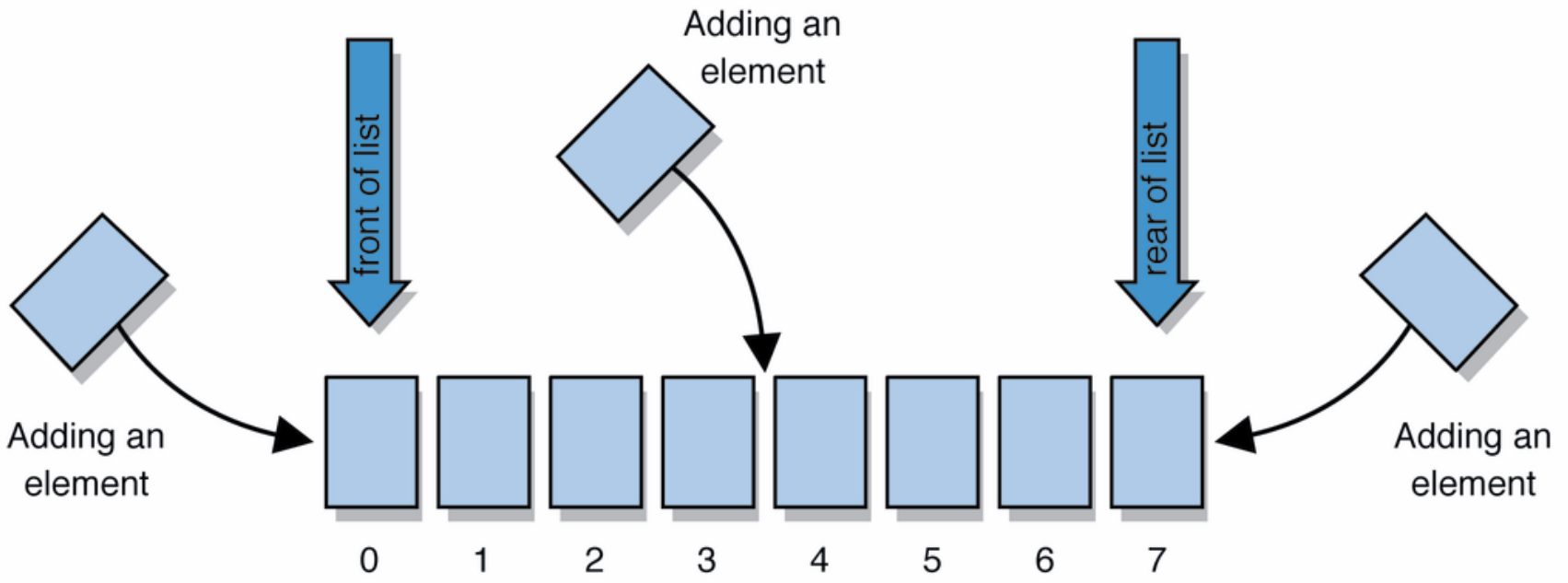
Method name	Description
`boolean add(`*value*`)`**	adds the given value to the collection
`boolean remove(`*value*`)`**	removes the first occurrence in this list of the specified element
`boolean addAll(`*collection*`)`**	adds all elements from given collection to this one
`void clear()`	removes all elements
`boolean contains(`*value*`)`**	returns **`true`** if the element is in the collection
`boolean containsAll(`*collection*`)`**	**`true`** if this collection contains all elements from the other
`boolean isEmpty()`	**`true`** if the collection does not contain any elements
`boolean removeAll(`*collection*`)`**	removes all values contained in the given collection from this collection
`boolean retainAll(`*collection*`)`**	removes all values not contained in the given collection from this collection
`int size()`	returns the number of elements in the collection
`Object[] toArray()`	returns an array containing the elements of this collection
`Iterator<E> iterator`	returns a special object for examining the elements of the list in order

List Interface

- List: là một collection có thứ tự (đôi khi còn được gọi là một chuỗi). List có thể chứa các phần tử trùng lặp. Thường có quyền kiểm soát chính xác vị trí các phần tử được chèn vào và có thể truy cập chúng bằng chỉ số.
- List Interface cũng được kế thừa và có đầy đủ các phương thức của Collection Interface: ArrayList and LinkedList
  + ArrayList là một mảng có kích thước thay đổi được, vì tương tự mảng nên nó rất nhanh trong việc tạo và thêm vào hay xóa đi theo thứ tự nhưng lại chậm hơn nếu thêm vào hay xóa ở 1 vị trí bất kì.
  + LinkedList: là một class dạng list hoạt động trên cơ sở của cấu trúc dữ liệu danh sách liên kết đôi (double-linked list)
- Thông thường các phần tử được thêm vào cuối danh sách (mặc định). Các phần tử có thể thêm vào tại vị trí chỉ định, thêm, xoá, sửa, tìm theo vị trí,...
- Khi sử dụng List người dùng không cần quan tâm tới kích thước của mảng.

List Interface

- Vị trí bắt đầu là 0. Líst là một trong những collections cơ bản và được sử dụng rất nhiều.

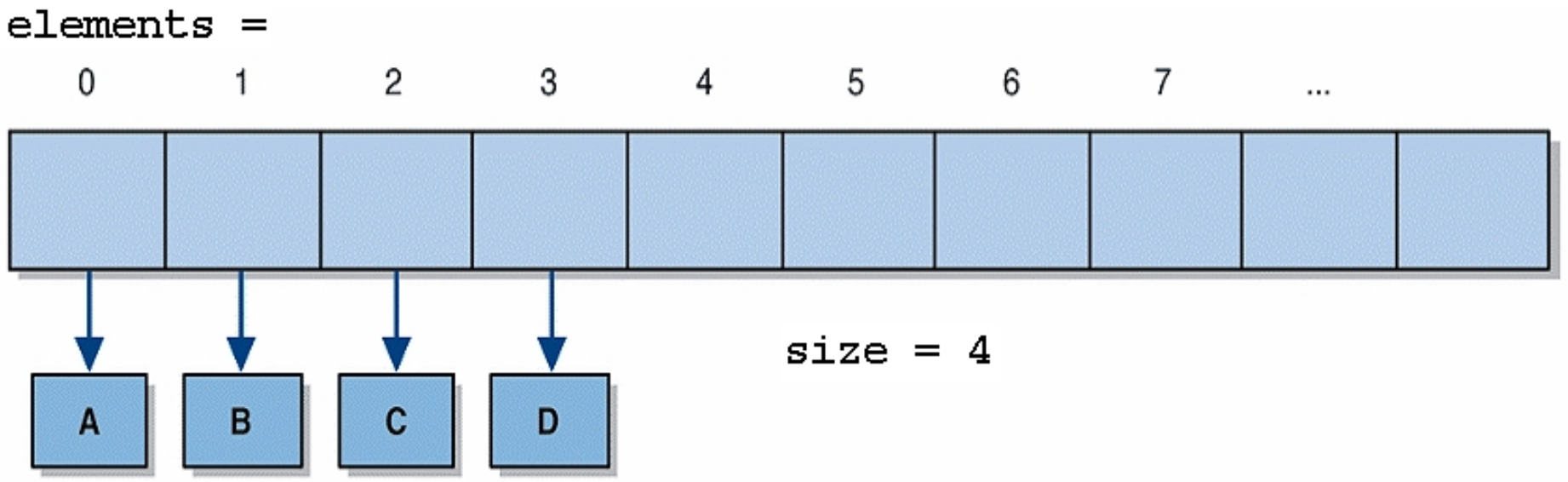


List Interface

- **`List<E>`** có thể được dùng để lưu một danh sách thuộc kiểu đối tượng. Một số phương thức thường dùng
- **`public void add(int index, E element)`**
- **`public E get(int index)`**
- **`public int indexOf(Object o):`** Trả về vị trí trong danh sách về lần xuất hiện đầu tiên của phần tử đã chỉ định hoặc -1 nếu danh sách không chứa nó.
- **`public int lastIndexOf(Object o:`** Trả về vị trí trong danh sách về lần cuối xuất hiện của phần tử đã chỉ định hoặc -1 nếu danh sách không chứa nó.
- **`public E remove(int index)`**

The `ArrayList` class

- **`ArrayList<E>` cần khai báo `java.util.*`**



The `ArrayList` class

- **ArrayList như một mảng tự động thay đổi kích thước có thể chứa bất kỳ loại đối tượng nào, với nhiều phương thức thuận tiện**
- **duy trì hầu hết các lợi ích của mảng, chẳng hạn như truy cập ngẫu nhiên nhanh**

ArrayList methods

Method name	Description
`add(`*value***`)`**	adds the given value to the end of the list
`add(`*index*`, `*value***`)`**	inserts the given value before the given index
`clear()`	removes all elements
`contains(`*value***`)`**	returns **`true`** if the given element is in the list
`get(`*index***`)`**	returns the value at the given index
`indexOf(`*value***`)`**	returns the first index at which the given element appears in the list (or -1 if not found)
`lastIndexOf(`*value***`)`**	returns the last index at which the given element appears in the list (or -1 if not found)
`remove(`*index***`)`**	removes value at given index, sliding others back
`size()`	returns the number of elements in the list

Generic classes

- Generic: Tham số hóa dữ liệu được viết để tạo ra và sử dụng một class, interface, method với nhiều kiểu dữ liệu khác nhau.
- Các lớp chung ("tham số hóa") đã được thêm vào Java để cải thiện mức độ an toàn của Colections trong Java.
  Các tham số hóa được đặt trong dấu < và >.

- ArrayList <E> là một lớp chung.
  <E> với E là loại phần tử bạn muốn lưu trữ trong ArrayList.

- Thí dụ:
  ArrayList <String> words = new ArrayList <String> ();

  Bây giờ các phương thức của words sẽ thao tác và trả về Strings.

Wrapper classes

`ArrayList`s chỉ chứa kiểu đối tượng.

e.g. **`ArrayList<int>`** không chấp nhận

Nếu bạn muốn lưu trữ các kiểu nguyên thủy trong một ArrayList, bạn phải khai báo nó bằng cách sử dụng lớp Wrapper.

Primitive type	Wrapper class
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

- Ví dụ: **`ArrayList<Integer> list = new ArrayList<Integer>();`**

Wrapper classes

- Ví dụ

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(13);
list.add(47);
list.add(15);
list.add(9);
int sum = 0;
for (int n : list) {
    sum += n;
}
System.out.println("list = " + list);
System.out.println("sum = " + sum);
```

- Output:

```
list = [13, 47, 15, 9]
sum = 84
```

ArrayList và array

- array

```
String[] names = new String[5];
names[0] = "Jennifer";
String name = names[0];
```

- ArrayList

```
ArrayList<String> namesList = new ArrayList<String>();
namesList.add("Jennifer");
String name = namesList.get(0);
```

Thêm phần tử

Phần tử được thêm vào danh sách

```
ArrayList<String> list = new ArrayList<String>();
System.out.println("list = " + list);
list.add("Tool");
System.out.println("list = " + list);
list.add("Phish");
System.out.println("list = " + list);
list.add("Pink Floyd");
System.out.println("list = " + list);
```

Output:

```
list = []
list = [Tool]
list = [Tool, Phish]
list = [Tool, Phish, Pink Floyd]
```

Xóa phần tử

- Xóa phần tử bởi vị trí:

```
System.out.println("before remove list = " +
 list);
list.remove(0);
list.remove(1);
System.out.println("after remove list = " +
 list);
```

- Output:

```
before remove list = [Tool, U2, Phish, Pink
 Floyd]
after remove list = [U2, Pink Floyd]
```

Lưu ý rằng khi mỗi phần tử được loại bỏ, các phần tử khác sẽ dịch chuyển xuống vị trí để lấp đầy chỗ lại ngay.

Do đó, loại bỏ Phish, không phải U2.

Tìm phần tử

- Tìm kiếm trong danh sách

```
int index = list.indexOf("Phish");
if (index > -1) {
    System.out.println(index + " " + list.get(index));
}

if (list.contains("Madonna")) {
    System.out.println("Madonna is in the list");
} else {
    System.out.println("Madonna is not found.");
}
```

- Output:

```
2 Phish
Madonna is not found.
```

Contains cho bạn biết một phần tử có trong danh sách hay không

indexOf cho bạn biết chỉ số nào bạn có thể tìm thấy nó

Sắp xếp phần tử

Sắp xếp lại các giá trị trong danh sách theo một thứ tự nhất định (thường theo thứ tự tăng dần tự nhiên của chúng).

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	15	2	8	1	17	10	12	5

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	1	2	5	8	10	12	15	17

Sắp xếp phần tử

- Java cung cấp 2 phương thức sắp xếp
  - **`Arrays.sort`** (for an array)

```
// demonstrate the Arrays.sort method
String[] strings = {"c", "b", "g", "h", "d", "f", "e", "a"};
System.out.println(Arrays.toString(strings));
Arrays.sort(strings);
System.out.println(Arrays.toString(strings));
```

  - Output:

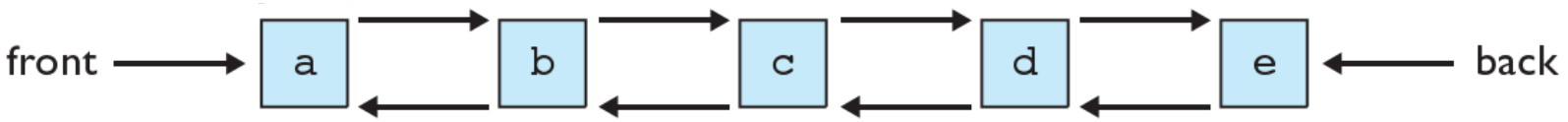
```
[c, b, g, h, d, f, e, a]
[a, b, c, d, e, f, g, h]
```

  - **`Collections.sort`** (for a **`List`**)

Linked list

Linked list: danh sách được triển khai bằng chuỗi giá trị được liên kết.

Mỗi giá trị được lưu trữ trong một đối tượng nhỏ gọi là nút và được tham chiếu đến các nút lân cận của nó



Linked list

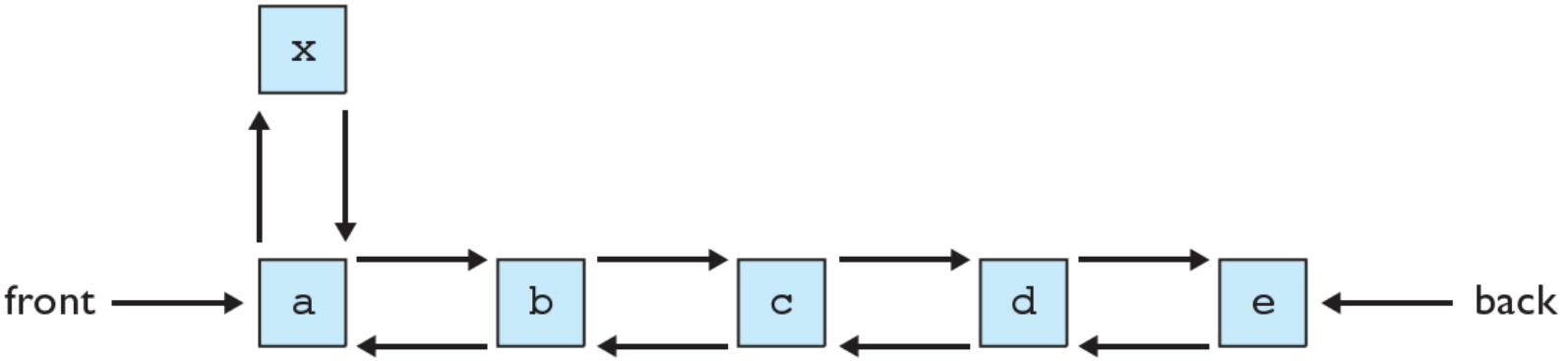
- A **`LinkedList`** dùng như **`ArrayList`**:

```
LinkedList <String> words = new
  LinkedList<String>();
words.add("hello");
words.add("goodbye");
words.add("this");
words.add("that");
```

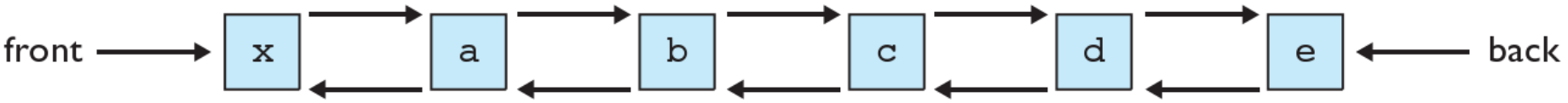
Linked list

1. Make a new node to hold the new element.

2. Connect the new node to the other nodes in the list.



3. Change the front of the list to point to the new node.



Set Interface

- Set: một bộ sưu tập không có thứ tự không có bản sao. Thường được sử dụng để mô tả các tập hợp như bộ bài, thời khóa biểu của học sinh hay tiến trình của máy tính.
- Mục đích chính của một Set là tìm kiếm, để kiểm tra các đối tượng để trở thành thành viên trong Set.
- Đây là một interface.
- Có hai triển khai Set trong Java:TreeSet và Hashset

Set Interface

- Hashset, LinkedHashset và Treeset là các class kế thừa của Set interface. Trong đó:
  + Hashset được sử dụng trong hầu hết các trường hợp.
  + LinkedHashset là sự kết hợp giữa Hashset và List, các phần tử được sắp xếp theo thứ tự như dánh sách nhưng không cho phép các phần tử trùng lặp.
  + Treeset các phần tử trong set đã được sắp xếp.

Set Interface

- Ví dụ:

```
Set<String> stooges = new HashSet<String>();
stooges.add("Larry");
stooges.add("Moe");
stooges.add("Curly");
stooges.add("Moe");    // duplicate, won't be
 added
stooges.add("Shemp");
stooges.add("Moe");    // duplicate, won't be
 added
System.out.println(stooges);
```

- Output:

```
[Moe, Shemp, Larry, Curly]
```

Lưu ý rằng thứ tự của các stooges không khớp với thứ tự mà chúng được thêm vào, cũng không phải là thứ tự chữ cái tự nhiên.

TreeSet vs. HashSet

- Dùng **`TreeSet`**:

```
Set<String> stooges = new
 TreeSet<String>();
...
System.out.println(stooges);
```

- Output:

```
[Curly, Larry, Moe, Shemp]
```

- **`TreeSet`** VS. **`HashSet`**:

Một TreeSet lưu trữ các yếu tố của nó theo thứ tự bảng chữ cái tự nhiên.

TreeSet chỉ có thể được sử dụng với các phần tử có thứ tự (ví dụ: nó không thể dễ dàng lưu trữ các đối tượng Point).

TreeSet hơi chậm (thường không đáng) hơn Hashset.

Map Interface

- map: một bộ sưu tập không có thứ tự, liên kết một tập hợp các giá trị phần tử với một bộ key để có thể tìm thấy các phần tử rất nhanh
- Mỗi khóa có thể xuất hiện tối đa một lần (không có khóa trùng lặp)
- Một ánh xạ chính tới nhiều nhất một giá trị
- các hoạt động chính:
- put (key, value)
  "Ánh xạ khóa này tới giá trị đó."
- get (key)
  “ giá trị của key”
- Maps được thể hiện bằng Java bởi Map <K, V>.
- Hai triển khai: HashMap và TreeMap
- Map Interface không kế thừa từ Collection Interface mà đây là 1 interface độc lập với các phương thức của riêng mình

Map Interface

Đôi khi muốn tạo ánh xạ giữa các phần tử của một bộ và một bộ khác

Vd: ánh xạ từ để đếm từ trong cuốn sách

- "the" --> 325
- "whale" --> 14

vd: ánh xạ người đến số điện thoại của họ

- "Marty Stepp" --> "692-4540"
- "Jenny" --> "867-5309"

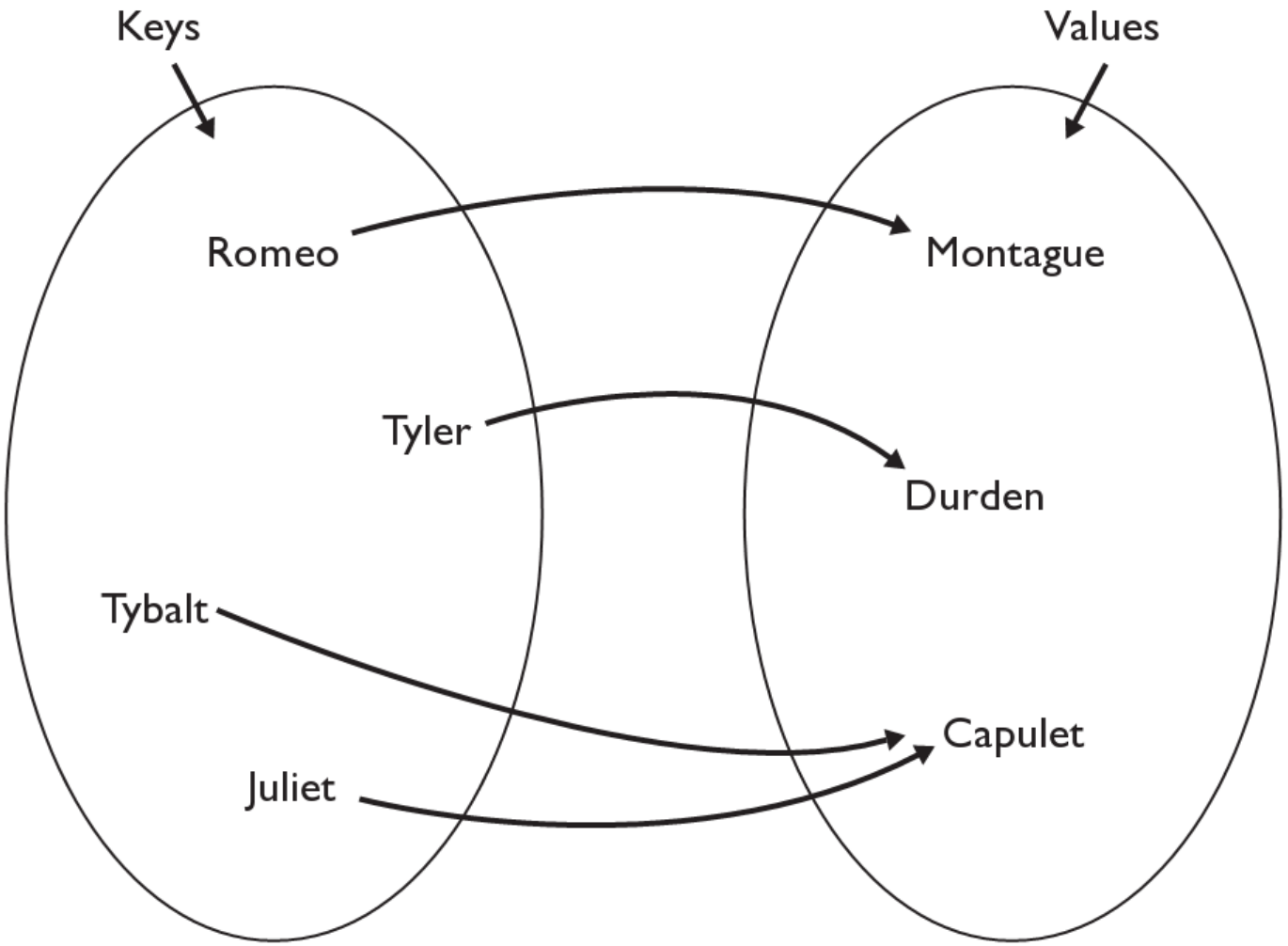
Làm thế nào chúng ta sẽ làm điều này với một danh sách?

Một danh sách không ánh xạ mọi người đến số điện thoại

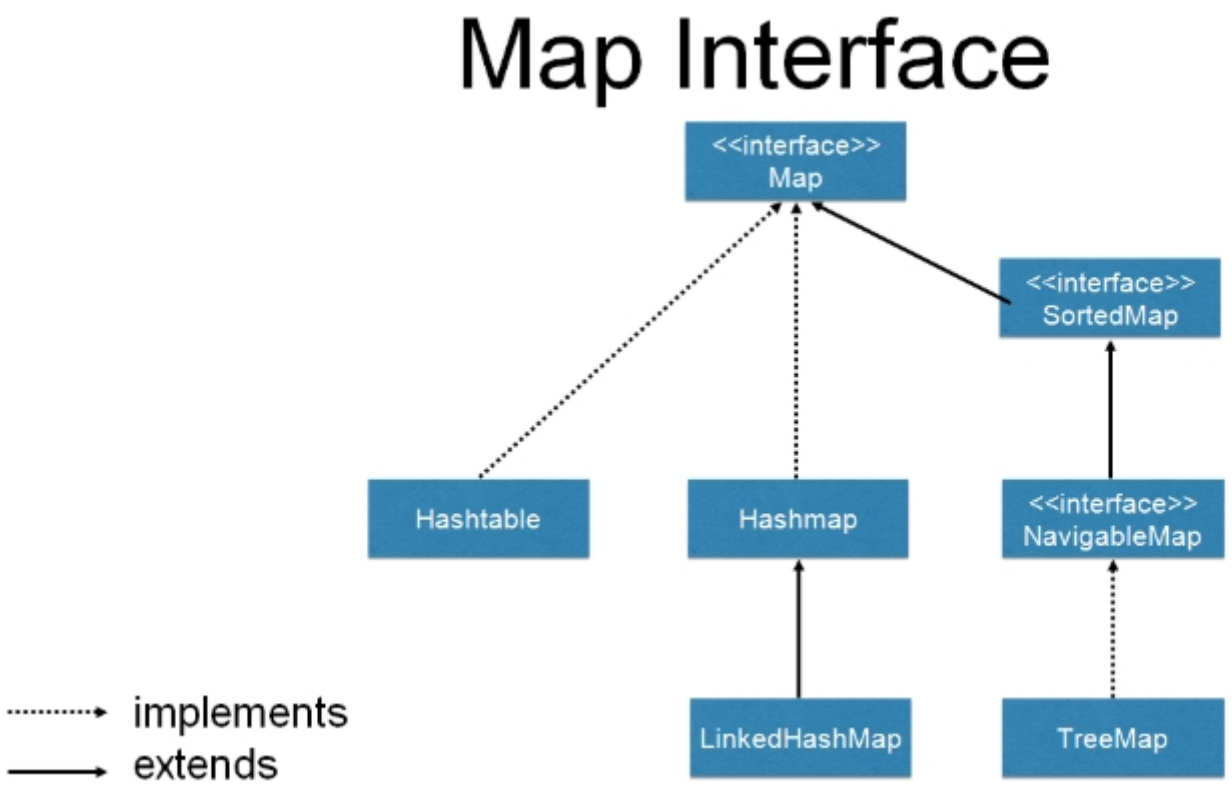
Chúng ta có thể ánh xạ một số int thành tên của một người và cùng một int với số điện thoại của người đó?

Làm thế nào chúng ta sẽ tìm thấy một số điện thoại, được đặt tên của người đó?

Map Interface



Map Interface



Map Interface

Map không inplement Collection interface nhưng chúng có các phương thức chung sau:

Method name	Description
`clear()`	removes all keys and values from the map
`containsKey(`*key*`)`	returns `true` if the given key exists in the map
`containsValue(`*value*`)`	returns `true` if the given value exists in the map
`get(`*key*`)`	returns the value associated with the given key (`null` if not found)
`isEmpty()`	returns `true` if the map has no keys or values
`keySet()`	returns a collection of all keys in the map
`put(`*key*`, `*value*`)`	associates the given key with the given value
`putAll(`*map*`)`	adds all key/value mappings from given map
`remove(`*key*`)`	removes the given key and its associated value
`size()`	returns the number of key/value pairs in the map
`values()`	returns a collection of all values in the map

Map Interface

- HashMap là triển khai mặc định được sử dụng trong hầu hết các trường hợp, thứ tự các phần tử không dựa theo thứ tự lúc thêm vào
- Map không đảm bảo về cách thức lưu trữ các yếu tố bên trong. LinkedHashMap là một ngoại lệ, cho phép lặp lại bản đồ theo thứ tự chèn (thứ tự các phần tử dựa theo thứ tự lúc thêm vào)
- TreeMap thì thứ tự các phần tử được sắp xếp theo chiều tăng dần của khóa

Map Interface

Maps được khai báo với hai tham số loại, một cho các khóa và một cho các giá trị:

```
Map<String, Double> salaryMap = new HashMap<String, Double>();
salaryMap.put("Stuart", 20000.00);
salaryMap.put("Marty", 15500.00);
salaryMap.put("Jenny", 86753.09);
System.out.println(salaryMap);

// search the map for a name
if (salaryMap.containsKey("Jenny")) {
    double salary = salaryMap.get("Jenny");
    System.out.println("Jenny's salary is $" + salary);
} else {
    System.out.println("I don't have a record for Jenny");
}
```

- Output:

```
{Jenny=86753.09, Stuart=20000.0, Marty=15500.0}
Jenny's salary is $86753.09
```

Map Interface

```
import java.util.*;

public class Birthday {
    public static void main(String[] args){
        Map<String, Integer> m = new HashMap<String, Integer>();
        m.put("Newton", 1642);
        m.put("Darwin", 1809);
        System.out.println(m);

        Set<String> keys = m.keySet();
        Iterator<String> itr = keys.iterator();
        while (itr.hasNext()) {
            String key = itr.next();
            System.out.println(key + " => " + m.get(key));
        }
    }
}
```

- Output:

```
{Darwin=1809, Newton=1642}
Darwin => 1809
Newton => 1642
```

Map Interface

```
public static void main(String[] args) {
    Map<Integer, String> hashMap = new HashMap<>();
     hashMap.put(1, "One");
    hashMap.put(0, "Zero");
    hashMap.put(2, "Two");
    hashMap.put(4, "Four");
    hashMap.put(21, "Twenty first");
    hashMap.put(5, "Five");
     Map<Integer, String> linkedHashMap = new LinkedHashMap<>();
    linkedHashMap.put(1, "One");
    linkedHashMap.put(0, "Zero");
    linkedHashMap.put(2, "Two");
    linkedHashMap.put(4, "Four");
    linkedHashMap.put(5, "Five");
    linkedHashMap.put(21, "Twenty first");
     Map<Integer, String> treeMap = new TreeMap<>();
     treeMap.put(1, "One");
    treeMap.put(0, "Zero");
    treeMap.put(2, "Two");
    treeMap.put(4, "Four");
    treeMap.put(21, "Twenty first");
    treeMap.put(5, "Five");
        //in ra
}
```

```
Các phần tử có trong hashMap:
{0=Zero, 1=One, 2=Two, 4=Four, 21=Twenty first, 5=Five}
Các phần tử có trong linkedHashMap:
{1=One, 0=Zero, 2=Two, 4=Four, 5=Five, 21=Twenty first}
Các phần tử có trong treeMap:
{0=Zero, 1=One, 2=Two, 4=Four, 5=Five, 21=Twenty first}
```

Map Interface

Viết mã để đảo ngược Map; nghĩa là, để làm cho các giá trị thành các khóa và làm cho các khó
trở thành các giá trị.

```
Map<String, String> byName =
    new HashMap<String, String>();
byName.put("Darwin", "748-2797");
byName.put("Newton", "748-9901");

Map<String, String> byPhone = new HashMap<String, String>();
// ... your code here!
System.out.println(byPhone);
```

Output:

```
{748-2797=Darwin, 748-9901=Newton}.
```

Bài tập tuần 6

Viết chương trình đếm từ trong tệp văn bản (tìm hiểu phần đọc file trước), sử dụng HashMap để lưu trữ số lần xuất hiện của mỗi từ