

# HUST

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



ĐẠI HỌC  
BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY  
OF SCIENCE AND TECHNOLOGY

# Bài 9: Lập trình tổng quát

ONE LOVE. ONE FUTURE.

# Mục tiêu

- Giới thiệu về lập trình tổng quát và cách thực hiện trong các ngôn ngữ lập trình
- Giới thiệu về collection framework với các cấu trúc tổng quát: List, HashMap, Tree, Set, Vector,...
- Định nghĩa và sử dụng Template và ký tự đại diện (wildcard)
- Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



# Nội dung

---

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập



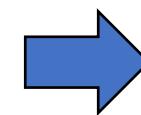
1. **Giới thiệu về lập trình tổng quát**
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập

# 1. Giới thiệu về lập trình tổng quát

- Lập trình tổng quát (Generic programming) : Tổng quát hóa chương trình để có thể hoạt động với các kiểu dữ liệu khác nhau, kể cả kiểu dữ liệu trong tương lai
  - Thuật toán đã xác định
- Ví dụ:

Phương thức sort()

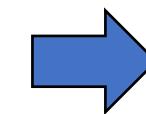
- Số nguyên int
- Xâu ký tự String
- Đối tượng số phức Complex object
- ...



**Thuật toán giống nhau, chỉ khác về kiểu dữ liệu**

Lớp lưu trữ kiểu ngăn xếp (Stack)

- Lớp IntegerStack → đối tượng Integer
- Lớp StringStack → đối tượng String
- Lớp AnimalStack → đối tượng animal,...



**Các lớp có cấu trúc tương tự, khác nhau về kiểu đối tượng xử lý**



# 1. Giới thiệu về lập trình tổng quát

---

- Lập trình tổng quát
  - C: dùng con trỏ không định kiểu (con trỏ void)
  - C++: dùng template
  - Java 1.5 trở về trước: lợi dụng upcasting, downcasting và lớp object
  - Java 1.5: đưa ra khái niệm về template



# 1. Giới thiệu về lập trình tổng quát

---

- Ví dụ C: hàm **memcpy()** trong thư viện **string.h**

**void\* memcpy(void\* region1, const void\* region2, size\_t n);**

- Hàm memcpy() bên trên được khai báo tổng quát bằng cách sử dụng các con trỏ void\*
- Điều này giúp cho hàm có thể sử dụng với nhiều kiểu dữ liệu khác nhau
  - Dữ liệu được truyền vào một cách tổng quát thông qua địa chỉ và kích thước kiểu dữ liệu
  - Hay nói cách khác, để sao chép dữ liệu, ta chỉ cần địa chỉ và kích cỡ của chúng



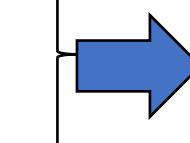
# 1. Giới thiệu về lập trình tổng quát

- Ví dụ: Lập trình tổng quát từ trước Java 1.5

```
public class ArrayList {  
    public Object get(int i) { . . . }  
    public void add(Object o) { . . . }  
    . . .  
    private Object[] elementData;  
}
```

- Lớp Object là lớp cha tổng quát nhất → có thể chấp nhận các đối tượng thuộc lớp con của nó

```
List myList = new ArrayList();  
myList.add("Fred");  
myList.add(new Dog());  
myList.add(new Integer(42));
```



Các đối tượng  
trong một danh  
sách khác hẳn nhau

- Hạn chế: Phải ép kiểu → có thể ép sai kiểu (run-time error)

```
String name = (String) myList.get(1); //Dog!!!
```



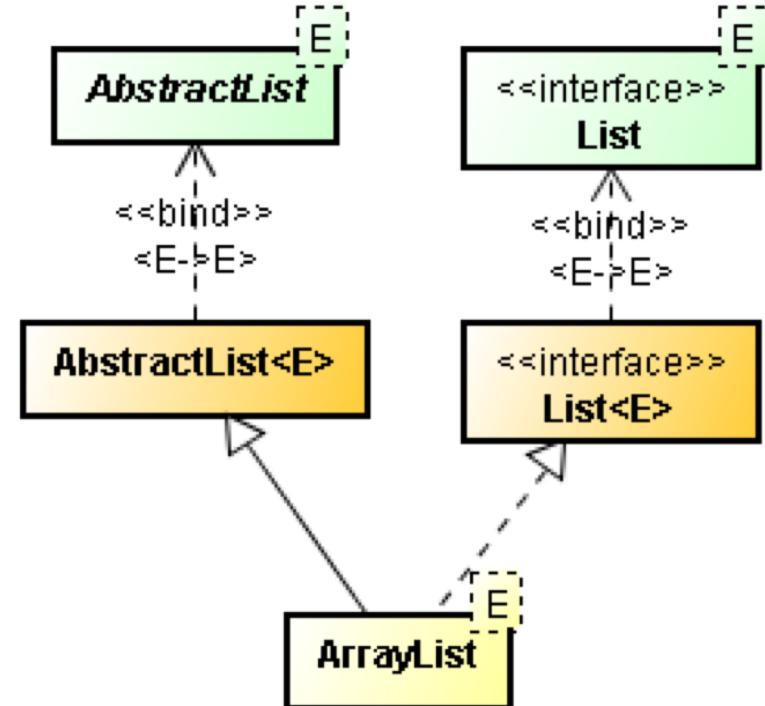
# 1. Giới thiệu về lập trình tổng quát

- Ví dụ: Lập trình Generic từ Java 1.5
  - Java 1.5 Template

Danh sách chỉ chấp nhận các đối tượng có kiểu là Integer



```
List<Integer> myList =  
    new LinkedList<Integer>();  
myList.add(new Integer(0));  
Integer x = myList.iterator().next(); //Không cần ép kiểu  
  
myList.add(new String("Hello")); //Compile Error
```



# Nội dung

---

1. Giới thiệu về lập trình tổng quát
2. **Định nghĩa và sử dụng Template**
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập



# Lớp tổng quát

- Lớp tổng quát (generic class) là lớp có thể nhận kiểu dữ liệu là một lớp bất kỳ
- Cú pháp
  - Tên Lớp <kiểu 1, kiểu 2, kiểu 3...> {  
    }
- Các phương thức hay thuộc tính của lớp tổng quát có thể sử dụng các kiểu được khai báo như mọi lớp bình thường khác



# Lớp tổng quát

Tên kiểu, sẽ được thay thế bằng một kiểu cụ thể khi sử dụng

- Ví dụ:

```
public class Information<T> {  
    private T value;  
    public Information(T value) {  
        this.value = value;  
    }  
    public T getValue() {  
        return value;  
    }  
}  
  
Information<String> mystring =  
    new Information<String>("hello");  
Information<Circle> circle =  
    new Information<Circle>(new Circle());  
Information<2DShape> shape =  
    new Information<>(new 2DShape());
```



# Lớp tổng quát

- Quy ước đặt tên kiểu

Tên kiểu	Mục đích
E	Các thành phần trong một collection
K	Kiểu khóa trong Map
V	Kiểu giá trị trong Map
T	Các kiểu thông thường
S, U	Các kiểu thông thường khác

- Chú ý: Không sử dụng các kiểu dữ liệu nguyên thủy cho các lớp tổng quát

```
Information<int> integer =  
    new Information<int>(2012);           // Error  
Information<Integer> integer =  
    new Information<Integer>(2012); // OK
```



# Phương thức tổng quát

- Phương thức tổng quát (generic method) là các phương thức tự định nghĩa kiểu tham số của nó
- Có thể được viết trong lớp bất kỳ (tổng quát hoặc không)
- Cú pháp
  - (chỉ định truy cập) <kiểu1, kiểu 2...> (kiểu trả về) tên phương thức (danh sách tham số)  
{  
    //...  
}
- Ví dụ

```
public static <E> void print(E[] a) { ... }
```



# Ví dụ Phương thức tổng quát

```
public class ArrayTool {  
    // Phương thức in các phần tử trong mảng String  
    public static void print(String[] a) {  
        for (String e : a) System.out.print(e + " ");  
        System.out.println();  
    }  
  
    // Phương thức in các phần tử trong mảng với kiểu  
    // dữ liệu bất kỳ  
    public static <E> void print(E[] a) {  
        for (E e : a) System.out.print(e + " ");  
        System.out.println();  
    }  
}
```



# Ví dụ Phương thức tổng quát

---

...

```
String[] str = new String[5];
```

```
Point[] p = new Point[3];
```

```
int[] intnum = new int[2];
```

```
ArrayTool.print(str);
```

```
ArrayTool.print(p);
```

// Không dùng được với kiểu dữ liệu nguyên thủy

```
ArrayTool.print(intnum);
```



# Giới hạn kiểu dữ liệu tổng quát

- Có thể giới hạn các kiểu dữ liệu tổng quát sử dụng phải là dẫn xuất của một hoặc nhiều lớp
- Giới hạn 1 lớp  
`<type_param extends bound>`
- Giới hạn nhiều lớp  
`<type_param extends bound_1 & bound_2 & ...>`



# Giới hạn kiểu dữ liệu tổng quát

- Ví dụ:

Chấp nhận các kiểu là lớp con  
của 2DShape

```
public class Information<T extends 2DShape> {  
    private T value;  
    public Information(T value) {  
        this.value = value;  
    }  
    public T getValue() {  
        return value;  
    }  
}  
  
Information<Point> pointInfo =  
    new Information<Point>(new Point()); // OK  
  
Information<String> stringInfo =  
    new Information<String>(); // error
```



1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
- 3. Lập trình tổng quát trong Java collections framework**
4. Ký tự đại diện (Wildcard)
5. Ví dụ và bài tập

### 3. Java Collections Framework

---

- Collection là đối tượng có khả năng chứa các đối tượng khác.
- Các thao tác thông thường trên collection
  - Thêm/Xoá đối tượng vào/khỏi collection
  - Kiểm tra một đối tượng có ở trong collection không
  - Lấy một đối tượng từ collection
  - Duyệt các đối tượng trong collection
  - Xoá toàn bộ collection



### 3. Java Collections Framework

---

- Các collection đầu tiên của Java:
  - Mảng
  - Vector: Mảng động
  - Hastable: Bảng băm
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.



### 3. Java Collections Framework

---

- 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. Java Collections Framework

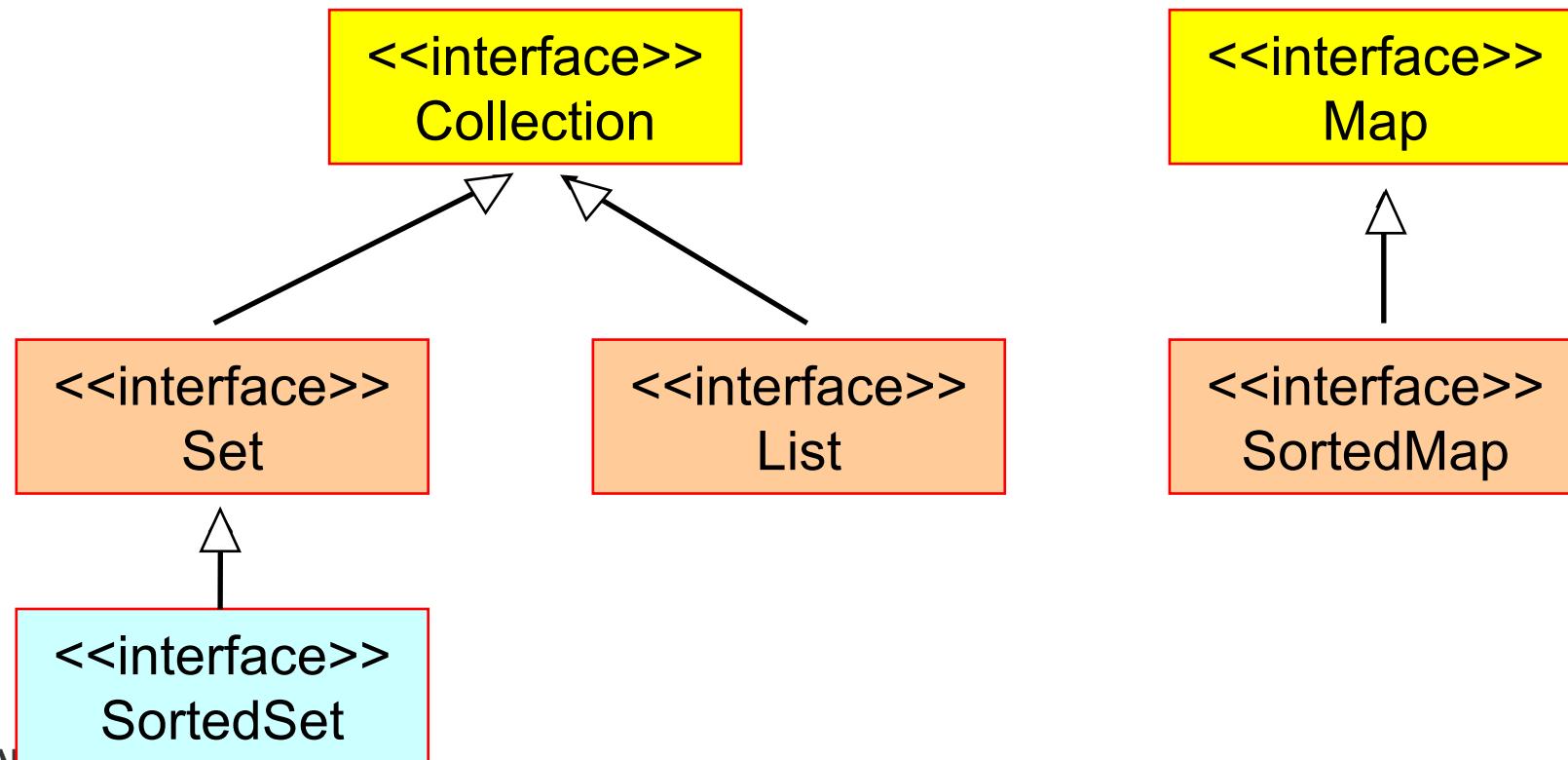
---

- Collections Framework bao gồm
  - 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 tinh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

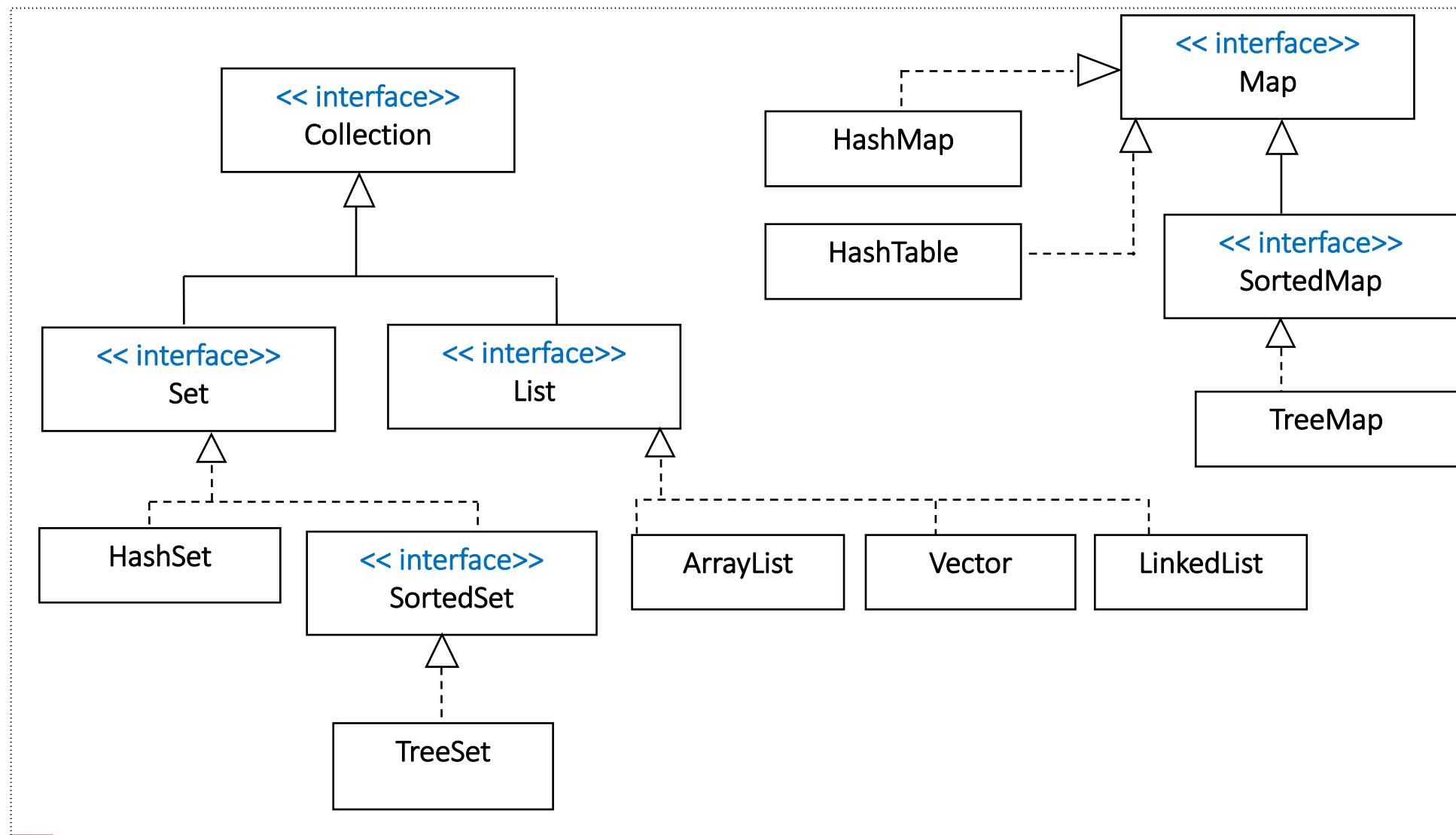


# Interfaces trong Java collections framework

- List: Tập các đối tượng tuần tự, kế tiếp nhau, có thể lặp lại
- Set: Tập các đối tượng không lặp lại
- Map: Tập các cặp khóa-giá trị (key-value) và không cho phép khóa lặp lại



# 3. Java Collections Framework



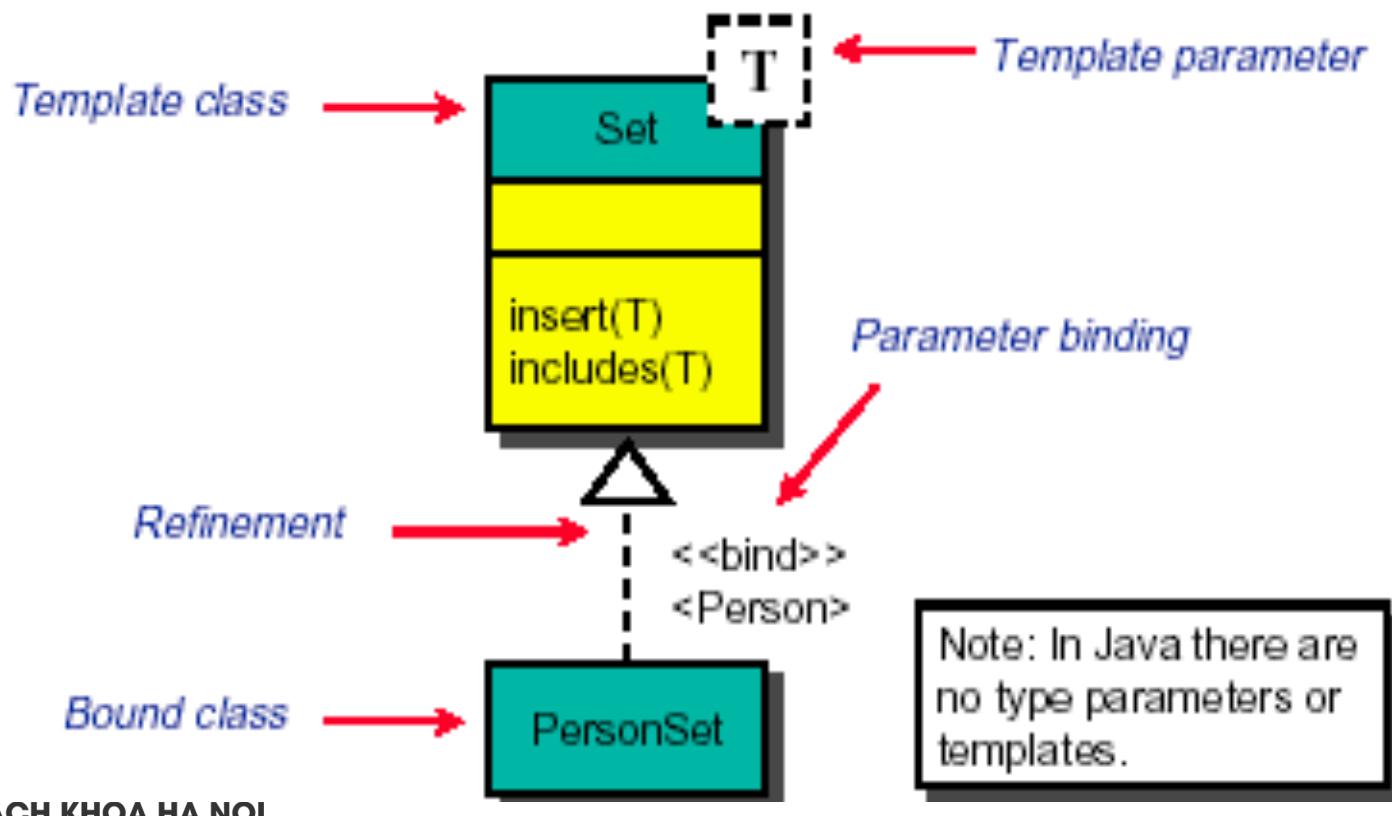
# So sánh Collection và Array

Collection	Array
Collection (có thể) truy xuất theo dạng ngẫu nhiên	Mảng truy xuất 1 cách tuần tự
Collection có thể chứa nhiều loại đối tượng/dữ liệu khác nhau	Mảng chứa 1 loại đối tượng/dữ liệu nhất định
Dùng Java Collection, chỉ cần khai báo và gọi những phương thức đã được định nghĩa sẵn	Dùng tổ chức dữ liệu theo mảng phải lập trình hoàn toàn
Duyệt các phần tử tập hợp thông qua Iterator	Duyệt các phần tử mảng tuần tự thông qua chỉ số mảng



### 3. Java Collections Framework

- Các giao diện và lớp thực thi trong Collection framework của Java đều được xây dựng theo template
  - Cho phép xác định tập các phần tử cùng kiểu nào đó bất kỳ
  - Cho phép chỉ định kiểu dữ liệu của các Collection → hạn chế việc thao tác sai kiểu dữ liệu



# Ví dụ

```
public interface List<E> {  
    void add(E x);  
    Iterator<E> iterator();  
}
```

```
List<String> myList = new ArrayList<String>();  
myList.add("Fred"); // OK  
myList.add(new Dog()); //Compile error!
```

```
String s = myList.get(0);
```



# Giao diện Collection

- Xác định giao diện cơ bản cho các thao tác với một tập các đối tượng
  - Thêm vào collection
  - Xóa khỏi collection
  - Kiểm tra có là thành viên
- Chứa các phương thức thao tác trên các phần tử riêng lẻ hoặc theo khối
- Cung cấp các phương thức cho phép thực hiện duyệt qua các phần tử trên collection (lặp) và chuyển tập hợp sang mảng



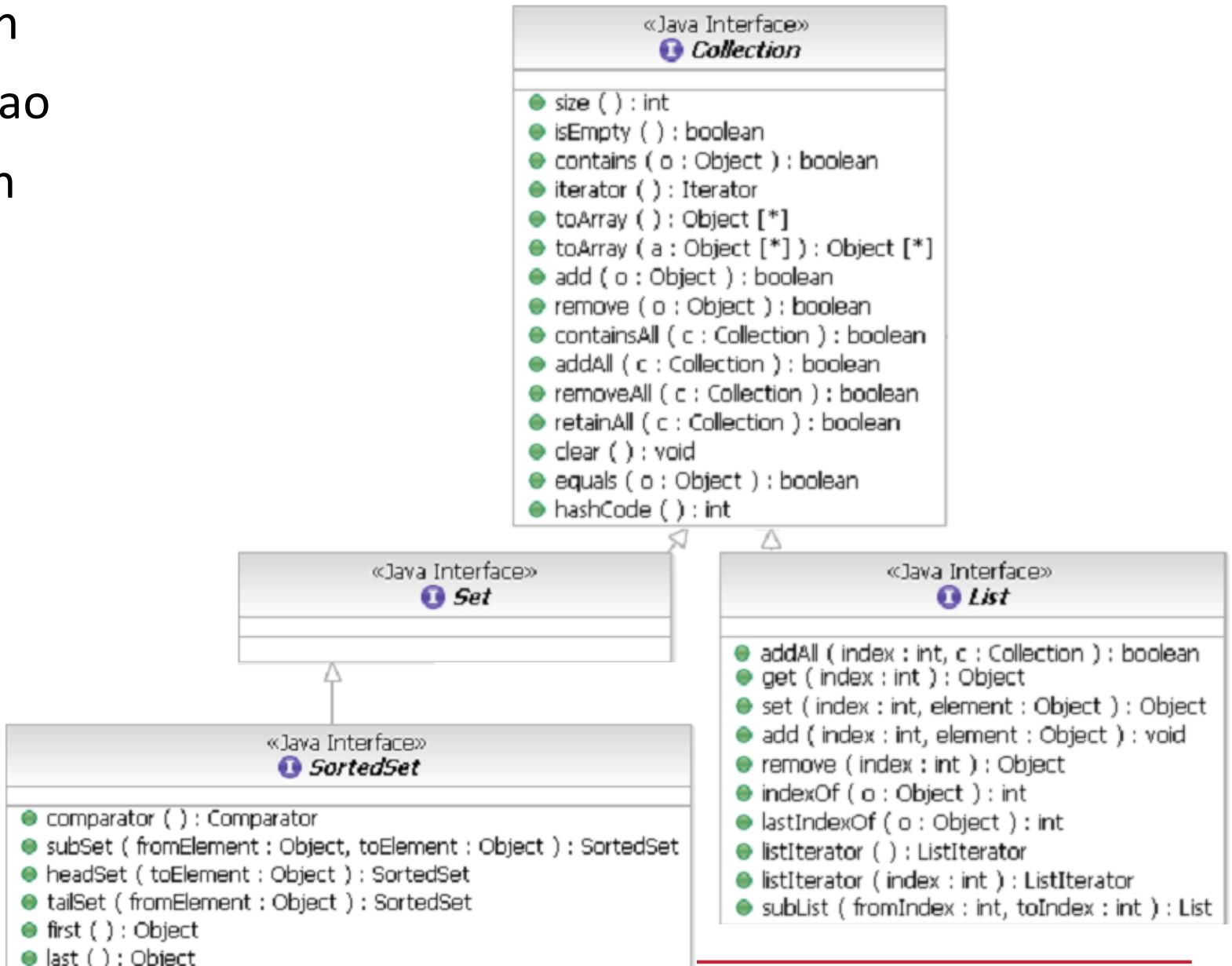
# Giao diện Collection

```
public interface Collection {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(Object element);  
    boolean remove(Object element);  
    Iterator iterator();  
  
    // Bulk Operations  
    boolean addAll(Collection c);  
    boolean removeAll(Collection c);  
    boolean retainAll(Collection c);  
    ...  
    // Array Operations  
    Object[] toArray();  
    Object[] toArray(Object a[]);  
}
```



# Giao diện Collection

- Các giao diện con kế thừa giao diện Collection



# Giao diện Set

- Set kế thừa từ Collection nên cũng hỗ trợ toàn bộ các thao tác xử lý trên Collection
- Ví dụ:
  - Set of cars:
    - {BMW, Ford, Jeep, Chevrolet, Nissan, Toyota, VW}
    - Nationalities in the class
      - {Chinese, American, Canadian, Indian}
- Set là một tập hợp các phần tử **không được trùng lặp**.
- Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.



# Giao diện SortedSet

- **SortedSet:** kế thừa giao diện Set
  - Các phần tử được sắp xếp theo một thứ tự
  - Không có các phần tử trùng nhau
  - Cho phép một phần tử là null
  - Các đối tượng đưa vào trong một SortedSet phải cài đặt giao diện Comparable hoặc lớp cài đặt SortedSet phải nhận một Comparator trên kiểu của đối tượng đó
- **Một số phương thức:**
  - first( ): lấy phần tử đầu tiên (nhỏ nhất)
  - 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



# Giao diện List

- List kế thừa từ Collection. List 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
- Một số phương thức của List
  - Object get(int index);
  - Object set(int index, Object o);
  - void add(int index, Object o);
  - Object remove(int index);
  - int indexOf(Object o);
  - int lastIndexOf(Object o);



# Giao diện Map

- Xác định giao diện cơ bản để thao tác với một tập hợp bao gồm cặp khóa-giá trị
  - Thêm một cặp khóa-giá trị
  - Xóa một cặp khóa-giá trị
  - Lấy về giá trị với khóa đã có
  - Kiểm tra có phải là thành viên (khóa hoặc giá trị)
- Cung cấp 3 cách nhìn cho nội dung của tập hợp:
  - Tập các khóa
  - Tập các giá trị
  - Tập các ánh xạ khóa-giá trị



# Giao diện Map

- Giao diện Map cung cấp các thao tác xử lý trên các bảng ánh xạ
  - Bảng ánh xạ lưu các phần tử theo khoá và không được có 2 khoá trùng nhau
- Một số phương thức của Map
  - Object put(Object key, Object value);
  - Object get(Object key);
  - Object remove(Object key);
  - boolean containsKey(Object key);
  - boolean containsValue(Object value);
  - ...



# Giao diện SortedMap

---

- Giao diện SortedMap
  - thừa kế giao diện Map
  - các phần tử được sắp xếp theo thứ tự
  - tương tự SortedSet, tuy nhiên việc sắp xếp được thực hiện với các khóa
- Phương thức: Tương tự Map, bổ sung thêm:
  - **firstKey( )**: returns the first (lowest) value currently in the map
  - **lastKey( )**: returns the last (highest) value currently in the map



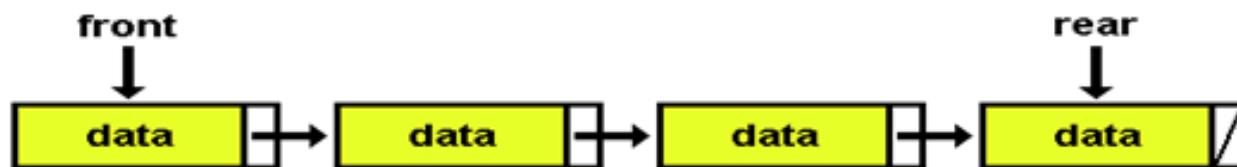
# Các lớp thực thi giao diện Collection

- Java đã xây dựng sẵn một số lớp thực thi các giao diện Set, List và Map và cài đặt các phương thức tương ứng

		IMPLEMENTATIONS				
INTERFACES	Set	HashTable	Resizable Array	Balanced Tree	LinkedList	Legacy
	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		HashTable, Properties

# Các lớp thực thi giao diện Collection

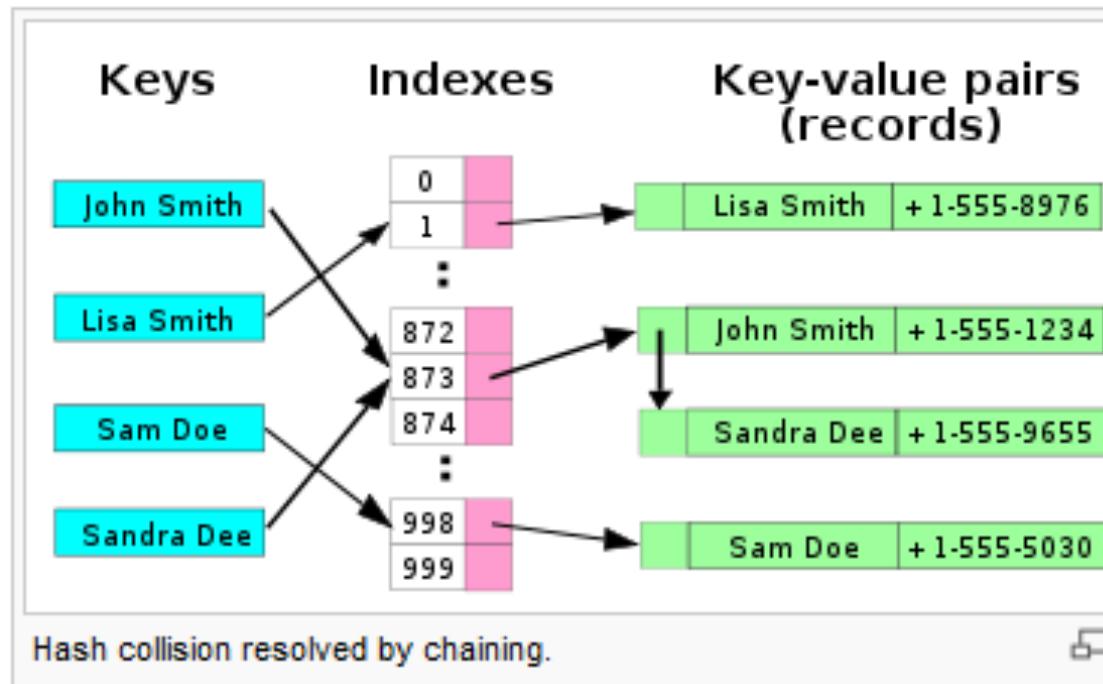
- **ArrayList**: Mảng động, nếu các phần tử thêm vào vượt quá kích cỡ mảng, mảng sẽ tự động tăng kích cỡ
- **LinkedList**: Danh sách liên kết
  - Hỗ trợ thao tác trên đầu và cuối danh sách
  - Được sử dụng để tạo ngăn xếp, hàng đợi, cây...



# Các lớp thực thi giao diện Collection

- HashSet: Bảng băm

- Lưu các phần tử trong một bảng băm
- Không cho phép lưu trùng lặp
- Cho phép phần tử null



# Các lớp thực thi giao diện Collection

- **LinkedHashSet**: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử
  - Thừa kế HashSet và thực thi giao diện Set
  - Khác HashSet ở chỗ nó lưu trữ trong một danh sách mốc nối đôi
  - Thứ tự các phần tử được sắp xếp theo thứ tự được insert vào tập hợp
- **TreeSet**: Cho phép lấy các phần tử trong tập hợp theo thứ tự đã sắp xếp
  - Các phần tử được thêm vào TreeSet tự động được sắp xếp
  - Thông thường, ta có thể thêm các phần tử vào HashSet, sau đó convert về TreeSet để duyệt theo thứ tự nhanh hơn



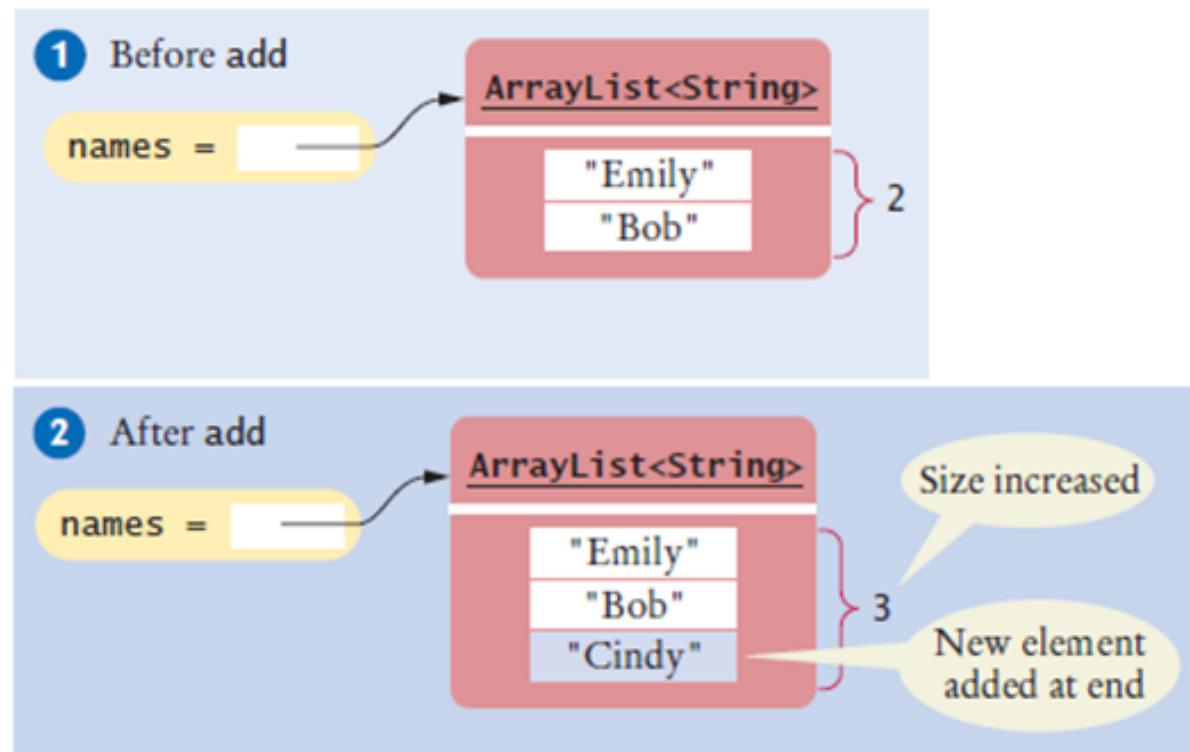
# Các lớp thực thi giao diện Collection

- **HashMap**: Bảng băm (cài đặt của Map)
- **LinkedHashMap**: Bảng băm kết hợp với linked list nhằm đảm bảo thứ tự các phần tử (cài đặt của Map)
- **TreeMap**: Cây (cài đặt của Map)
- **Legacy Implementations**
  - Là các lớp cũ được cài đặt bổ sung thêm các collection interface.
  - **Vector**: Có thể thay bằng ArrayList
  - **Hashtable**: Có thể thay bằng HashMap



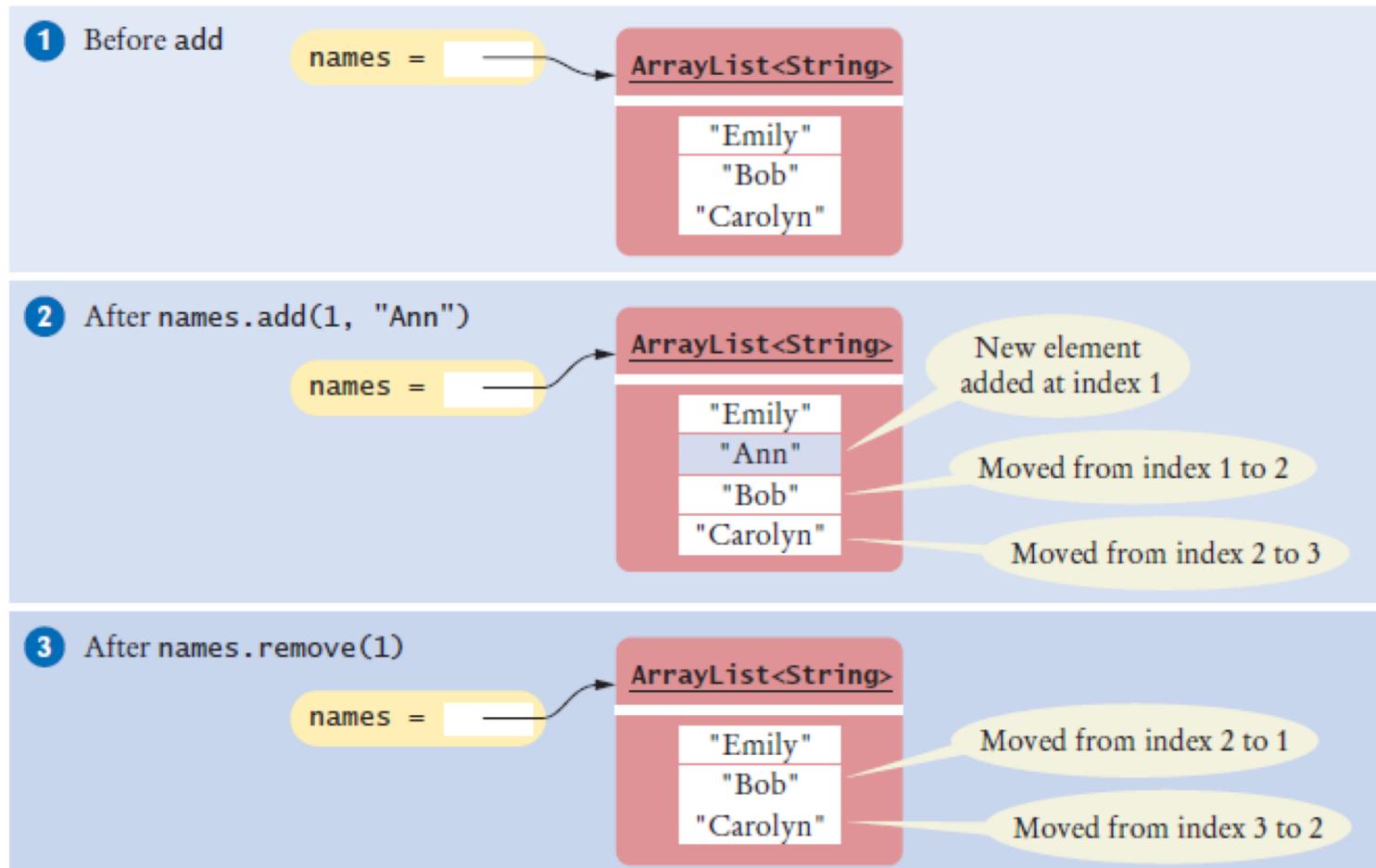
# Ví dụ

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Emily");  
names.add("Bob");  
names.add("Cindy");
```



# Ví dụ

```
String name = names.get(0);  
names.add(1, "Ann");  
names.remove(1);
```



# Bài tập 1



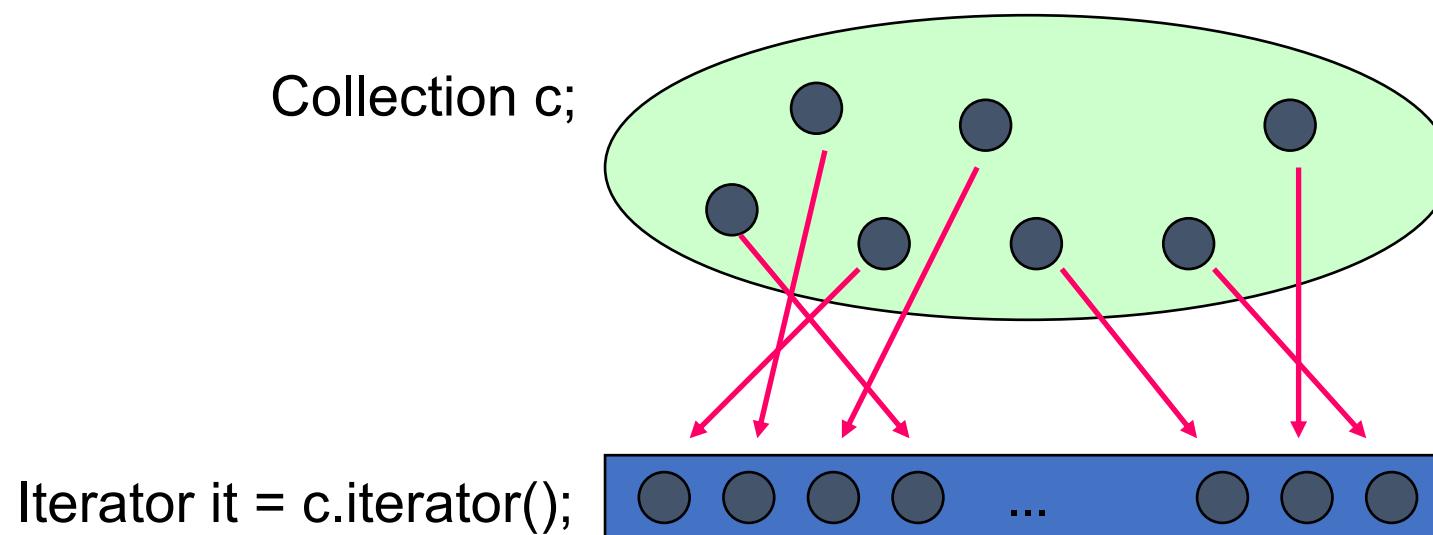
- Sau khi thực hiện đoạn chương trình sau, danh sách names có chứa các phần tử nào?

```
ArrayList<String> names = new ArrayList<String>;  
names.add("Bob");  
names.add(0, "Ann");  
names.remove(1);  
names.add("Cal");
```



# Giao diện Iterator và Comparator

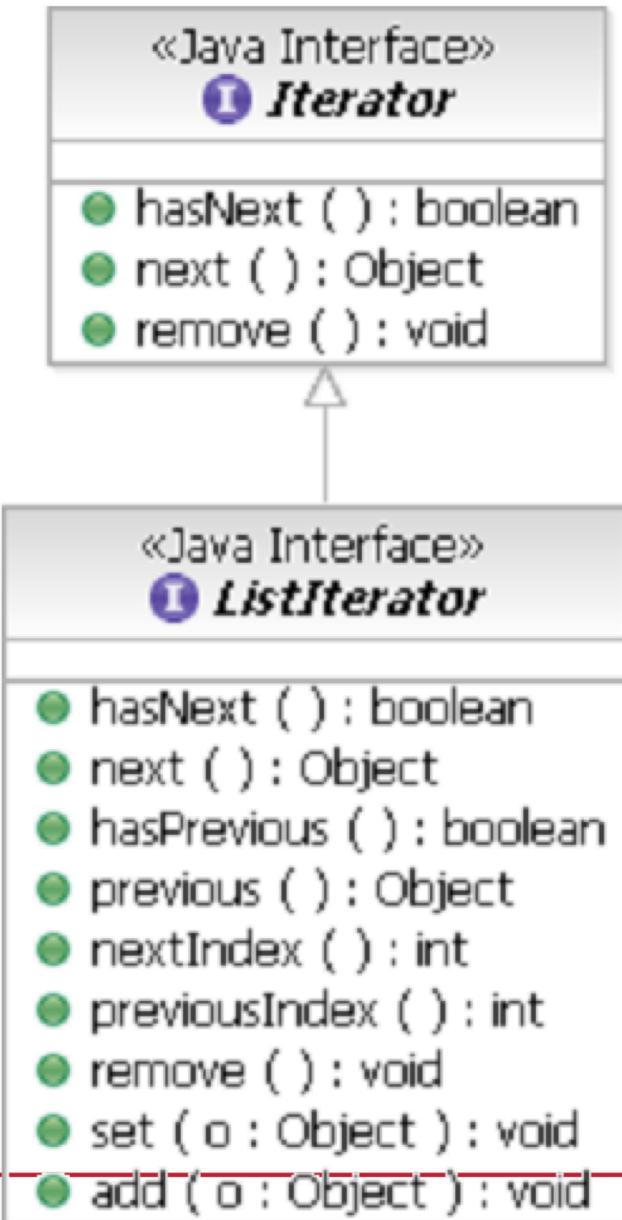
- Sử dụng để duyệt và so sánh trên các Collection
- Iterator
  - Các phần tử trong collection có thể được duyệt thông qua Iterator



# Giao diện Iterator và Comparator

- Iterator

- Cung cấp cơ chế thuận tiện để duyệt (lặp) qua toàn bộ nội dung của tập hợp, mỗi lần là một đối tượng trong tập hợp
  - Giống như SQL cursor
- Iterator của các tập hợp đã sắp xếp duyệt theo thứ tự tập hợp
- ListIterator thêm các phương thức đưa ra bản chất tuần tự của danh sách cơ sở



# Giao diện Iterator và Comparator

---

- Iterator : Các phương thức
  - iterator( ): yêu cầu container trả về một iterator
  - next( ): trả về phần tử tiếp theo
  - hasNext( ): kiểm tra có tồn tại phần tử tiếp theo hay không
  - remove( ): xóa phần tử gần nhất của iterator



# Giao diện Iterator và Comparator

- Iterator: Ví dụ

- Định nghĩa iterator

```
public interface Iterator {  
    boolean hasNext();  
    Object next();  
    void remove();  
}
```

- Sử dụng iterator

```
Collection c;
```

```
Iterator i = c.iterator();  
while (i.hasNext()) {  
    Object o = i.next();  
    // Process this object  
}
```

Tương tự vòng lặp for

```
for (String name : names) {  
    System.out.println(name);  
}
```



# Giao diện Iterator và Comparator

---

- Giao diện Comparator được sử dụng để cho phép so sánh hai đối tượng trong tập hợp
- Một Comparator phải định nghĩa một phương thức compare( ) lấy 2 tham số Object và trả về -1, 0 hoặc 1
- Không cần thiết nếu tập hợp đã có khả năng so sánh tự nhiên (vd. String, Integer...)



# Giao diện Iterator và Comparator

- Ví dụ lớp Person:

```
class Person {  
    private int age;  
    private String name;  
  
    public void setAge(int age) {  
        this.age=age;  
    }  
    public int getAge() {  
        return this.age;  
    }  
    public void setName(String name) {  
        this.name=name;  
    }  
    public String getName() {  
        return this.name;  
    }  
}
```



# Giao diện Iterator và Comparator

- Ví dụ Cài đặt AgeComparator :

```
class AgeComparator implements Comparator {  
    public int compare(Object ob1, Object ob2) {  
        int ob1Age = ((Person)ob1).getAge();  
        int ob2Age = ((Person)ob2).getAge();  
  
        if(ob1Age > ob2Age)  
            return 1;  
        else if(ob1Age < ob2Age)  
            return -1;  
        else  
            return 0;  
    }  
}
```



# Giao diện Iterator và Comparator

- Ví dụ Sử dụng AgeComparator :

```
public class ComparatorExample {  
    public static void main(String args[]) {  
        ArrayList<Person> lst = new  
            ArrayList<Person>();  
        Person p = new Person();  
        p.setAge(35); p.setName("A");  
        lst.add(p);  
  
        p = new Person();  
        p.setAge(30); p.setName("B");  
        lst.add(p);  
  
        p = new Person();  
        p.setAge(32); p.setName("C");  
        lst.add(p);  
    }  
}
```



# Giao diện Iterator và Comparator

- Ví dụ Sử dụng AgeComparator :

```
System.out.println("Order before sorting");
for (Person person : lst) {
    System.out.println(person.getName() +
        "\t" + person.getAge());
}

Collections.sort(lst, new AgeComparator());
System.out.println("\n\nOrder of person" +
    "after sorting by age");

for (Iterator<Person> i = lst.iterator();
    i.hasNext();) {
    Person person = i.next();
    System.out.println(person.getName() + "\t" +
        person.getAge());
} //End of for
} //End of main
} //End of class
```



# Nội dung

---

1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
- 4. Ký tự đại diện (Wildcard)**
5. Ví dụ và bài tập



## 4. Ký tự đại diện (Wildcard)

- Quan hệ thừa kế giữa hai lớp không có ảnh hưởng gì đến quan hệ giữa các cấu trúc tổng quát dùng cho hai lớp đó.
- Ví dụ:
  - Dog và Cat là các lớp con của Animal
  - Có thể đưa các đối tượng Dog và Cat vào một ArrayList<Animal> (sử dụng phương thức add)
  - Tuy nhiên, ArrayList<Dog>, ArrayList<Cat> lại không có quan hệ gì với ArrayList<Animal>



## 4. Ký tự đại diện (Wildcard)

---

- Không thể ép kiểu `ArrayList<Child>` về kiểu `ArrayList<Parent>`

```
class Parent { }
```

```
class Child extends Parent { }
```

```
ArrayList<Parent> myList = new ArrayList<Child>();
```



# Ví dụ

---

```
public class Test {  
    public static void main(String args[]) {  
        List<String> lst0 = new LinkedList<String>();  
        List<Object> lst1 = lst0; // Error  
        printList(lst0); // Error  
    }  
  
    void static printList(List<Object> lst) {  
        Iterator it = lst.iterator();  
        while (it.hasNext())  
            System.out.println(it.next());  
    }  
}
```



## 4. Ký tự đại diện (Wildcard)

---

- Giải pháp: sử dụng kí tự đại diện (wildcard)
- Ký tự đại diện: **?** dùng để hiển thị cho một kiểu dữ liệu bất kỳ
- Khi biên dịch, dấu ? có thể được thay thế bởi bất kì kiểu dữ liệu nào.



# Ví dụ: Sử dụng Wildcards

```
public class Test {  
    void printList(List<?> lst) {  
        Iterator it = lst.iterator();  
        while (it.hasNext())  
            System.out.println(it.next());  
    }  
  
    public static void main(String args[]) {  
        List<String> lst0 = new LinkedList<String>();  
        List<Employee> lst1 = new LinkedList<Employee>();  
  
        printList(lst0); // String  
        printList(lst1); // Employee  
    }  
}
```



## 4. Ký tự đại diện (Wildcard)

- Lưu ý: cách làm sau là không hợp lệ

```
ArrayList<?> list = new ArrayList<String>();  
list.add("a1"); //compile error  
list.add(new Object()); //compile error
```

- Nguyên nhân: Vì không biết list là danh sách liên kết cho kiểu dữ liệu nào, nên không thể thêm phần tử vào list, kể cả đối tượng của lớp Object



## 4. Ký tự đại diện (Wildcard)

---

- "? extends Type": Xác định một tập các kiểu con của Type. Đây là wildcard hữu ích
- "? super Type": Xác định một tập các kiểu cha của Type
- "?": Xác định tập tất cả các kiểu hoặc bất kỳ kiểu nào



## 4. Ký tự đại diện (Wildcard)

- Ví dụ:
  - `? extends Animal` có nghĩa là kiểu gì đó thuộc loại Animal (là Animal hoặc con của Animal)
- Lưu ý: Hai cú pháp sau là tương đương:

```
public void foo( ArrayList<? extends Animal> a)  
public <T extends Animal> void foo( ArrayList<T>  
a)
```



# Khác biệt giữa print1 và print2?

---

```
public void print1(List<Employee> list) {  
    for (Employee e : list) {  
        System.out.println(e);  
    }  
}
```

```
public void print2(List<? extends Employee> list) {  
    for (Employee e : list) {  
        System.out.println(e);  
    }  
}
```



1. Giới thiệu về lập trình tổng quát
2. Định nghĩa và sử dụng Template
3. Lập trình tổng quát trong Java collections framework
4. Ký tự đại diện (Wildcard)
5. **Ví dụ và bài tập**

## Bài tập 2

---

- Trừu tượng hoá mô tả sau: một quyển sách là tập hợp các chương, chương là tập hợp các trang.
  - Phác họa các lớp Book, Chapter, và Page
  - Tạo các thuộc tính cần thiết cho các lớp, sử dụng Collection
  - Tạo các phương thức cho lớp Chapter cho việc thêm trang và xác định một chương có bao nhiêu trang
  - Tạo các phương thức cho lớp Book cho việc thêm chương và xác định quyển sách có bao nhiêu chương, và số trang cho quyển sách



# Bài tập 3

- Xây dựng lớp Stack tổng quát với các kiểu dữ liệu

StackOfChars
- elements: char[]
- size: int
+ StackOfChars()
+ StackOfChars (capacity: int)
+ isEmpty(): boolean
+ isFull(): boolean
+ peak(): char
+ push(value:char): void
+ pop(): char
+ getSize(): int

StackOfIntegers
- elements: int[]
- size: int
+ StackOfIntegers()
+ StackOfIntegers (capacity: int)
+ isEmpty(): boolean
+ isFull(): boolean
+ peak(): int
+ push(value:int): void
+ pop(): int
+ getSize(): int

...







**HUST**

**THANK YOU !**