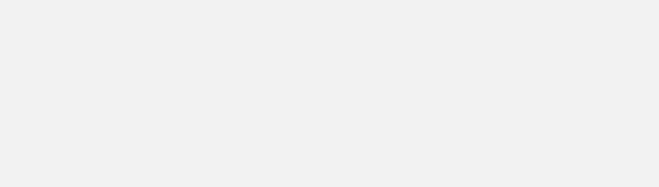


Bài 9

# Lập trình tổng quát



# Nội dung

1. Lập trình tổng quát
2. Tập hợp đối tượng
3. Các giao diện  
Collection
4. Cài đặt của các giao  
diện Collection
5. Iterator và Comparator



# 1

## Lập trình tổng quát

Generic programming



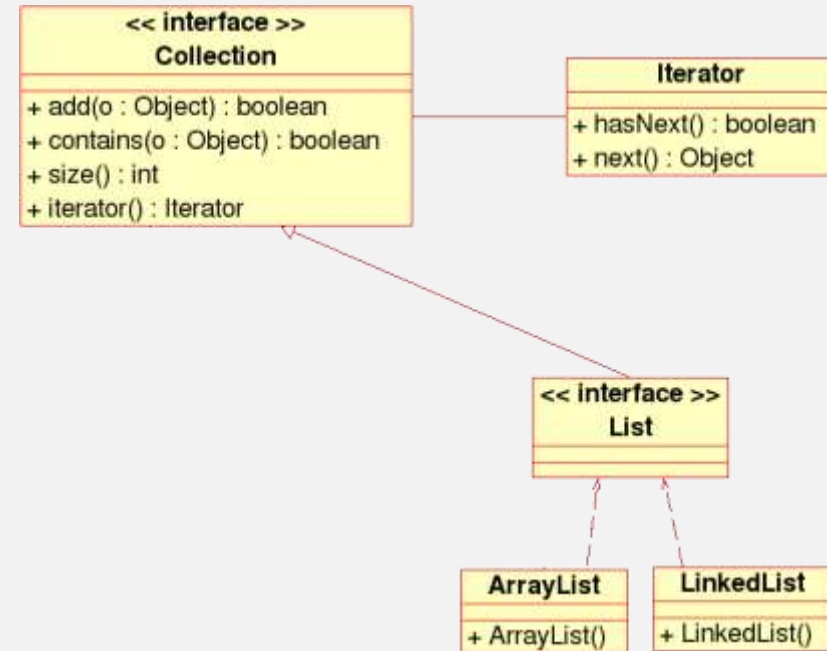
# Lập trình tổng quát

- 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:
  - C: dùng con trỏ void
  - C++: dùng template
  - Java: lợi dụng upcasting
  - Java 1.5: template

# Java 1.5 Template

- Không dùng Template

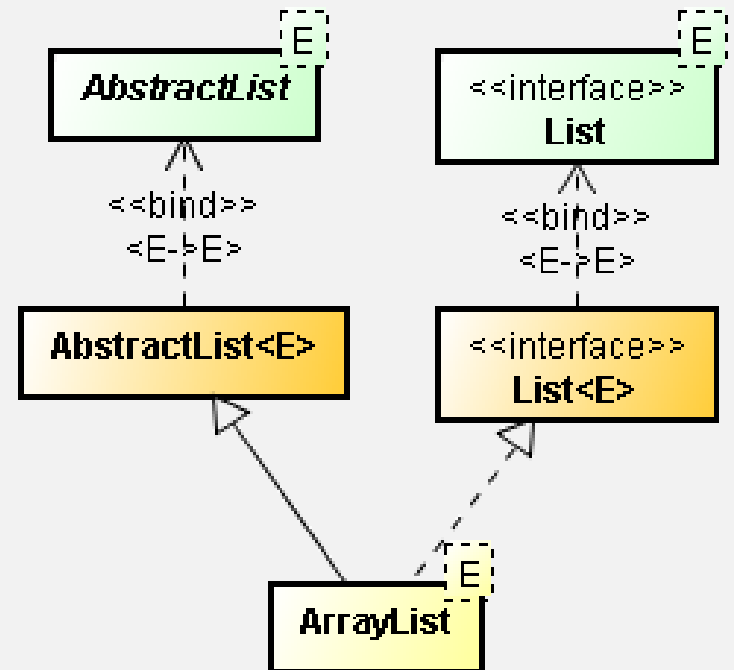
```
List myList = new LinkedList();  
myList.add(new Integer(0));  
Integer x = (Integer)  
myList.iterator().next();
```



# Java 1.5 Template

- Dùng Template:

```
List<Integer> myList = new  
    LinkedList<Integer>();  
myList.add(new Integer(0));  
Integer x =  
    myList.iterator().next();  
  
myList.add(new Long(0)); // Error
```



# Upcasting về object

- Tất cả các lớp đều dẫn xuất từ lớp **Object** → có thể *up-casting* các đối tượng lên **Object**

```
class MyStack {  
    ...  
    public void push(Object obj) {...}  
    public Object pop() {...}  
}
```

```
public class TestStack{  
    MyStack s = new MyStack();  
    Point p = new Point();  
    Circle c = new Circle();  
    s.push(p); s.push(c);  
    Circle c1 = (Circle) s.pop();  
    Point p1 = (Point) s.pop();  
}
```

# Ví dụ: equals của lớp tự viết

```
class MyValue {  
    int i;  
}  
public class EqualsMethod2 {  
    public static void main(String[] args) {  
        MyValue v1 = new MyValue();  
        MyValue v2 = new MyValue();  
        v1.i = v2.i = 100;  
        System.out.println(v1.equals(v2));  
        System.out.println(v1==v2);  
    }  
}
```





# Ví dụ: equals của lớp tự viết

```
class MyValue {  
    int i;  
    public boolean equals(Object obj) {  
        return (this.i == ((MyValue) obj).i);  
    }  
}  
  
public class EqualsMethod2 {  
    public static void main(String[] args) {  
        MyValue v1 = new MyValue();  
        MyValue v2 = new MyValue();  
        v1.i = v2.i = 100;  
        System.out.println(v1.equals(v2));  
        System.out.println(v1==v2);  
    }  
}
```

# Lớp tổng quát

- Lớp tổng quát (generic class) là lớp có thể nhận kiểu dữ liệu là một lớp bất kỳ
- Có thể lợi dụng up-casting về Object để xây dựng lớp tổng quát

```
public class Information {  
    private Object object;  
    public void set(Object object) {  
        this.object = object;  
    }  
    public Object get() {  
        return object;  
    }  
}
```

# 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

- Ví dụ

```
public class Information<T> {  
    private T value;  
    public Information(T value) {  
        this.value = value;  
    }  
    public T getValue() {  
        return value;  
    }  
}
```

```
Information<String> string = new Information<String>("hello");  
Information<Circle> circle =  
    new Information<Circle>(new Circle());  
Information<2DShape> shape =  
    new Information<>(new 2DShape());
```

# Quy ước đặt tên kiểu

<i>Tên kiểu</i>	<i>Mục đích</i>
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

# Lớp tổng quát

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

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

# Phương thức tổng quát

- Phương thức tổng quát 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 <E> static void print(E[] a) {    ... }
```

# Ví dụ

```
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

...

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

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

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

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

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

```
// 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 & ...>`

# Ví dụ

```
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
```

# 2

## Tập hợp đối tượng

Tổng quan về Collection trong Java

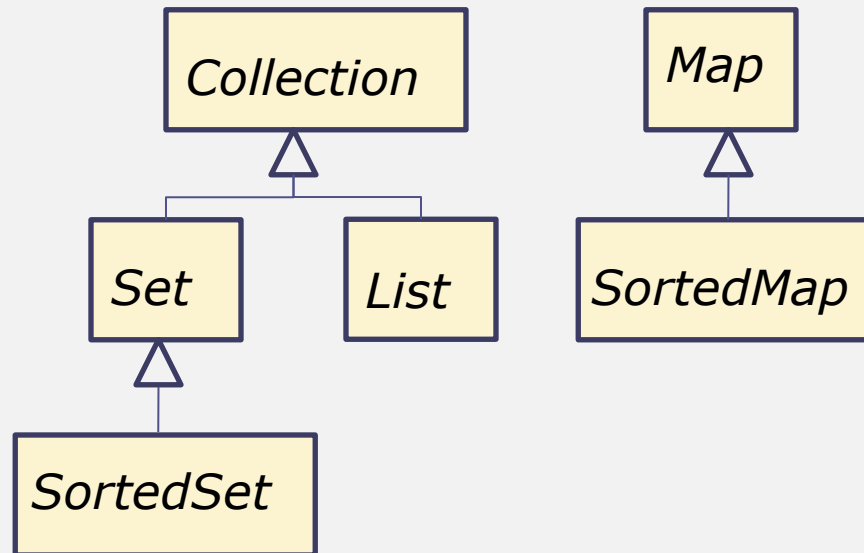


# Collection

- Collection – tập hợp: Nhóm các đối tượng lại thành một đơn vị duy nhất
- Java Collections Framework:
  - Biểu diễn các tập hợp
  - Cung cấp giao diện tiêu chuẩn cho hầu hết các tập hợp cơ bản
  - Xây dựng dựa trên
    - + Interface: thể hiện các loại tập hợp cơ bản
    - + Class: các thực thi của các giao diện
    - + Thuật toán: cài đặt một số thao tác đơn giản như tìm kiếm, sắp xếp...

# Cây cấu trúc giao diện Collection

- Các giao diện trong Collection framework thể hiện các chức năng khác nhau của tập hợp



# Cây cấu trúc giao diện Collection

- Collection: Tập các đối tượng
  - 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
  - Liên kết các đối tượng trong tập này với đối các đối tượng trong tập khác như tra từ điển/danh bạ điện thoại.

# Cây cấu trúc giao diện Collection

- Tóm lược về các giao diện trong Collection framework

<i>Tên giao diện</i>	<i>Được sắp xếp</i>	<i>Cho phép trùng</i>	<i>Cặp khóa-giá trị</i>
<b>Collection</b>	✗	✓	✗
<b>Set</b>	✗	✗	✗
<b>List</b>	✓	✓	✗
<b>Map</b>	✗	✗	✓
<b>SortedSet</b>	✓	✗	✗
<b>SortedMap</b>	✓	✗	✓



# 3








## Các giao diện Collection

Các giao diện trong Collection framework



# Giao diện Collection

- Xác định giao diện cơ bản cho các thao tác với một tập các đối tượng
  - Thêm vào tập hợp
  - Xóa khỏi tập hợp
  - 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 tập hợp (lặp) và chuyển tập hợp sang mảng

«Java Interface»  <b>Collection</b>	
	size ( ) : int
	isEmpty ( ) : boolean
	contains ( o : Object ) : boolean
	iterator ( ) : Iterator
	toArray ( ) : Object [ * ]
	toArray ( a : Object [ * ] ) : Object [ * ]
	add ( o : Object ) : boolean
	remove ( o : Object ) : boolean
	containsAll ( c : Collection ) : boolean
	addAll ( c : Collection ) : boolean
	removeAll ( c : Collection ) : boolean
	retainAll ( c : Collection ) : boolean
	clear ( ) : void
	equals ( o : Object ) : boolean
	hashCode ( ) : int

# 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 Set

- Là một **Collection** nhưng không được trùng lặp
- Ví dụ:
  - Set of cars:
    - + {BMW, Ford, Jeep, Chevrolet, Nissan, Toyota, VW}
  - Nationalities in the class
    - + {Chinese, American, Canadian, Indian}
- Methods:
  - Tương tự Collection nhưng được quy định không được trùng lặp

# 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**.
- Phương thức: tương tự **Set**, bổ sung thêm 2 phương thức
  - **first( )**: returns the first (lowest) element currently in the collection
  - **last( )**: returns the last (highest) element currently in the collection

# Giao diện List

- Kế thừa từ **Collection**
- Các phần tử được sắp xếp theo thứ tự
- Một List có thể có các phần tử trùng nhau
- Ví dụ:
  - List of first name in the class sorted by alphabetical order:
    - + Eric, Fred, Fred, Greg, John, John, John
  - List of cars sorted by origin:
    - + Ford, Chevrolet, Jeep, Nissan, Toyota, BMW, VW

# Giao diện List

- Các phương thức: Tương tự **Collection**
- Bổ sung:  
`void add(int index, Object element);`  
`boolean addAll(int index, Collection c);`  
`Object get(int index);`  
`Object remove(int index);`  
`Object set(int index, Object element);`  
`int lastIndexOf(Object o);`  
`int indexOf(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 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

# 4

## Cài đặt của các giao diện Collection

Các lớp `ArrayList`, `LinkedList`, `HashMap`...



# Các giao diện và các cài đặt

- 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

<i>Interfaces</i>	<i>Implementations</i>				
	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

# Set Implementations

- HashSet:
  - Lưu các phần tử trong một bảng băm
  - Không cho phép lưu trùng lặp
  - Cho phép phần tử null
- 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
- LinkedHashSet:
  - Cài đặt của cả HashTable và LinkedList
  - Thừa kế HashSet và thực thi giao diện Set
  - Khác HashSet ở chỗ nó lưu trữ trong một danh sách móc nối đôi
  - Thứ tự các phần tử được sắp xếp theo thứ tự được insert vào tập hợp

# List Implementations

- ArrayList, Vector: cài đặt mảng của List
  - Đối tượng của Vector mặc định được đồng bộ
  - Vector được phát triển từ Java 1.0 trước khi Collection framework được giới thiệu
  - ArrayList tốt hơn và được sử dụng nhiều hơn Vector
- LinkedList: cài đặt danh sách móc nối của List
  - Được sử dụng để tạo ngăn xếp, hàng đợi, cây...

# Map implementations

- **HashMap:**
  - Được sử dụng để thực hiện một số thao tác cơ bản như thêm, xóa và tìm kiếm phần tử trong Map
- **TreeMap:**
  - Thích hợp khi chúng ta muốn duyệt các khóa của tập hợp theo một thứ tự được sắp xếp
  - Các phần tử được thêm vào TreeMap phải có thể sắp xếp được
  - Thông thường, thêm các phần tử vào HashMap rồi convert về TreeMap để duyệt các khóa sẽ nhanh hơn
- **LinkedHashMap:**
  - Thừa kế HashMap cài đặt danh sách móc nối đôi hỗ trợ sắp xếp các phần tử
  - Các phần tử trong LinkedHashMap có thể được lấy ra theo
    - Thứ tự thêm vào, hoặc
    - Thứ tự truy cập

# 5

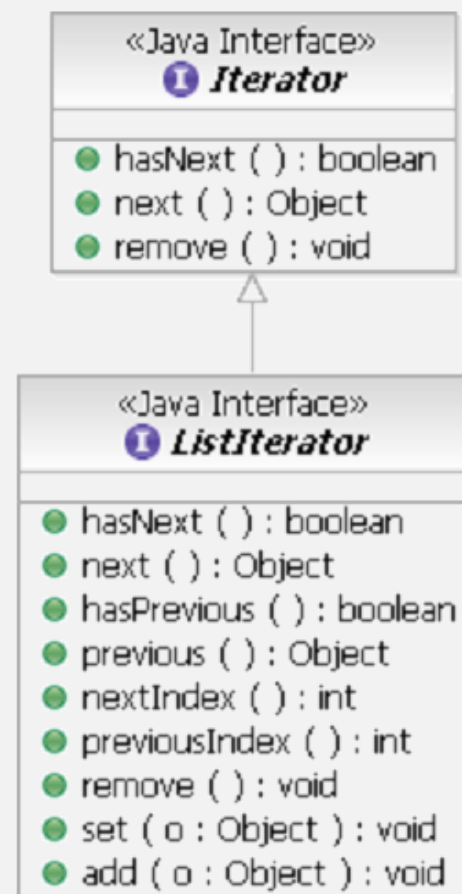
## Iterator và Comparator

Sử dụng để duyệt và so sánh trên các  
Collection



# 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
- ListIterator thêm các phương thức đưa ra bản chất tuần tự của danh sách cơ sở
- Iterator của các tập hợp đã sắp xếp duyệt theo thứ tự tập hợp





# Các phương thức

- Các phương thức của Iterator:
  - `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

# Ví dụ

- Định nghĩa iterator

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

- Sử dụng iterator

```
Collection c;  
// Some code to build the collection  
  
Iterator i = c.iterator();  
while (i.hasNext()) {  
    Object o = i.next();  
    // Process this object  
}
```

# 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...)

# 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;  
    }  
}
```

# 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;  
        }  
    }
```

# Ví dụ

```
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);  
    }  
}
```

# Ví dụ

```
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
```

# Thank you!

Any questions?

