

# BÀI 9. Lập trình tổng quát

# Nội dung

1. Giới thiệu về lập trình tổng quát
2. Giới thiệu về collection framework
3. Giới thiệu về các cấu trúc tổng quát
4. Định nghĩa và sử dụng Template

# 1. Giới thiệu về lập trình tổng quát (Generic programming)

# Giới thiệu về 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

**Tổng quát hoá chương trình**

■ 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**

# Giới thiệu về lập trình tổng quát

- ❖ **Lập trình Generic có nghĩa** là lập trình mà có thể tái sử dụng cho nhiều kiểu dữ liệu
  - Cho phép trừu tượng hóa kiểu dữ liệu
- ❖ Giải pháp trong các ngôn ngữ lập trình:
  - C: dùng con trỏ không định kiểu (con trỏ void)
  - C++: dùng template
  - Java 1.5 trở về trước: lợi dụng upcasting và kiểu tổng quát object
  - Java 1.5: đưa ra khái niệm về template

# 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

# Giới thiệu về lập trình tổng quát

❖ Ví dụ: Lập trình Generic 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!!!
```

# Giới thiệu về lập trình tổng quát

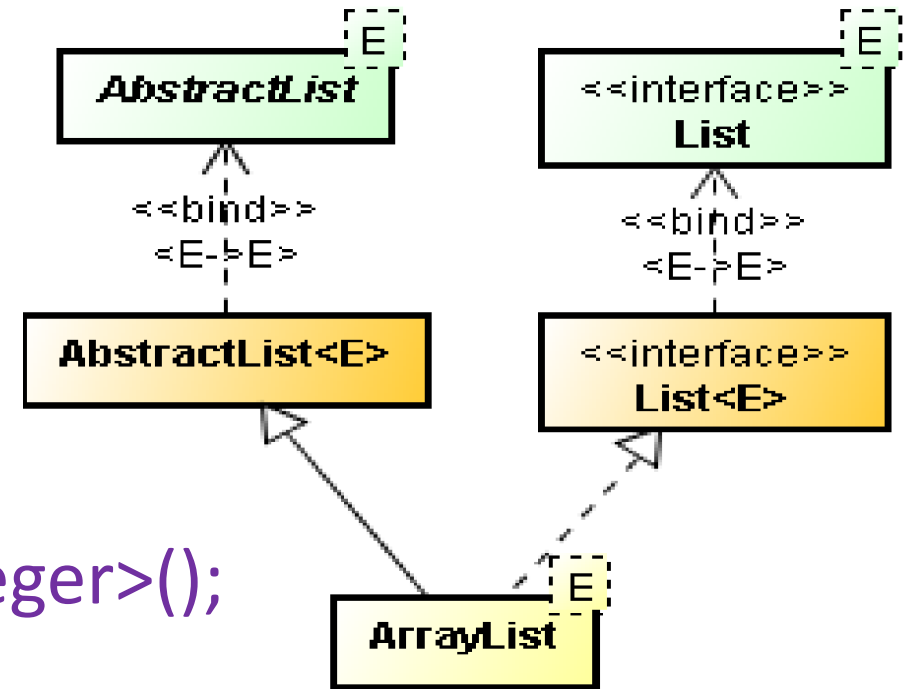
❖ Ví dụ: Lập trình Generic từ Java 1.5

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
```





# Giới thiệu về lập trình tổng quát

❖ Chúng ta xem đoạn chương trình sau:  
**KHÔNG DÙNG GENERIC**

```
public class Demo {  
    public static void main(String[] args) {  
        Integer[] intArr = {1, 2, 3, 4};  
        Double[] doubleArr = {1.1, 1.2,  
1.3};  
        String[] strArr = {"nguyen", "huy", "hung"};  
        printArray(intArr);  
        printArray(doubleArr);  
        printArray(strArr);  
    }  
    public static void printArray(Integer[] arr)  
{
```

```
    for (int i = 0; i < arr.length; i++) {  
        System.out.println(arr[i]);  
    }  
}
```

```
    public static void printArray(Double[] arr) {  
        for (int i = 0; i < arr.length; i++) {  
            System.out.println(arr[i]);  
        }  
    }
```

```
    public static void printArray(String[] arr) {  
        for (int i = 0; i < arr.length; i++) {  
            System.out.println(arr[i]);  
        }  
    }  
} } }
```

# Giới thiệu về lập trình tổng quát

❖ Chúng ta xem đoạn chương trình sau: DÙNG GENERIC

```
public class Demo {  
    public static void main(String[] args) {  
        Integer[] intArr = {1, 2, 3, 4};  
        Double[] doubleArr = {1.1, 1.2, 1.3};  
        String[] strArr = {"nguyen", "huy",  
"hung"};  
        printArray(intArr);  
        printArray(doubleArr);  
        printArray(strArr);  
    }  
}
```

```
public static void printArray(T[] arr) {  
    for (int i = 0; i < arr.length; i++) {  
        System.out.println(arr[i]);  
    }  
}
```

# 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
- ❖ 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> 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	Mục đích
T	Kiểu dữ liệu
E	Phần tử trong một Collection
K	Kiểu khóa trong Map
V	Kiểu giá trị trong Map
N	Number

❖ 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) { ... }
```

# Phương thức tổng quát - 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();  
    }  
}
```

# Phương thức tổng quát - Ví dụ (tt)

```
...  
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
```

# 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) { ... }**

Ký tự đại diện (Wildcard)

# Ký tự đại diện (Wildcard)

- ❖ Làm thế nào để xây dựng các tập hợp dành cho kiểu bất kì là lớp con của lớp cụ thể nào đó? → Giải pháp là sử dụng kí tự đại diện (wildcard)
- ❖ Ký tự đại diện: **?** dùng để hiển thị cho một kiểu dữ liệu chưa biết trong collection

```
void printCollection(Collection<?> c) {  
    for(Object e : c) {  
        System.out.println(e);  
    }  
}
```

- ❖ Khi biên dịch, dấu ? có thể được thay thế bởi bất kì kiểu dữ liệu nào.

# Ký tự đại diện (Wildcard)

❖ Tuy nhiên viết như thế này là không hợp lệ

```
Collection<?> c = new ArrayList<String>();  
c.add("a1"); //compile error, null
```

❖ Vì không biết c đại diện cho tập hợp kiểu dữ liệu nào → không thể thêm phần tử vào c

# 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  
    }  
}
```



# Ký tự đại diện (Wildcard)

Các ký tự đại diện nằm ở vị trí khác nhau có ý nghĩa khác nhau:

❖ **"? extends Type": Xác định một tập các kiểu con của Type.**

❖ Ví dụ:

- **? extends Animal** có nghĩa là kiểu gì đó thuộc loại Animal (là Animal hoặc con của Animal)
- **List<? extends Number>** mô tả một danh sách, nơi mà các phần tử là kiểu Number hoặc con của Number
- 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)**

# Ký tự đại diện (Wildcard)

❖ **"? super Type": Xác định một tập các kiểu cha của Type.**

**Ví dụ: Comparator<? Super String>**

→ mô tả một bộ so sánh Comparator mà thông số phải là String hoặc cha của String

❖ **"?": Xác định tập tất cả các kiểu hoặc bất kỳ kiểu nào.**

**Ví dụ: Collection <?>**

→ mô tả chấp nhận các loại đối số kiểu String, Integer, Boolean,...

# LỢI THẾ CỦA GENERICS TRONG JAVA

❖ **Kiểu đối tượng an toàn:** chúng ta chỉ lưu một kiểu đối tượng duy nhất trong generics. Nó không cho phép lưu trữ 2 đối tượng có kiểu dữ liệu khác nhau.

❖ **Không cần phải ép kiểu**

// trước generics → ép kiểu

Ví dụ: `List list = new ArrayList();`

`list.add("hello");`

`String s = (String) list.get(0)`

// Sau generics → không cần ép kiểu

`List <String> list = new  
ArrayList<String>();`

`list.add("hello");`

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

# LỢI THẾ CỦA GENERICS TRONG JAVA

- ❖ Giúp chúng ta nhận diện một số lỗi tại thời điểm biên dịch chương trình, mà không cần phải chờ đến khi chạy chương trình (thời điểm thực thi).

Ví dụ:

```
List <String> list = new ArrayList<String>();  
list.add("hello");  
list.add(32); // compile time error
```

# Câu hỏi

```
public void draw(List<Shape> shape) {  
    for(Shape s: shape) {  
        s.draw(this);  
    }  
}
```

→ Khác như thế nào với:

```
public void draw(List<? extends Shape> shape){  
    for(Shape s: shape) {  
        s.draw(this);  
    }  
}
```

## 2. Giới thiệu collection framework

# Collection framework

- ❖ Collection – tập hợp: 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

# Collection framework

## ❖ Các collection đầu tiên của Java:

- Mảng
- Vector: Mảng động
- Hashtable: 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.



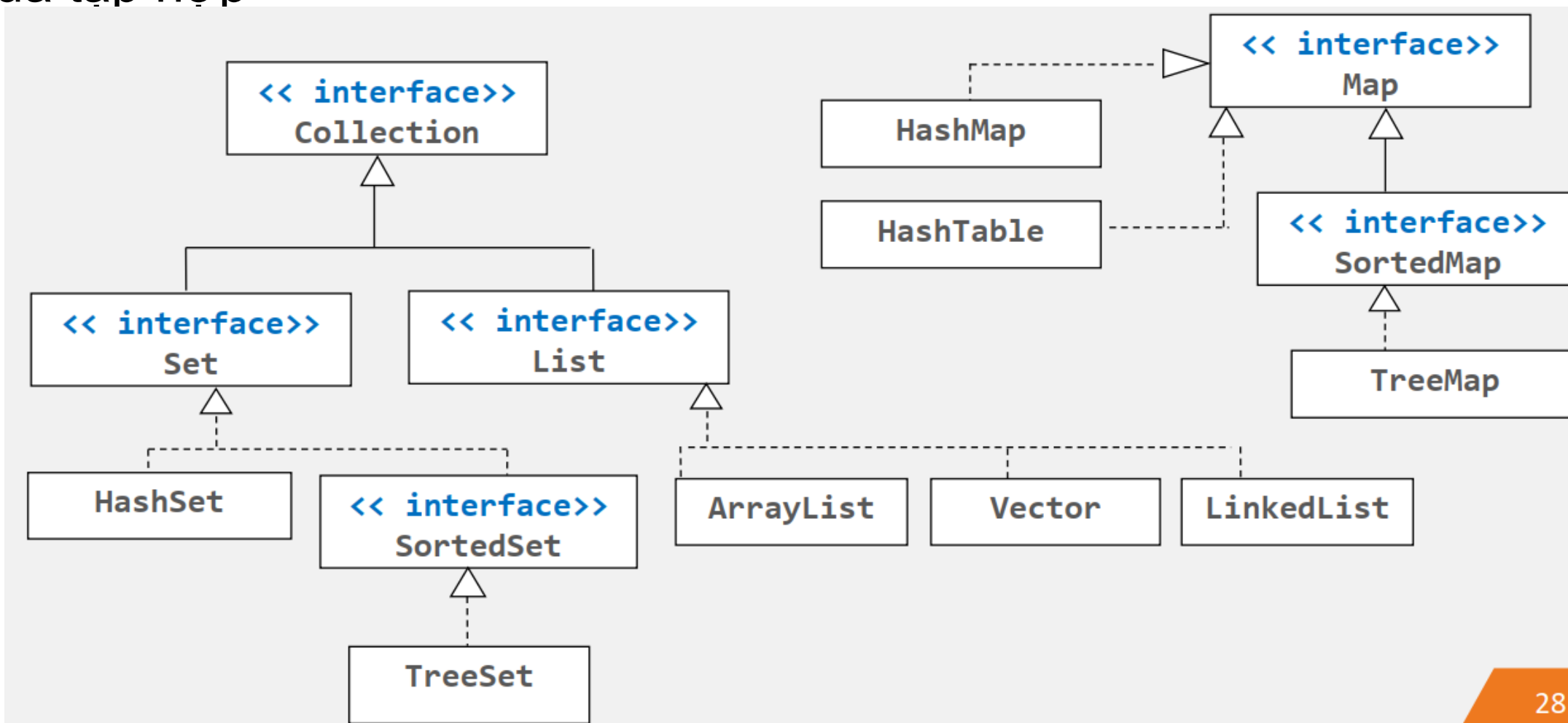
# Collection framework

❖ 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 tĩnh để xử lý trên collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

# Cây cấu trúc giao diện Collection

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



# Iterator (lặp) trong Java

- ❖ Với Collection chúng ta sử dụng một cách mới để duyệt qua các phần tử của một Collection đó là Iterator.
- ❖ Đối với Collection, Iterator là một interface cung cấp một số phương thức để duyệt (lặp) qua các phần tử của bất kỳ tập hợp nào.
- ❖ Ngoài ra, Iterator còn có khả năng xóa những phần tử từ một tập hợp trong quá trình lặp
- ❖ Ví dụ: 

```
List<String> list = new ArrayList<String>();  
list.add("an"); list.add("tam"); list.add("khoa");  
Iterator iter = list.iterator();  
While(iter.hasNext()){  
    String i= iter.next();}
```

# 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

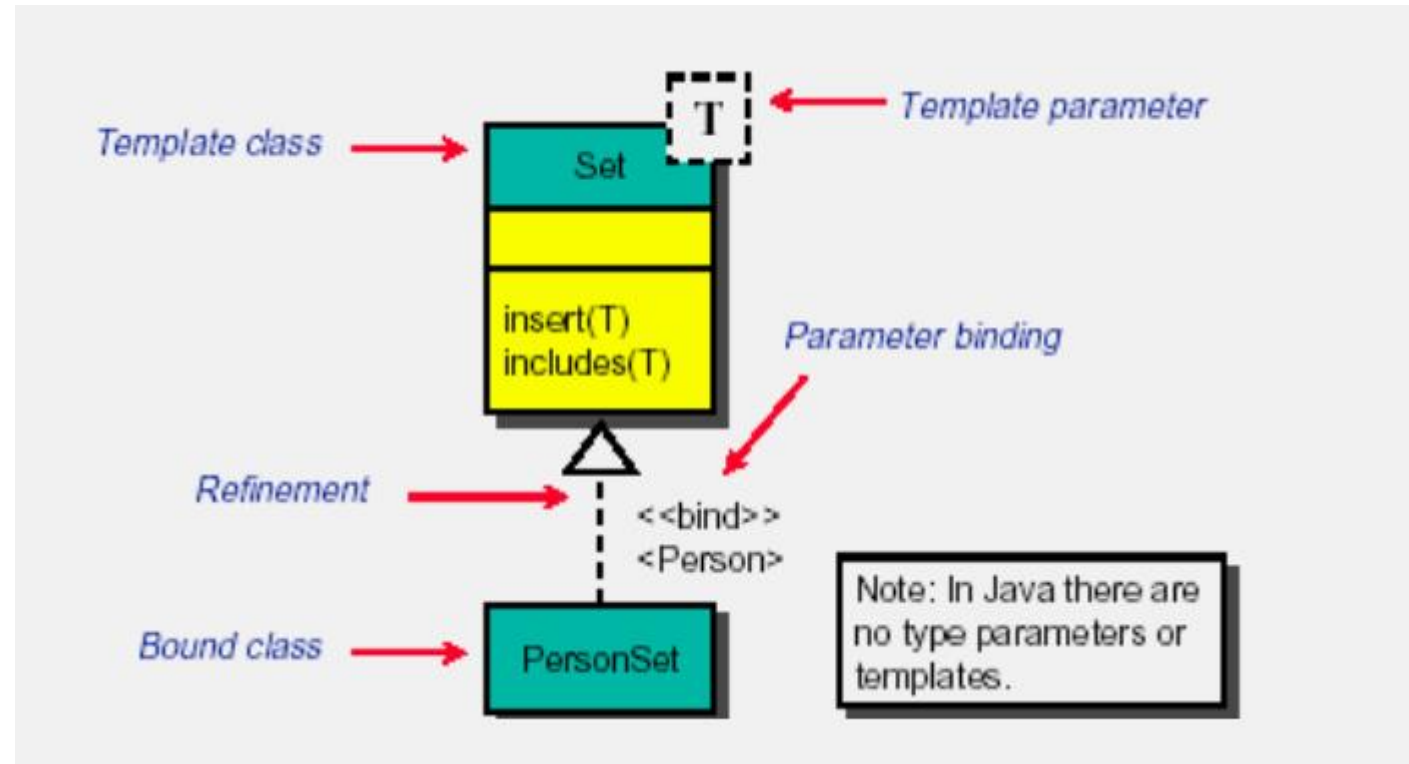
<i>Tên giao diện</i>	<i>Được sắp xếp</i>	<i>Cho phép trùng</i>	<i>Cặp khóa-giá trị</i>
<b>Collection</b>	X	✓	X
<b>Set</b>	X	X	X
<b>List</b>	✓	✓	X
<b>Map</b>	X	X	✓
<b>SortedSet</b>	✓	X	X
<b>SortedMap</b>	✓	X	✓

# 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

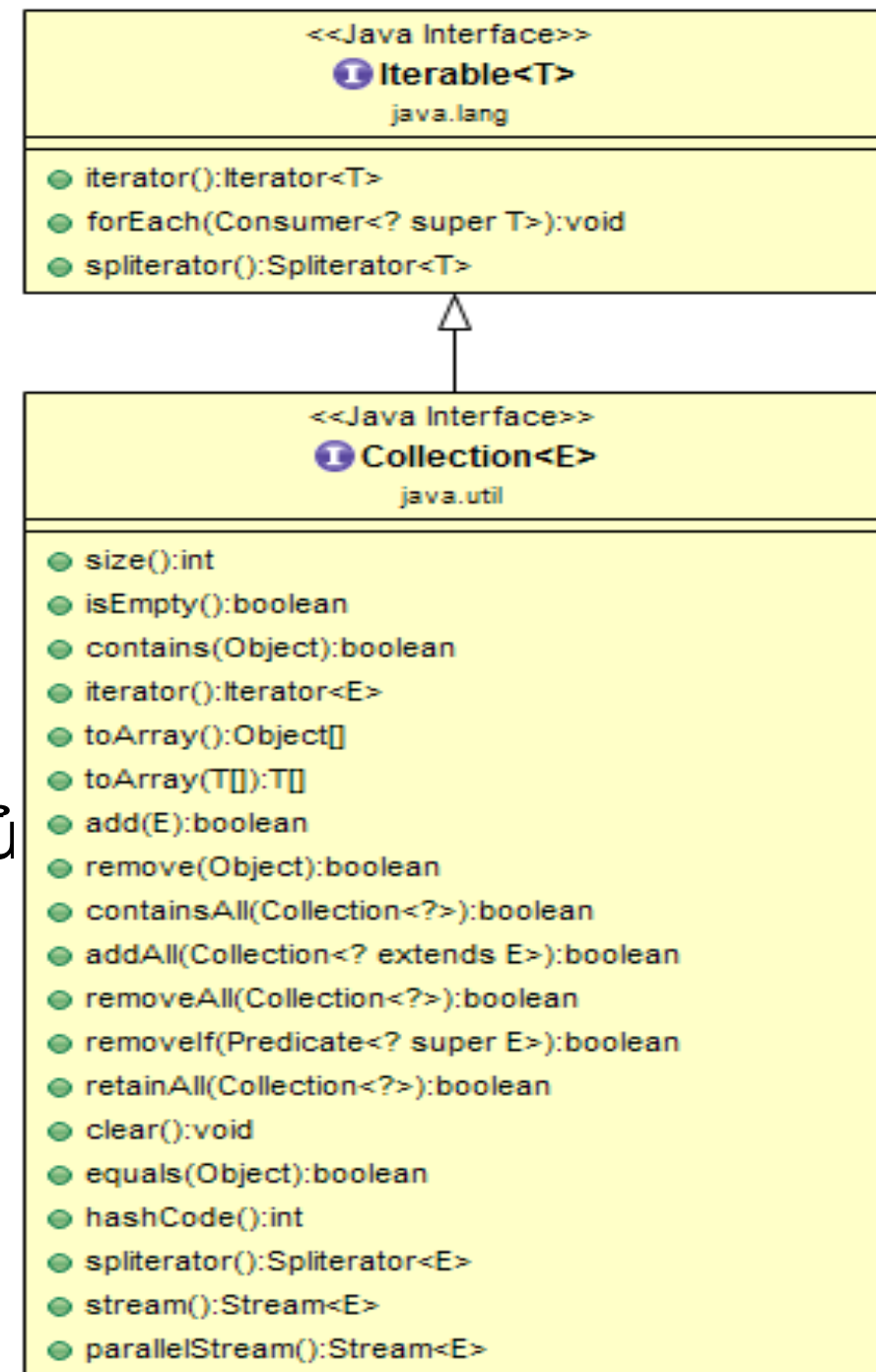
# Java Collections Framework

- ❖ 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



# Java Collections Framework

- ❖ 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



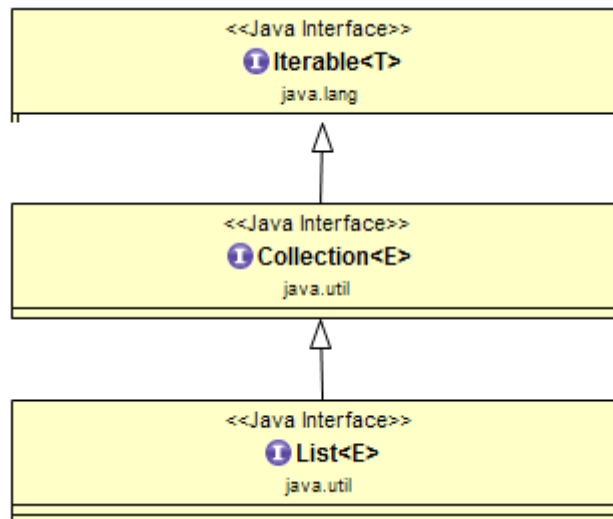


# Giao diện Collection

```
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[]);  
}
```

# Cây cấu trúc giao diện Collection - List

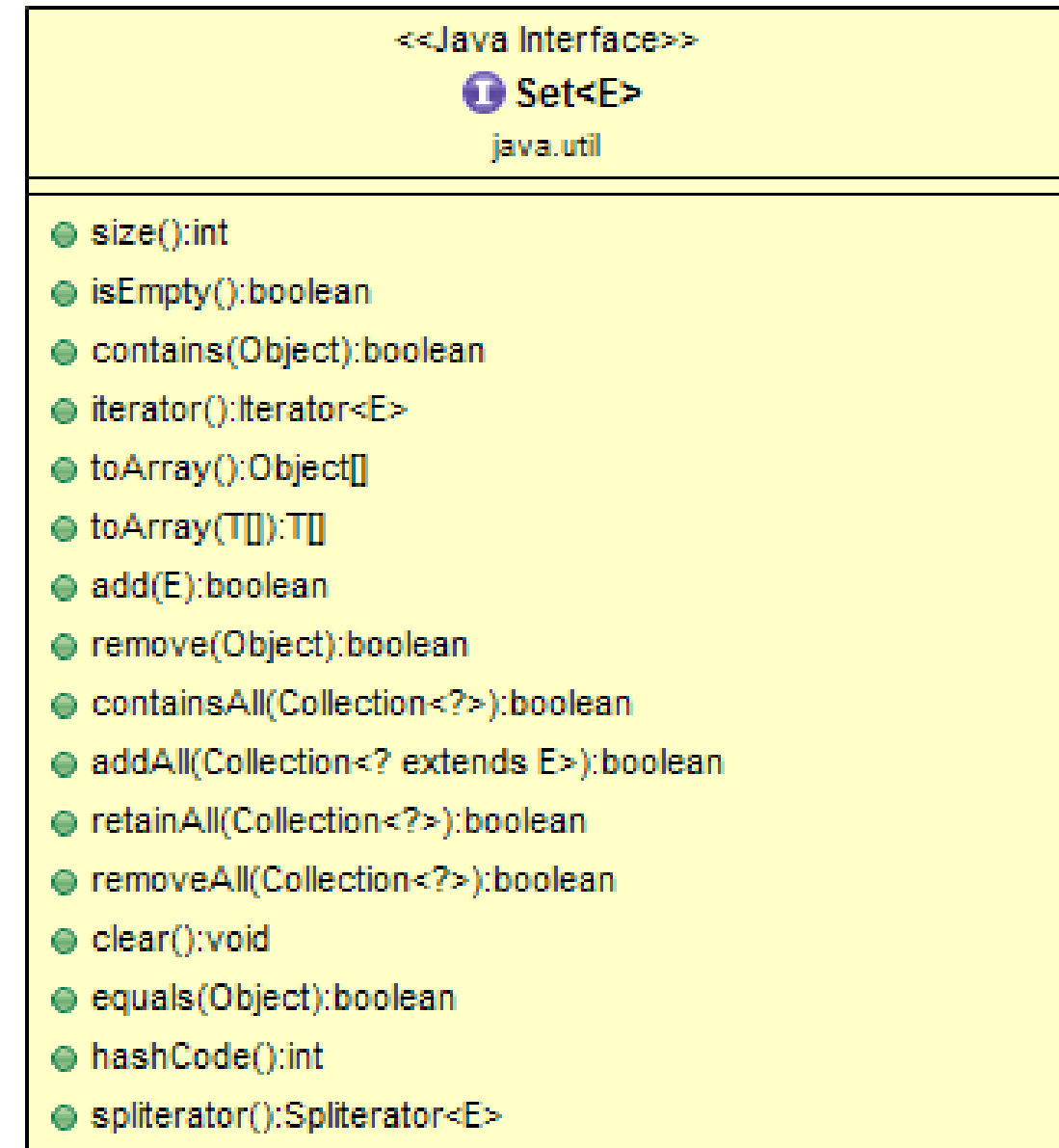
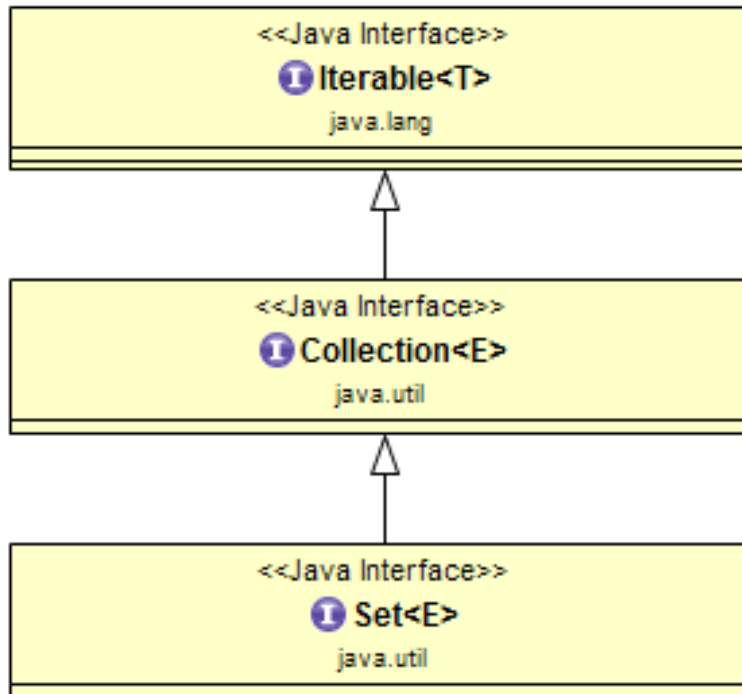
❖ Các giao diện trong Collection framework thể hiện các chức năng khác nhau của tập hợp



<div>&lt;&lt;Java Interface&gt;&gt;</div> <div><b>List&lt;E&gt;</b></div> <div>java.util</div>
<div><ul style="list-style-type: none"><li>size():int</li><li>isEmpty():boolean</li><li>contains(Object):boolean</li><li>iterator():Iterator&lt;E&gt;</li><li>toArray():Object[]</li><li>toArray(T[]):T[]</li><li>add(E):boolean</li><li>remove(Object):boolean</li><li>containsAll(Collection&lt;?&gt;):boolean</li><li>addAll(Collection&lt;? extends E&gt;):boolean</li><li>addAll(int,Collection&lt;? extends E&gt;):boolean</li><li>removeAll(Collection&lt;?&gt;):boolean</li><li>retainAll(Collection&lt;?&gt;):boolean</li><li>replaceAll(UnaryOperator&lt;E&gt;):void</li><li>sort(Comparator&lt;? super E&gt;):void</li><li>clear():void</li><li>equals(Object):boolean</li><li>hashCode():int</li><li>get(int):E</li><li>set(int,E):E</li><li>add(int,E):void</li><li>remove(int):E</li><li>indexOf(Object):int</li><li>lastIndexOf(Object):int</li><li>listIterator():ListIterator&lt;E&gt;</li><li>listIterator(int):ListIterator&lt;E&gt;</li><li>subList(int,int):List&lt;E&gt;</li><li>splitIterator():SplitIterator&lt;E&gt;</li></ul></div>

# Cây cấu trúc giao diện Collection - Set

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



# Cây cấu trúc giao diện Collection - Map

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

<<Java Interface>>

**1** Map<K,V>

java.util

- size():int
- isEmpty():boolean
- containsKey(Object):boolean
- containsValue(Object):boolean
- get(Object):V
- put(K,V):V
- remove(Object):V
- putAll(Map<? extends K,? extends V>):void
- clear():void
- keySet():Set<K>
- values():Collection<V>
- entrySet():Set<Entry<K,V>>
- equals(Object):boolean
- hashCode():int
- getOrDefault(Object,V):V
- forEach(BiConsumer<? super K,? super V>):void
- replaceAll(BiFunction<? super K,? super V,? extends V>):void
- putIfAbsent(K,V):V
- remove(Object,Object):boolean
- replace(K,V,V):boolean
- replace(K,V):V
- computeIfAbsent(K,Function<? super K,? extends V>):V
- computeIfPresent(K,BiFunction<? super K,? super V,? extends V>):V
- compute(K,BiFunction<? super K,? super V,? extends V>):V
- merge(K,V,BiFunction<? super V,? super V,? extends V>):V

### 3. Giới thiệu một số cấu trúc tổng quát

# Interface collections

Tên Interface	Đặc điểm khái quát
List Interface	Các phần tử trong List Interface được sắp xếp có thứ tự và có thể có giá trị giống nhau
Set	Các phần tử trong Set là duy nhất
SortedSet	Là một dạng riêng của Set Interface, trong đó giá trị các phần tử mặc định được sắp xếp tăng dần
Map	Giá trị của mỗi phần tử trong Map bao gồm 2 phần đó là khóa (key) và giá trị tương ứng của key đó (value). Khóa của các phần tử này là duy nhất
SortedMap	Là một dạng riêng của Map Interface, trong đó giá trị key được sắp xếp tăng dần

# List Interface

- ❖ Kế thừa từ **Collection**
- ❖ Các phần tử được sắp xếp theo thứ tự và giá trị các phần tử có thể trùng nhau
- ❖ Phần tử trong List có thể truy cập, thêm, sửa hoặc xóa thông qua vị trí của nó trong danh sách, và phần tử đầu tiên trong List có chỉ số là 0.
- ❖ Tạo mới một List Interface:  

```
List<String> list1= new ArrayList<String>();  
List<String> list2= new LinkedList<String>();
```

# List Interface – Hiện thị các phần tử trong List

❖ Sử dụng vòng lặp for thông thường

```
for(int i=0; i<list.size();i++){  
    System.out.println(list.get(i));  
}
```

❖ Sử dụng vòng lặp for cải tiến duyệt theo đối tượng danh sách

```
for(String element: list){  
    System.out.println(element);  
}
```



# List Interface – Hiện thị các phần tử trong List

❖ Sử dụng Iterator:

```
Iterator<String> iter=list.iterator();  
while(iter.hasNext()){  
    System.out.println(iter.next());  
}
```

# Các thao tác trên List

❖ Thêm phần tử:

```
list.add("Minh");
```

❖ Xóa phần tử

```
list.remove(...);
```

❖ .....

# Set Interface

- ❖ Là một Interface Collection nhưng không được trùng lặp (**duy nhất**)
- ❖ Sử dụng Set khi ta muốn lưu trữ các phần tử không trùng, hoặc khi ta không quan tâm đến thứ tự các phần tử trong danh sách.
- ❖ Tạo mới một Set Interface

```
Set<Integer> s1= new HashSet<>();  
Set<Integer> s2= new TreeSet<>();  
Set<Integer> s3= new TreeSet<>(listInteger);
```

# Set Interface

❖ HashSet khả năng lưu trữ các phần tử mặc định là 16 phần tử. Vì vậy, nếu cần lưu trữ nhiều hơn thì khai báo số phần tử cần dùng.

❖ Ví dụ:

```
Set<Float> setFloat = new HashSet<>(1000);  
setFloat.add(7.0f);
```

# Map Interface

- ❖ Map là một tập các cặp khóa – giá trị (key – value). Giá trị của các phần tử có thể giống nhau, nhưng khóa thì không giống nhau.
- ❖ Áp dụng truy xuất, cập nhật hoặc tìm kiếm phần tử thông qua khóa của phần tử đó.
- ❖ Ví dụ:
  - Mỗi Người quản lý (key) → danh sách nhân viên (value)
  - Quản lý lớp học (key) → danh sách sinh viên thuộc lớp học (value)

# Map Interface

## ❖ Tạo mới Map Interface

```
Map <Integer, String> map = new HashMap<>();
```

```
Map <Integer, String> map2 = new LinkedHashMap<>();
```

```
Map <Integer, String> map3 = new TreeMap<>();
```

## ❖ Thêm phần tử vào Map: `map3.put(1,"one");`

## ❖ Cách lấy giá trị của Map

- Sử dụng for cải tiến
- Lấy toàn bộ các entry của Map

```
Set<Map.Entry<Integer,String>> set = map3.entrySet();
```

```
map3.forEach((keynumber, valueString) -> System.out.println(  
    "key = "+ keynumber+", value = "+ valueString));
```

# Map Interface

❖ Lấy toàn bộ Key của Map

```
for(Integer key: map3.keySet()){  
    System.out.println("Key = "+key);  
}
```

❖ Lấy toàn bộ Value của Map

```
for(Integer value: map3.values()){  
    System.out.println("Value = "+ value);  
}
```

# Map Interface

- ❖ Lấy toàn bộ Key của Map dùng **Iterator**

```
Iterator<Integer> iter= map3.keySet().iterator();  
while(iter.hasNext()){  
    System.out.println(iter.next());  
}
```

- ❖ Lấy dữ liệu value khi biết được Key

```
Integer value = map3.get(1);  
System.out.println(value); → "one"
```

- ❖ Kiểm tra một value có trong Map hay không

```
boolean y = map3.containsValue("one");  
System.out.println(y); → true
```



# Map Interface

## ❖ Thay thế value của 1 entry trong Map

- Cú pháp 1:

- `replace(K key, V value);`

- `map3.replace(1, "Mot");`

- `Replace (K key, V oldValue, V newValue);`

- `map3.replace(1, "one", "Mot");`

## ❖ Xóa 1 entry trong Map

- `Remove(K key);`

# SortedMap Interface

- ❖ SortedMap Interface là một dạng riêng của Map Interface nên nó chứa những đặc điểm của Map
  - Gồm 1 cặp khóa – giá trị (key-value)
  - Giá trị các phần tử trong SortedMap có thể giống nhau, key thì không
  - Các entry trong SortedMap được sắp xếp tăng dần theo khóa

# SortedMap Interface

- ❖ Tạo mới và hiện thị các phần tử của 1 SortedMap

```
SortedMap<Integer, String> sortedMap= new TreeMap<>();
```

- ❖ Trích xuất một phần tử trong SortedMap

```
Map<Integer, String> headMap=sortedMap.subMap(4,"Thu tu");
```

- ❖ Lấy các phần tử có key của nó nhỏ hơn toKey cho trước

```
Map<Integer, String> headMap=sortedMap.headMap(5);
```

- ❖ Lấy các phần tử có key lớn hơn hoặc bằng fromKey cho trước

```
Map<Integer, String> tailMap=sortedMap.tailMap(5);
```

# SortedMap Interface

❖ Tìm giá trị khóa nhỏ nhất trong SortedMap

Integer key = **sortedMap.firstKey();**

❖ Tìm giá trị khóa lớn nhất trong SortedMap

Integer key = **sortedMap.lastKey();**

# 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

# SortedSet (TreeSet) - Ví dụ

## ❖ Ví dụ: SortedSet

```
import java.util.SortedSet;
import java.util.TreeSet;

public class Exam {

    public static void main(String[]
args) {

        // Create a SortedSet object.

        SortedSet<String> map = new
TreeSet<String>();

            map.add("B");
            map.add("A");
```

```
map.add("F");
```

```
map.add("D");
```

```
map.add("E");
```

```
System.out.println(map);
}
```

➔ Output???

# Set (HashSet) - Ví dụ

## ❖ Ví dụ: HashSet

```
import java.util.Set;
import java.util.HashSet;

class Main {
    public static void main(String[] args) {
        // Creating a set using the HashSet class
        Set<Integer> set1 = new HashSet<>();
        // Add elements to the set1
        set1.add(2);
        set1.add(3);
        System.out.println("Set1: " + set1);
```

```
// Creating another set using the
HashSet class
Set<Integer> set2 = new HashSet<>();
// Add elements
set2.add(1);
set2.add(2);
System.out.println("Set2: " + set2);
// Union of two sets
set2.addAll(set1);
System.out.println("Union is: " + set2); }
} ➔ Output???
```

# Set (TreeSet) - Ví dụ

## ❖ Ví dụ: TreeSet

```
import java.util.Set;
import java.util.TreeSet;
import java.util.Iterator;
class Main {
public static void main(String[] args) {
// Creating a set using the TreeSet class
Set<Integer> numbers = new TreeSet<>();
// Add elements to the set
numbers.add(2);
numbers.add(3);
numbers.add(1);
System.out.println("Set using TreeSet: " +
numbers);
```

```
// Access Elements using iterator()
System.out.print("Accessing elements using
iterator(): ");
Iterator<Integer> iterate =
numbers.iterator();
while(iterate.hasNext()) {
    System.out.print(iterate.next());
    System.out.print(", ");
}
}
}
```

➔ **Output???**



# List (ArrayList) – ví dụ

```
import java.util.ArrayList;
import java.util.Collection;
public class CollectionDemo {
    public static void main(String[] args) {
        Collection < String > fruitCollection =
new ArrayList < > ();
        fruitCollection.add("banana");
        fruitCollection.add("apple");
        fruitCollection.add("mango");
        System.out.println(fruitCollection);
        fruitCollection.remove("banana");
        System.out.println(fruitCollection);
```

```
        System.out.println(fruitCollection.contains("
apple"));
        fruitCollection.forEach((element) -> {
            System.out.println(element);
        });
        fruitCollection.clear();

        System.out.println(fruitCollection);
    }
}
➔ Output
```

# List (ArrayList) – Ví dụ (tt)

```
import java.util.ArrayList;
import java.util.List;
public class Exam {
    public static void main(String[] args) {
        List < String > list = new ArrayList < > ();
        // List allows you to add duplicate elements
        list.add("element1");
        list.add("element1");
        list.add("element2");
        list.add("element2");
        System.out.println(list);

        // List allows you to have 'null' elements.
        list.add(null);
        list.add(null);
        System.out.println(list);
    }
}
```

```
list.add("element1"); // 0
list.add("element2"); // 1
list.add("element4"); // 2
list.add("element3"); // 3
list.add("element5"); // 4

System.out.println(list);

// access elements from list

System.out.println(list.get(0));
System.out.println(list.get(4));
    }
}
```

➔ Output

# List (LinkedList) - Ví dụ

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class Exam {
    public static void main(String[] args) {
        List<String> progLangs = new LinkedList<>();

        progLangs.add("C");
        progLangs.add("C++");
        progLangs.add("Core Java");
        progLangs.add("Java EE");
```

```
        progLangs.add("Spring Framework");
        progLangs.add("Hibernate Framework");

        System.out.println("\n=== Iterate over a
LinkedList using iterator() ===");

        Iterator<String> iterator =
progLangs.iterator();

        while (iterator.hasNext()) {
            String speciesName =
iterator.next();
            System.out.println(speciesName);
        }
    }
} → Output
```

# Map (HashMap) - Ví dụ

```
import java.util.HashMap;
import java.util.Map;
public class Exam {
    public static void main(String[] args) {
        // Creating a HashMap
        Map<String, Integer> numberMapping = new HashMap<>();
        // Adding key-value pairs to a HashMap
        numberMapping.put("One", 1);
        numberMapping.put("Two", 2);
        numberMapping.put("Three", 3);
        // Add a new key-value pair only if the key does not exist in the
        // HashMap, or is mapped to `null`
        numberMapping.putIfAbsent("Four", 4);
        System.out.println(numberMapping); ➔Output???
    }
}
```

# Quiz 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");
```

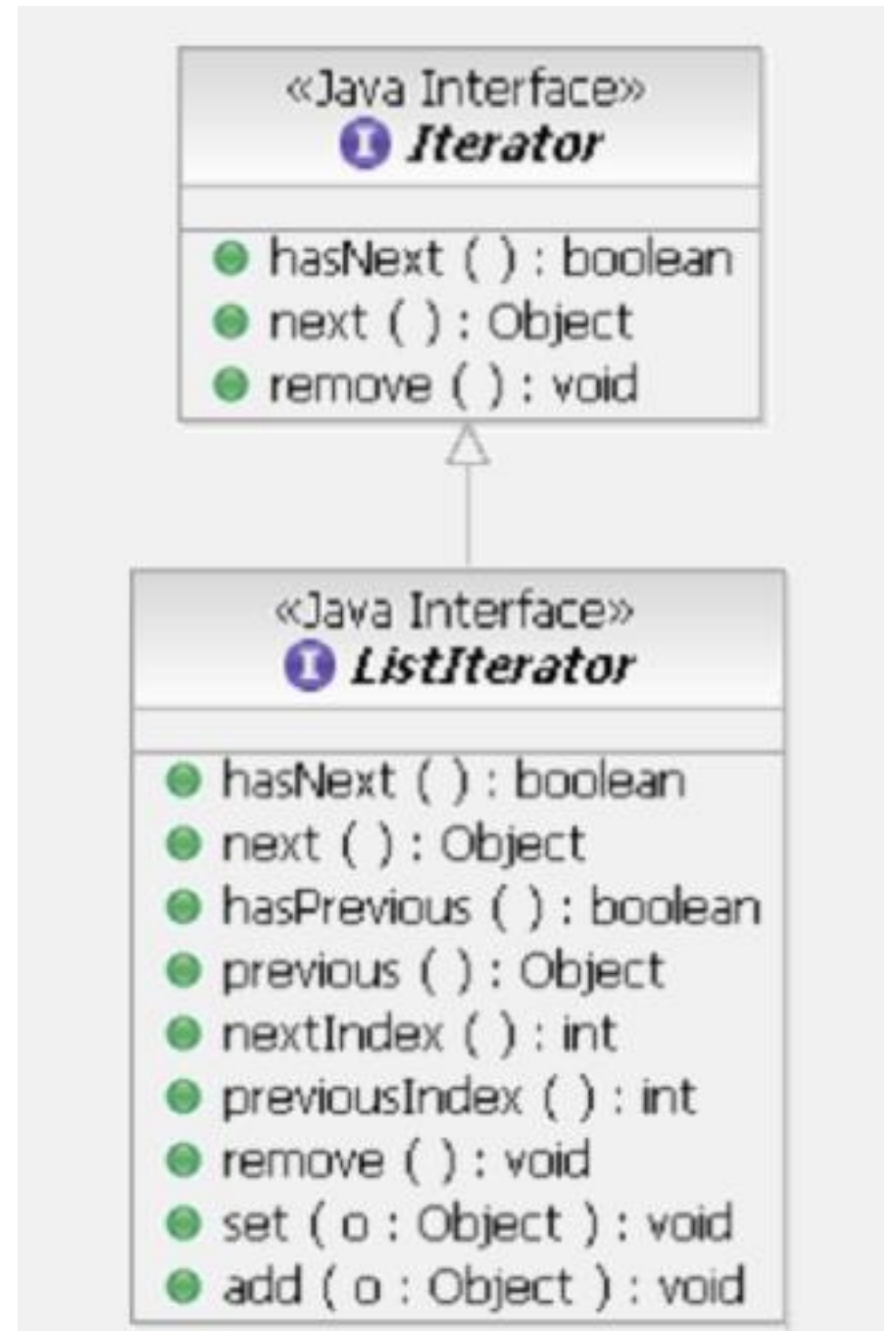
# Class Collections

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

		IMPLEMENTATIONS				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Legacy
I N T E R F A C E S	Set	HashSet		TreeSet		
	List		ArrayList		LinkedList	Vector, Stack
	Map	HashMap		TreeMap		HashTable, Properties

# 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
- ❖ 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



# 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);  
}
```

# 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...)

# 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;  
    }  
}
```

# 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;  
    }  
}
```

# 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);  
    }  
}
```

# 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
```

## 4. Định nghĩa và sử dụng Template

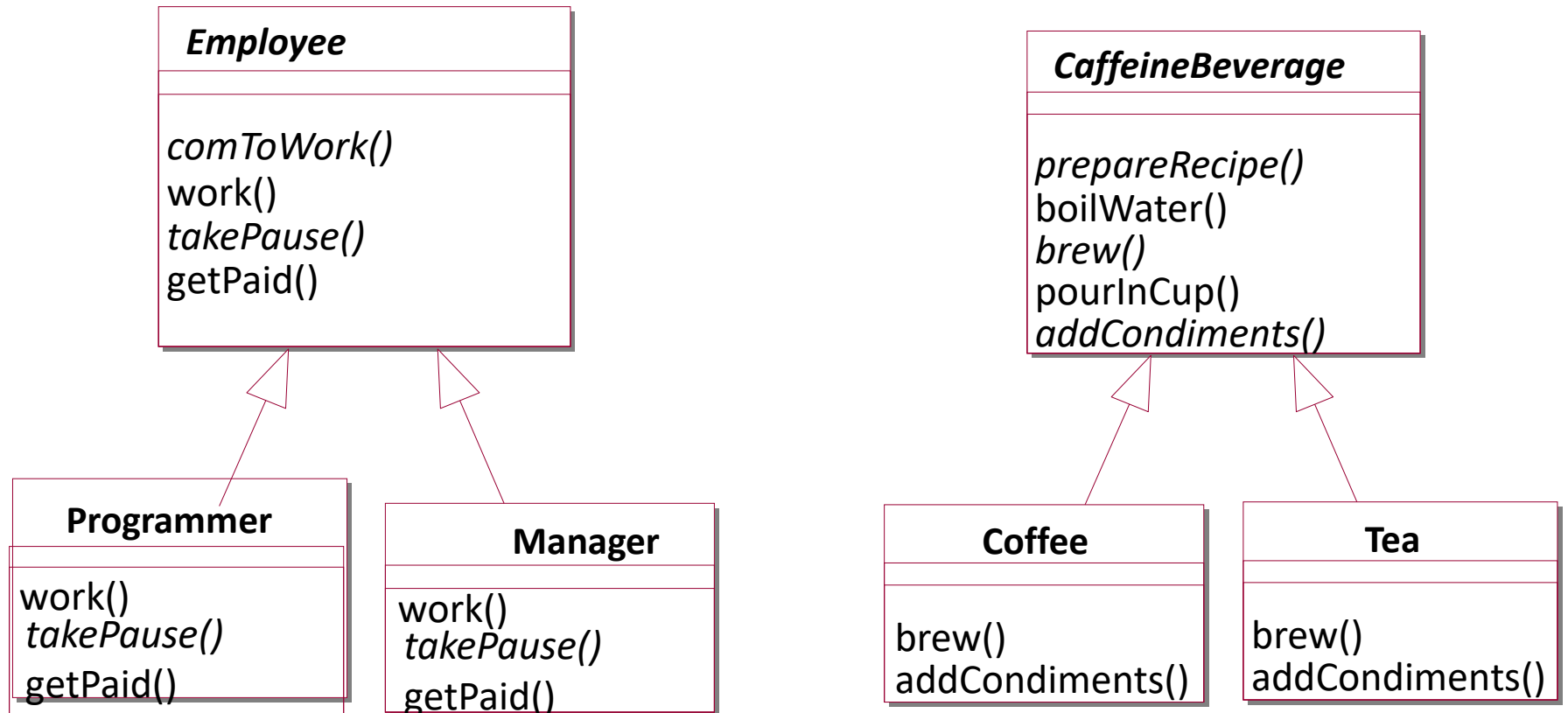
# Template method là gì?

- ❖ Template method còn được gọi là Template Pattern được sử dụng để xác định một class abstract (trừu tượng), cung cấp các cách để chạy phương thức của nó.
- ❖ Các lớp con kế thừa các phương thức này cũng phải tuân theo các định nghĩa bên trong nó



# Chương trình đơn giản với Template method

❖ Trong chương trình mô tả cách thực hiện công việc, trách nhiệm của các nhân viên trong công ty.



# Employee.java

```
public abstract class Employee {  
    public final void comeToWork() {  
        work();  
        takePause();  
        work();  
        getPaid();  
    }  
    abstract void work();  
    abstract void takePause();  
    abstract void getPaid();  
}
```

# Manager.java

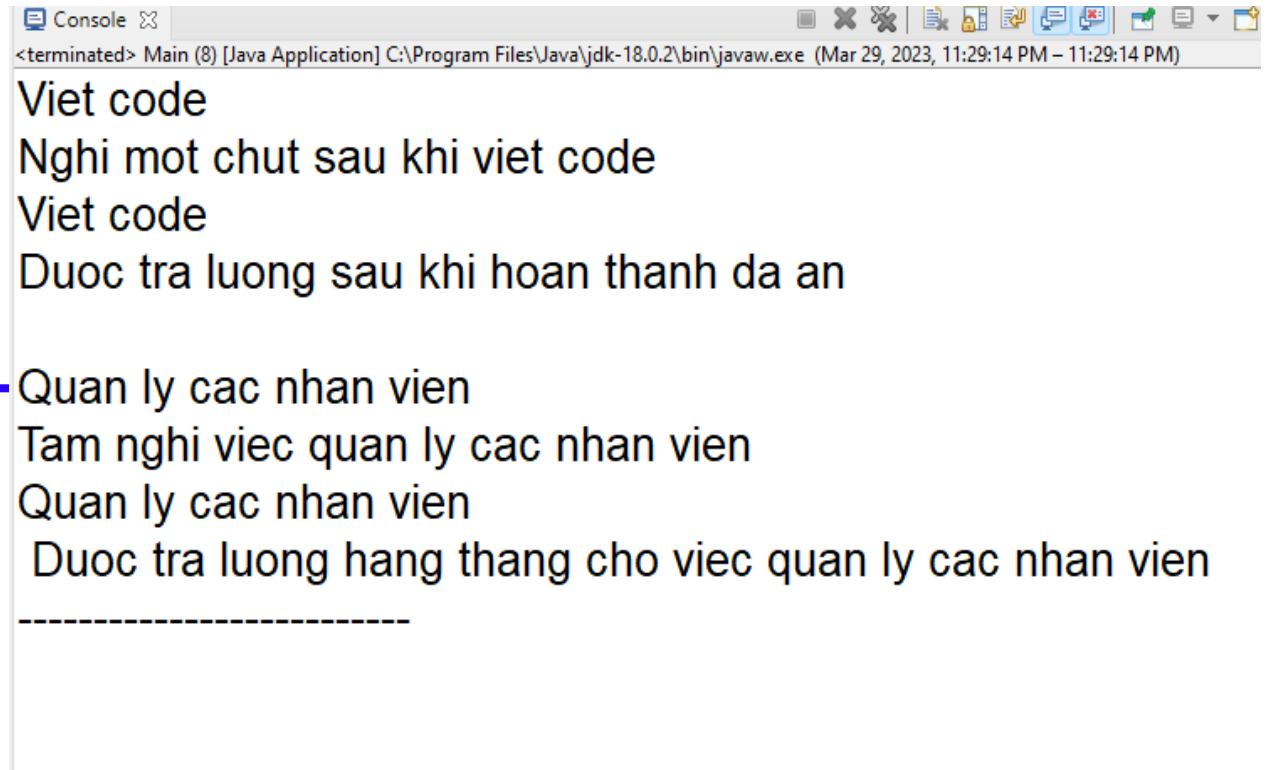
```
public class Manager extends Employee{
    @Override
    void work() {
        System.out.println("Quan ly cac nhan vien");
    }
    @Override
    void takePause() {
        System.out.println("Tam nghi viec quan ly cac nhan vien");
    }
    @Override
    void getPaid() {
        System.out.println(" Duoc tra luong hang thang cho viec quan ly cac nhan vien");
    }
}
```

# Programmer.java

```
public class Programmer extends Employee{
    @Override
    void work() {
        System.out.println("Viet code");
    }
    @Override
    void takePause() {
        System.out.println("Nghỉ một chút sau khi viết code");
    }
    @Override
    void getPaid() {
        System.out.println("Được trả lương sau khi hoàn thành đã ăn");
    }
}
```

# Main.java

```
public class Main {  
    public static void main(String[] args) {  
        Employee employee= new Programmer();  
        employee.comeToWork();  
        System.out.println();  
        employee = new Manager();  
        employee.comeToWork();  
        System.out.println("-----"  
    }  
}
```



The screenshot shows a Java console window titled "Console" with the following output:

```
<terminated> Main (8) [Java Application] C:\Program Files\Java\jdk-18.0.2\bin\javaw.exe (Mar 29, 2023, 11:29:14 PM - 11:29:14 PM)  
Viet code  
Nghỉ một chút sau khi viết code  
Viet code  
Được trả lương sau khi hoàn thành đã ăn  
  
Quản lý các nhân viên  
Tạm nghỉ việc quản lý các nhân viên  
Quản lý các nhân viên  
Được trả lương hàng tháng cho việc quản lý các nhân viên  
-----
```

Tổng kết

# Tổng kết

- ❖ 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 với thuật toán đã xác định
- ❖ Trong Java sử dụng Template
- ❖ 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 (giao diện, lớp thực thi, thuật toán)
- ❖ Lớp tổng quát (generic class) là lớp có thể nhận kiểu dữ liệu là một lớp bất kỳ

# Bài tập



# Bài 1

- ❖ Trừu tượng hoá mô tả sau: một quyển sách là tập hợp các chương, chương là tập hợp các trang.
  - Phác hoạ các lớp Book, Chapter, và Page
  - Tạo các thuộc tính cần thiết cho các lớp, hãy tận dụng tập hợp như là thuộc tính của lớp
  - 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