



# **Môn: Lập trình Hướng đối tượng (Object Oriented Programming)**

**Chương 5. Tập hợp trên Java**

# Nội dung

---

- 5.1. Khái niệm về Tập hợp
- 5.2. So sánh Tập hợp và mảng
- 5.3. Các Lớp Tập hợp trong Java
- 5.4. Ứng dụng của Tập hợp trong lập trình

## 5.1. Khái niệm về Tập hợp

- Tập hợp dùng lưu trữ, thao tác trên một nhóm các đối tượng.
- Collection/Tập hợp là đối tượng có khả năng chứa các đối tượng khác.
- Các đối tượng của tập hợp có thể thuộc nhiều loại dữ liệu khác nhau
- Các thao tác thông thường trên tập hợp
  - Thêm/Xoá đối tượng vào/ra tập hợp
  - Kiểm tra một đối tượng có ở trong tập hợp hay không
  - Lấy một đối tượng từ tập hợp
  - Duyệt các đối tượng trong tập hợp
  - Xoá toàn bộ tập hợp

## 5.1. Khái niệm về Tập hợp (tt)

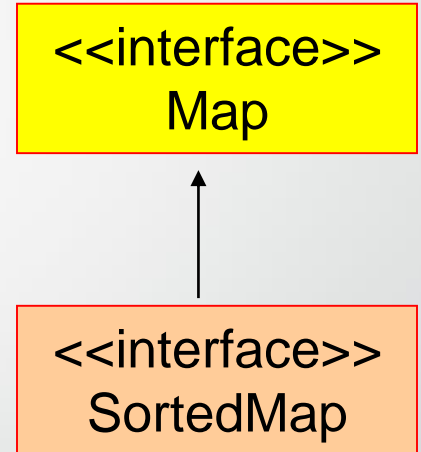
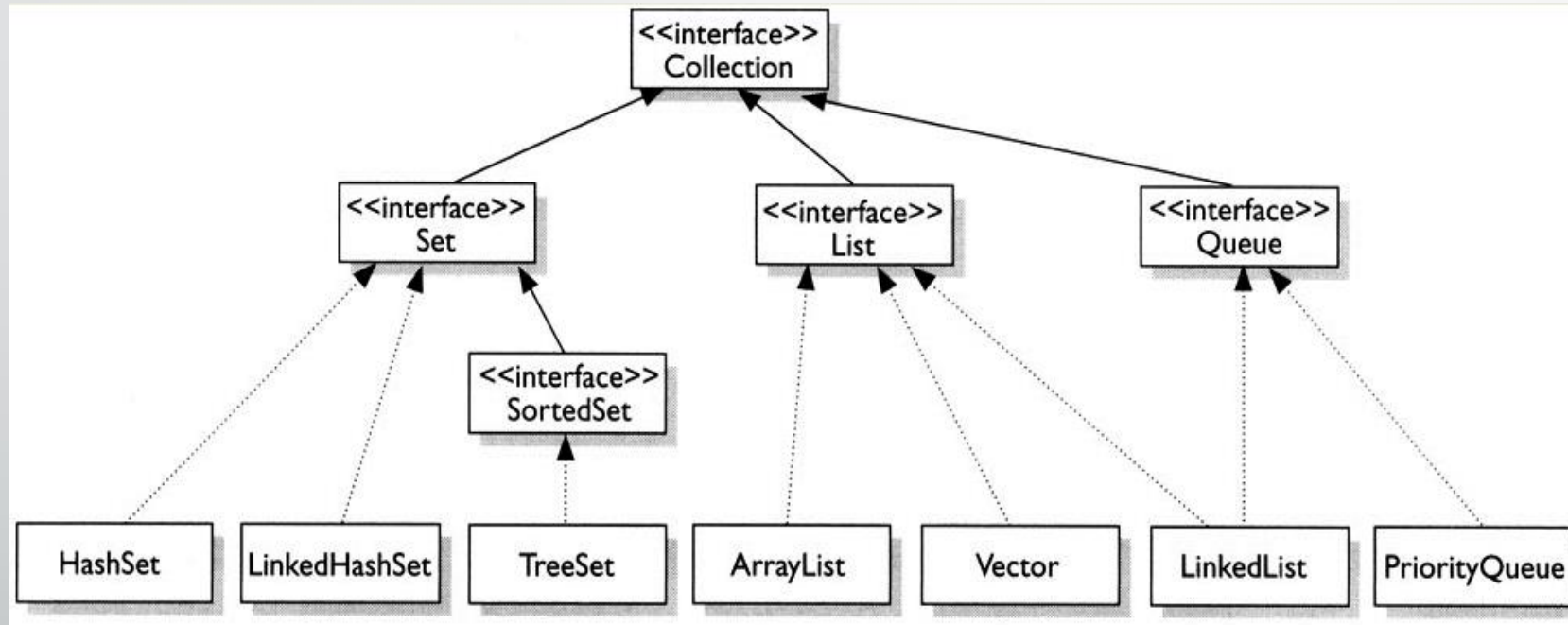
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các collection.
  - Giúp cho việc xử lý các collection độc lập với biểu diễn chi tiết bên trong của chúng.
- Một số lợi ích của Collections Framework
  - Giảm thời gian lập trình
  - Tăng cường hiệu năng chương trình
  - Dễ mở rộng các collection mới
  - Sử dụng lại mã chương trình

## 5.1. Khái niệm về Tập hợp (tt)

- Collections Framework bao gồm
  - Interfaces: Là các interface 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...

## 5.1. Khái niệm về Tập hợp (tt)

- Các interfaces của interface Collection, Map



## 5.1. Khái niệm về Tập hợp (tt)

- Các interfaces của interface Collection
  - List
    - Lưu trữ các phần tử theo thứ tự được thêm vào
    - Truy xuất các phần tử theo chỉ mục(index)
    - Các phần tử trong List có thể trùng nhau.
  - Set
    - Các phần tử trong Set lưu trữ không theo thứ tự đã thêm vào .
    - Không chấp nhận các phần tử trùng.
  - SortedSet
    - Thừa kế từ Set
    - Lưu trữ các phần tử theo thứ tự tăng.
    - Không chấp nhận các phần tử trùng.
  - Queue

## 5.1. Khái niệm về Tập hợp (tt)

- Một số phương thức của interface Collection

### Method Summary

boolean	<u><a href="#">add</a></u> ( <u><a href="#">E</a></u> o) Ensures that this collection contains the specified element (optional operation).
boolean	<u><a href="#">addAll</a></u> ( <u><a href="#">Collection</a></u> <? extends <u><a href="#">E</a></u> > c) Adds all of the elements in the specified collection to this collection (optional operation).
void	<u><a href="#">clear</a></u> () Removes all of the elements from this collection (optional operation).
boolean	<u><a href="#">contains</a></u> ( <u><a href="#">Object</a></u> o) Returns true if this collection contains the specified element.
boolean	<u><a href="#">containsAll</a></u> ( <u><a href="#">Collection</a></u> <?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	<u><a href="#">isEmpty</a></u> () Returns true if this collection contains no elements.
<u><a href="#">Iterator</a></u> < <u><a href="#">E</a></u> >	<u><a href="#">iterator</a></u> () Returns an iterator over the elements in this collection.
boolean	<u><a href="#">remove</a></u> ( <u><a href="#">Object</a></u> o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	<u><a href="#">removeAll</a></u> ( <u><a href="#">Collection</a></u> <?> c) Removes all this collection's elements that are also contained in the specified collection (optional operation).
int	<u><a href="#">size</a></u> () Returns the number of elements in this collection.



# 5.1. Khái niệm về Tập hợp (tt)

## Method Summary

boolean	<u>add</u> ( <u>E</u> o) Appends the specified element to the end of this list (optional operation).
void	<u>add</u> (int index, <u>E</u> element) Inserts the specified element at the specified position in this list (optional operation).
boolean	<u>addAll</u> ( <u>Collection</u> <? extends <u>E</u> > c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned
boolean	<u>addAll</u> (int index, <u>Collection</u> <? extends <u>E</u> > c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	<u>clear</u> () Removes all of the elements from this list (optional operation).
<u>E</u>	<u>get</u> (int index) Returns the element at the specified position in this list.
<u>E</u>	<u>set</u> (int index, <u>E</u> element) Replaces the element at the specified position in this list with the specified element (optional operation).
<u>E</u>	<u>remove</u> (int index) Removes the element at the specified position in this list (optional operation).
boolean	<u>remove</u> ( <u>Object</u> o) Removes the first occurrence in this list of the specified element (optional operation).
boolean	<u>removeAll</u> ( <u>Collection</u> <?> c) Removes from this list all the elements that are contained in the specified collection (optional operation).
<u>List</u> < <u>E</u> >	<u>subList</u> (int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.

Một số phương thức của interface List:

## 5.1. Khái niệm về Tập hợp (tt)

- Interface Set
  - Set kế thừa từ Collection, hỗ trợ các thao tác xử lý trên tập hợp (Một tập hợp yêu cầu các phần tử phải không được trùng lặp).
  - Set không có thêm phương thức riêng ngoài các phương thức kế thừa từ Collection.
- Interface SortedSet
  - SortedSet kế thừa từ Set, hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được. Các đối tượng đưa vào trong một SortedSet phải implements interface Comparable hoặc lớp cài đặt SortedSet phải nhận một Comparator trên kiểu của đối tượng đó.

## 5.1. Khái niệm về Tập hợp (tt)

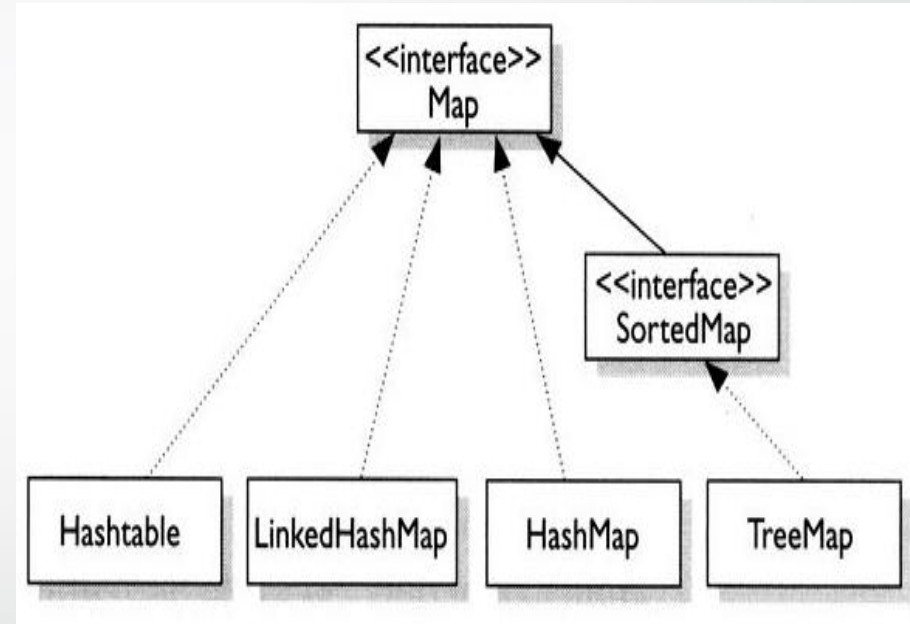
- Interface SortedSet (tt)
  - Một số phương thức của SortedSet:

### Method Summary

<a href="#">Comparator</a> <? super <a href="#">E</a> >	<a href="#">comparator</a> () Returns the comparator associated with this sorted set, or null if it uses its elements' natural ordering.
<a href="#">E</a>	<a href="#">first</a> () Returns the first (lowest) element currently in this sorted set.
<a href="#">SortedSet</a> < <a href="#">E</a> >	<a href="#">headSet</a> ( <a href="#">E</a> toElement) Returns a view of the portion of this sorted set whose elements are strictly less than toElement.
<a href="#">E</a>	<a href="#">last</a> () Returns the last (highest) element currently in this sorted set.
<a href="#">SortedSet</a> < <a href="#">E</a> >	<a href="#">subSet</a> ( <a href="#">E</a> fromElement, <a href="#">E</a> toElement) Returns a view of the portion of this sorted set whose elements range from fromElement, inclusive, to toElement, exclusive.
<a href="#">SortedSet</a> < <a href="#">E</a> >	<a href="#">tailSet</a> ( <a href="#">E</a> fromElement) Returns a view of the portion of this sorted set whose elements are greater than or equal to fromElement.

## 5.1. Khái niệm về Tập hợp (tt)

- Interface Map
  - Interface 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).
  - MAP lưu trữ dữ liệu theo từng cặp: khóa – giá trị (key-value)
  - Các giá trị được lấy từ MAP thông qua khóa của nó.
  - Các khóa trong MAP phải duy nhất.



## 5.1. Khái niệm về Tập hợp (tt)

- Interface Map
  - Các phương thức của interface Map

### Method Summary

boolean	<code>containsKey(Object key)</code> Returns true if this map contains a mapping for the specified key.
boolean	<code>containsValue(Object value)</code> Returns true if this map maps one or more keys to the specified value.
<code>V</code>	<code>get(Object key)</code> Returns the value to which this map maps the specified key.
<code>V</code>	<code>put(K key, V value)</code> Associates the specified value with the specified key in this map (optional operation).
void	<code>putAll(Map&lt;? extends K, ? extends V&gt; t)</code> Copies all of the mappings from the specified map to this map (optional operation).
<code>V</code>	<code>remove(Object key)</code> Removes the mapping for this key from this map if it is present (optional operation).
int	<code>size()</code> Returns the number of key-value mappings in this map.
<code>Collection&lt;V&gt;</code>	<code>values()</code> Returns a collection view of the values contained in this map.

## 5.1. Khái niệm về Tập hợp (tt)

- Interface Map (tt)
  - Map cung cấp 3 cách view dữ liệu:
    - View các khoá:  
`Set keySet();` // Trả về các khoá
    - View các giá trị:  
`Collection values();` // Trả về các giá trị
    - View các cặp khoá-giá trị  
`Set entrySet();` // Trả về các cặp khoá-giá trị

## 5.1. Khái niệm về Tập hợp (tt)

- Interface SortedMap
  - Giao tiếp SortedMap kế thừa từ Map, cung cấp thao tác trên các bảng ánh xạ với khoá có thể so sánh được.
  - Giống như SortedSet, các đối tượng khoá đưa vào trong SortedMap phải implements interface Comparable hoặc lớp cài đặt SortedMap phải nhận một Comparator trên đối tượng khoá.

## 5.1. Khái niệm về Tập hợp (tt)

- Interface Queue
  - Queue: Các phần tử được truy xuất theo thứ tự First In First Out (FIFO).
  - Priority queue (hàng đợi ưu tiên): Thứ tự truy xuất các phần tử phụ thuộc vào giá trị của chúng.
  - Các phương thức của Queue

### Method Summary

<u>E</u>	<u>element</u> ()	Retrieves, but does not remove, the head of this queue.
boolean	<u>offer</u> ( <u>E</u> o)	Inserts the specified element into this queue, if possible.
<u>E</u>	<u>peek</u> ()	Retrieves, but does not remove, the head of this queue, returning <code>null</code> if this queue is empty.
<u>E</u>	<u>poll</u> ()	Retrieves and removes the head of this queue, or <code>null</code> if this queue is empty.
<u>E</u>	<u>remove</u> ()	Retrieves and removes the head of this queue.



## 5.2. So sánh Tập hợp và mảng

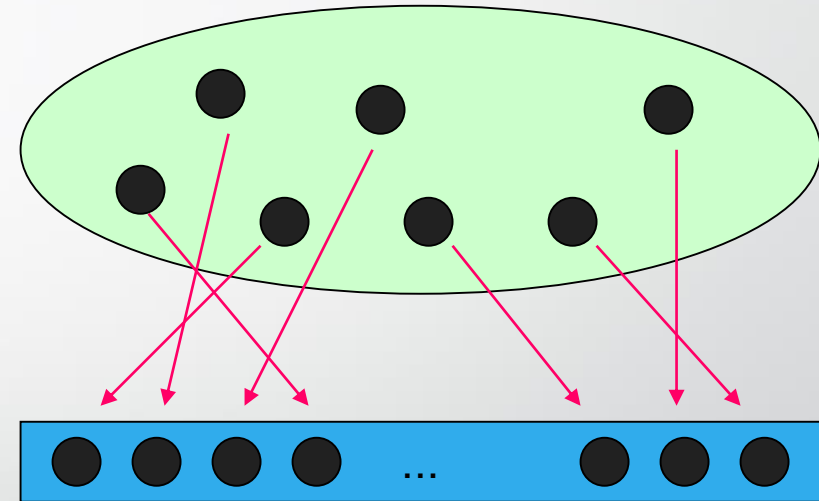
- Mảng truy xuất 1 cách tuần tự còn tập hợp (có thể) truy xuất theo dạng ngẫu nhiên.
- Mảng chứa 1 loại đối tượng/dữ liệu nhất định. Tập hợp có thể chứa nhiều loại đối tượng/dữ liệu khác nhau.
- Dùng tổ chức dữ liệu theo mảng phải lập trình hoàn toàn, dùng theo kiểu tập hợp xây dựng sẵn của Java chỉ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

## 5.2. So sánh Tập hợp và mảng (tt)

### Duyệt tập hợp

- Iterator cho phép duyệt các phần tử của một collection.
- Các phương thức của Iterator:
  - boolean hasNext();
  - Object next();
  - void remove();

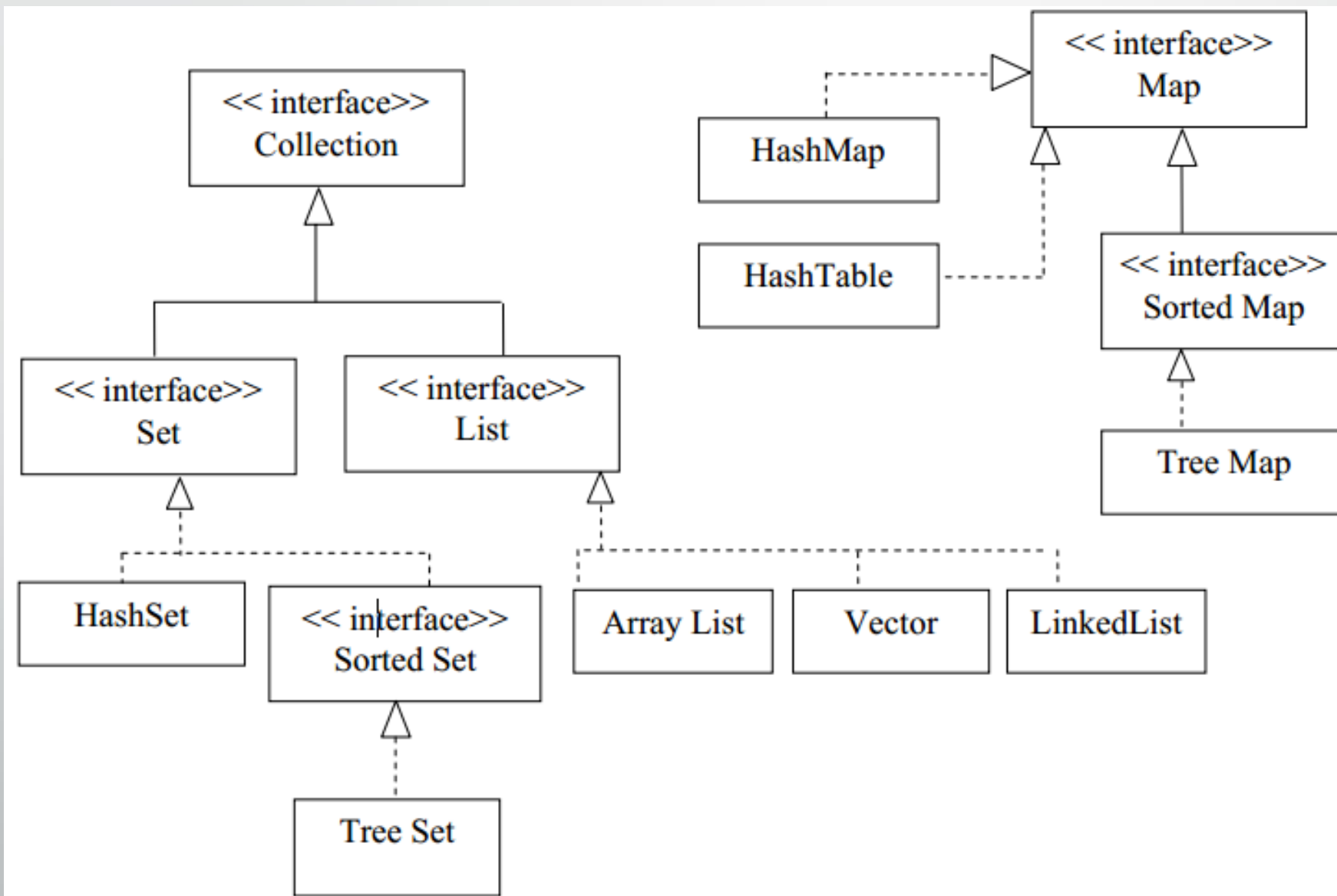
Collection c;



Iterator it = c.iterator();

```
Iterator it = c.iterator();
while ( it.hasNext() )
{
    Point p = (Point) it.next();
    System.out.println( p.toString() );
}
```

## 5.3. Các lớp tập hợp trong Java



## 5.3. Các lớp tập hợp trong Java

### 1. Lớp ArrayList

- Là một “thực thi” của giao diện List
- Phù hợp khi cần truy xuất ngẫu nhiên các phần tử trong tập hợp .

#### Constructor Summary

[`ArrayList\(\)`](#)

Constructs an empty list with an initial capacity of ten.

[`ArrayList\(Collection<? extends E> c\)`](#)

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

[`ArrayList\(int initialCapacity\)`](#)

Constructs an empty list with the specified initial capacity.

## 5.3. Các lớp tập hợp trong Java (tt)

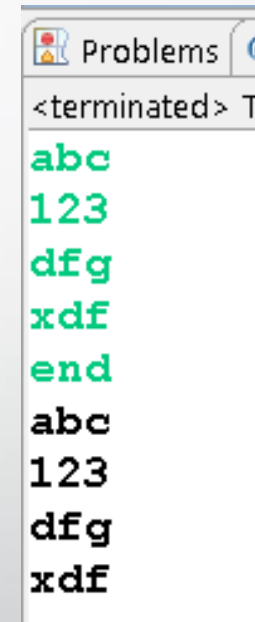
### 1. Lớp ArrayList (tt)

```
public static void main(String[] args)
{
    ArrayList list = new ArrayList();

    while (true)
    {
        Scanner scan = new Scanner(System.in);
        String s = scan.next();
        if (s.equalsIgnoreCase("end"))
        {
            break;
        }
        list.add(s);
    }

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

### Output



Problems

<terminated> T

abc  
123  
dfg  
xdf  
end  
abc  
123  
dfg  
xdf

## 5.3. Các lớp tập hợp trong Java (tt)

### 2. Lớp Vector

- Tương tự ArrayList
- Các phương thức của vector được đồng bộ → an toàn khi được sử dụng trong các Thread.

#### Constructor Summary

Vector()

Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.

Vector(Collection<? extends E> c)

Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Vector(int initialCapacity)

Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.

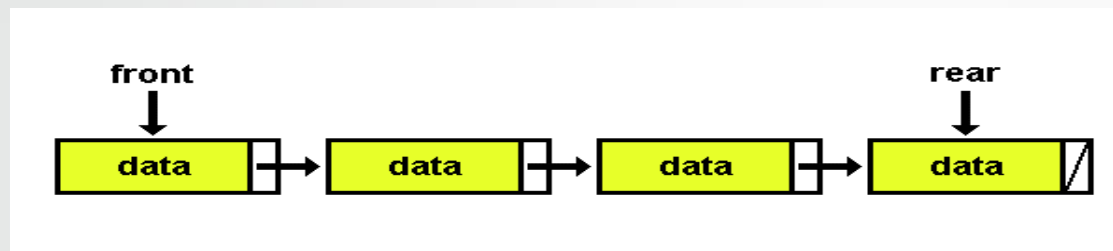
Vector(int initialCapacity, int capacityIncrement)

Constructs an empty vector with the specified initial capacity and capacity increment.

## 5.3. Các lớp tập hợp trong Java (tt)

### 3. Lớp LinkedList

- Các phần tử được lưu trữ dạng một danh sách liên kết.



#### Constructor Summary

[`LinkedList\(\)`](#)

Constructs an empty list.

[`LinkedList\(Collection<? extends E> c\)`](#)

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

## 5.3. Các lớp tập hợp trong Java (tt)

### 3. Lớp LinkedList (tt)

- Các phương thức của lớp LinkedList

#### Method Summary

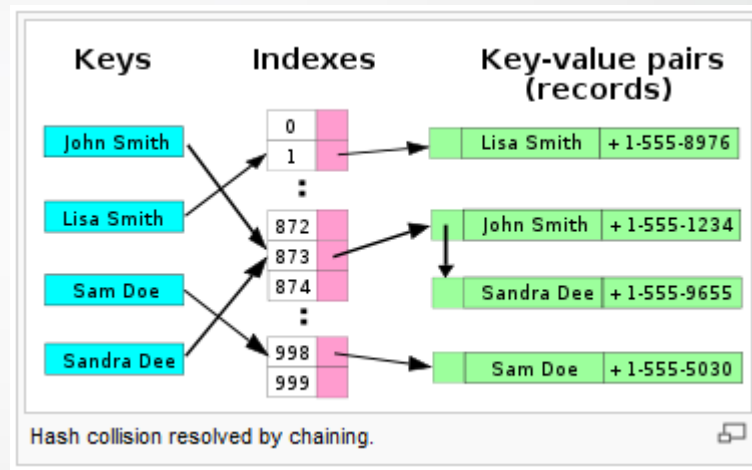
void	<a href="#"><u>addFirst</u></a> ( <a href="#"><u>E</u></a> o) Inserts the given element at the beginning of this list.
void	<a href="#"><u>addLast</u></a> ( <a href="#"><u>E</u></a> o) Appends the given element to the end of this list.
<a href="#"><u>E</u></a>	<a href="#"><u>getFirst</u></a> () Returns the first element in this list.
<a href="#"><u>E</u></a>	<a href="#"><u>getLast</u></a> () Returns the last element in this list.
<a href="#"><u>E</u></a>	<a href="#"><u>removeFirst</u></a> () Removes and returns the first element from this list.
<a href="#"><u>E</u></a>	<a href="#"><u>removeLast</u></a> () Removes and returns the last element from this list.



## 5.3. Các lớp tập hợp trong Java (tt)

### 4. Lớp HashSet

- Thực thi giao diện Set
- Sử dụng Hash Table để lưu dữ liệu.
- Constructor của HashSet



### Constructor Summary

#### `HashSet()`

Constructs a new, empty set; the backing `HashMap` instance has default initial capacity (16) and load factor (0.75).

#### `HashSet(Collection<? extends E> c)`

Constructs a new set containing the elements in the specified collection.

#### `HashSet(int initialCapacity)`

Constructs a new, empty set; the backing `HashMap` instance has the specified initial capacity and default load factor, which is 0.75.

#### `HashSet(int initialCapacity, float loadFactor)`

Constructs a new, empty set; the backing `HashMap` instance has the specified initial capacity and the specified load factor.

## 5.3. Các lớp tập hợp trong Java (tt)

### 5. Lớp `LinkedHashSet`

- Kết hợp giữa `HashSet` và `LinkedList`
- Sử dụng một List để duy trì thứ tự của các phần tử như khi chúng được thêm vào

#### Constructor Summary

`LinkedHashSet()`

Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).

`LinkedHashSet(Collection<? extends E> c)`

Constructs a new linked hash set with the same elements as the specified collection.

`LinkedHashSet(int initialCapacity)`

Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).

`LinkedHashSet(int initialCapacity, float loadFactor)`

Constructs a new, empty linked hash set with the specified initial capacity and load factor.

## 5.3. Các lớp tập hợp trong Java (tt)

### 5. Lớp LinkedHashSet, HashSet – Ví dụ

```
public void testHashSet()
{
    HashSet hs = new HashSet();
    hs.add("XYS");
    hs.add("A");
    hs.add("B");

    System.out.println("HashSet content:");
    for (Iterator i = hs.iterator(); i.hasNext();)
    {
        System.out.println(i.next());
    }
}

public void testLinkedHashSet()
{
    LinkedHashSet lhs = new LinkedHashSet();
    lhs.add("XYS");
    lhs.add("A");
    lhs.add("B");

    System.out.println("LinkedHashSet content:");
    for (Iterator i = lhs.iterator(); i.hasNext();)
    {
        System.out.println(i.next());
    }
}
```

HashSet content:  
A  
XYS  
B

LinkedHashSet content:  
XYS  
A  
B

## 5.3. Các lớp tập hợp trong Java (tt)

### 6. TreeSet

- Lưu giữ liệu theo cấu trúc “cây”.
- Các phần tử được lưu trữ theo thứ tự tăng dần

#### Constructor Summary

[`TreeSet\(\)`](#)

Constructs a new, empty set, sorted according to the elements' natural order.

[`TreeSet\(Collection<? extends E> c\)`](#)

Constructs a new set containing the elements in the specified collection, sorted according to the elements' *natural order*.

[`TreeSet\(Comparator<? super E> c\)`](#)

Constructs a new, empty set, sorted according to the specified comparator.

[`TreeSet\(SortedSet<E> s\)`](#)

Constructs a new set containing the same elements as the specified sorted set, sorted according to the same ordering.

## 5.3. Các lớp tập hợp trong Java (tt)

### 7. HashMap

- Thực thi giao diện Map
- Ví dụ:

```
public void testHashMap()
{
    HashMap hMap = new HashMap();

    hMap.put("K1", "Hi");
    hMap.put("K2", "Hello");
    hMap.put("K3", "Morning");
    hMap.put("K3", "Bonjour");

    System.out.println("HashMap content:");

    Set keySet = hMap.keySet();
    for (Iterator i = keySet.iterator(); i.hasNext();)
    {
        Object key = i.next();
        System.out.println(key + "- " + hMap.get(key));
    }
}
```

```
HashMap content:
K3- Bonjour
K1- Hi
K2- Hello
```

## 5.3. Các lớp tập hợp trong Java (tt)

### 8. TreeMap

- Lưu trữ các phần tử theo cấu trúc cây
- Các phần tử sắp xếp dựa trên giá trị của khóa.

#### Constructor Summary

[TreeMap](#)()

Constructs a new, empty map, sorted according to the keys' natural order.

[TreeMap](#)([Comparator](#)<? super [K](#)> c)

Constructs a new, empty map, sorted according to the given

[TreeMap](#)([Map](#)<? extends [K](#), ? extends [V](#)> m)

Constructs a new map containing the same mappings as the

[TreeMap](#)([SortedMap](#)<[K](#), ? extends [V](#)> m)

Constructs a new map containing the same mappings as the

#### Method Summary

[K](#) [firstKey](#)()

Returns the first (lowest) key currently in this sorted map.

[K](#) [lastKey](#)()

Returns the last (highest) key currently in this sorted map.

[SortedMap](#)<[K](#),[V](#)> [headMap](#)([K](#) toKey)

Returns a view of the portion of this map whose keys are strictly less than toKey.

[SortedMap](#)<[K](#),[V](#)> [tailMap](#)([K](#) fromKey)

Returns a view of the portion of this map whose keys are greater than or equal to fromKey.

## 5.3. Các lớp tập hợp trong Java (tt)

### 8. TreeMap (tt)

```
public void testTreeMap()
{
    TreeMap treeMap = new TreeMap();
    treeMap.put("101", "Hello");
    treeMap.put("102", "Hi");
    treeMap.put("103", "Morning");
    treeMap.put("104", "Bonjour");

    // Get first element
    Object fkey = treeMap.firstKey();
    System.out.println("First element: " + treeMap.get(fkey));

    // Get last element
    Object lkey = treeMap.lastKey();
    System.out.println("Last element: " + treeMap.get(lkey));

    System.out.println("Elements before key 103");
    SortedMap smap = treeMap.headMap("103");
    Set hMapKeys = smap.keySet();
    for (Iterator i = hMapKeys.iterator(); i.hasNext(); )
    {
        Object key = (Object) i.next();
        System.out.println(smap.get(key));
    }
}
```

```
First element: Hello
Last element: Bonjour
Elements before key 103
Hello
Hi
```

## 5.3. Các lớp tập hợp trong Java (tt)

### 9. LinkedHashMap

- Các phần tử trong tập hợp được duy trì thứ tự như khi chúng được thêm vào.

#### Constructor Summary

[`LinkedHashMap\(\)`](#)

Constructs an empty insertion-ordered `LinkedHashMap` instance with a default capacity (16) and load factor (0.75).

[`LinkedHashMap\(int initialCapacity\)`](#)

Constructs an empty insertion-ordered `LinkedHashMap` instance with the specified initial capacity and a default load factor (0.75).

[`LinkedHashMap\(int initialCapacity, float loadFactor\)`](#)

Constructs an empty insertion-ordered `LinkedHashMap` instance with the specified initial capacity and load factor.

[`LinkedHashMap\(int initialCapacity, float loadFactor, boolean accessOrder\)`](#)

Constructs an empty `LinkedHashMap` instance with the specified initial capacity, load factor and ordering mode.

[`LinkedHashMap\(Map<? extends K, ? extends V> m\)`](#)

Constructs an insertion-ordered `LinkedHashMap` instance with the same mappings as the specified map.

#### Method Summary

void	<a href="#"><code>clear()</code></a>	Removes all mappings from this map.
boolean	<a href="#"><code>containsValue(Object value)</code></a>	Returns <code>true</code> if this map maps one or more keys to the specified value.
<code>V</code>	<a href="#"><code>get(Object key)</code></a>	Returns the value to which this map maps the specified key.
protected boolean	<a href="#"><code>removeEldestEntry(Map.Entry&lt;K, V&gt; eldest)</code></a>	Returns <code>true</code> if this map should remove its eldest entry.



## 5.3. Các lớp tập hợp trong Java (tt)

### 10. Lớp PriorityQueue

- Các phần tử được sắp xếp theo thứ tự tự nhiên hoặc dựa vào một comparator.
- Không chấp nhận phần tử có giá trị null.

#### Constructor Summary

[PriorityQueue\(\)](#)

Creates a `PriorityQueue` with the default initial capacity (11) that orders its elements according to their natural ordering (using `Comparable`).

[PriorityQueue\(Collection<? extends E> c\)](#)

Creates a `PriorityQueue` containing the elements in the specified collection.

[PriorityQueue\(int initialCapacity\)](#)

Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their natural ordering (using `Comparable`).

[PriorityQueue\(int initialCapacity, Comparator<? super E> comparator\)](#)

Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.

[PriorityQueue\(PriorityQueue<? extends E> c\)](#)

Creates a `PriorityQueue` containing the elements in the specified collection.

[PriorityQueue\(SortedSet<? extends E> c\)](#)

Creates a `PriorityQueue` containing the elements in the specified collection.

## 5.3. Các lớp tập hợp trong Java (tt)

### 10. Lớp PriorityQueue (tt)

- Các phương thức

#### Method Summary

boolean	<a href="#">add</a> ( <a href="#">E</a> o) Adds the specified element to this queue.
void	<a href="#">clear</a> () Removes all elements from the priority queue.
<a href="#">Comparator</a> <? super <a href="#">E</a> >	<a href="#">comparator</a> () Returns the comparator used to order this collection, or <code>null</code> if this collection is sorted according to its <code>Comparable</code> implementation.
<a href="#">Iterator</a> < <a href="#">E</a> >	<a href="#">iterator</a> () Returns an iterator over the elements in this queue.
boolean	<a href="#">offer</a> ( <a href="#">E</a> o) Inserts the specified element into this priority queue.
<a href="#">E</a>	<a href="#">peek</a> () Retrieves, but does not remove, the head of this queue, returning <code>null</code> if this queue is empty.
<a href="#">E</a>	<a href="#">poll</a> () Retrieves and removes the head of this queue, or <code>null</code> if this queue is empty.
boolean	<a href="#">remove</a> ( <a href="#">Object</a> o) Removes a single instance of the specified element from this queue, if it is present.
int	<a href="#">size</a> () Returns the number of elements in this collection.

## 5.3. Các lớp tập hợp trong Java (tt)

### 10. Lớp PriorityQueue (tt)

```
public void testPriorityQueue()
{
    PriorityQueue pQueue = new PriorityQueue();

    pQueue.offer("Hello");
    pQueue.offer("Bonjour");
    pQueue.offer("Konichiowa");
    pQueue.offer("Abc");

    System.out.println("1. Comparator: " + pQueue.comparator());
    System.out.println("2. Content of Priority Queue");
    for (Iterator i = pQueue.iterator(); i.hasNext(); )
    {
        System.out.print(i.next() + " - ");
    }
    System.out.println("");
    System.out.println("3. Retrieve and remove head element: " + pQueue.poll());

    System.out.println("4. Now, content of Priority Queue is:");
    for (Iterator i = pQueue.iterator(); i.hasNext(); )
    {
        System.out.print(i.next() + " - ");
    }
}
```

#### Output

```
1. Comparator: null
2. Content of Priority Queue
Abc - Bonjour - Konichiowa - Hello -
3. Retrieve and remove head element: Abc
4. Now, content of Priority Queue is:
Bonjour - Hello - Konichiowa -
```

## 5.3. Các lớp tập hợp trong Java (tt)

### 11. Lớp Arrays (tt)

```
public void testArrays()
{
    int a[] = new int[3];
    a[0] = 9;
    a[1] = 6;
    a[2] = 3;

    Arrays.sort(a);

    System.out.println("Array after sorted:");
    for (int i = 0; i < a.length; i++)
    {
        System.out.println(a[i]);
    }
}
```

#### Output

Array after sorted:

3  
6  
9

## 5.3. Các lớp tập hợp trong Java (tt)

### 11. Lớp Arrays

- Chứa các phương thức cho phép thao tác trên mảng (sorting, searching)
- Các phương thức:

```
■ equals(<type>[] arrObj1, <type>[] arrObj2)
```

```
■ fill(<type>[] array, <type> value)
```

```
■ fill (<type>[] array, int fromIndex, int toIndex, type value)
```

```
■ sort(<type> [] array)
```

```
■ sort(<type> [] array, int startindex, int endIndex)
```

```
■ toString()
```

## 5.3. Các lớp tập hợp trong Java (tt)

### 12. Các lớp bao (wrapper classes)

- Collection chỉ làm việc trên các Object. Những kiểu dữ liệu cơ bản như: byte, short, int, long, double, float, char, boolean không thể đưa được trực tiếp vào Collection mà phải thông qua các lớp bao.
- Các lớp bao: Byte, Short, Int, Long, Double, Float, Char, Boolean.
- Ví dụ:
  - `Integer intObject = new Integer(9);`
  - `int value = intObject.intValue();`

## 5.4. Ứng dụng của Tập hợp trong lập trình

### Bài tập

1. Cài đặt các xử lý Exception cần thiết cho các phương thức trong LinkedList, Stack, Queue, Tree.
2. Viết chương trình cho phép nhập một chuỗi ký tự từ bàn phím, sau đó hiển thị chuỗi này theo thứ tự ngược lại (dùng Stack).
3. Viết chương trình cho phép nhập một danh sách sinh viên sau đó sắp xếp danh sách theo thứ tự tăng dần. Dùng ArrayList và Collections.sort().
4. Giải các bài toán ứng dụng trong môn Cấu trúc dữ liệu bằng cách sử dụng Collections Framework.

