Chương 5

TẬP HỢP TRÊN JAVA

Mục tiêu

- Phân biệt tập hợp và mảng
- Phân biệt các đặc trưng của các Collection interface
- Chọn loại tập hợp thích hợp để giải quyết bài toán

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. Case study

5.1. Khái niệm về tập hợp

- Tập hợp là đối tượng có khả năng chứa các đối tượng khác
- Các thao tác thông thường trên tập hợp
  - Thêm đối tượng vào tập hợp
  - Xoá đối tượng khỏi tập hợp
  - Kiểm tra một đối tượng có tồn tại 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 Collections Framework

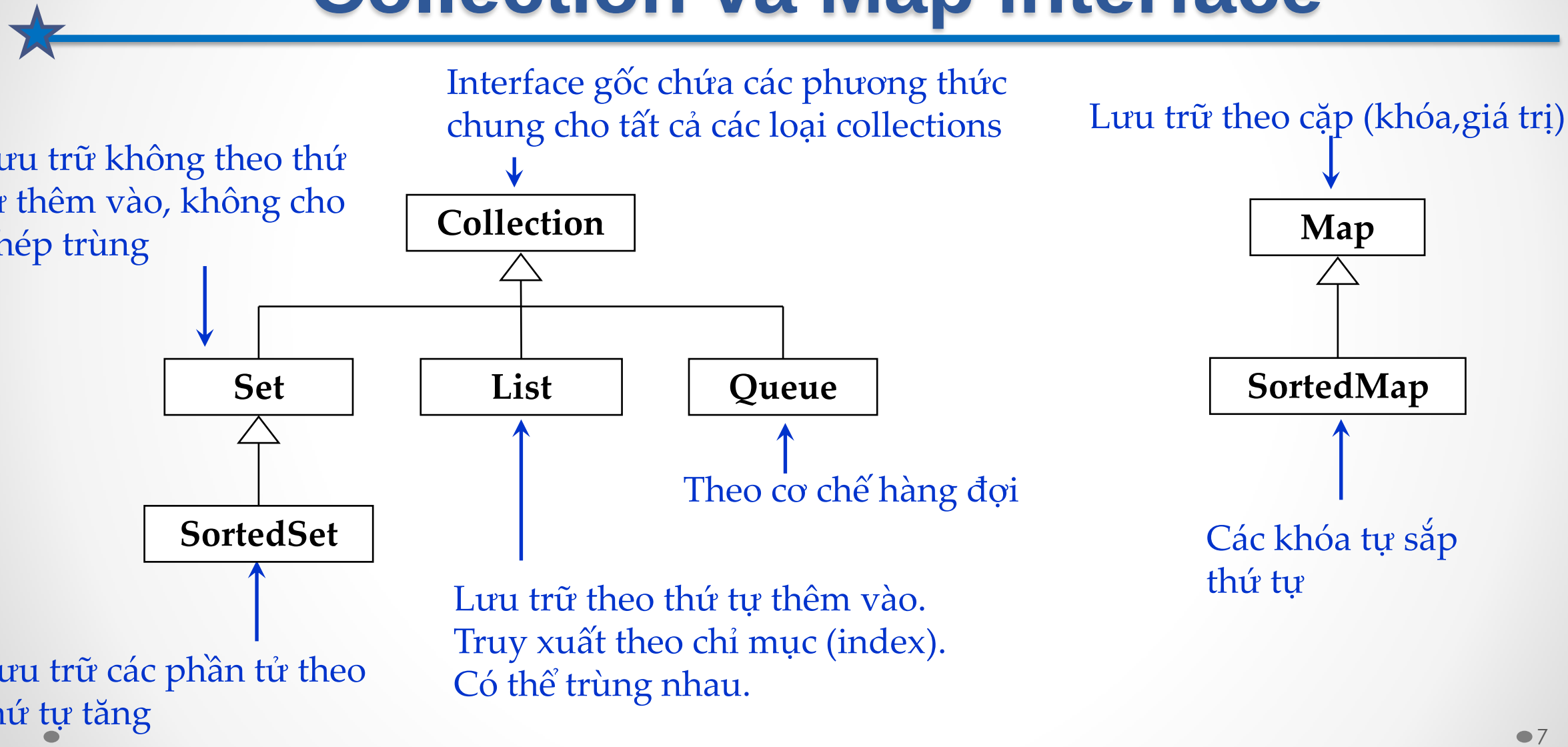
- Collections Framework (từ Java 1.2)
  - Là một kiến trúc hợp nhất để biểu diễn và thao tác trên các loại tập hợp
  - Giúp cho việc xử lý tập hợp độc lập với biểu diễn chi tiết bên trong

- 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

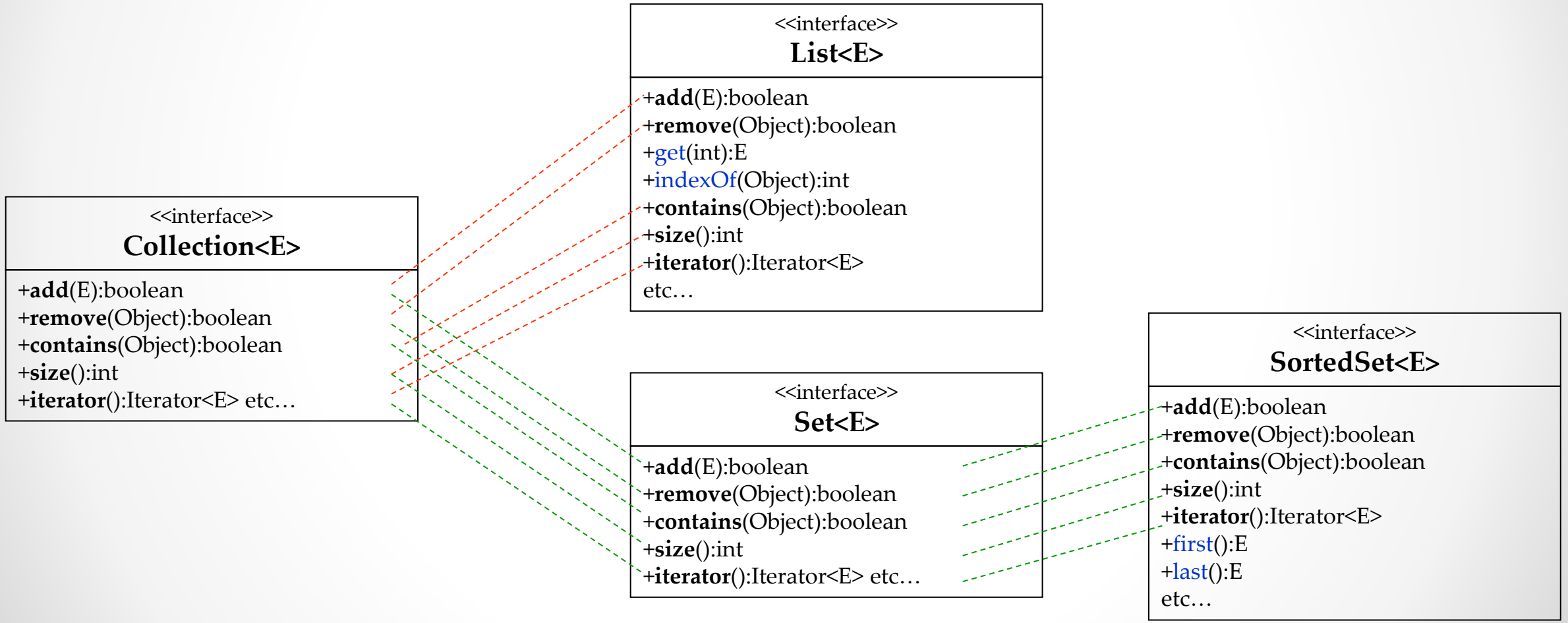
5.1. Khái niệm về tập hợp Collections Framework

- Collections Framework được cung cấp từ gói `java.util`, 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 như LinkedList, HashSet,…
  - Algorithms: Là các phương thức tĩnh để xử lý collection, ví dụ: sắp xếp danh sách, tìm phần tử lớn nhất...

5.1. Khái niệm về tập hợp Collection và Map interface



5.1. Khái niệm về tập hợp
So sánh một số interface



5.1. Khái niệm về tập hợp

Các phương thức của Collection interface

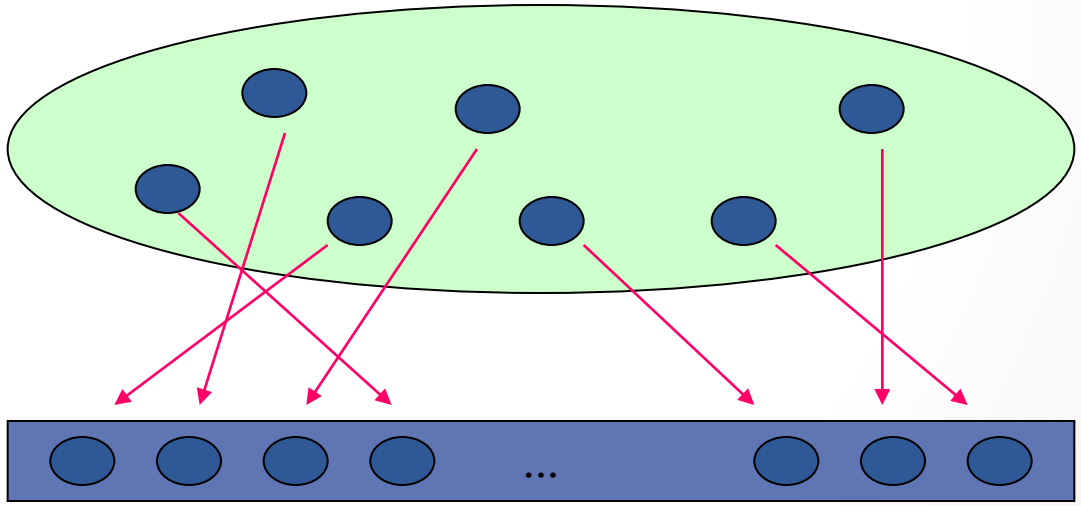
Method Summary	
boolean	**add**(E o) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	**clear**() Removes all of the elements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

5.1. Khái niệm về tập hợp
Duyệt collection

- Các phần tử trong collection có thể được duyệt tuần tự thông qua Iterator interface
- Các lớp cài đặt Collection cung cấp phương thức trả về Iterator trên các phần tử của chúng

Collection c;

Iterator it = c.**iterator**();



5.1. Khái niệm về tập hợp
Duyệt collection

- Các phương thức của Iterator:
  - boolean hasNext(): trả về *true* nếu còn phần tử chưa duyệt
  - Object next(): trả về phần tử kế
  - void remove(): xóa phần tử đang duyệt
- Cách sử dụng:

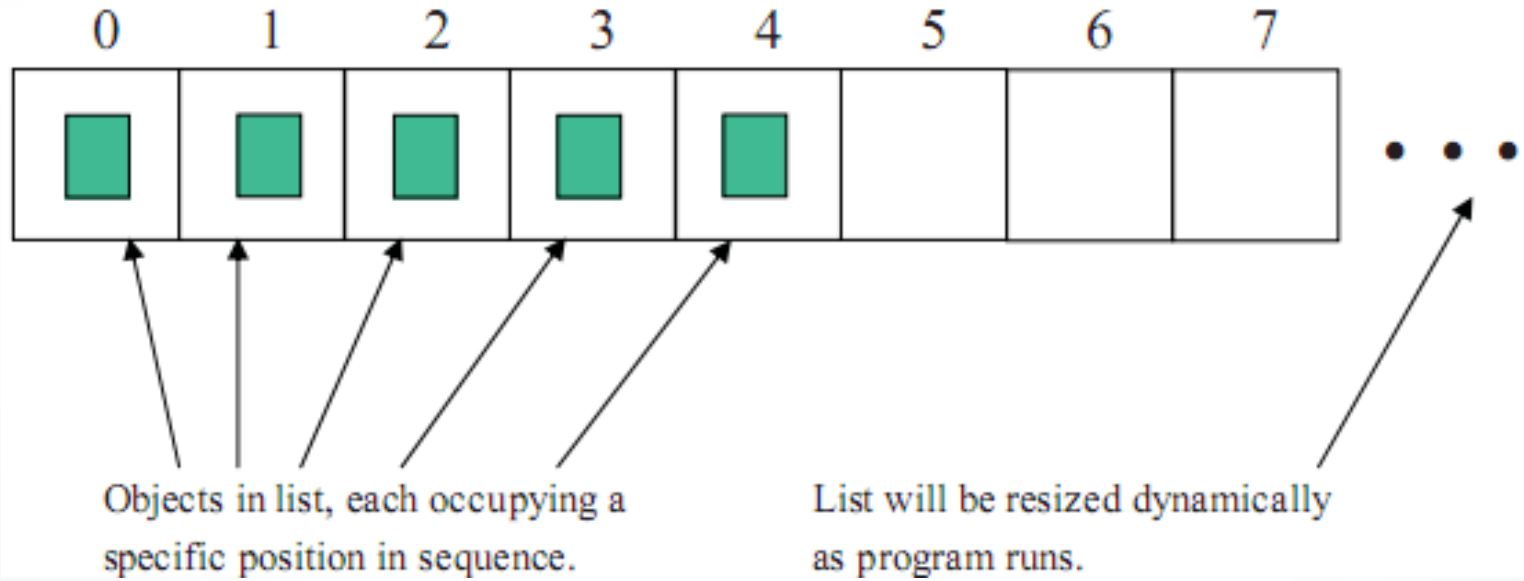
```
Iterator<Item> it = items.iterator();
while(it.hasNext()) {
      Item item = it.next();
      System.out.println(item);
}
```

=

```
for (Item item : items) {
   System.out.println(item);
}
```

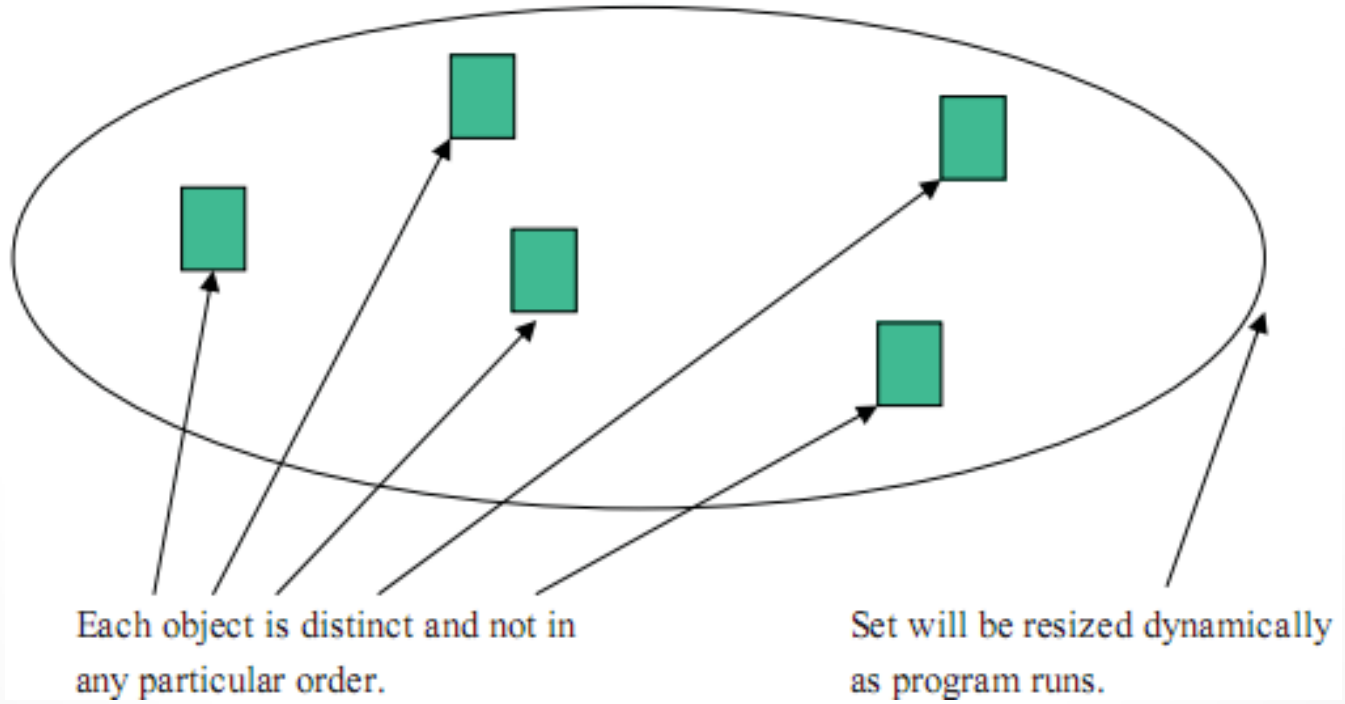
5.1. Khái niệm về tập hợp
List interface

- Giống như array nhưng linh hoạt hơn, vì có thể tự điều chỉnh kích thước khi thêm dữ liệu
- Lưu trữ dữ liệu theo thứ tự thêm vào, có thể truy xuất theo chỉ mục (index)



5.1. Khái niệm về tập hợp
Set interface

- Giống như “bag” hơn là “list”
- Các phần tử lưu trong Set không được trùng và không quan tâm thứ tự thêm vào



5.1. Khái niệm về tập hợp SortedSet interface

- Hỗ trợ thao tác trên tập hợp các phần tử có thể so sánh được
- Các đối tượng đưa vào SortedSet phải có khả năng so sánh được, tức là phải implements Comparable interface HOẶC lớp cài đặt SortedSet phải nhận một Comparator trên đối tượng khoá

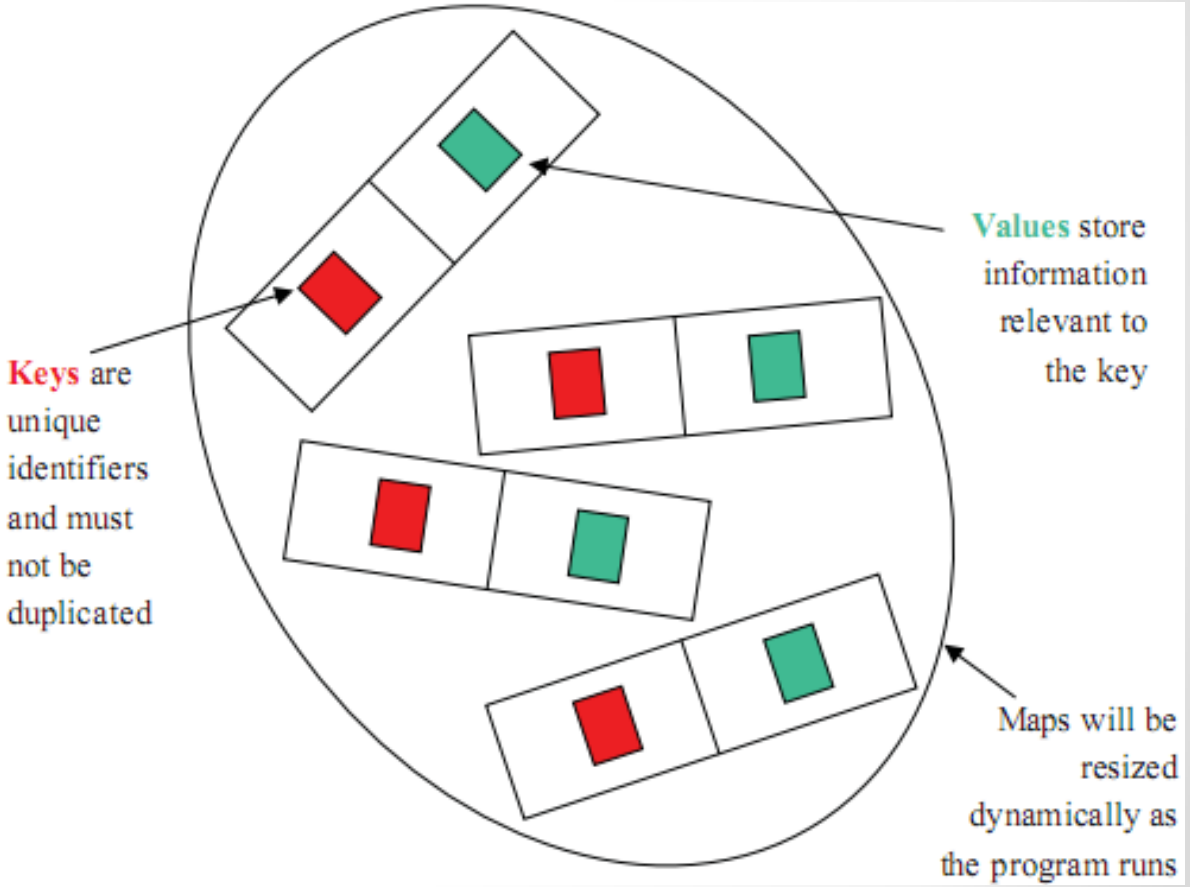
5.1. Khái niệm về tập hợp Queue interface

- Các phần tử được truy xuất theo thứ tự FIFO
- Các phương thức của Queue

Method Summary	
E	**element**() Retrieves, but does not remove, the head of this queue.
boolean	**offer**(E o) Inserts the specified element into this queue, if possible.
E	**peek**() Retrieves, but does not remove, the head of this queue, returning `null` if this queue is empty.
E	**poll**() Retrieves and removes the head of this queue, or `null` if this queue is empty.
E	**remove**() Retrieves and removes the head of this queue.

5.1. Khái niệm về tập hợp
Map interface

- 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
- Giống như Set, các phần tử trong Map không có thứ tự



5.1. Khái niệm về tập hợp
Map interface (tt)

- Hai phương thức truy cập:
  - V put(K key, V value) // Thêm cặp key-value vào map
  - V get(Object key) // Trả về value tương ứng với key cho trước
- Ba cách xem dữ liệu:
  - Lấy tất cả các khoá:
    - Set keySet(); // Trả về các khoá
  - Lấy tất cả các giá trị:
    - Collection values(); // Trả về các giá trị
  - Lấy tất cả các cặp khoá-giá trị
    - Set entrySet(); // Trả về các cặp khoá-giá trị

5.1. Khái niệm về tập hợp SortedMap interface

- 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

5.2. So sánh tập hợp và mảng

Mảng	Tập hợp
Có kích thước cố định	Kích thước có tính "tự giãn nở"
Duyệt các phần tử thông qua chỉ số mảng	Có thể duyệt các phần tử thông qua Iterator
Phải lập trình hoàn toàn	Gọi những phương thức đã được định nghĩa

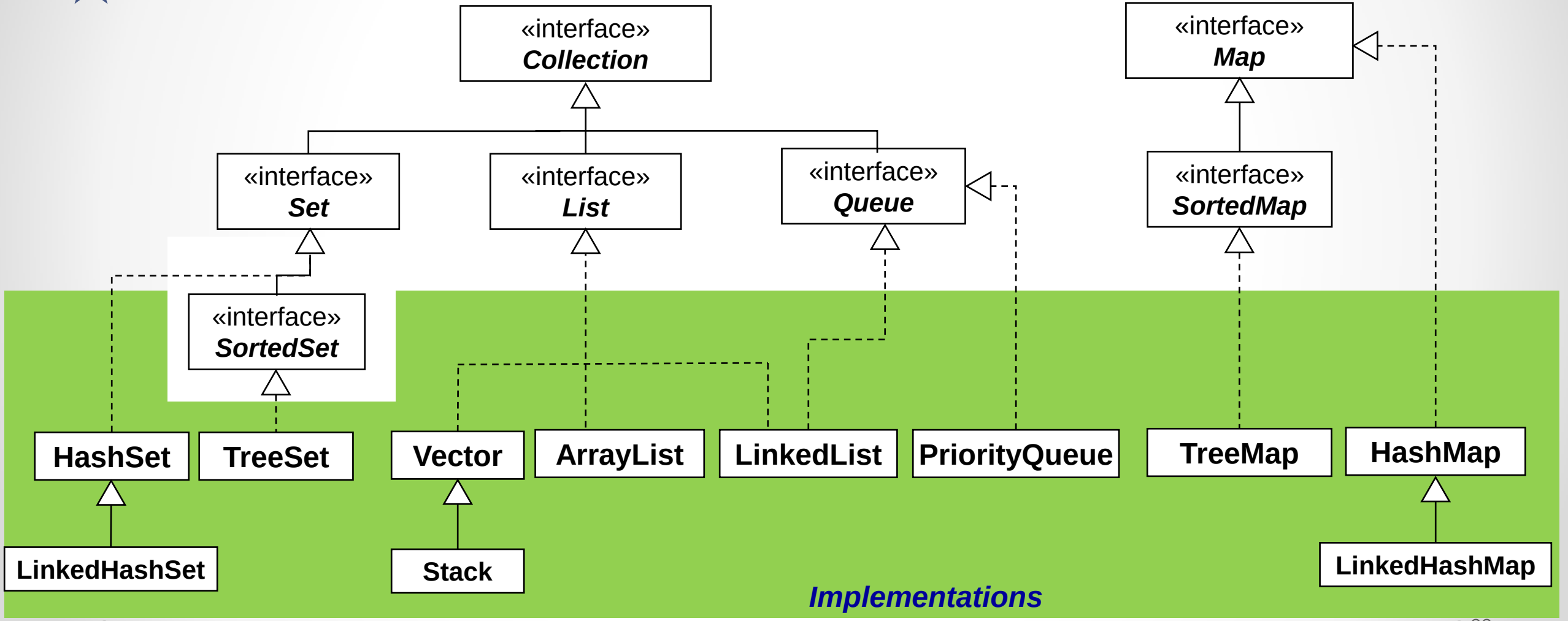
Review questions 1 [1/2]

1. `List` và `Set` khác nhau ở điểm đặc trưng nào.
2. Các phần tử trong `SortedSet`, `SortedMap` được tự động sắp theo thứ tự tăng, dựa vào tiêu chí nào.
3. Cần chú ý gì khi sử dụng `SortedSet`, `SortedMap`.
4. Làm thế nào để duyệt các phần tử trong tập hợp.
5. Mỗi trường hợp sau đây nên dùng collection nào (`List`, `Set`, `Map`,…):
   a) Ghi nhận các thành viên của câu lạc bộ
   b) Ghi nhận các khoản tiền đã chi
   c) Ghi nhận chi tiết thông tin tài khoản ngân hàng, với mỗi tài khoản có một số hiệu duy nhất

Review questions 1 [2/2]

5. (tt) Mỗi trường hợp sau đây nên dùng collection nào (List, Set, Map,…):
   d) Công ty XYZ cần lưu tên của các nhân viên của mình. Mỗi tháng một nhân viên sẽ được chọn ngẫu nhiên để nhận một quà tặng.
   e) Công ty XYZ cần đặt tên cho sản phẩm mới, tên sản phẩm được chọn từ tên của nhân viên, vì vậy tên không được trùng, tên chỉ được dùng có một lần. Chọn loại collection để lưu danh sách tên sản phẩm.
   f) Công ty XYZ muốn cho nhân viên đi du lịch, chính sách được tạo ra là ưu tiên cho những người đăng ký trước.
   g) Công ty XYZ muốn tạo danh sách các khách hàng theo thứ tự tăng dần theo doanh số.

5.3. Các lớp tập hợp trong Java



5.3. Các lớp tập hợp trong Java
Lớp ArrayList

- Thực thi List interface
- Phù hợp khi cần truy xuất ngẫu nhiên các phần tử trong tập hợp
- Các hàm khởi tạo:

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
Lớp ArrayList: Ví dụ

```
import java.util.*;
public class ArrayListTest {
  public static void main(String[] args) {
    ArrayList<String> obj = new ArrayList<String>();
    obj.add("Ajeet");
    obj.add("Harry");
    obj.add("Steve");
    obj.add("Anuj");
    /* Displaying array list elements */
    System.out.println("Currently the array list has following elements:" + obj);
    /* Add element at the given index */
    obj.add(1, "Justin");
    /* Remove elements from array list like this */
    obj.remove("Harry");
    System.out.println("Current array list is:" + obj);
    /* Remove element from the given index */
    obj.remove(1);
    System.out.println("Current array list is:" + obj);
  }
}
```

```
Currently the array list has following elements:[Ajeet, Harry, Steve, Anuj]
Current array list is:[Ajeet, Justin, Steve, Anuj]
Current array list is:[Ajeet, Steve, Anuj]
```

5.3. Các lớp tập hợp trong Java
Lớp ArrayList: Quiz

- Cho đoạn mã sau:

```
ArrayList myArrayList = new ArrayList( );
myArrayList.add("One");
myArrayList.add("Two");
myArrayList.add("Three");
myArrayList.add("Four");
```

Đoạn lệnh nào sau đây làm thay đổi myArrayList thành: One; Two; Four

a. `myArrayList.remove (myArrayList.get(3));`
b. `myArrayList.remove (myArrayList.indexOf("Three"));`
c. `myArrayList.remove (Three);`
d. `myArrayList.remove (myArrayList.get(2));`

5.3. Các lớp tập hợp trong Java
Lớp ArrayList: Quiz

- Cho đoạn mã sau:

```
ArrayList myArrayList = new ArrayList( );
myArrayList.add("One");
myArrayList.add("Two");
myArrayList.add("Three");
myArrayList.add("Four");
```

Đoạn lệnh nào sau đây làm thay đổi myArrayList thành: One; Two; Three; Five

```
a. myArrayList[3] = "Five";
b. myArrayList[4] = "Five";
c. myArrayList.set (myArrayList.indexOf("Four"), "Five");
d. myArrayList.set (myArrayList.indexOf("Five"), "Four");
```

5.3. Các lớp tập hợp trong Java
Lớp ArrayList: Quiz

- Cho đoạn mã sau:

```
ArrayList myArrayList = new ArrayList();
myArrayList.add(1);
myArrayList.add(3);
myArrayList.add(7);
```

Đoạn lệnh nào sau đây làm thay đổi myArrayList thành: 1 3 5 7

```
a. myArrayList.add(5);
b. myArrayList.add(2, 5);
c. myArrayList.add(4, 5);
d. myArrayList.add(3, 5);
```

5.3. Các lớp tập hợp trong Java
Lớp ArrayList: Demo

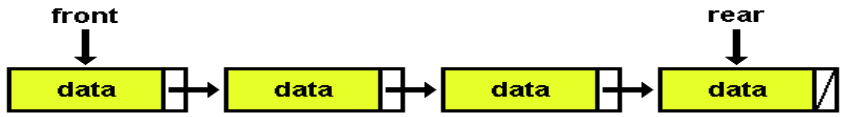
- Tạo một ArrayList chứa tập các sinh viên, mỗi sinh viên gồm các thông tin *mã, họ tên, năm sinh*. Thêm vào danh sách 5 sinh viên tùy ý (không được trùng mã). Viết các phương thức:
  - in danh sách sinh viên dạng bảng
  - thêm một sinh viên mới
  - xóa sinh viên khi biết mã
  - sửa thông tin sinh viên (không sửa mã)
  - tìm kiếm sinh viên theo mã, theo tên,…
  - sắp xếp danh sách theo mã tăng dần,…
- Xây dựng lớp với các thao tác trên tập hợp nêu trên.

5.3. Các lớp tập hợp trong Java
Lớp Vector

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

5.3. Các lớp tập hợp trong Java
Lớp LinkedList

- Thực thi List và Queue interface
- Các phần tử được lưu trữ dạng một danh sách liên kết



- Các phương thức

Method Summary

void	**addFirst**(E o) Inserts the given element at the beginning of this list.
void	**addLast**(E o) Appends the given element to the end of this list.
E	**getFirst**() Returns the first element in this list.
E	**getLast**() Returns the last element in this list.
E	**removeFirst**() Removes and returns the first element from this list.
E	**removeLast**() Removes and returns the last element from this list.

5.3. Các lớp tập hợp trong Java
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
- Các hàm:

Method Summary

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

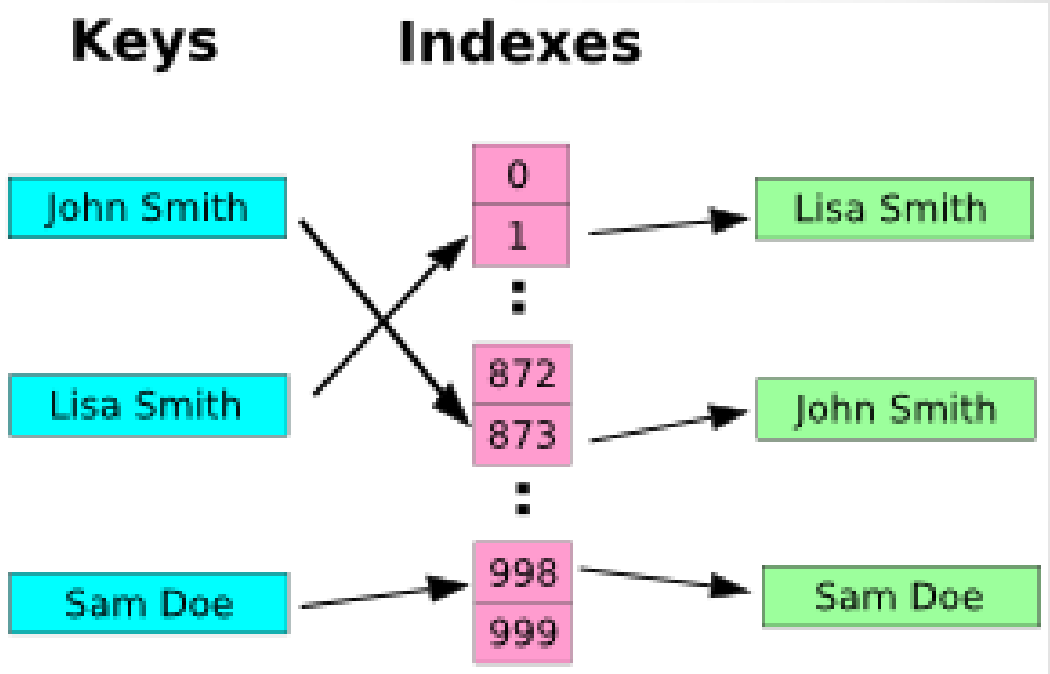
5.3. Các lớp tập hợp trong Java
Lớp PriorityQueue: Ví dụ

```
public static void main(String[] args) {
    PriorityQueue<String> pQueue = new PriorityQueue<String>();
    pQueue.offer("Hello");
    pQueue.offer("Bonjour");
    pQueue.offer("Konichiowa");
    pQueue.offer("Abc");
    System.out.println("Content:" + pQueue);

    System.out.println("Retrieve and remove head element: " + pQueue.poll());

    System.out.println("Current content:" + pQueue);
}
```

```
Content:[Abc, Bonjour, Konichiowa, Hello]
Retrieve and remove head element: Abc
Current content:[Bonjour, Hello, Konichiowa]
```

5.3. Các lớp tập hợp trong Java
Lớp HashSet

- Thực thi Set interface
- HashSet lưu trữ các phần tử bằng cách sử dụng một cơ chế gọi là băm (hashing)
- HashSet không cho phép lưu trùng lặp



5.3. Các lớp tập hợp trong Java
Lớp HashSet: Ví dụ

```
import java.util.*;
public class HashSetTest {
  public static void main(String[] args) {
     Set<String> words = new HashSet<String>();
     words.add("Bats");
     words.add("Ants");
     words.add("Crabs");
     words.add("Ants");
     System.out.println(words);
  }
}
```

5.3. Các lớp tập hợp trong Java
Lớp TreeSet

- Thực thi SortedSet interface
- Lưu giữ liệu theo cấu trúc “cây”
- Chậm hơn HashSet nhưng có khả năng tự sắp xếp: các phần tử được lưu trữ theo thứ tự tăng dần
- Có thể quy định thứ tự bằng:
  - Comparable (theo thứ tự tự nhiên: số, chuỗi)
  - Comparator

5.3. Các lớp tập hợp trong Java
Lớp TreeSet: Ví dụ 1

- String có thứ tự tự nhiên:

```
Set<String> words = new TreeSet<String>();
words.add("Bats");
words.add("Ants");
words.add("Crabs");
System.out.println(words);
```

What's the output?
`[Ants, Bats, Crabs]`

5.3. Các lớp tập hợp trong Java
Lớp TreeSet: Ví dụ 2

- Quy định thứ tự bằng Comparable:

```
public class Student implements Comparable<Student> {
    private String name;
    private int gpa;
    //…
    @Override
    public int compareTo(Student o) {
        return this.name.compareToIgnoreCase(o.name);
    }
}
```

Student
- name - gpa

```
Set<Student> studentSet = new TreeSet<Student>();
studentSet.add(new Student("Fred", 3));
studentSet.add(new Student("Sam", 4));
studentSet.add(new Student("Steve", 3));
studentSet.add(new Student("Laura", 2));
for (Student st: studentSet) {
   System.out.println(st);
}
```

What's the output?

```
Fred-3
Laura-2
Sam-4
Steve-3
```

5.3. Các lớp tập hợp trong Java
Lớp TreeSet: Ví dụ 3

- Quy định thứ tự bằng Comparator (truyền vào constructor của TreeSet)

```
public class StudentNameComparator implements Comparator<Student> {
    @Override
    public int compare(Student o1, Student o2) {
        return o1.getName().compareToIgnoreCase(o2.getName());
    }
}
```

Student
- name - gpa

```
Set<Student> studentSet = new TreeSet<Student>(new StudentNameComparator());
studentSet.add(new Student("Fred", 3));
studentSet.add(new Student("Sam", 4));
studentSet.add(new Student("Steve", 3));
studentSet.add(new Student("Laura", 2));
for (Student st: studentSet) {
   System.out.println(st);
}
```

What's the output?

```
Fred-3
Laura-2
Sam-4
Steve-3
```

5.3. Các lớp tập hợp trong Java
Lớp HashMap

- Thực thi Map interface
- Hoạt động dựa trên nguyên lý của việc băm dữ liệu (hashing)
- Lưu trữ dữ liệu theo từng cặp: khóa-giá trị (key-value)
- Các khóa trong Map phải duy nhất
- Có thể có một khóa null và nhiều giá trị null
- Lưu trữ không theo trật tự thêm vào

5.3. Các lớp tập hợp trong Java
Lớp HashMap: Ví dụ

- Tạo và xuất HashMap:

```
HashMap<Integer, String> hmap = new HashMap<Integer, String>();
hmap.put(24, "Hà Nội");
hmap.put(236, "Đà Nẵng");
hmap.put(258, "Khánh Hòa");
hmap.put(28, "Thành Phố Hồ Chí Minh");
System.out.println(hmap);
```

```
{258=Khánh Hòa, 24=Hà Nội, 236=Đà Nẵng, 28=Thành Phố Hồ Chí Minh}
```

- Lấy phần tử có khóa tùy ý:

```
String var = hmap.get(28);
System.out.println("Value at index 28 is: " + var);
```

```
Value at index 2 is: Thành Phố Hồ Chí Minh
```

- Lấy ra tập key rồi duyệt dựa trên nó:

```
for (int key : hmap.keySet()) {
  System.out.println("Key = " + key + " & Value = " + hmap.get(key));
}
```

```
Key = 258 & Value = Khánh Hòa
Key = 24 & Value = Hà Nội
Key = 236 & Value = Đà Nẵng
Key = 28 & Value = Thành Phố Hồ Chí Minh
```

5.3. Các lớp tập hợp trong Java
Lớp TreeMap

- Thực thi SortedMap interface
- Lưu trữ dữ liệu dưới dạng cặp key-value, theo cấu trúc cây
- Các phần tử sắp xếp dựa trên giá trị của khóa: các đối tượng khoá đưa vào trong TreeMap phải implements interface Comparable HOẶC lớp cài đặt TreeMap phải nhận một Comparator trên đối tượng khoá
- Không cho phép key là null nhưng có thể có nhiều giá trị null

5.3. Các lớp tập hợp trong Java
Lớp TreeMap: Ví dụ

- Tạo và xuất TreeMap:

```
TreeMap<Integer, String> tmap = new TreeMap<Integer, String>();
tmap.put(24, "Hà Nội");
tmap.put(236, "Đà Nẵng");
tmap.put(258, "Khánh Hòa");
tmap.put(28, "Thành Phố Hồ Chí Minh");
System.out.println(tmap);
```

```
{24=Hà Nội, 28=Thành Phố Hồ Chí Minh, 236=Đà Nẵng, 258=Khánh Hòa}
```

- Lấy phần tử có khóa tùy ý:

```
String var = tmap.get(28);
System.out.println("Value at index 28 is: " + var);
```

```
Value at index 2 is: Thành Phố Hồ Chí Minh
```

- Lấy ra tập key rồi duyệt dựa trên nó:

```
for (int key : tmap.keySet()) {
  System.out.println("Key = " + key + " & Value = " + tmap.get(key));
}
```

```
Key = 24 & Value = Hà Nội
Key = 28 & Value = Thành Phố Hồ Chí Minh
Key = 236 & Value = Đà Nẵng
Key = 258 & Value = Khánh Hòa
```

5.3. Các lớp tập hợp trong Java
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
Lớp Arrays: Ví dụ

```
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
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 tương ứng: Byte, Short, Integer, Long, Double, Float, Char, Boolean

- Ví dụ:

```
Integer intObject = new Integer(9);
int value = intObject.intValue();
```

Review questions 2

- Trong `ArrayList`, làm thế nào để thêm, xóa, sửa, tìm kiếm, lấy phần tử bất kỳ.
- So sánh `HashSet`, `TreeSet` với các đặc điểm: ordering, performance, null value
- `ArrayList`, `Vector`, `LinkedList` có đặc điểm chung gì.
- `TreeSet`, `TreeMap` có đặc điểm chung gì.
- `PriorityQueue` và `ArrayList` giống nhau và khác nhau ở điểm nào.

5.4. Case study

Giả sử một nhà xuất bản phát hành hai loại: **sách** và **đĩa CD**. Để quản lý chúng, người ta cần *mã, tựa đề* và *giá* của ấn phẩm, *số trang* của sách và *thời gian* tính theo phút của đĩa. Người quản lý còn theo dõi *tình trạng quá lớn* của ấn phẩm, với quy ước sách dày hơn 999 trang, băng từ quá 180 phút được xem là các ấn phẩm quá lớn; theo dõi ấn phẩm *có hợp pháp hay không* thông qua việc xem xét *đã có giấy phép xuất bản* chưa.

Thiết kế và cài đặt cho mỗi loại ấn phẩm được mô tả trên.

Chọn loại danh sách phù hợp và cài đặt (ArrayList, LinkedList, Vector) để lưu trữ tập các ấn phẩm. Trong đó cho phép: **thêm** (không được trùng mã), **xóa** (theo mã), **sửa** ấn phẩm (theo mã), **in** toàn bộ ấn phẩm, **tìm kiếm** theo tựa đề, **xuất** các ấn phẩm quá lớn và các ấn phẩm không hợp pháp, **sắp xếp** các ấn phẩm theo giá, **sắp xếp** theo số trang của sách…

Tạo menu lựa chọn để thực hiện các chức năng ở danh sách trên.

Solution

Thường kỳ LO23_2

Yêu cầu:

- Câu 1. (2 điểm)

Sinh viên trình bày *các bước* cần thiết để giải quyết bài toán được mô tả trên (*sinh viên cần ghi rõ thứ tự lớp nào viết trước - lớp nào viết sau, lớp sau có mối quan hệ gì với lớp trước, lớp nào là lớp trừu tượng*).

- Câu 2. (8 điểm)

a. Ở mỗi lớp được nêu ở câu 1, sinh viên liệt kê đầy đủ thông tin chi tiết theo mô tả bài toán gồm: **tên lớp**, **các thuộc tính của lớp** (cần ghi rõ kiểu dữ liệu), **các phương thức getter/setter**, **constructor đầy đủ tham số**, **các phương thức khác của lớp**. Cần g*hi rõ phương thức nào là trừu tượng, phương thức nào là override.*

b. Ghi rõ phương thức nào cần kiểm tra ràng buộc dữ liệu + mô tả ràng buộc theo quy định như sau: kiểu chuỗi không được phép rỗng, kiểu số phải >=0.

Mô tả yêu cầu

- Cho mô tả bài toán sau:

Giả sử một nhà xuất bản phát hành hai loại ấn phẩm: **sách** và **đĩa CD**. Để quản lý chúng, người ta cần *mã, tựa đề, giá tiền* và *tình trạng có giấy phép xuất bản hay chưa* của ấn phẩm, *số trang* của sách và *thời gian* tính theo phút của đĩa.

Người quản lý còn phải theo dõi *tình trạng quá lớn* của ấn phẩm (với quy ước sách dày hơn 999 trang, băng từ quá 180 phút được xem là các ấn phẩm quá lớn); và theo dõi ấn phẩm *có hợp pháp hay không* thông qua việc xem xét *đã có giấy phép xuất bản* chưa.

Mô tả yêu cầu

Ngoài ra, người ta cũng muốn tạo một lớp để quản lý danh sách các ấn phẩm (dùng ArrayList). Trong lớp đó cho phép: thêm một ấn phẩm (không cho phép trùng mã); xóa ấn phẩm theo mã; sửa ấn phẩm theo mã (chỉ sửa thông tin của ấn phẩm, không sửa mã); lấy thông tin toàn bộ ấn phẩm; đếm số lượng các ấn phẩm quá lớn; lấy danh sách các ấn phẩm không hợp pháp; tìm ấn phẩm theo tựa đề (tìm tương đối); tìm số trang lớn nhất của sách.