NGÔN NGỮ LẬP TRÌNH JAVA

Nội dung

Cơ bản về ngôn ngữ lập trình Java

- Lập trình hướng đối tượng
- Biến, từ khoá, kiểu dữ liệu
- Biểu thức, các cấu trúc điều khiển
- Dữ liệu kiểu mảng

Các khía cạnh nâng cao của lập trình hướng đối tượng

- Thiết kế lớp
- Thiết kế lớp nâng cao
- Xử lý ngoại lệ
- Generics

- Collection Framework

THE JAVA COLLECTION FRAMEWORK

Nội dung

1. Giới thiệu về Java Collection Framework
2. Các interfaces của Collection Framework
3. List<E> interface
4. Thứ tự, sắp xếp và tìm kiếm

1. Giới thiệu về JCF

- Để lưu trữ các phần tử có cùng kiểu (primitives hoặc objects):
  - Sử dụng mảng:
    - Kích thước cố định;
    - không hỗ trợ cấp phát động (dynamic allocation)
    - Cấu trúc tuyến tính
  - Nhiều ứng dụng yêu cầu cấu trúc dữ liệu phức tạp hơn : linked list, stack, hash table, sets và trees.
- Trong Java, các cấu trúc dữ liệu cấp phát động được hỗ trợ trong một kiến trúc thống nhất
  - The Collection Framework

1. Giới thiệu về JCF

- Định nghĩa: 1 Collection là 1 đối tượng lưu trữ một tập hợp, một nhóm các đối tượng khác.
- Mỗi đối tượng trong collection được gọi là 1 phần tử (element).
- Java Collection Framework là một hệ thống interfaces đồng nhất để lưu trữ, tìm kiếm và thay đổi các phần tử của 1 collection, mà ko quan tâm đến kiểu dữ liệu của phần tử đó.

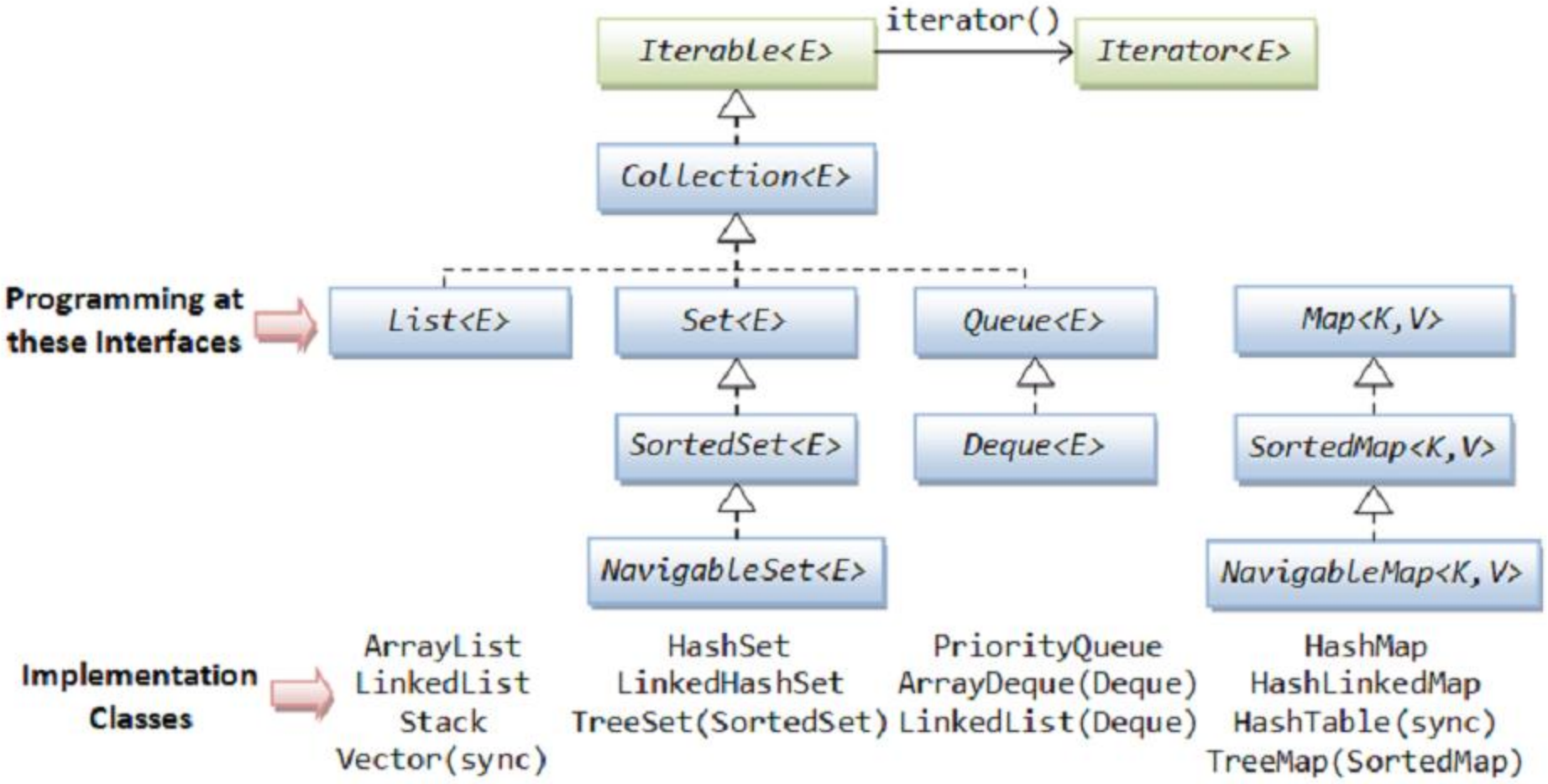
1. Giới thiệu về JCF

- Lợi ích của việc sử dụng JCF:
  - Tăng hiệu suất và chất lượng lập trình
  - Giảm thời gian học và sử dụng các API mới
  - Giảm thời gian thiết kế các API mới
  - Thúc đẩy tái sử dụng phần mềm
- Gói “java.util” cài đặt JCF, bao gồm:
  - Một tập hợp các interfaces
  - Các lớp cài đặt
  - Giải thuật (vd: tìm kiếm, sắp xếp)

Ví dụ

- Định nghĩa: 1 Collection là 1 đối tượng lưu trữ một tập hợp, một nhóm các đối tượng khác.
- Mỗi đối tượng trong collection được gọi là 1 phần tử (element).
- 1 framework là một tập hợp các interfaces dùng để một tập các thiết kế thực.
- 1 framework được thiết kế tốt làm tăng hiệu suất và khả năng bảo trì

2. Các interfaces của JCF



2.1.Iterator<E> interface

- Iterable interface:
  - Đối số là kiểu generics E
  - Định nghĩa cơ chế để duyệt qua các phần tử của một collection thông qua đối tượng Iterator
  - Chỉ chứa 1 abstract method,
    - Iterator iterator();
    - Trả về đối tượng Iterator<E> gắn với tất cả các collections
- Iterator interface
  - Gồm các abstract methods :

```
boolean hasNext()  // returns true if it has more elements
Object next()      // returns the next element
void remove()      // removes the last element returned by the iterator
```

Ví dụ Iterator

```
List<String> lst = new ArrayList<String>();
lst.add("alpha");
lst.add("beta");
lst.add("charlie");

// Retrieve the Iterator associated with this List via the iterator() method
Iterator<String> iter = lst.iterator();
// Transverse thru this List via the Iterator
while (iter.hasNext()) {
   // Retrieve each element and process
   String str = iter.next();
   System.out.println(str);
}
```

2.2.Collection<E> interface

- Collection<E> interface:
  - Là interface gốc của mọi lớp trong JCF
  - Gồm các abstract methods :

```
// Basic Operations
int size()                      // Returns the number of elements of this Collection
void clear()                    // Removes all the elements of this Collection
boolean isEmpty()               // Returns true if there is no element in this Collection
boolean add(E element)          // Ensures that this Collection contains the given element
boolean remove(Object element)  // Removes the given element, if present
boolean contains(Object element)  // Returns true if this Collection contains the given element

// Bulk Operations with another Collection
boolean containsAll(Collection<?> c)       // Collection of any "unknown" object
boolean addAll(Collection<? extends E> c)  // Collection of E or its sub-types
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)

// Comparison - Objects that are equal shall have the same hashCode
boolean equals(Object o)
int hashCode()

// Array Operations
Object[] toArray()       // Convert to an Object array
<T> T[] toArray(T[] a)   // Convert to an array of the given type T
```

2.3.List<E>, Set<E>, Queue<E>

- List<E> :
  - Mô phỏng Resizable linear array
  - Truy cập bằng chỉ số index của mảng
  - Có thể chứa duplicate elements
  - Implementations: ArrayList, LinkedList, Vector & Stack.
- Set<E>:
  - Mô phỏng k/n tập hợp trong toán học
  - Không tồn tại duplicate elements
  - Implementations: HashSet, LinkedHashSet
- SortedSet<E>:
  - Mô phỏng tập có thứ tự và đã được sắp xếp.
  - Implementations: TreeSet

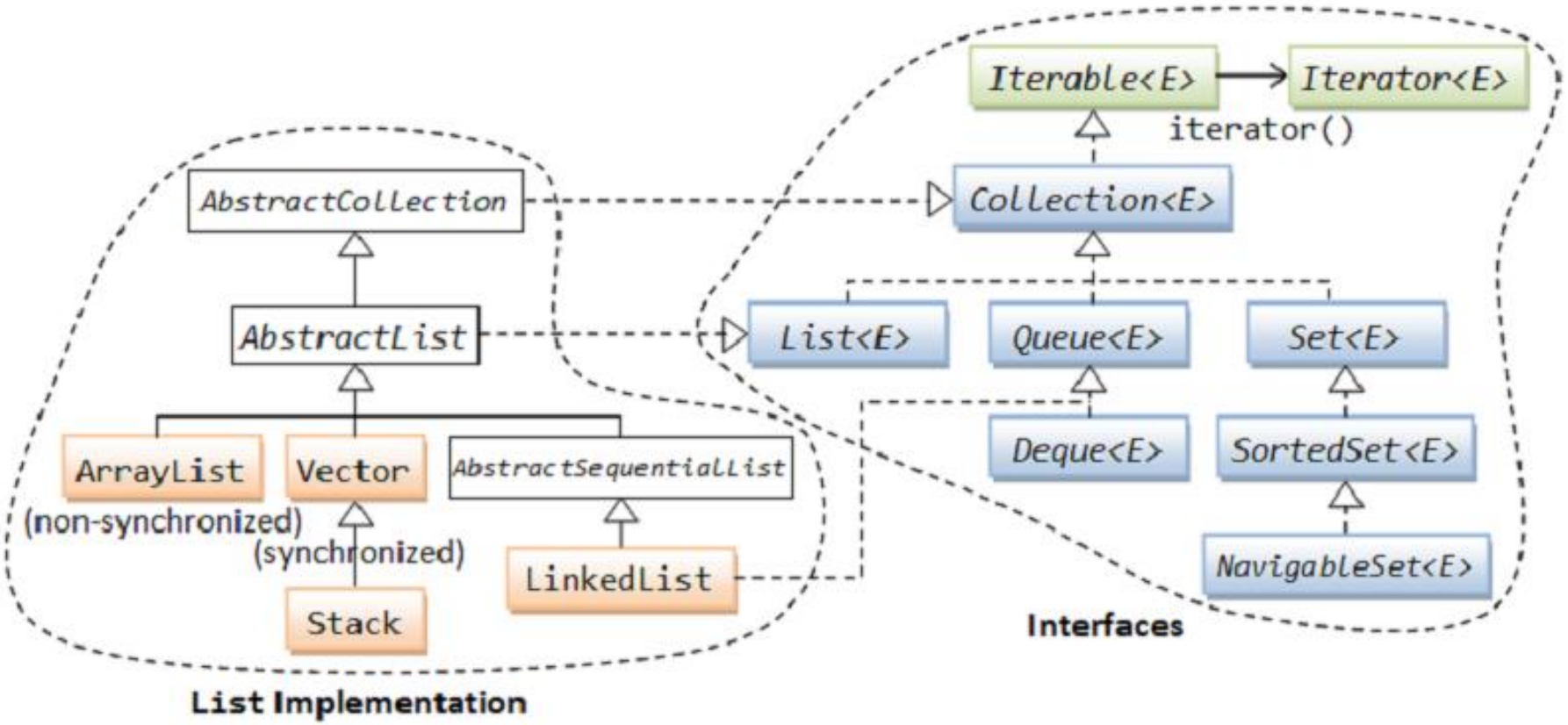
2.3.List<E>, Set<E>, Queue<E>

- Queue<E>:
  - Mô phỏng k/n hàng đợi FIFO trong toán học
  - Deque<E>:
    - Mô phỏng hàng đợi 2 đầu
  - Implementations:
    - PriorityQueue,
    - ArrayDeque,
    - LinkedList

2.4.Map<K,V>

- Mô phỏng một tập hợp các cặp Key-Value
- Lấy 2 đối số kiểu generics <K,V>
- Không tồn tại 2 key trùng nhau
- Implementations: HashMap, HashTable, LinkedHashMap
- SortedMap<K,V>:
  - Mô phỏng một map có thứ tự và đã được sắp xếp.

3. List<E> interface & implementations



3. List<E> interface & implementations

```
// Operations at a specified index position
void add(int index, E element)     // add
E set(int index, E element)        // replace
E get(int index)                   // retrieve without remove
E remove(int index)                // remove last retrieved
int indexOf(Object obj)
int lastIndexOf(Object obj)
// Operations on a range fromIndex (inclusive) toIndex (exclusive)
List<E> subList(int fromIndex, int toIndex)
......

// Operations inherited from Collection<E>
int size()
boolean isEmpty()
boolean add(E element)
boolean remove(Object obj)
boolean contains(Object obj)
void clear();
......
```

3. List<E> : ArrayList vs Vector

ArrayList<E>:

- Không đồng bộ
- Nên sử dụng ArrayList nếu ko liên quan tới vấn đề đa luồng

Vector<E>

- Có đồng bộ
- Dùng với đa luồng

4. Sắp xếp và tìm kiếm
4.1 Sắp xếp

- Để sắp xếp được các phần tử thì cần tiêu chí sắp thứ tự
- Có 2 cách để sắp thứ tự:
  1. Cài đặt interface java.lang.Comparable
     - Cài đè phương thức compareTo() để xác định thứ tự giữa hai đối tượng
  2. Tạo đối tượng đặc biệt java.lang.Comparator với phương thức compare() để xác định thứ tự giữa 2 đối tượng

Cách 1: Sử dụng interface java.lang.Comparable

```
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class TestComparable {
   public static void main(String[] args) {
      // Sort and search an "array" of Strings
      String[] array = {"Hello", "hello", "Hi", "HI"};

      // Use the Comparable defined in the String class
      Arrays.sort(array);
      System.out.println(Arrays.toString(array));  // [HI, Hello, Hi, hello]

      // Try binary search - the array must be sorted
      System.out.println(Arrays.binarySearch(array, "Hello")); // 1
      System.out.println(Arrays.binarySearch(array, "HELLO")); // -1 (insertion at index 0)

      // Sort and search a "List" of Integers
      List<Integer> lst = new ArrayList<Integer>();
      lst.add(22);  // auto-box
      lst.add(11);
      lst.add(44);
      lst.add(33);
      Collections.sort(lst);    // Use the Comparable of Integer class
      System.out.println(lst);  // [11, 22, 33, 44]
      System.out.println(Collections.binarySearch(lst, 22)); // 1
      System.out.println(Collections.binarySearch(lst, 35)); // -4 (insertion at index 3)
   }
}
```

Cách 2: Sử dụng đối tượng java.lang.Comparator

```
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class TestComparator {

   // Define a Comparator<String> to order strings in case-insensitive manner
   public static class StringComparator implements Comparator<String> {
      @Override
      public int compare(String s1, String s2) {
         return s1.compareToIgnoreCase(s2);
      }
   }

   // Define a Comparator<Integer> to order Integers based on the least significant digit
   public static class IntegerComparator implements Comparator<Integer> {

       @Override
       public int compare(Integer s1, Integer s2) {
          return s1%10 - s2%10;
       }
    }
```

Cách 2: Sử dụng đối tượng java.lang.Comparator

```
   public static void main(String[] args) {
      // Use a customized Comparator for Strings
      Comparator<String> compStr = new StringComparator();

      // Sort and search an "array" of Strings
      String[] array = {"Hello", "Hi", "HI", "hello"};
      Arrays.sort(array, compStr);
      System.out.println(Arrays.toString(array));  // [Hello, hello, Hi, HI]
      System.out.println(Arrays.binarySearch(array, "Hello", compStr)); // 1
      System.out.println(Arrays.binarySearch(array, "HELLO", compStr)); // 1 (case-insensitive)

      // Use a customized Comparator for Integers
      Comparator<Integer> compInt = new IntegerComparator();

      // Sort and search a "List" of Integers
      List<Integer> lst = new ArrayList<Integer>();
      lst.add(42);  // auto-box
      lst.add(21);
      lst.add(34);
      lst.add(13);
      Collections.sort(lst, compInt);
      System.out.println(lst);  // [21, 42, 13, 34]
      System.out.println(Collections.binarySearch(lst, 22, compInt)); // 1
      System.out.println(Collections.binarySearch(lst, 35, compInt)); // -5 (insertion at index 4)
   }
}
```

4. Sắp xếp và tìm kiếm
4.2 Tìm kiếm

- Giải thuật tìm kiếm: tìm kiếm nhị phân
  - Arrays.binarySearch()