CMC UNIVERSITY

Java Iterative data types

PhD. Ngo Hoang Huy

Monday, July 31, 2023

CMC UNIVERSITY

Objectives

Collections:

List: ArrayList,

Set: HashSet, TreeSet.

Map: HashMap, TreeMap

Iterator with Collections:

CMC UNIVERSITY

Collections

- The Java 2 platform includes a new collections framework.
- A collection is an object that represents a group of objects.
- The Collections Framework is a unified architecture for representing and manipulating collections.

CMC UNIVERSITY

The Collections Framework

- Reducing programming effort: It provides pre-implemented data structures and algorithms, saving developers from having to write them from scratch.
- Improving performance: The framework includes high-performance implementations of data structures and algorithms, enhancing the overall efficiency of the code.
- Ensuring interoperability: It establishes a common language for passing collections between unrelated APIs, promoting compatibility and smooth data exchange.
- Simplifying API learning: With a standardized collection framework, developers don't need to learn multiple ad hoc collection APIs, making the learning process more straightforward.
- Streamlining API design and implementation: By eliminating the need to produce ad hoc collection APIs, the framework reduces the effort required to design and implement APIs.
- Encouraging software reuse: The provision of a standard interface for collections and algorithms facilitates software reuse, allowing components to be easily integrated and reused in different projects.

Collection Interfaces

- java.lang.**Iterable**<T>
  - java.util.**Collection**<E>
    - java.util.**List**<E>
    - java.util.**Queue**<E>
      - java.util.**Deque**<E>
    - java.util.**Set**<E>
      - java.util.**SortedSet**<E>
        - java.util.**NavigableSet**<E>
- java.util.**Map**<K,V>
  - java.util.**SortedMap**<K,V>
    - java.util.**NavigableMap**<K,V>

Methods declared in these interfaces can work on a list containing elements which belong to arbitrary type. T: type, E: Element, K: Key, V: Value

Details of this will be introduced in the topic Generic

3 types of group:
List can contain duplicate elements
Set can contain distinct elements only
Map :can store key-value pairs for efficient searching based on the key

Queue, Deque contains methods of restricted list.

Common methods on group are: Add, Remove, Search, Clear,…

CMC UNIVERSITY

Collection Interfaces

Central Interfaces

- java.util.**Collection**<E>
  - java.util.**List**<E>
  - java.util.**Queue**<E>
    - java.util.**Deque**<E>
  - java.util.**Set**<E>
    - java.util.**SortedSet**<E>
      - java.util.**NavigableSet**<E>
  - java.util.**Map**<K,V>
    - java.util.**SortedMap**<K,V>
      - java.util.**NavigableMap**<K,V>

Common Used Classes

- java.util.**ArrayList**<E>
- java.util.**Vector**<E>

Dynamic array
Use index to access an element.

- java.util.**HashSet**<E>
- java.util.**TreeSet**<E>

Specific structure/tree
Use iterator to access elements

java.lang.Comparable interface

- java.util.**HashMap**<K,V>
- java.util.**TreeMap**<K,V>

keySet()
values()

A TreeSet stores elements in ascending order using natural ordering for numbers and lexicographic ordering for strings. To use TreeSet with custom objects, you need to implement the compareTo(Object) method from the Comparable interface..

CMC UNIVERSITY

Lists

a List is a data structure that maintains its elements in the order in which they were added. Each element in the List is assigned an index, starting from 0 for the first element, 1 for the second, and so on.

Common methods associated with a List include:

void add(int index, Object x): This method inserts the specified element (Object x) at the specified index in the List. Any existing elements at or after the index are shifted to accommodate the new element.

Object get(int index): This method retrieves the element at the specified index from the List. It returns the element at the given index without removing it from the List.

int indexOf(Object x): This method returns the index of the first occurrence of the specified element (Object x) in the List. If the element is not found in the List, it returns -1.

Object remove(int index): This method removes the element at the specified index from the List and returns the removed element. Any elements after the removed element are shifted to fill the gap.

CMC UNIVERSITY

Classes Implementing the interface List

AbstractList:

AbstractList is an abstract class that provides a partial implementation of the List interface. It extends the AbstractCollection class and implements most of the methods defined in the List interface.

ArrayList:

ArrayList is a dynamic array-based implementation of the List interface. It uses an underlying array to store its elements and automatically resizes the array when needed to accommodate more elements. ArrayList provides fast access and retrieval of elements using their index, making it an efficient choice for random access operations. However, inserting or deleting elements at the beginning or middle of the ArrayList can be relatively slow, as it requires shifting elements to accommodate the change in size.

LinkedList:

LinkedList is a doubly-linked list implementation of the List interface. Unlike ArrayList, LinkedList does not use an array to store elements. Instead, each element is stored in a node that contains references to the previous and next nodes in the list. This structure allows for efficient insertion and deletion of elements at any position in the list, making
LinkedList suitable for implementing stacks, queues, and double-ended queues (deque). However, accessing elements by index in a LinkedList is slower compared to ArrayList, as it requires traversing the list from the beginning or end.

CMC UNIVERSITY

ArrayList

This is a simpler syntax to loop through elements of an array or a Collection. It is used to sequentially iterate through elements and cannot modify the structure of the array or Collection during iteration.

```
import java.util.ArrayList;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> names;
        names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        // Using a for loop
        for (int i = 0; i < names.size(); i++) {
            String name = names.get(i);
            System.out.println(name);
        }
    }
```

CMC UNIVERSITY

Hash Table

In arrays, elements are stored in contiguous memory blocks, which makes linear search slow. Binary search is an improvement for faster search operations.

On the other hand, in a hash table, elements can be stored in different memory blocks. The index of an element is determined by a hash function. This allows for very fast add and search operations, with a time complexity of O(1).

CMC UNIVERSITY

Maps

Java's two most important Map classes:

HashMap (mapping keys are unpredictable order – hash table is used, hash function is pre-defined in the Java Library).

TreeMap (mapping keys are natural order)-> all keys must implement Comparable (a tree is used to store elements).

```
import java.util.HashMap;
public class HashMapExample {
    public static void main(String[] args) {
        // Create a new HashMap
        HashMap<String, Integer> ageMap;
        ageMap = new HashMap<>();
        // Add key-value pairs to the HashMap
        ageMap.put("John", 25);
        ageMap.put("Emily", 30);
        ageMap.put("Michael", 22);
        ageMap.put("Sophia", 28);
        // Access and print the value for a specific key
        System.out.println("John's age: " + ageMap.get("John"));
        // Check if a key exists in the HashMap
        if (ageMap.containsKey("Sophia"))
            System.out.println("Sophia's age: " + ageMap.get("Sophia"));
        else
            System.out.println("Sophia's age not found.");

    }//main

}//Main
```

CMC UNIVERSITY

Iterator

It is an interface in the java.util package, providing methods to traverse elements in a collection like ArrayList, LinkedList, HashSet, and TreeMap. An Iterator is used to sequentially iterate through elements and cannot modify the structure of the collection during iteration.

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
    public static void main(String[] args) {
        ArrayList<String> names;
        names= new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            String name = iterator.next();
            System.out.println(name);
        }
    }
}
```

CMC UNIVERSITY

Iterator with List

List and Iterator are two distinct concepts in the Java Collection Framework, but they are often used together to iterate through the elements of a list.

In this example, we use an Iterator to traverse through the elements of the List (list of names) and print them to the screen.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
publc class Main {
    public static void main(String[] args) {
        List<String> names;
        names = new ArrayList<>();
        names.add("Alice");
        names.add("Bob");
        names.add("Charlie");
        // Sử dụng Iterator để duyệt qua các phần tử trong List
        Iterator<String> iterator = names.iterator();
        while (iterator.hasNext()) {
            String name = iterator.next();
            System.out.println(name);
        }
    }
}
```

CMC UNIVERSITY

Iterator with Hash Table

In this example, we use Iterator and entrySet() to obtain the set of key-value pairs in the hash table. Then, we use a while loop to iterate through the elements in the set and print the corresponding key and value of each element

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
public class Main {
    // Hàm hashFunction để chuyển đổi chuỗi key thành giá trị hash
    public static int hashFunction(String key) {
        int hash = 0;
        for (int i = 0; i < key.length(); i++)
            hash = (hash + key.charAt(i)) % 100;
        return hash;
    }//hashFunction
    public static void main(String[] args) {
        // Tạo một hash table sử dụng HashMap trong Java
        HashMap<String, Integer> hashTable = new HashMap<>();
        // Thêm các phần tử vào hash table
        hashTable.put("apple", 10);
        hashTable.put("banana", 20);
        hashTable.put("orange", 15);
        hashTable.put("grape", 25);
```

Iterator with Hash Table

```
        // Lấy giá trị từ hash table bằng key
        String key = "apple";
        int value = hashTable.get(key);
        System.out.println("Value of " + key + ": " + value);
        // Xóa một phần tử khỏi hash table
        hashTable.remove("orange");
        // Kiểm tra xem key có tồn tại trong hash table không
        String checkKey = "orange";
        boolean containsKey = hashTable.containsKey(checkKey);
        System.out.println("Key " + checkKey + " exists in the hash table: " + containsKey);
        // Duyệt qua tất cả các phần tử trong hash table bằng vòng lặp thông thường
        Iterator<Map.Entry<String, Integer>> iterator = hashTable.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> entry = iterator.next();
            String k = entry.getKey();
            int v = entry.getValue();
            System.out.println("Key: " + k + ", Value: " + v);
        }//while
    }//main
}// Main
```

CMC UNIVERSITY

Iterator with Hash Map

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
public class Main {
    public static void main(String[] args) {
        // Create a new HashMap
        HashMap<String, Integer> ageMap = new HashMap<>();
        // Add key-value pairs to the HashMap
        ageMap.put("John", 25);
        ageMap.put("Emily", 30);
        ageMap.put("Michael", 22);
        ageMap.put("Sophia", 28);
        // Create an iterator for the HashMap's entry set
        Iterator<Map.Entry<String, Integer>> iterator ;
        iterator= ageMap.entrySet().iterator();
        // Iterate through the HashMap using the iterator
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> entry = iterator.next();
            String name = entry.getKey();
            int age = entry.getValue();
            System.out.println(name + "'s age: " + age);
        }
    }
}
```

CMC UNIVERSITY

THANK YOU