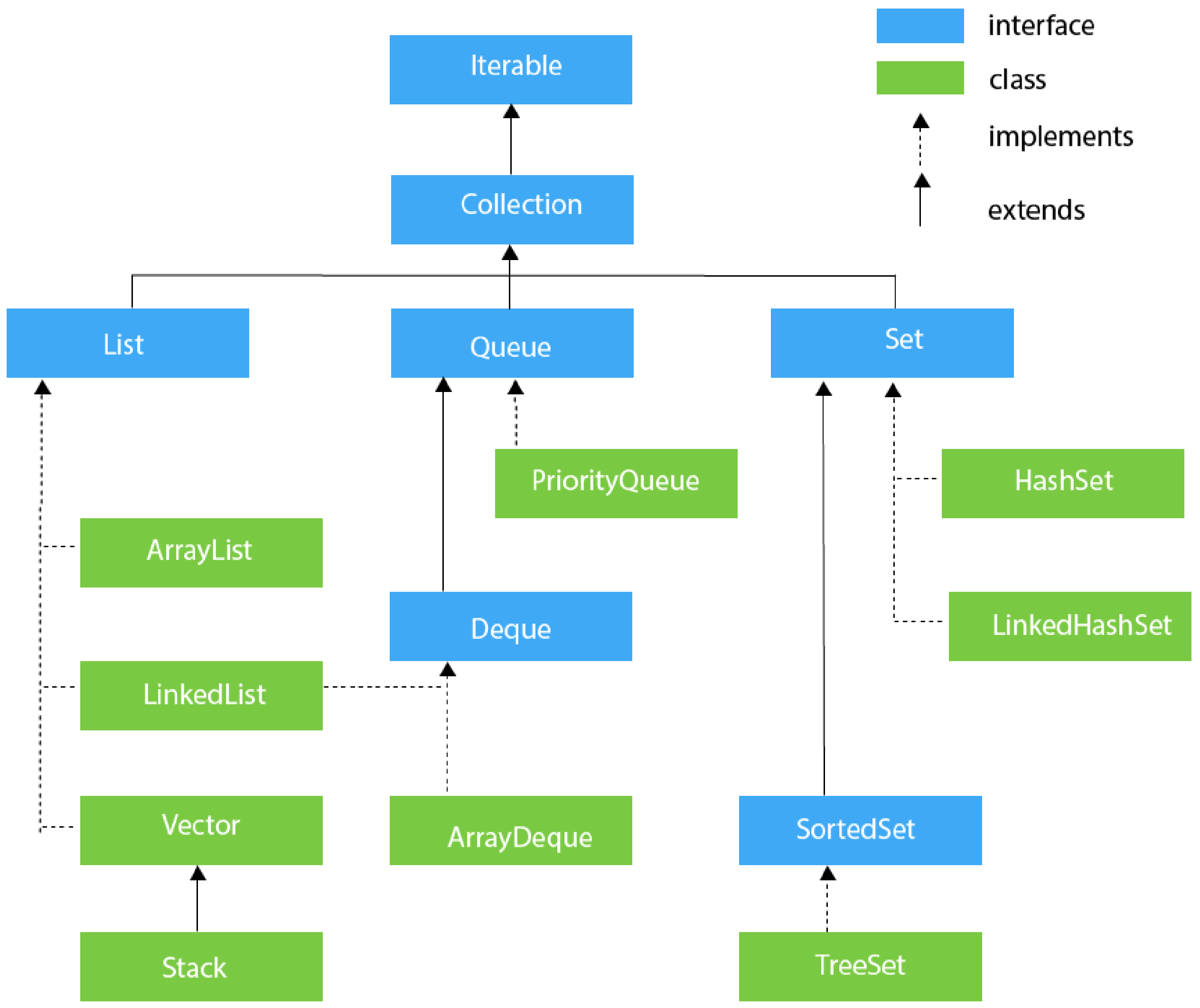
Collection in java

Contents

Overview 3
Interface 3
    Iterator interface 3
    Collection interface 4
    List Interface 4
    Queue Interface 5
    Deque Interface 5
    Set Interface 5
    SortedSet Interface 5
Compare List with List 5
    For immutable use equal 5
    Operator == 5
    Equal Method: 5
    removeAll method 5
    retainAll method 5
    Stream filter 6
Sort with List 6
    For Class Immutable 6
    For class customize (Sefl-defination): 6
        Comparator 6
        Comparable : 6
Class Implements 7
    ArrayList 7
    LinkedList 8
    Vector 8
    Stack 8
    HashSet 9
        Example working Hasing: 9
    LinkedHashSet 10
    TreeSet 11

Compare HashSet, LinkedHashSet, TreeSort 12
PriorityQueue 12
ArrayQueue 12
Reference 12
Github 12

Overview



Interface

Iterator interface

It include methods below:

public boolean hasNext() return true if interator has more element otherwise it return false.

public Object next() return object in element current of interator

public void remove() delete element current of interator

Collection interface

Methods of Collection Interface

1 public boolean **add**(E e) : It use to insert an element in this collection

2 public boolean **addAll**(Collection<? extends E> c) : it use to insert a list collection into collection.

3 public boolean **remove**(Object element): remove an element in collection by object

4 public boolean **removeAll**(Collection<?> c) : remove all element specified collection from collection others.

5 default boolean **removeIf**(Predicate<? super E> filter): remove all element if satisfy th specified predicate.

6 public void **clear**(): remove all element of Collection

7 public boolean **retainAll**(Collection<?> c): it retain all Collection param and remove the element others.

8 public int **size**(): it return length of Collection start 0

9 public boolean **contains**(Object element): return true if element param has in Collection, otherwise return false

10 public boolean **containsAll**(Collection<?> c): return true if Collection has in Collection this, otherwise return false.

11 public Iterator iterator(): return iterator to get or delete element in Collection

12 public Object[] **toArray**() & **toArray**(T[] a): convert Collection to array. Not for reference type, if primitive type use array normal (https://stackoverflow.com/questions/9572795/convert-list-to-array-in-java)

```
Java 8 String[] strings = list.stream().toArray(String[]::new);
Java 11  String[] strings = list.toArray(String[]::new);
```

13 public boolean **isEmpty**() : return true if collection empty

14 default Stream<E> **parallelStream**(): get a paralleStream

15 default Stream<E> **stream**() : get sequetial stream with Collection as its source

16 default Spliterator<E> **spliterator**() : It supports Parallel Programming. It generates a Spliterator over the specified elements in the collection.

17 public boolean **equals**(Object element): It matches two collections.

18 public int **hashCode**(): It returns the hash code number of the collection.

List Interface

List interface is child interface of Collection interface. Element can store the ordered collection of objects. It can duplicate values.

List interface is implemented by classes **ArrayList**, **LinkedList**, **Vector**, and **Stack**.

Queue Interface

Queue interface maintains the first-in-first-out order. There are various classes like **PriorityQueue**, **Deque** and **ArrayDeque** which implements the Queue interface.

Deque Interface

Deque interface extends the Queue interface. In deque, we can remove and add the elements from both the side. Deque stands for a double-ends queue which enables us to perform the operations at both the ends. **ArrayDeque** is implement it.

Set Interface

It extends the Collection interface. It not allow duplicate element. It represent unordered set. We can store null only item in Set. Set has implements **HashSet**, **LinkedHashSet** and **TreeSet**.

SortedSet Interface

It is alternative Set interface , it was sorted all element in Set. **TreeSet** is implemented of SortedSet.

Compare List with List

For immutable use equal list we simple use list.equal list it return true if content same.

```
List<Integer> listOne = new ArrayList<>(Arrays.asList(1, 4, 6, 3, 66, 3, 33));
List<Integer> listTwo = new ArrayList<>(Arrays.asList(1, 4, 6, 3, 66, 3, 33));
System.out.println("WrapperClass listOne.equals(listTwo): " + listOne.equals(listTwo));
```

We can compare list with list by various ways:

Operator == : true if two list similar memory (oparator =) example below.

List a = list b

return a == b return true.

Equal Method: True if element is immutable class, if class is customize(self defination) we must override Equal & hascode method of class element, if not return false all case.

removeAll method then check size: It method dependent method equal check delete so we must override Equal & hashcode method of class element.

retainAll method then compare size: It method dependent method equal check delete defference so we must override Equal & hashcode method of class element.

Stream filter ListCheckContain Collect: Use predicate of inside method fillter of Stream it indepent of method equal & hashcode of class element so we must override them.

```
View Class collectionsinJava.listinterface.comparelist.CompareToArrayLists
```

Sort with List

For Class Immutable is wrapper class Use Collections.sort normal not use comparator

```
static void sortClassImmutable() {
    List<Integer> listOne = new ArrayList<>(Arrays.asList(1, 4, 6, 3, 66, 3, 33));
    System.out.println("List original : " + listOne);
    Collections.sort(listOne);
    System.out.println("List sorted ascending : " + listOne);
    Collections.sort(listOne, Comparator.reverseOrder());
    System.out.println("List sorted descending : " + listOne);
}
```

For class customize (Sefl-defination): Two sort with list : Use Comparator and Comparable

Comparator : Create Comparator<Class> then use Collections.sort(list, comparator) sort on self-list current.

```
// create Comparator for class Vehicle
static Comparator<Vehicle> ascendingVehicle = (o1, o2) -> {
    if (o1.getId() > o2.getId()) { // max last
        return 1;
    }
    return -1;
};
```

```
// comparator for class descending vehicle
static Comparator<Vehicle> descendingVehicle = Collections.reverseOrder(ascendingVehicle);
```

Use it

```
System.out.println("List original : " + listOriginal);
Collections.sort(listOriginal, ascendingVehicle);
System.out.println("List sorted ascending : " + listOriginal);
Collections.sort(listOriginal, descendingVehicle);
System.out.println("List sorted descending : " + listOriginal);
```

Comparable : When sort use Comparable we implement Comparable and Override method compareTo(Class) in class element

```
class Vehicle2  implements  Comparable<Vehicle2>{

    private Integer id;
```

```
   private String name;
   private Address2 address;

  @Override
  public int compareTo(Vehicle2 o) {
   if (this.getId() > o.getId()) { // max last
      return 1;
   }
   return -1;
  }

   // getter, setter , constructor
}
```

Use It.

```
Collections.sort(listOriginal);
Collections.sort(listOriginal, Comparator.reverseOrder());
```

```
View: collectionsinJava.listinterface.sorts
```

Class Implements

ArrayList

It uses a dynamic to store in duplicate element of the difference data types. Arraylist maintains ordered insertation in list and non-sychonized. Random access because it works at the index basic. Compare with LinkedList it access to quickly then but manipulation slower as remove or add element because Linkedlist not shfting when computation.

```
subListDemo
```

subList return reference of list current

can Remove element reference from subList of List Current

```
retainAll
```

retainAll element exist join list current and list argument list current 1 2 3

list argument 2 4

result list current is: 2

LinkedList

Java LinkedList class can contain duplicate elements.

Java LinkedList class is non synchronized.

It uses a doubly linked list internally to store the elements. (sử dụng danh sách liên kết).

In Java LinkedList class, manipulation is fast because no shifting needs to occur.

Methods:

Offer method insert element.

Peek method retrive element. Return null if list empty.

Poll method retrive element and remove. Return null if list empty.

Push: push an element into first list as stack.

Pop: pop an element first and remove it.

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList.

However, It is synchronized and contains many methods that are not the part of Collection framework.

It is recommended to use the Vector class in the thread-safe implementation only.

If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.

Example: 2 thread similar access data of vector but in time only thread manupulation on vector.

Stack

It work Last-in first out. It extend Vector, it class synchronized . it has method fundemantal :

Push, pop, peek get top element without remove element…

HashSet

HashSet contains unique elements only. Hashset stores element by using a mechanism called Hasing. Hashset is non sychronized. Hashset dose **not maintains** order insertion. Hashset best for approach for seach operations. The initial default capacity of HashSet is 16, and the load factor is 0.75. HashSet allows null value.

It has included methods as add, clear, clone (shaddow copy), contains, iterator,…

Example working Hasing:

When work with mechanism Hasing we must notice :

When override equal and hashcode, but only use field is Identiy of class example : id …

If we use all field, Can you push set error.

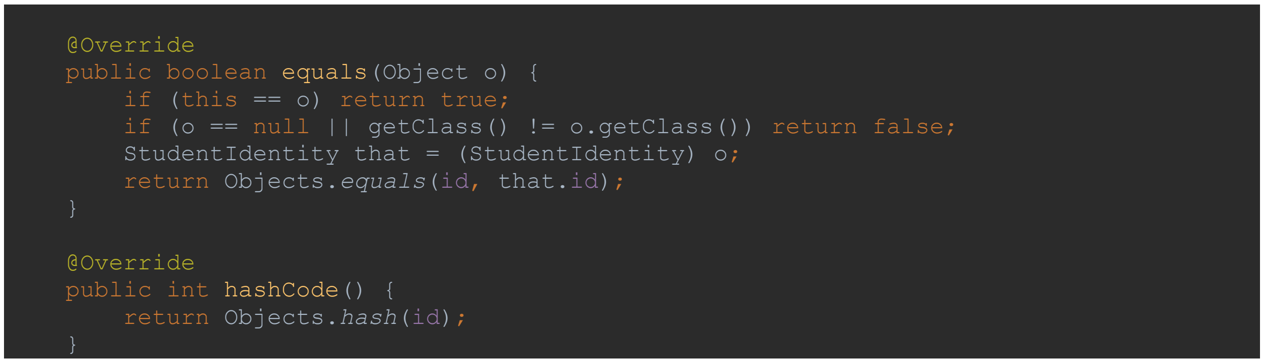
Ex: We has two class

Student use all field in class to override equal and hashcode

StudentIdentity only use field id for function equal and hascode

```java
class Student {
    private Integer id;
    private String name;
    private Address address;

    //case full item of class
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Student student = (Student) o;
        return Objects.equals(id, student.id) && Objects.equals(name, student.name) &&
Objects.equals(address, student.address);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name, address);
    }
```

```java
class StudentIdentity {

    private Integer id;
    private String name;
    private Address address;
```



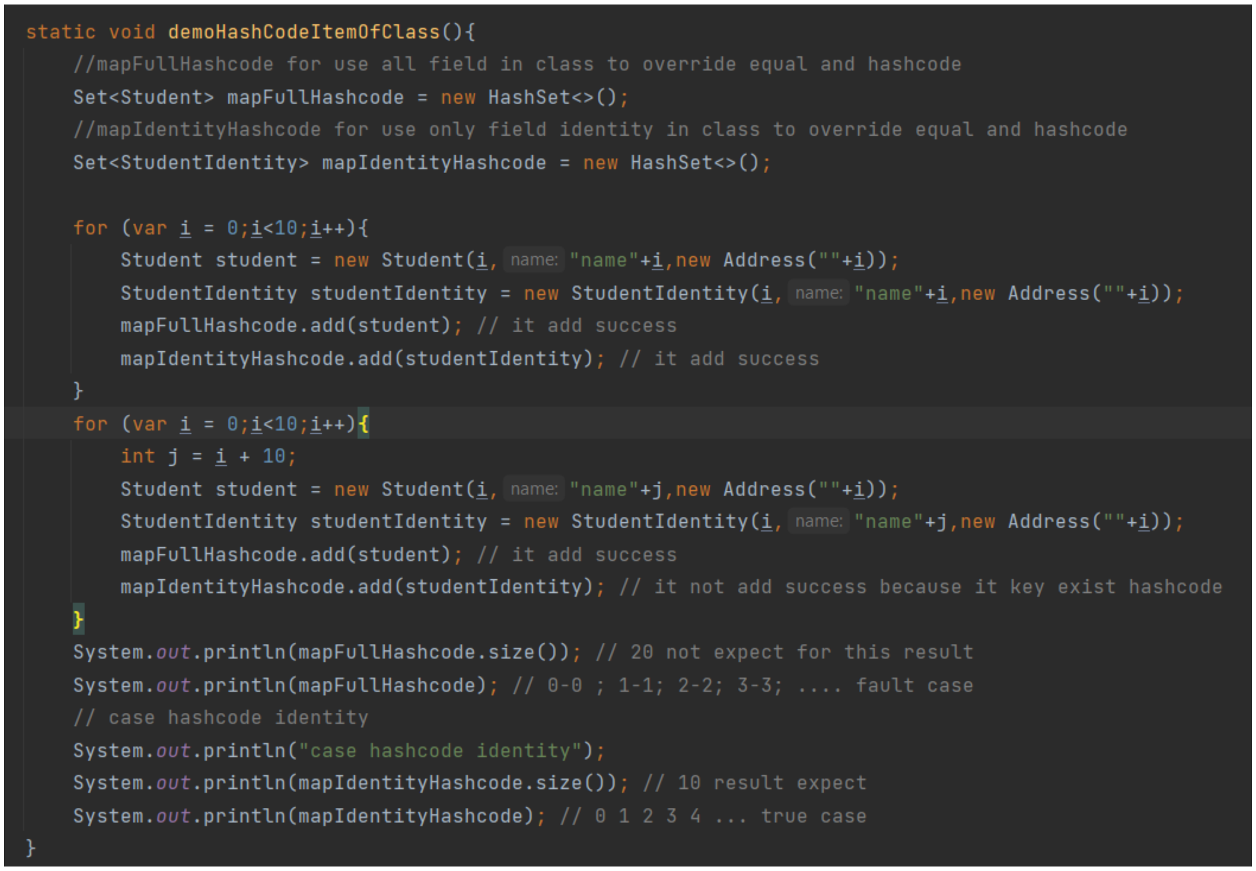
```java
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        StudentIdentity that = (StudentIdentity) o;
        return Objects.equals(id, that.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
```

Use

In loop first, the both map is push element into inside.

In loop second, create element same identity but difference field name with element in loop first.

only mapFullHashcode push into but mapIdentityHashcode not push because element hash exist hashcode in map.



```java
static void demoHashCodeItemOfClass(){
    //mapFullHashcode for use all field in class to override equal and hashcode
    Set<Student> mapFullHashcode = new HashSet<>();
    //mapIdentityHashcode for use only field identity in class to override equal and hashcode
    Set<StudentIdentity> mapIdentityHashcode = new HashSet<>();

    for (var i = 0;i<10;i++){
        Student student = new Student(i, name: "name"+i,new Address(""+i));
        StudentIdentity studentIdentity = new StudentIdentity(i, name: "name"+i,new Address(""+i));
        mapFullHashcode.add(student); // it add success
        mapIdentityHashcode.add(studentIdentity); // it add success
    }
    for (var i = 0;i<10;i++){
        int j = i + 10;
        Student student = new Student(i, name: "name"+j,new Address(""+i));
        StudentIdentity studentIdentity = new StudentIdentity(i, name: "name"+j,new Address(""+i));
        mapFullHashcode.add(student); // it add success
        mapIdentityHashcode.add(studentIdentity); // it not add success because it key exist hashcode
    }
    System.out.println(mapFullHashcode.size()); // 20 not expect for this result
    System.out.println(mapFullHashcode); // 0-0 ; 1-1; 2-2; 3-3; .... fault case
    // case hashcode identity
    System.out.println("case hashcode identity");
    System.out.println(mapIdentityHashcode.size()); // 10 result expect
    System.out.println(mapIdentityHashcode); // 0 1 2 3 4 ... true case
}
```

LinkedHashSet

LinkedHashSet class contant unique element only like HashSet. LinkedHashSet class provides all operation set and **permit null** element. Java LinkedHashSet class is non synchronized. Java LinkedHashSet **class maintains insertion order**.

TreeSet

TreeSet implement Set interface that uses tree for storage. The object in TreeSet is sorted by ascending order. Java TreeSet class contains unique elements only like HashSet. Java TreeSet class **doesn't allow null element**. Java TreeSet class is non synchronized. Java TreeSet class maintains ascending order.

Classes wapper is sorted not implement comparable, default has implement Comparable. The Class use-defined must implement comparable inteface.

Methods:

```
System.out.println(treeSet); // [a, b, e, i, n, o, q, t, u]
//get descending order
System.out.println(treeSet.descendingSet()); // [u, t, q, o, n, i, e, b, a]
//ceiling bigger closet (lớn gần nhất) hoặc bằng nếu ko có return null;
System.out.println(treeSet.ceiling("f")); // i

//get smaller closet (nhỏ gần nhất) hoặc bằng nếu ko có return null;
System.out.println(treeSet.floor("d")); // b

//get head to element , if element not found get all smaller
System.out.println(treeSet.headSet("d", true)); //[a, b]

//get element to tail , if element not found get all bigger
System.out.println(treeSet.tailSet("m", true)); //[n, o, q, t, u]

//same ceilling
System.out.println(treeSet.higher("f")); // i

// get first (min) and remove
System.out.println(treeSet.pollFirst()); // a

// get first (max) and remove
System.out.println(treeSet.pollLast()); // u

System.out.println(treeSet); //[b, e, i, n, o, q, t]

// get first (min) and without remove
System.out.println(treeSet.first()); // b

// get first (max) and without remove
System.out.println(treeSet.last()); // t

System.out.println(treeSet); // [b, e, i, n, o, q, t]
```

Demo Sort in TreeSort : we can implement Comparable in class object user-defined

```
class Student implements Comparable<Student> {
    private Integer id;
    private String name;
    private Address address;

    @Override
    public int compareTo(Student o) {
        if(this.id > o.getId()){
            return  1;
        }
        return -1;
    }
```
Then create TreeSort and add normal object

```
treeSetStudent.add(student);
```

Compare HashSet, LinkedHashSet, TreeSort

Similar: non synchronized, unique element, use hasing mechanism.

Defference:

HashSet: not maintain order insertation unordered, allow null

LinkedHashSet: it uses a doubly linked list to maintain the insertion order, allow null.

TreeSet: Not allow null, Sorted ascending default when add object.

PriorityQueue

PriorityQueue doesn't permit null. We can't create PriorityQueue of Objects that are non-comparable. Priotiry by order of comparable in class. It print not following by order but when it work following order, example queue.remove() it remove head first (min value if comparable ascending).

ArrayQueue

Work with First-in-First-out unlike Priority must order class element. Unlike Queue, we can add or remove elements from both sides. Null elements are not allowed in the ArrayDeque. ArrayDeque is not thread safe, in the absence of external synchronization. Order view print same Order work.

Reference Doc: https://www.javatpoint.com/java-tutorial

Github:
https://github.com/nguyenthinhit996/sharefullcode/tree/master/Learn%20Java/Java%20Basic/ListCollection/src