什么是类集框架

- 类集框架是一组类和接口。位于 java.util 包中。
- 主要用于存储和管理对象。
- 主要分为三类——集合(Set)、列表(List)和映射(Map)。

类集框架种类

1. 集合(Set)

   集合中的对象不安特定的方式排序，并且没有重复对象。

2. 列表(List)

   集合中的对象按照索引位置排序，可以有重复对象。

3. 映射(Map)

   集合众的每一个元素包含一个键对象和一个值对象，键不可以重复，值可以重复。

类集框架的基础结构

Iterable

```
public interface Iterable<T>
参数类型：
通过返回的迭代器 T-元素类型
```

实现该接口允许一个对象成为“每个循环”语句的目标。

该接口里面有一个方法 Iterator 能返回一个迭代器

```
Iterator<T> iterator()
返回类型 T 元素的迭代器。
//Iterator 详细说明
public interface Iterator<E>
参数
E 元素迭代器返回的类型
```

该接口下几个常用的方法：

1. hasNext()
   boolean hasNext()
   返回 true 则说明集合中下一个元素存有内容。
2. next()
   E next()
   返回迭代器中的下一个元素

Collection

```
public interface Collection<E> extends Iterable<E>
参数
E 集合中元素的类型
```

该接口下几个重要的方法：

1. add(E e)
   boolean add(E e)
   参数
   e 集合中保存元素的类型
   向集合中增加元素。
2. remove(Object o)
   boolean remove(Object o)
   参数
   o 要从集合中删除的元素
   从集合中删除一个对象的引用
3. clear()
   void clear()
   从这个集合中移除所有的元素
4. size()
   int size()
   返回此集合中的迭代器
5. iterator()
   iterator iterator()
   返回此集合中的迭代器
6. isEmpty()
   boolean isEmpty()
   判断该集合是否包含元素，返回 true 则该集合内无元素。

Set

```
public interface Set<E> extends Collection<E>
```

该接口继承 Collection 接口，故 Collection 接口中的方法在该接口中都存在。

实现该接口的常用类有：HashSet

Set 常用方法：
1.

HashSet

```
public class HashSet<E> extends AbstractSet<E> implements Set<E>,Cloneable,Serializable
```

HashSet 类构造方法：

1. HashSet()

public HashSet()
构造一个新的，空的设置；支持 HashSet 实例具有默认初始容量(16)和负载因子(0.75)。

2. HashSet(Collection c)
   public HashSet(Collection c)
   参数
   c的元素都被放置在该集合中
3. HashSet(int initialCapacity,float loadFactor)
   public HashSet(int initialCapacity,float loadFactor)
   参数
   initialCapacity 哈希映射的初始容量
   loadFactor 哈希 Map 加载因子
4. HashSet(int initialCapacity)
   public HashSet(int initialCapacity)
   参数
   initialCapacity 哈希映射的初始容量

例如：

```
import java.util.*;
class Test{
  public static void main(String[] args) {
    Set<String> hs=new HashSet<String>();//对象上转型
    hs.add("qwe");//增加元素
    hs.add("asd");
    hs.add("zxc");
    System.out.println(hs.isEmpty());//判断是否有元素
    Iterator<String> it=hs.iterator();//获得迭代器
    //由于集合不按特定的方式排序，所以我们使用迭代器输出集合里面存的对象
    while(true){
      if (it.hasNext()) {
        System.out.println(it.next());
      }else{
        break;
      }
    }
    System.out.println(hs.size());//输出集合大小
  }
}
执行结果:
false
asd
zxc
qwe
3
```

List

```
public interface List<E> extends Collection<E>
```

List 常用子类：ArrayList

List 常用方法：

1. add(int index,E element)
   void add(int index,E element)
   将 element 元素插入到 index 位置。
2. remove(int index)
   E remove(int index)

移除指定位置的元素

3. remove(Object o)
   boolean remove(Object o)
   移除指定元素的第一个引用。
4. contains(Object o)
   void contains(Object o)
   返回 true 表示该列表包含 o 元素。
5. get(int index)
   E get(int index)
   返回此列表中 index 指定位置的元素。
6. iterator()
   Iterator iterator()
   在该列表中的元素上返回一个正确的顺序。
7. listIterator()
   ListIterator listIterator()
   返回列表元素的列表迭代器(适当顺序)。
8. listIterator(int index)
   ListIterator listIterator(int index)
   从 index 位置返回列表元素的列表迭代器(适当顺序)。
9. set(int index,E element)
   E set(int index,E element)
   用指定元素替代指定位置的元素。

ArrayList

```
public class ArrayList<E> extends AbstractList<E> implements List<E>,RandomAccess,Cloneable,Serializable
```

ArrayList 的构造方法：

1. ArrayList()
   public ArrayList()
   构造一个初始为十的空列表。
2. ArrayList(Collection c)
   public ArrayList(Collection c)
   构造一个包含指定集合的元素的列表，它们在迭代器返回的顺序中返回。
3. ArrayList(int initialCapacity)
   public ArrayList(int initialCapacity)
   用指定的初始容量构造一个空列表。

例如：

```
import java.util.*;
class Test{
  public static void main(String[] args) {
    ArrayList<String> al=new ArrayList<String>();
    List<String> list=al;
    list.add(0,"qwe");
    list.add(1,"asd");
    list.add(2,"zxc");
    System.out.println(list.get(2));
    list.set(2,"vbn");
    for(int i=0;i<list.size();i++){
      System.out.println(list.get(i));
    }
    System.out.println(list.contains("qwe"));
    al.remove("qwe");
    System.out.println(al.get(0));
```

```
    Iterator it=list.iterator();
    while(true){
        if(it.hasNext()){
          System.out.println(it.next());
        }else{
          break;
        }
    }
  }
}
执行结果：
zxc
qwe
asd
vbn
true
asd
asd
vbn
```

Map

```
public interface Map<k,v>
参数
k 钥匙的 Map 保持性
v 映射值得类型
```

Map 常用方法：

1. clear()
   void clear()
   从这个映射中移除所有的映射。
2. containsKey(Object key)
   boolean containsKey(Object key)
   如果该映射中包含一个指定 key 键则返回 true。
3. containsValue(Object value)
   boolean containsValue(Object value)
   如果该映射存在映射到该 value 的值的一个或多个键则返回 true。
4. get(Object key)
   V get(Object key)
   返回指定键映射的值，如果返回 null 则表示该映射不包含该键映射。
5. getDefault(Object key,V defaultValue)
   default v getDefault(Object key,V defaultValue)
   返回指定键映射的值，如果不包含该键映射返回 defaultValue。
6. put(K key,V value)
   V put(K key,V value)
   将指定的键值相关联。
7. remove(Oject key)
   V remove(Object key)
   从该映射中移除一个键的映射。
8. remove(Object key,Object value)
   default boolean remove(Object key,Object value)
   移除指定的键的值。
9. replace(K key,V Value)
   default V replace(K key,V Value)
   仅当当前映射到某一值时，替换指定的键的条目。
10. replace(K key,V oldValue,V new Value)
    default boolean replace(K key,V oldValue,V new Value)

仅当当前映射到指定的值时，替换指定的键的条目。

Map 常用的子类：HashMap

HashMap

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>,Cloneable,Serializable
```

HashMap 构造方法：

1. HashMap()
   public HashMap()
   构造一个默认初始容量的空 HashMap (16)和默认的加载因子 (0.75)。
2. HashMap(int initialCapacity)
   public HashMap(int initialCapacity)
   构建一个具有指定初始容量和加载因子空 HashMap(0.75)。
3. HashMap(Map m)
   public HashMap(Map m)
   构造一种新的 HashMap 与指定的 Map 相同的映射。

例如：

```
import java.util.*;
class Test{
  public static void main(String[] args) {
    Map<String,String> map=new HashMap<String,String>();
    map.put("1001","zhangsan");
    map.put("1002","lisi");
    map.put("1003","wanger");
    map.put("1004","zhaowu");
    System.out.println(map.get("1004"));
    map.replace("1001","hh");
    System.out.println(map.get("1001"));
    //利用内部类 Map.Entry<K,V>以及for-each 遍历该map
    for(Map.Entry<String,String> entry:map.entrySet()){
      String key=entry.getKey();
      String value=entry.getValue();
      System.out.println("key:"+key+",value:"+value);
    }
    //并对该map进行排序
    List<Entry<String,String>> list=new ArrayList<Map.Entry<String,String>>();
    Collection.sort(list,new Comparator<Map.Entry<String,String>>(){
      public int compare(Entry<String, String> o1, Entry<Integer, String> o2) {
                return o1.getKey().compareTo(o2.getKey());
            }
    });
    Iteratot<Entry<String,String>> it=list.iterator();
    while(true){
            if(it.hasNext()){
                Entry<String,String> entry=it.next();
        String i=entry.getKey();
        String s=entry.getValue();
        System.out.println(i+":"+s);
            }else{
                break;
            }
        }
  }
}
```