

Support Classes

Objectives

- Support Classes: Collections, Arrays
- Use the Collections class
 - Sorting/ Shuffling
 - Routine Data Manipulation(copy, reverse, swap, addAll...)
 - Searching (binarySearch)/
Composition(frequency, disjoint, min, max)
 - Finding Extreme Values (to find min,max value by comparator)
- Use the Arrays class

Introduction

- java.lang.**Object**
 - java.util.**Arrays**
 - java.util.**Collections**

- An algorithm on a list can be applied on some lists although the type of elements in each list can be different.
- The *polymorphic algorithms* described here are pieces of reusable functionality provided by the Java platform.
- All of them come from the **Collections** class and the **Arrays** class (support classes), and all take the form of static methods whose first argument is the collection on which the operation is to be performed.

The Collections class

- A support class containing static methods which accept **collections as their parameters**.
- <file:///J:/Softs/JavaSofts/JavaDocs/docs-Java8/api/java/util/Collections.html>

Collections Demo.

```
import java.util.ArrayList;
import java.util.Vector;
import java.util.Collections;
import java.util.Random;
public class CollectionsDemo {
    public static void main(String[] args){
        ArrayList ar= new ArrayList();
        Vector v = new Vector();
        Random rd= new Random(); // MAXIMUM VALUE= 29
        for (int i=1; i<=10; i++){
            ar.add(rd.nextInt(30));
            v.add(rd.nextInt(30));
        }
        System.out.println("ar=" + ar);
        System.out.println("v=" + v);
        boolean dis= Collections.disjoint(ar, v);
        System.out.println("ar and v are disjoint: " + dis);
        Collections.addAll(v, ar.toArray());
        System.out.println("After adding, v=" + v);
        int minVal= (int)Collections.min(v);
        int maxVal= (int) Collections.max(v);
    }
}
```

Collections Demo.

```

System.out.println("min= " + minVal + ", max= " + maxVal);
int fre= Collections.frequency(v, 8);
System.out.println("Occurences of 8: " + fre);
Collections.sort(v);
System.out.println("After sorting, v=" + v);
int pos = Collections.binarySearch(v, 8);
System.out.println("Position of 8: " + pos);
Collections.shuffle(v);
System.out.println("After shuffling, v=" + v);
    }
}

```

run:

ar=[16, 22, 13, 29, 12, 8, 23, 8, 17, 10]

v=[3, 2, 24, 13, 24, 18, 22, 8, 3, 1]

ar and v are disjunct: false

After adding, v=[3, 2, 24, 13, 24, 18, 22, 8, 3, 1, 16, 22, 13, 29, 12, 8, 23, 8, 17, 10]

min= 1, max= 29

Occurences of 8: 3

After sorting, v=[1, 2, 3, 3, 8, 8, 8, 10, 12, 13, 13, 16, 17, 18, 22, 22, 23, 24, 24, 29]

Position of 8: 4

After shuffling, v=[3, 3, 17, 8, 23, 8, 12, 24, 13, 18, 2, 24, 1, 29, 22, 16, 22, 13, 10, 8]

Sorting

- The sort algorithm reorders a List so that its elements are in ascending order according to an ordering relationship.
- Example

```
public class Sort {  
    public static void main(String[] args) {  
        List<String> list = Arrays.asList(args);  
        Collections.sort(list);  
        System.out.println(list);  
    }  
}
```

Comparator Interface

- A comparison function, which imposes a total ordering on some collection of objects
- The following demonstration will show you the way to sort a list based on your own criteria: A list of employees will be sorted based on descending salaries then ascending IDs.

Comparator Interface – Demo.

```
package sort;

import java.lang.Comparable;
import java.util.Comparator;

public class Employee implements Comparable {
    String ID="", name="";
    int salary=0;

    public Employee(String id, String n, int s){
        ID= id; name= n; salary=s;
    }

    @Override
    public String toString(){
        return ID + "," + name + "," + salary;
    }

    @Override // standard comparing
    public int compareTo(Object emp){
        return ID.compareTo(((Employee) emp).ID);
    }
}
```

Based on ID

Comparator Interface- Demo.

Comparing 2 employees based on descending salaries then ascending IDs

```
// comparing on salary descending then ID
public static Comparator compareObj= new Comparator() {
    @Override
    public int compare(Object e1, Object e2){
        Employee emp1 = (Employee) e1;
        Employee emp2 = (Employee) e2;
        int d= emp1.salary - emp2.salary;
        if (d>0) return -1; // lower salary -> move upper
        if (d==0) return emp1.ID.compareTo(emp2.ID);
        return 1;
    }
};
```

Create an
anonymous
object for
comparing 2
employees

Comparator Interface- Demo.

```
package sort;
import java.util.ArrayList;
import java.util.Collections;
public class SortDemo {
    public static void main(String[] args){
        ArrayList<Employee> list= new ArrayList<Employee>();
        list.add(new Employee("ID004", "Michel", 400));
        list.add(new Employee("ID001", "Helen", 200));
        list.add(new Employee("ID003", "Hemming", 400));
        System.out.println("Sorting on IDs ascending");
        Collections.sort(list);
        System.out.println(list);
        System.out.println("Sorting on descending salary then ascending IDs");
        Collections.sort(list, Employee.compareObj);
        System.out.println(list);
    }
}
```

run:

Sorting on IDs ascending

[ID001,Helen,200, ID003,Hemming,400, ID004,Michel,400]

Sorting on descending salary then ascending IDs

[ID003,Hemming,400, ID004,Michel,400, ID001,Helen,200]

Routine Data Manipulation (1)

- The Collections class provides five algorithms for doing routine data manipulation on List objects, including:
 - reverse()
 - fill()
 - copy()
 - swap()
 - addAll()

Searching

- Condition: The list in ascending order
- The binarySearch algorithm searches for a specified element in a sorted List.
 - Return $\text{pos} \geq 0 \rightarrow \text{Present}$
 - Return $\text{pos} < 0 \rightarrow \text{Absent}$

Composition

- `frequency` — counts the number of times the specified element occurs in the specified collection.
- `disjoint` — determines whether two `Collections` are disjoint; that is, whether they contain no elements in common.

Finding Extreme Values

- Methods: `min(...)`, `max(...)`

The Arrays Class

- It is similar to the Collections class, but it accepts arrays as its parameters.
- <file:///J:/Softs/JavaSofts/JavaDocs/docs-Java8/api/java/util/Arrays.html>

Arrays Class: Demo

```
import java.util.Arrays;

public class ArraysDemo {

    public static void main(String[] args)
    {
        int ar1[] = {5,1,4,7,9,3,4,5,3};
        int ar2[] = {5,6,7,8,9};
        int ar3[] = {5,6,7,8,9};

        System.out.println("ar1=" + Arrays.toString(ar1));
        System.out.println("ar2=" + Arrays.toString(ar2));
        System.out.println("ar3=" + Arrays.toString(ar3));
        boolean eq = Arrays.equals(ar1, ar2);
        System.out.println("ar1=ar2: " + eq);
        eq = Arrays.equals(ar2, ar3);
        System.out.println("ar2=ar3: " + eq);
        int numElements=3, from=2, before=6;
        int ar4[] = Arrays.copyOf(ar1, numElements);
        System.out.println("ar4=" + Arrays.toString(ar4));
        int ar5[] = Arrays.copyOfRange(ar1, from, before);
        System.out.println("ar5=" + Arrays.toString(ar5));
        Arrays.sort(ar1);
        System.out.println("After sorting, ar1=" + Arrays.toString(ar1));
        int pos = Arrays.binarySearch(ar1, 7);
        System.out.println("Binary search 7, pos= " + pos);
    }
}
```

```
run:
ar1=[5, 1, 4, 7, 9, 3, 4, 5, 3]
ar2=[5, 6, 7, 8, 9]
ar3=[5, 6, 7, 8, 9]
ar1=ar2: false
ar2=ar3: true
ar4= [5, 1, 4]
ar5=[4, 7, 9, 3]
After sorting, ar1=[1, 3, 3, 4, 4, 5, 5, 7, 9]
Binary search 7, pos= 7
```

Summary

- Support Classes: Collections, Arrays
- Use the Collections class
 - Sorting/ Shuffling
 - Routine Data Manipulation
 - Searching/ Composition
 - Finding Extreme Values
- Use the Arrays class