# Container Classes

## Sets, Maps

## COMP2603
## Object Oriented Programming 1

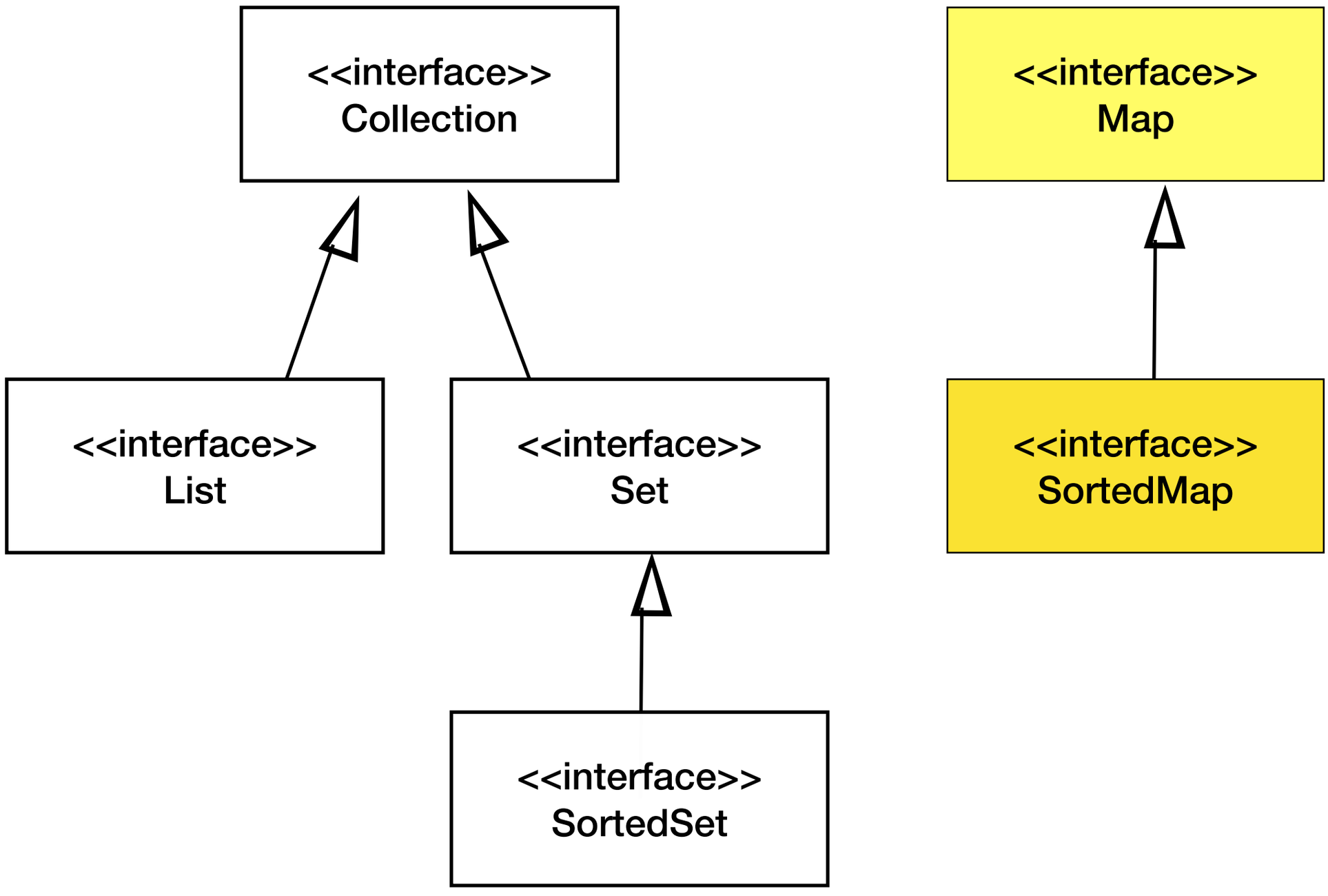
Week 9

# Outline

- Java Collections Framework
- Collection Interface
  - List
    - Linked List ✓
    - ArrayList ✓
    - Vector ✓
  - Set
    - SortedSet
- Map
  - SortedMap

Reading: Chapter 13 - Container Classes: P. Mohan 2013

# Interfaces in the Java Collections Framework



# The Set Interface

The Set interface represents an unordered collection of objects that contains no duplicate elements.

It therefore cannot contain two elements e1 and e2 where e1.equals(e2).

A Set can contain at most one null element.

The Set interface declares the same methods as its super-interface, Collection.

# The Set Interface
## (same as Collection)

| Method | Description |
|---|---|
| boolean add ( E o) | Inserts the object of the specified type into the collection; returns true if the object was added, false otherwise |
| boolean addAll (Collection c) | Inserts all the objects from the specified collection into the current collection |
| void clear( ) | Removes all the elements from the collection |
| boolean contains (Object o) | Returns true id the specified object is present in the colleciton, and false otherwise |
| boolean isEmpty( ) | Returns true if there are no elements in the collection, and false otherwise |
| boolean remove( Object o) | Deletes the specified object from the collection |
| int size( ) | Returns the number of elements currently in the collection |

**https://docs.oracle.com/javase/7/docs/api/java/util/Set.html**

# The Set Interface - Adding Elements

The Set interface  does not allow the add( ) and addAll( ) methods to duplicate elements to the Set.

If a Set already contains the element being added, its add( ) and addAll( ) methods return false.

In order for a Set to determine if it already contains an element, it uses the equals( ) method of the element to check if the element is equal to an existing element in the Set.

It is therefore necessary to override the equals( ) method of the objects that will be inserted into a Set.

# The Sorted Set Interface

The SortedSet interface is a Set that sorts its elements and guarantees that elements are enumerated in sorted order.

It declares a few methods of its own such as first( ) and last( ) which return the lowest and highest elements in the set, respectively (as determined by the sort order).

# Comparable Interface

This interface imposes a total ordering on the objects of each class that implements it.

This ordering is referred to as the class's natural ordering, and the class's **compareTo** method is referred to as its natural comparison method.

**https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html**

# int compareTo(Object obj )

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

**https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html**

# Comparable Interface

The natural ordering for a class C is said to be consistent with equals if and only if e1.compareTo(e2) == 0 has the same boolean value as e1.equals(e2) for every e1 and e2 of class C.

Note that null is not an instance of any class, and e.compareTo(null) should throw a NullPointerException even though e.equals(null) returns false.

**https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html**

# Example 1 - Plant Class

```
public class Plant implements Comparable{
    private String name;
    …
    …
    // Compare by Name – ascending A–Z
    public int compareTo(Object obj){
      if(obj instanceof Plant){
        Plant p = (Plant)obj;
         return name.compareTo(p.name);
      }
      throw new ClassCastException(“Not a Plant”);
    }
}
```

# Example 2 - Plant Class

```
public class Plant implements Comparable{
    private int ID;
    …
    …// 1 , 2
    // Compare by ID – ascending order
    public int compareTo(Object obj){
       if(obj instanceof Plant){
         Plant p = (Plant)obj;
          if(this.ID == p.ID) return 0;
          if(this.ID > p.ID) return 1;
          if(this.ID < p.ID) return -1;
       }
       throw new ClassCastException(“Not a Plant”);
    }
}
```

# The Map Interface

The Map interface represents a collection of mappings between key objects and value objects. Hash tables are examples of maps.

The set of key objects in a Map must not have any duplicates. However, the collection of value objects may contain duplicates.

# The Map Interface

| Method | Description |
|---|---|
| boolean containsKey( Object key) | Returns true if the Map contains a mapping for the specified key, and false otherwise |
| V get (Object key) | Returns the value object associated with the specified key or null if there is no mapping for the key |
| Set<E> keySet( ) | Returns a Set of all the key objects in the Map |
| V put(K key, V value) | Creates a key/value mapping in the Map. If the key already exists in the Map, put( ) replaces the value currently in the Map with the value supplied as an argument and returns the value replaced; otherwise it returns the value. |
| Collection<V> values( ) | Returns a Collection of all the value objects in the Map |

**https://docs.oracle.com/javase/7/docs/api/java/util/Map.html**

# Example - Map

```
public class Plant{
    private String name;

    public Plant(String n){
       name = n;
    }
    public String getName(){
       return name;
    }

}
```

# Example - Map
# Adding objects

```
Map<Plant> plants;
//assumed plants is initialised

Plant aloe = new Plant(“Aloe”); // value
String plantName = aloe.getName(); // key
plants.put(plantName, aloe);

Plant basil = new Plant(“Basil”);
plants.put(basil.getName(), basil);
```

# The Map Interface
# Preventing duplicate keys

To ensure that the set of key objects in the Map does not contain duplicates, it is important that the Key objects override the equals( ) method of the Object class, based on the content of the key.

Duplicates depend on how the equals( ) method is defined.

# Example - equals( )

```
public class Plant{
    private String name;
    …

    public boolean equals(Object obj){
       if(obj instanceof Plant){
          Plant p = (Plant) obj;
          if(this.name.equals(p.name))
             return true;
       }
       return false;
    }

}
```

# Example - hashCode( )

```
public class Plant{
    private String name;
    …

    public int hashCode( ){
       //use the String class hashCode
       return name.hashCode( );
    }

}
```

# The Sorted Map Interface

The SortedMap interface represents a Map object that keeps its set of key objects in sorted order. Its keySet( ) and values( ) methods inherited from Map return collection that can be traversed in sorted order of the key.

It also declares methods of its own such as firstKey( ) and lastKey() that return the lowest and highest key values in the SortedMap.

# Classes in the Java Collections Framework