SORTING

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

20: Comparators

Alexandra Papoutsaki

LECTURES

Mark Kampe

LABS

Lecture 20: Comparators

- Interface `Comparable`
- Interface `Comparator`
- Sorting

Some slides adopted from Algorithms 4th Edition or COS226

`Comparable`

- Interface with a single method that we need to implement:
  `public int compareTo(T that)`
- Implement it so that `v.compareTo(w)`:
  - Returns >0 if v is greater than w.
  - Returns <0 if v is smaller than w.
  - Returns 0 if v is equal to w.
- Corresponds to natural ordering.

Lecture 20: Comparators

- Interface `Comparable`
- Interface `Comparator`
- Sorting

Comparator

- Sometimes the natural ordering is not the type of ordering we want.
- Comparator is an interface which allows us to dictate that kind of ordering we want by implementing the method:

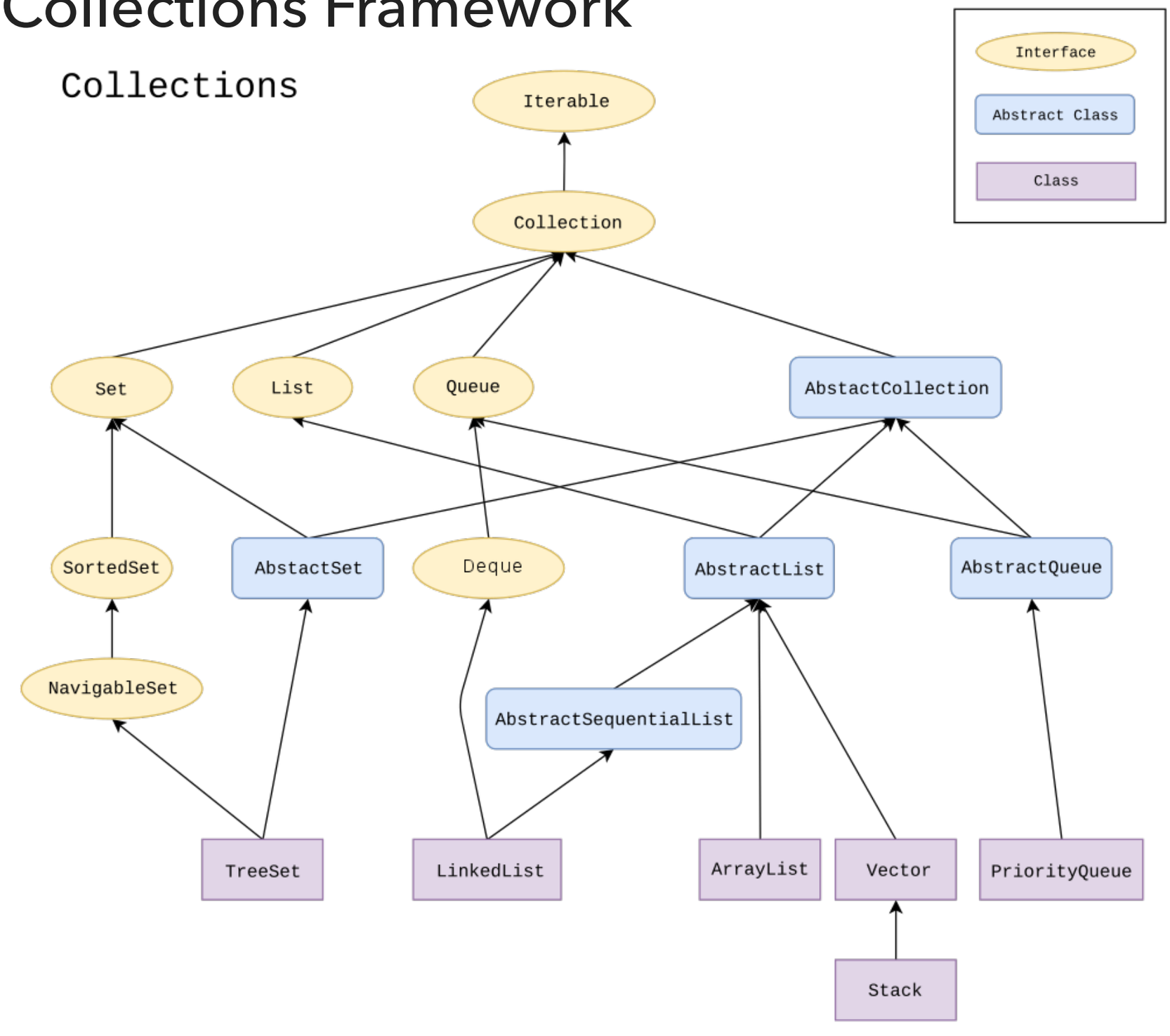
```
public int compare(T this, T that)
```

- Implement it so that `compare(v, w)`:
  - Returns >0 if v is greater than w.
  - Returns <0 if v is smaller than w.
  - Returns 0 if v is equal to w.

Lecture 20: Comparators

- Interface `Comparable`
- Interface `Comparator`
- Sorting

The Java Collections Framework



https://en.wikipedia.org/wiki/Java_collections_framework

Sorting Collections

- Collections class contains:
  - `public static <T extends Comparable<? super T>> void sort(List<T> list)`
  - Generic methods introduce their own type parameters.
    - Use extends with generics, even if the type parameter implements an interface.
- The class T itself or one of its ancestors implements Comparable.
- `Collections.sort(list)`
  - Implemented as optimized mergesort
  - If list's elements do not implement `Comparable`, throw `ClassCastException`.

Example: How can we sort `Association` objects?

- ```
  public class Association<K, V>
      protected K theKey; // key of the key-value pair
      protected V theValue; // value of key-value pair
  ```
- We want associations where we can order by key.
  ```

Make a comparable class.

```
public class ComparableAssociation<K extends Comparable<K>,V>
extends Association<K,V> implements
Comparable<ComparableAssociation<K,V>>{
    public ComparableAssociation(K key, V value) {
        super(key,value);
    }
    public int compareTo(ComparableAssociation<K,V> that) {
        return this.getKey().compareTo(that.getKey());
    }
    ...
}
```

- Now we can use `sort` method!

Sort a collection of ComparableAssociations

```
public static void main(String[] argv) {
      List<ComparableAssociation<Integer, String>> classesTaken = new
ArrayList<ComparableAssociation<Integer, String>>();

      // add professors and classes taken to a vector
      classesTaken.add(new ComparableAssociation<Integer, String>(1, "Barbara"));
      classesTaken.add(new ComparableAssociation<Integer, String>(3, "Bill"));
      classesTaken.add(new ComparableAssociation<Integer, String>(2, "Duane"));
      classesTaken.add(new ComparableAssociation<Integer, String>(1, "Tom"));
      Collections.sort(classesTaken);
      for (ComparableAssociation<Integer, String> classX : classesTaken) {
         System.out.println(classX);
      }
}
```

Altenative Sorting of `Collections`

- Collections class contains:
  - `static <T> void sort(List<T> list, Comparator<? super T> c)`
- `Collections.sort(list, someComparator);`
  - If list's elements do not implement `Comparable` or cannot be compared with `Comparator`, throw `ClassCastException`.

Example: Alternative sorting for Employees

```
public class Employee implements Comparable<Employee> {

    private int id;
    private String name;
    private int age;
    private long salary;

    // Many sort sequences can be created with different names.
    public static Comparator<Employee> NameComparator = new Comparator<Employee>() {
        @Override
        public int compare(Employee e1, Employee e2) {
            return e1.getName().compareTo(e2.getName());
        }
    };
    public static Comparator<Employee> idComparator = new Comparator<Employee>() {
        @Override
        public int compare(Employee e1, Employee e2) {
            return Integer.valueOf(e1.getId()).compareTo(Integer.valueOf(e2.getId()));
        }
    };

    public Employee() { }
    public Employee(int id, String name, int age, long salary){
        this.id = id;
        this.name = name;
        this.age = age;
        this.salary = salary;
    }
    //… setters and getters.
```

https://stackoverflow.com/questions/2266827/when-to-use-comparable-and-comparator

Example: Alternative sorting for Employees

```
    // Only one sort sequence can be created with in the class.
    @Override
    public int compareTo(Employee e) {
    //return Integer.valueOf(this.id).compareTo(Integer.valueOf(e.id));
    //return Character.toString(this.name.charAt(0)).compareToIgnoreCase(Character.toString(e.name.charAt(0)));
        if (this.id > e.id) {
            return 1;
        }else if(this.id < e.id){
            return -1;
        }else {
            return Character.toString(this.name.charAt(0)).compareToIgnoreCase(Character.toString(e.name.charAt(0)));
        }

    }

    public static void main(String[] args) {

        Employee e1 = new Employee(5, "Yash", 22, 1000);
        Employee e2 = new Employee(8, "Tharun", 24, 25000);

        List<Employee> list = new ArrayList<Employee>();
        list.add(e1);
        list.add(e2);
        Collections.sort(list); // call @compareTo(o1)
        Collections.sort(list, Employee.nameComparator); // call @compare (o1,o2)
        Collections.sort(list, Employee.idComparator); // call @compare (o1,o2)
    }
}
```

https://stackoverflow.com/questions/2266827/when-to-use-comparable-and-comparator

Lecture 20: Comparators

- Interface `Comparable`
- Interface `Comparator`
- Sorting

Readings:

- Textbook:
  - Chapter 2.1 (Page 247), Chapter 2.5 (Pages 338-339)
- Oracle Documentation:
  - Comparable: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
  - Comparator: https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html