# *Interfaces: Comparable & Comparator*

Supplemental Material

CS 2110

# *Additional Material: [Comparable and Comparator]*

*(This information supplements the main slides; you are responsible for the material contained within these slides – don't hesitate to stop by office hours if you have any questions!)*

# *Can you sort an ArrayList of Strings?*

- Go to API for class String (Google: "Java 10 String API")
- On "All Implemented Interfaces" – one of them is **`comparable`**!
  - String class implements the Comparable Interface!
- Scroll to "Method Summary" – has **`compareTo()`** method!
- So you can definitely sort an ArrayList of Strings using Collections.sort() method

`https://docs.oracle.com/javase/10/docs/api/java/lang/String.html`

# *What are we getting at?*

- The method **sort** can take an **ArrayList** as an argument (an ArrayList of something) Of what? It can take an ArrayList of anything that meets this interface called Comparable
- The parameter has to be an ArrayList of any class that implements the comparable interface – that means it has the compareTo() method – which means inside of **sort**, it knows it can call get() to get 2 items from the ArrayList and use the compareTo() method – its **guaranteed** to have it because that class implements the Comparable interface

# *Under the Hood for Sorting*

- How might a **sort()** or any other method use this? Imagine:
  - Perhaps items stored as a list of Objects:

```
List theList …;
```

- Inside a loop, code might look like this:

```
Comparable item1 = (Comparable) theList.get(i);
Comparable item2 = (Comparable) theList.get(j);
int cmpResult = item1.compareTo(item2);
```

- Such code will work when the list stores any class that implements Comparable!
- But, what happens if list-elements are of <u>different</u> classes (still Comparable, but different)?
  - compareTo() fails!

# *Take note!*

```
public class TestingTypes{
      public static void main(String []args){
            String s = "Hello!";

            System.out.println(s instanceof String);
            System.out.println(s instanceof Object);
            System.out.println(s instanceof Comparable);
      }
}
```

Output:
true
true
true

# *compareTo()String Examples*

- Consider the following code:

```
System.out.println("cat".compareTo("dog"));
System.out.println("eagle".compareTo("cat"));
System.out.println("cats".compareTo("catcher"));
System.out.println("bed".compareTo("bedroom"));
```

| Prints | What it means |
|---|---|
| -1 | cat < dog |
| 2 | eagle > cat |
| 16 | cats > catcher |
| -4 | bed < bedroom |

- When the first world comes alphabetically before the second word, the result is negative
- When the first world comes alphabetically after the second word, the result is positive
- If we use two identical strings, the result will be **zero**

# *More about compareTo()*

- Digits are less than letters

```
System.out.println("999".compareTo("AAA")); //negative
```

- Capital letters are less than lowercase letters

```
System.out.println("AAA".compareTo("aaa")); //negative
```

# *Using Comparable (compareTo()) with TreeSet*

- TreeSets are inherently sorted. As we add elements, they are stored in a particular way
  - As a result, printing elements of a TreeSet<String> will be in alphabetical order
    - With the exception around **case**

- To place items in a TreeSet, that class must implement the Comparable interface
  - Example, if we have a class student, the first line must be:

```
public class Student implements Comparable<Student>
```

# A Comparator Example: Student class

- Consider the Student class
  - What if we want to sort in a more specific way:
    - Sort by scores in descending order
    - If two students have the same score, sort by name
- We can create a Comparator as follows:

```
1
2  import java.util.Comparator;
3
4  public class StudentScoreNameComparator implements Comparator<Student> {
5      public int compare(Student s0, Student s1) {
6          if (s0.score == s1.score) {
7              return s0.name.compareTo(s1.name);
8          } else {
9              return s1.score - s0.score;
10         }
11     }
12 }
```

# *Creating Comparators for classes you didn't write*

- What if we wanted to sort Strings *ignoring* case?
- Well, we can:

```
import java.util.Comparator;

public class StringIgnoreCaseComparator implements Comparator<String> {
    public int compare(String o1, String o2) {
        return o1.toUpperCase().compareTo(o2.toUpperCase());
    }
}
```

- `Collections.sort(listOfStrings, new StringIgnoreCaseComparator());`

# *Using Comparator (compare()) with TreeSet*

- By default, the TreeSet will use a Comparable class's compareTo() method to sort elements.
- However, we can change that when we create the TreeSet
- Example

```
TreeSet<Student> students = new TreeSet<Student>(new
                                                 StudentNameComparator());
```

- Note that we can only do this when we create the TreeSet
  - We cannot change it later
- You can sort an ArrayList as many times as you want