

# CSSE 220

Comparable/Comparator

Import *MergeSortSimple* project from Git

# Today's Plan

- How to use Java's sort functions (Comparable and Comparator)
- Project worktime

# How to Sort in Java

- For arrays:

```
Arrays.sort(myArray);
```

- For ArrayLists or other stuff:

```
Collections.sort(myArrayList)
```

- For stuff like Strings and ints, the expected sorting is already built in. But what if you have a new class you want to sort?

# When Your Object is Sortable

- You should implement the `Comparable<YourObjectType>` interface
- You need to implement 1 method:  
`compareTo(other)`

Requirements: compares data like `.equals()`, but returns an integer such that:

- `a.compareTo(b) < 0` when `a < b`**
- `a.compareTo(b) > 0` when `a > b`**
- `a.compareTo(b) == 0` when `a == b`.**

Section 10.3 of your text has more details

# A Sort of a Different Order

- Java libraries provide efficient sorting algorithms
  - `Arrays.sort(...)` and `Collections.sort(...)`
- But suppose we want to sort by something other than the “natural order” given by `compareTo()`
- Look at the ugly code duplication if the way to sort is embedded in the sort (next slide)!

# Code duplication again!

## Sort by length of string

```
public void sort(String[] array) {
    final int n = array.length;
    for (int j= 0; j< n - 1; j++) {
        int indexOfSmallestLeft = j;
        String smallestLeft = array[indexOfSmallestLeft];
        for (int i = j+ 1; i < n; i++) {
            if (array[i].length() < smallestLeft.length()) {
                indexOfSmallestLeft = i;
                smallestLeft = array[i];
            }
        }
        array[indexOfSmallestLeft] = array[j];
        array[j] = smallestLeft;
    }
}
```

## Sort by second character

```
public void sort(String[] array) {
    final int n = array.length;
    for (int j= 0; j< n - 1; j++) {
        int indexOfSmallestLeft = j;
        String smallestLeft = array[indexOfSmallestLeft];
        for (int i = j+ 1; i < n; i++) {
            if (array[i].charAt(1) < smallestLeft.charAt(1)) {
                indexOfSmallestLeft = i;
                smallestLeft = array[i];
            }
        }
        array[indexOfSmallestLeft] = array[j];
        array[j] = smallestLeft;
    }
}
```

So close! Can we let the “way to sort” be a parameter to the method?

# Solution: Function Objects

- Objects defined to just “wrap up” functions so we can pass them to other (library) code
- For sorting we can create a function object that implements Comparator  
`Arrays.sort(people, new ByAgeComparator())`
- What goes into the ByAgeComparator class?
- Let's try it!
- Examples on next slide if you get stuck

# Examples

```
String[] colors = new String[] {"red", "orange", "yellow", "green", "blue", "indigo", "violet"};
Arrays.sort(colors);
System.out.println("Sort [default]: " + Arrays.toString(colors));
```

```
Comparator<String> bySecondLetter = new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        char first = s1.charAt(1);
        char second = s2.charAt(1);
        return first - second;
    }
};
Arrays.sort(colors, bySecondLetter);
System.out.println("Sort [second letter]: " + Arrays.toString(colors));
```

```
Comparator<String> byFirstEPosition = new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        int first = s1.indexOf("e");
        int second = s2.indexOf("e");
        return first - second;
    }
};
Arrays.sort(colors, byFirstEPosition);
System.out.println("Sort [first e position]: " + Arrays.toString(colors));
```