Sorting a List of your Objects: **Comparable Interface and its compareTo method**

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 10.2
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/) & thanks to Ms Martin.

Sorting Lists

- Collections.sort(List<T> list)
  - A static method in the Collections class
  - Can sort any kind of list
  - MUST pass in a variable of type List (like an ArrayList or others)
- But how does it know how to order those elements?
- The List's element type (an Object) must implement the **Comparable interface** … with the **compareTo** method

```
public interface Comparable {
    public int compareTo(Object obj);
}
```

The `compareTo` method

The standard way for a Java class to define a comparison function for its objects is to define a `compareTo` method.

```
this.compareTo(object parameter_name)
```

- Compares `this` with the parameter
  - returns negative number if `this` is less than parameter
  - returns zero if they are equal
  - returns positive number if `this` is greater than parameter

  Example: in the `String` class, there is a method:

  ```
  public int compareTo(String other)
  ```

- A call of **A**`.compareTo(`**B**`)` will return:

  a value < 0 if **A** comes "before" **B** in the ordering,

  a value > 0 if **A** comes "after" **B** in the ordering,

  or 0 if **A** and **B** are considered "equal" in the ordering.

Using `compareTo`

- `compareTo` can be used as a test in an `if` statement.

```
String a = "alice";
String b = "bob";
if (a.compareTo(b) < 0) {   // true
    ...
}
```

Primitives	Objects
`if (a < b) { ...`	`if (a.compareTo(b) < 0) { ...`
`if (a <= b) { ...`	`if (a.compareTo(b) <= 0) { ...`
`if (a == b) { ...`	`if (a.compareTo(b) == 0) { ...`
`if (a != b) { ...`	`if (a.compareTo(b) != 0) { ...`
`if (a >= b) { ...`	`if (a.compareTo(b) >= 0) { ...`
`if (a > b) { ...`	`if (a.compareTo(b) > 0) { ...`

Comparable (10.2)

```
public interface Comparable<E> {
    public int compareTo(E other);
}
```

- A class can implement the `Comparable` interface to define a natural ordering function for its objects.

- A call to your `compareTo` method should return:

  a value < 0 if the `other` object comes "before" `this` one,

  a value > 0 if the `other` object comes "after" `this` one,

  or 0 if the `other` object is considered "equal" to `this`.

- If you want multiple orderings, use a `Comparator` instead (see Ch. 13.1)

Comparable template

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

    ...

    public int compareTo(name other) {
        ...
    }
}
```

Comparable example

```
public class Point implements Comparable<Point> {
    private int x;
    private int y;
    …

    // sort by x and break ties by y
    public int compareTo(Point other) {
        if (x < other.x) {
            return -1;
        } else if (x > other.x) {
            return 1;
        } else if (y < other.y) {
            return -1;    // same x, smaller y
        } else if (y > other.y) {
            return 1;     // same x, larger y
        } else {
            return 0;     // same x and same y
        }
    }
}
```

compareTo tricks

- *subtraction trick* - Subtracting related numeric values produces the right result for what you want `compareTo` to return:

```
// sort by x and break ties by y
public int compareTo(Point other) {
    if (x != other.x) {
        return x - other.x;   // different x
    } else {
        return y - other.y;   // same x; compare y
    }
}
```

  - The idea:
    - if `x > other.x`, then `x - other.x > 0`
    - if `x < other.x`, then `x - other.x < 0`
    - if `x == other.x`, then `x - other.x == 0`
      - NOTE: This trick doesn't work for `double`s (but see `Math.signum`)

`compareTo` tricks 2

- *delegation trick* - If your object's fields are comparable (such as strings), use their `compareTo` results to help you:

```
// sort by employee name, e.g. "Jim" < "Susan"
public int compareTo(Employee other) {
    return name.compareTo(other.getName());
}
```

- *`toString` trick* - If your object's `toString` representation is related to the ordering, use that to help you:

```
// sort by date, e.g. "09/19" > "04/01"
public int compareTo(Date other) {
    return toString().compareTo(other.toString());
}
```