CSE 143
Section 14

`Comparable`

slides adapted from Marty Stepp and Hélène Martin
http://www.cs.washington.edu/143/

Ordering our own types

- We cannot binary search or make a `TreeSet`/`Map` of arbitrary types, because Java doesn't know how to order the elements.
  - The program compiles but crashes when we run it.

```
Set<Note> tags = new TreeSet<Note>();
tags.add(new Note(1.5, Pitch.A, 4, Accidental.Natural, false));
tags.add(new Note(0.6, Pitch.C, 4, Accidental.Natural, true));
...

Exception in thread "main" java.lang.ClassCastException
        at java.util.TreeSet.add(TreeSet.java:238)
```

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 `this` object comes "before" the `other` object,
  a value > 0 if `this` object comes "after" the `other` object,
  or 0 if `this` object is considered "equal" to the `other`.

- 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());
}
```