Sorting Objects and Comparisons

Common Java object methods

- Four methods underlie many of Java's built in functionality
  - equals – you should be sick of this one at this point
  - hashCode – we spent Monday talking about this
  - compare and compareTo – we'll talk about today

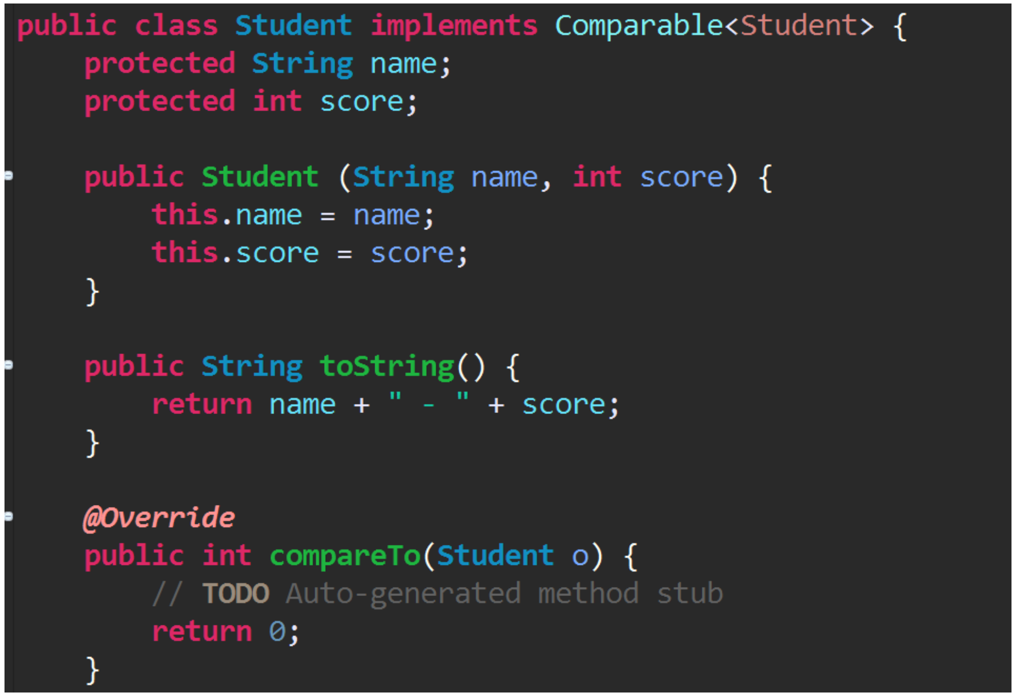
- Many built in Java objects define these, such as String
  - For your own objects, you must define them

Compare/CompareTo

- You should have already seen compareTo
  - String
  - firstItem.compareTo(secondItem)
    - 0 if they are equal
    - Negative if firstItem is LESS THAN secondItem
    - Positive if firstItem is GREATHER THAN secondItem

Making an object sortable

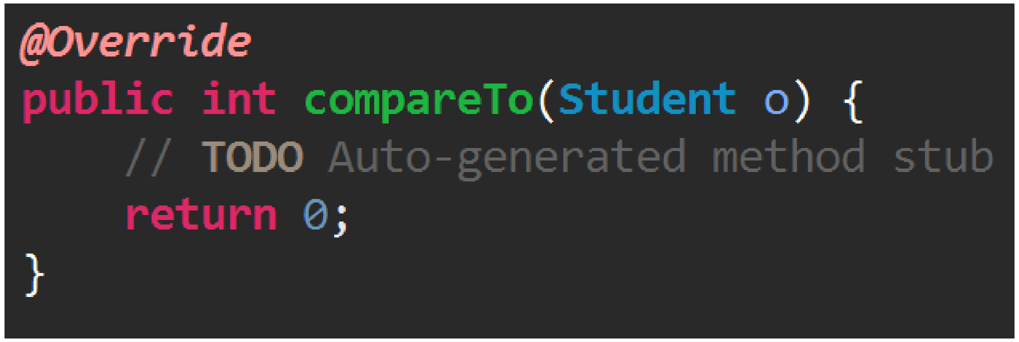
- Simply implement Comparable
  - Comparable says “this object can be sorted”
  - Comparable is generically typed, so you have to specify type



```java
public class Student implements Comparable<Student> {
    protected String name;
    protected int score;

    public Student (String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String toString() {
        return name + " - " + score;
    }

    @Override
    public int compareTo(Student o) {
        // TODO Auto-generated method stub
        return 0;
    }
```

Comparable interface

- Only one required method



```
@Override
public int compareTo(Student o) {
    // TODO Auto-generated method stub
    return 0;
}
```

- Example, first.compareTo(second)
- Return 0 if equal (first.equals(second) == true)
- Return negative if first < second
- Return positive if second < first
- The exact value is irrelevant, only the sign matters

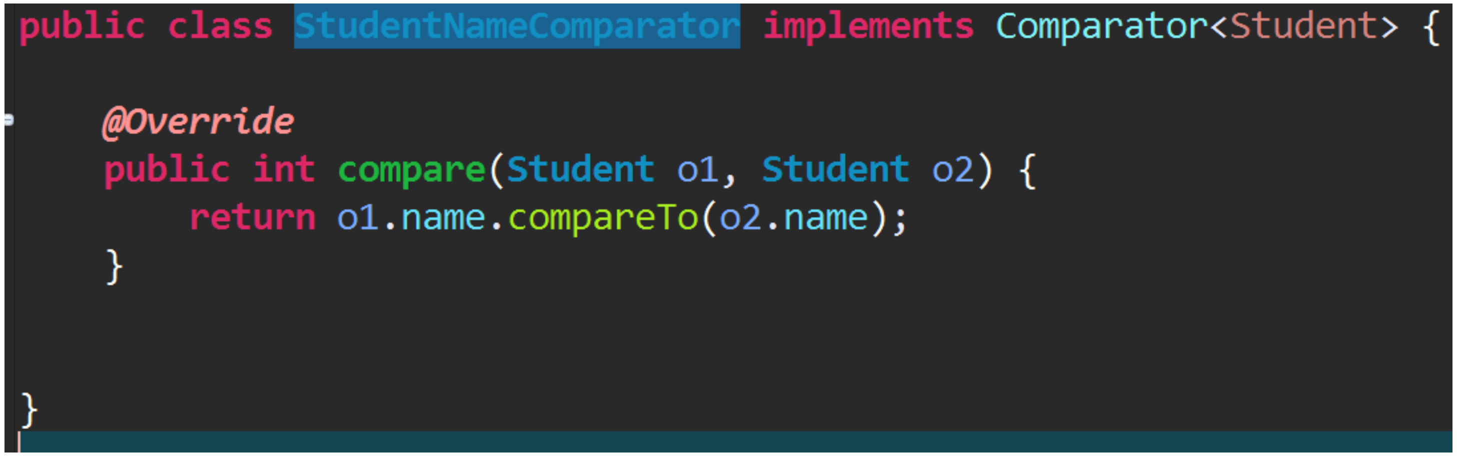
Sorting by score

- Collections.sort will sort using this compareTo in most cases

Using a Comparator

- The problem here is if we want to sort by student name, we have to rewrite our compareTo method.

- What if we want to sort at runtime?
  - Use a *Comparator*



```
public class StudentNameComparator implements Comparator<Student> {

    @Override
    public int compare(Student o1, Student o2) {
        return o1.name.compareTo(o2.name);
    }


}
```

Sorting an object by a field

- Comparator class that uses that field
  - Make sure your compare(Object o1, Object o2) method returns the right result

- Use Collections.sort(thingYouAreSorting, new CompartorYouAreUsing());

```
Collections.sort(students, new StudentNoComparator());
```