CSE 143
Computer Programming II

Outline

1. Understand How To Use Interfaces

2. Learn about the Comparable Interface

IntList Interface

For example: To be a List, which methods does another class need to have?
- Lists have an add method
- Lists have a remove method
- Lists have a get method
- Lists have a set method
- Lists have a size method
- ...

Normally, we specify a method and its implementation. Java allows us to just specify the header:

"public String toString();"

is a valid line of code.

Interfaces

An interface specifies a group of behaviors and gives them a name. Classes can choose to implement interfaces which require them to implement all of the methods in the interface.

Interfaces answer the question:

"To be an X, which methods does another class need to have?"

Interface Syntax

To Specify An Interface

1. public interface IntList {
2.   public void add(int value);
3.   public int remove(int index);
4.   public int get(int index);
5.   public void set(int index, int element);
6.   public int size();
7.   public boolean isEmpty();
8. }

To Use An Interface

Edit the first line of a class (say ArrayIntList or LinkedIntList):

public class ArrayIntList implements IntList {
...

Also, make sure it actually has all the methods the interface is supposed to have...
**How do sort and TreeSet work?**

How do sort and TreeSet KNOW the ordering?

If you were implementing sort for a type T, what would you need to be able to do with T a and T b?

We would need to be able to COMPARE a and b

That's just an interface! Java calls it "Comparable".

### Comparable

The Comparable interface allows us to tell Java how to sort a type of object:

```java
1 public interface Comparable<E> { 
2    public int compareTo(E other);
3 }
```

This says, "to be Comparable, classes must define compareTo".

---

**Thinking about compareTo**

Think about the following transformation when using compareTo:

```
this.compareTo(that) < 0
```

This works if you replace < with =, >, !=, ....:

<table>
<thead>
<tr>
<th>Normal</th>
<th>compareTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; b</td>
<td>a.compareTo(b) &lt; 0</td>
</tr>
<tr>
<td>a &lt;= b</td>
<td>a.compareTo(b) &lt;= 0</td>
</tr>
<tr>
<td>a == b</td>
<td>a.compareTo(b) == 0</td>
</tr>
<tr>
<td>a &gt;= b</td>
<td>a.compareTo(b) &gt;= 0</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>a.compareTo(b) &gt; 0</td>
</tr>
</tbody>
</table>

---

**Storing Multiple Choice Quizzes**

The text files:
- Each text file corresponds to answers for a multiple choice quiz.
- Each line contains one answer.
- For each quiz, answers.txt represents the correct answers.

### MQQuiz Class

```java
1 public class MQQuiz { 
2    private String studentName; 
3    private String quizName; 
4    private List<String> correctAnswers; 
5    private List<String> studentAnswers; 
6    public MQQuiz(String filename) throws FileNotFoundException { ... }
7    public String getName() { ... } 
8    public String getStudent() { ... } 
9    public int numberCorrect() throws IOException { ... } 
10    public int numberCorrect() { ... } 
11    public int numberCorrect() throws IOException { ... } 
12    public MQQuiz(String filename) throws FileNotFoundException { ... }
13    public int numberCorrect() throws IOException { ... } 
```

We would like to do the two following tasks:
- Print out the quizzes in worst-to-best order
- Collect all quizzes of each particular student together and display them (still from worst-to-best)

---

**MCQuiz: Defining compareTo**

### Attempt #1

```java
1 public class MCQuiz implements Comparable<MCQuiz> { 
2    ... 
3    public int compareTo(MCQuiz other) { 
4        return this.numberCorrect() - other.numberCorrect();
5    }
```

This doesn't work, because if we have a quiz where someone got 1/10 and another where someone else got 1/5, we treat them as the same.

### Attempt #2

```java
1 public class MCQuiz implements Comparable<MCQuiz> { 
2    ... 
3    public int compareTo(MCQuiz other) { 
4        return (double)this.numberCorrect()/this.correctAnswers.size() -
5            (double)other.numberCorrect()/other.correctAnswers.size();
6    }
```

This won't even compile! We need to return an int.

---

**Printing The Quizzes in Order**

### Client Code to Print The Quizzes

```java
1 List<MCQuiz> quizzes = createQuizzes(2);
2 Collections.sort(quizzes);
3 for (MCQuiz quiz : quizzes) { 
4    System.out.println(quiz);
5 }
```

This doesn't work, because Java doesn't know how to sort MCQuizzes.

### Comparable

The Comparable interface allows us to tell Java how to sort a type of object:

```java
1 public interface Comparable<E> { 
2    public int compareTo(E other);
3 }
```

This says, "to be Comparable, classes must define compareTo".

---

**Comparable: Tricks #1 & #2**

### int Fields

If we have a field int x in our class, and we want to compare with it, our code should look like:

```java
1 public class Sample implements Comparable<Sample> { 
2    public int compareTo(Sample other) { 
3        return this.x - other.x;
4    }
5 }
```

### Object Fields

If we have a field Thing x in our class, and we want to compare with it, our code should look like:

```java
1 public class Sample implements Comparable<Sample> { 
2    public int compareTo(Sample other) { 
3        return this.x.compareTo(other.x);
4    }
5 }
```

In other words, just use the existing compareTo on the field in the class!
Grouping the Quizzes by Student

The output looks like this:

```
>> TeresaHall: [TeresaHall (quiz0): 4/11]
>> BarbaraHarris: [BarbaraHarris (quiz1): 3/11, BarbaraHarris (quiz0): 4/11]
```

Why does Teresa only have one quiz? She scored the same on both of her quizzes and compareTo said they were the same!

Final Attempt

```
public class MCQuiz implements Comparable<MCQuiz> {
    public int compareTo(MCQuiz other) {
        Double thisPer = (double)this.numberCorrect() / this.correctAnswers.size();
        Double otherPer = (double)other.numberCorrect() / other.correctAnswers.size();
        int result = thisPer.compareTo(otherPer);
        if (result == 0) { result = this.studentName.compareTo(other.studentName); } 
        return result;
    }
}
```

Lesson: When you write compareTo, make sure that `a.compareTo(b) == 0` exactly when `a.equals(b)`.

Some Interface/Comparable Tips

- Understand multi-level structures
- Use the most general interface as possible
- When implementing compareTo, make sure to use all the fields that make it different (to put another way: `a.compareTo(b) == 0` exactly when `a.equals(b)`)
- Remember that inside classes, you can look at the fields of other instances of that class