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.

**MCQuiz Class**
- public static void sort(List<MCQuiz> list) {
- for (int i = 0; i < list.size(); i++) {
- int minIndex = i;
- for (int j = i; j < list.size(); j++) {
- if (list.get(j).compareTo(list.get(minIndex)) < 0) {
- minIndex = j;
- }
- }
- list.set(minIndex, list.get(i));
- }
}

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

Sorting A String List

Strings have a method called compareTo that works like < does on ints.
If we have two strings:

```
String hello = "hello" and String bye = "bye"
```
To do the test "hello < bye", we do the following:
- Write what we want: hello < bye
- Subtract the right from both sides: hello - bye < 0
- Replace the subtraction with compareTo:

```
hello.compareTo(bye) < 0
```
That's it!!

Sorting A String List

```
if (list.get(i).compareTo(list.get(minIndex)) < 0) {
    minIndex = j;
}
```

Sorting A MCQuiz List

```
if (list.get(i).compareTo(list.get(minIndex)) < 0) {
    minIndex = j;
}
```

Sorting A String List

```
if (list.get(i).compareTo(list.get(minIndex)) < 0) {
    minIndex = j;
}
```

Sorting Integers and Strings

Last lecture, we sorted the characters of a string. Let's sort more:

**Implementing A compareTo**
- public int compareTo(MCQuiz other) {
- return this.numberCorrect() - other.numberCorrect();
-
-}

Strings were easier, because they knew how to compare themselves.
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 and T ?

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

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

MCQuiz: Defining compareTo

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

This doesn't work, because 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
1 public class MCQuiz implements Comparable<MCQuiz> {  
2   public int compareTo(MCQuiz other) {  
3     return (double) this.numberCorrect() / (double) other.numberCorrect();  
4   }  

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

MCQuiz: Defining compareTo

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

This still doesn't work, because it doesn't take the names of the students into account. In particular, if two students both get 1/10 on a quiz, our compareTo method says "it doesn't matter which one goes first".

Attempt #4
1 public class MCQuiz implements Comparable<MCQuiz> {  
2   public int compareTo(MCQuiz other) {  
3     double thisPer = (double) this.numberCorrect() / (double) this.correctAnswers.size();  
4     double otherPer = (double) other.numberCorrect() / (double) other.correctAnswers.size();  
5     int result = thisPer.compareTo(otherPer);  
6     if (result == 0) { result = this.studentName.compareTo(other.studentName); }  
7     return result;  
8   }  

This still doesn't work, but it's not as clear why. Let's try the second task.

Printing The Quizzes in Order

Client Code to Print The Quizzes
1 List<MCQuiz> quizzes = createQuizzes(2);  
2 // ...  
3 Collections.sort(quizzes);  
4 for (MCQuiz quiz : quizzes) {  
5   System.out.println(quiz);  
6  }  

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

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

MCQuiz: Defining compareTo

Comparable: Tricks #1 & #2

Object Fields
If we have a field Thing x in our class, and we want to compare with it, our code should look like:
1 public class MCQuiz implements Comparable<MCQuiz> {  
2   public int compareTo(MCQuiz other) {  
3     return (x).compareTo(other.x);  
4   }  

In other words, just use the existing compareTo on the field in the class!

Grouping the Quizzes by Student

What data structure should we use to group the quizzes? A Map!

Map Question: "Which quizzes were taken by this student?"

Keys: Strings (the student names)
Values: Sets of MCQuiz (all the quizzes that student took).

MCQuiz: Defining compareTo

Attempt #3
1 public class MCQuiz implements Comparable<MCQuiz> {  
2   public int compareTo(MCQuiz other) {  
3     double thisPer = (double) this.numberCorrect() / (double) this.correctAnswers.size();  
4     double otherPer = (double) other.numberCorrect() / (double) other.correctAnswers.size();  
5     return thisPer.compareTo(otherPer);  
6   }  

This still 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:
1 public interface Comparable<E> {  
2   public int compareTo(E other);  
3 }  

This says, "to be Comparable, classes must define compareTo".
The output looks like this:

```
>> BarbaraHarris: [BarbaraHarris (quiz1): 3/11, BarbaraHarris (quiz0): 4/11]
>> JessicaHernan: [JessicaHernan (quiz1): 1/11, JessicaHernan (quiz0): 2/11]
>> TeresaHall: [TeresaHall (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

```java
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);
    }
    if (result == 0) {
      result = this.quizName.compareTo(other.quizName);
    }
    return result;
  }
}
```

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

---

### Some 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
Today, we will consider multiple new implementations of IntSet:

```java
public interface IntSet {
    public void add(int value);
    public void remove(int value);
    public boolean contains(int value);
}
```

A New Data Structure

Design a class RangeSet that represents a set which only allows numbers inside a fixed range.

You should have a constructor:

```java
RangeSet(int max) {
    this.data = new boolean[max];
}
```

And the following public methods:

```java
add(val) { ... remove(int value) {
    if (value < this.data.length && value >= 0) {
        this.data[value] = false;
    }
}
```

Generalizing to an Arbitrary Range

In RangeSet, when we got the number n, we mapped it to the index n. What if we had a function that took an input and mapped it to an index?

**Definition (HashCode)**

A hash code is a function that takes in a piece of data and maps it to an array index.

If we have an array of size 8, consider the following hashcode:

```java
public int hashCode(int value) {
    return value % 8;
}
```

Now, let’s insert the following data: 1, 4, 13

<table>
<thead>
<tr>
<th>Set Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>
public class IntHashSet implements IntSet {
    public final int DEFAULT_SIZE = 20;
    public Integer[] data;
    public IntHashSet() {
        this.data = new Integer[DEFAULT_SIZE];
    }
    public Integer[] data;  // This is redundant
    public void add(int value) {
        this.data[hashCode(value)] = value;
    }
    public boolean contains(int value) {
        return this.data[hashCode(value)] == value;
    }
    public void remove(int value) {
        this.data[hashCode(value)] = null;
    }
    public int hashCode() {
        int h = hash;
        if (h == 0 && value.length > 0) {
            char val[] = value;
            for (int i = 0; i < value.length; i++) {
                h = 31 * h + val[i];
            }
            hash = h;
        }
        return h;
    }
}

Consider the following insertions: 1, 21
First: 1 1 1 1 1
Then: 21 21 21 21 21
Uh oh! We've overwritten the one!
How can we fix this?
Instead of storing an integer, let's store a list of integers
1 21 1 1 1
set

String HashCode!

public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char val[] = value;
        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}