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.

```java
MCQuiz Class
public class MCQuiz {
    private String studentName;
    private String quizName;
    private List<String> correctAnswers;
    private List<String> studentAnswers;

    public MCQuiz(String studentName) {
        this.studentName = studentName;
        this.correctAnswers = new ArrayList<>();
        this.studentAnswers = new ArrayList<>();
    }

    public String getName() { ... }
    public String getStudent() { ... }
    public int numberCorrect() { ... }
    public void addCorrectAnswer(String answer) { ... }
    public void addStudentAnswer(String answer) { ... }
}
```

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 Strings

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 Multiple Choice Quizzes

A few lectures ago, we sorted the characters of a string. Let's sort more:
- Sorting An Integer List
- Sorting A String List

Sorting Integers and Strings

A few lectures ago, we sorted the characters of a string. Let's sort more:
- Sorting An Integer List
- Sorting A String List

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

Implementing A compareTo

```java
public static void sortMCQuizList(List<MCQuiz> list) {
    for (int i = 0; i < list.size(); i++) {
        for (int j = i + 1; j < list.size(); j++) {
            String temp = list.get(minIndex);
            list.set(minIndex, list.get(i));
            list.set(i, temp);
        }
    }
}
```
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:

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

This says, “to be Comparable, classes must define compareTo”.

MCQuiz: Defining compareTo

Attempt #1

```
public class MCQuiz implements Comparable<MCQuiz> {
    ...
    public int compareTo(MCQuiz other) {
        return this.numberCorrect() - other.numberCorrect();
    }
}
```

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

```
public class MCQuiz implements Comparable<MCQuiz> {
    ...
    public int compareTo(MCQuiz other) {
        return (double) this.numberCorrect() / this.correctAnswers.size() -
                 (double) other.numberCorrect() / other.correctAnswers.size();
    }
}
```

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

MCQuiz: Defining compareTo

Attempt #3

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

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

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

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

Client Code to Print The Quizzes

```
List<MCQuiz> quizzes = createQuizzes(2);
// First, let’s get a sorted list of the quizzes
Collections.sort(quizzes);
for (MCQuiz quiz : quizzes) {
    System.out.println(quiz);
}
```

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:

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

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:

```
public class Sample implements Comparable<Sample> {
    public int compareTo(Sample other) {
        return Integer.compare(this.x, other.x);
    }
}
```

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

Object Fields

```
public class Sample implements Comparable<Sample> {
    public int compareTo(Sample other) {
        return this.x.compareTo(other.x);
    }
}
```

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: Set<MCQuiz> (all the quizzes that student took)

```
List<MCQuiz> quizzes = createQuizzes(2);
Map<String, Set<MCQuiz>> quizzesByStudent = new TreeMap<>();
// We want to loop over all the quizzes, adding the one by one
for (MCQuiz quiz : quizzes) {
    String name = quiz.getStudent();
    if (!quizzesByStudent.containsKey(name)) {
        quizzesByStudent.put(name, new TreeSet<>(Arrays.asList(quiz)));
    } else {
        quizzesByStudent.get(name).add(quiz);
    }
}
```

```
System.out.println(student + " has " + quizzesByStudent.get(student));
```

Now, we want to print out the quizzes student by student:

```
for (String student : quizzesByStudent.keySet()) {
    System.out.println(student + " has " + quizzesByStudent.get(student));
}
```
Grouping the Quizzes by Student

The output looks like this:

```java
>> BarbaraHarris: [BarbaraHarris (quiz1): 3/11, BarbaraHarris (quiz0): 4/11]
>> 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 = this.numberCorrect()/this.correctAnswers.size();
        double otherPer = other.numberCorrect()/other.correctAnswers.size();
        if (Double.compare(thisPer, otherPer) == 0) {
            if (this.studentName.compareTo(other.studentName) == 0) {
                return result;
            } else {
                return 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