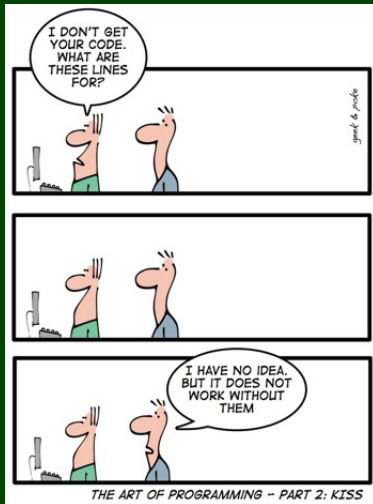


CSE 143

Computer Programming II

Comparable



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

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

We would like to do the two following tasks:

- 1 Print out the quizzes in worst-to-best order (e.g. sort the quizzes)
- 2 Collect all quizzes of each particular student together and display them (still from worst-to-best)

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

Sorting An Integer List

```
1 public static void sortIntList(List<Integer> list) {
2     for (int i = 0; i < list.size(); i++) {
3         int minIndex = i;
4         for (int j = i; j < list.size(); j++) {
5             if (list.get(j) < list.get(minIndex)) {
6                 minIndex = j;
7             }
8         }
9         int temp = list.get(minIndex);
10        list.set(minIndex, list.get(i));
11        list.set(i, temp);
12    }
13 }
```

Sorting A String List

```
1 public static void sortStringList(List<String> list) {
2     for (int i = 0; i < list.size(); i++) {
3         int minIndex = i;
4         for (int j = i; j < list.size(); j++) {
5             if (list.get(j) < list.get(minIndex)) {
6                 minIndex = j;
7             }
8         }
9         String temp = list.get(minIndex);
10        list.set(minIndex, list.get(i));
11        list.set(i, temp);
12    }
13 }
```

Sorting A String List

```
1 if (list.get(j) < list.get(minIndex)) {  
2     minIndex = j;  
3 }
```

compareTo

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:

- 1 Write what we want: `hello < bye`
- 2 Subtract the right from both sides: `hello - bye < 0`
- 3 Replace the subtraction with `compareTo`:
`hello.compareTo(bye) < 0`

That's it!

Sorting A String List

```
1 if (list.get(j).compareTo(list.get(minIndex)) < 0) {  
2     minIndex = j;  
3 }
```

Sorting A MCQuiz List

```
1 public static void sort(List<MCQuiz> list) {  
2     for (int i = 0; i < list.size(); i++) {  
3         int minIndex = i;  
4         for (int j = i; j < list.size(); j++) {  
5             if (list.get(j).numberCorrect() < list.get(minIndex).numberCorrect()) {  
6                 minIndex = j;  
7             }  
8         }  
9         MCQuiz temp = list.get(minIndex);  
10        list.set(minIndex, list.get(i));  
11        list.set(i, temp);  
12    }  
13 }
```

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

Implementing A compareTo

```
1 public int compareTo(MCQuiz other) {  
2     // From above: list.get(j).numberCorrect() < list.get(minIndex).numberCorrect()  
3     // Replacing: this.numberCorrect() < other.numberCorrect()  
4     // Converting: this.numberCorrect() - other.numberCorrect() < 0  
5     return this.numberCorrect() - other.numberCorrect();  
6 }
```

Sorting An MCQuiz List

```
1 if (list.get(j).compareTo(list.get(minIndex)) < 0) {  
2     minIndex = j;  
3 }
```

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:

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

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

Client Code to Print The Quizzes

```
1 List<MCQuiz> quizzes = createQuizzes(2);
2 // First, let's get a sorted list of the quizzes
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”.

Attempt #1

```
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

```
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**.

int Fields

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

```
1 public class Sample implements Comparable<Sample> {  
2     public int compareTo(Sample other) {  
3         return ((Integer)this.x).compareTo(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:

```
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!

Attempt #3

```
1 public class MCQuiz implements Comparable<MCQuiz> {  
2     ...  
3     public int compareTo(MCQuiz other) {  
4         Double thisPer = (double)this.numberCorrect()/this.correctAnswers.size();  
5         Double otherPer = (double)other.numberCorrect()/other.correctAnswers.size();  
6         return thisPer.compareTo(otherPer);  
7     }
```

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     ...  
3     public int compareTo(MCQuiz other) {  
4         Double thisPer = (double)this.numberCorrect()/this.correctAnswers.size();  
5         Double otherPer = (double)other.numberCorrect()/other.correctAnswers.size();  
6         int result = thisPer.compareTo(otherPer);  
7         if (result == 0) { result = this.studentName.compareTo(other.studentName); }  
8         return result;  
9     }
```

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

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).

```
1 List<MCQuiz> quizzes = createQuizzes(2);
2 Map<String, Set<MCQuiz>> quizzesByStudent = new TreeMap<>();
3
4 // We want to loop over all the quizzes, adding them one by one
5 for (MCQuiz quiz : quizzes) {
6     String name = quiz.getStudent();
7     if (!quizzesByStudent.containsKey(name)) {
8         quizzesByStudent.put(name, new TreeSet<MCQuiz>());
9     }
10    quizzesByStudent.get(name).add(quiz);
11 }
12
13 // Now, we want to print out the quizzes student by student:
14 for (String student : quizzesByStudent.keySet()) {
15     System.out.println(student + ": " + quizzesByStudent.get(student));
16 }
```

The output looks like this:

OUTPUT

```
>> AdamBlank: [AdamBlank (quiz1): 1/11, AdamBlank (quiz0): 4/11]
>> BarbaraHarris: [BarbaraHarris (quiz1): 3/11, BarbaraHarris (quiz0): 4/11]
>> ChrisHill: [ChrisHill (quiz0): 3/11, ChrisHill (quiz1): 4/11]
>> JessicaHerna: [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

```
1 public class MCQuiz implements Comparable<MCQuiz> {
2     ...
3     public int compareTo(MCQuiz other) {
4         Double thisPer = (double)this.numberCorrect()/this.correctAnswers.size();
5         Double otherPer = (double)other.numberCorrect()/other.correctAnswers.size();
6         int result = thisPer.compareTo(otherPer);
7         if (result == 0) {
8             result = this.studentName.compareTo(other.studentName);
9         }
10        if (result == 0) {
11            result = this.quizName.compareTo(other.quizName);
12        }
13        return result;
14    }
```

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

- 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