CSE 143X
Accelerated Computer Programming I/II
Comparable

I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?

I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM

THE ART OF PROGRAMMING - PART 2: KISS
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**

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

    public MCQuiz(String filename) throws FileNotFoundException {
        // ... }

    public String getStudent() {
        // ... }

    public String getName() {
        // ... }

    public int numberCorrect() {
        // ... }
}
```

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

```java
public static void sortIntList(List<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        int minIndex = i;
        for (int j = i; j < list.size(); j++) {
            if (list.get(j) < list.get(minIndex)) {
                minIndex = j;
            }
        }
        int temp = list.get(minIndex);
        list.set(minIndex, list.get(i));
        list.set(i, temp);
    }
}
```

Sorting A String List

```java
public static void sortStringList(List<String> list) {
    for (int i = 0; i < list.size(); i++) {
        int minIndex = i;
        for (int j = i; j < list.size(); j++) {
            if (list.get(j) < list.get(minIndex)) {
                minIndex = j;
            }
        }
        String temp = list.get(minIndex);
        list.set(minIndex, list.get(i));
        list.set(i, temp);
    }
}
```
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!
Strings were easier, because **they knew how to compare themselves**.

**Implementing A compareTo**

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

**Sorting An MCQuiz List**

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

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

This says, “to be Comparable, classes must define compareTo”.
Client Code to Print The Quizzes

```java
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:

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

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

MCQuiz: Defining compareTo

**Attempt #1**

```java
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**

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

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

In other words, just use the existing compareTo on the field in the class!
**MCQuiz: Defining compareTo**

Attempt #3

```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();
        return thisPer.compareTo(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

```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); }
        return result;
    }
}
```

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).

```java
List<MCQuiz> quizzes = createQuizzes(2);
Map<String, Set<MCQuiz>> quizzesByStudent = new TreeMap<>();

// We want to loop over all the quizzes, adding them one by one
for (MCQuiz quiz : quizzes) {
    String name = quiz.getStudent();
    if (!quizzesByStudent.containsKey(name)) {
        quizzesByStudent.put(name, new TreeSet<MCQuiz>());
    }
    quizzesByStudent.get(name).add(quiz);
}

// Now, we want to print out the quizzes student by student:
for (String student : quizzesByStudent.keySet()) {
    System.out.println(student + " : " + quizzesByStudent.get(student));
}
```
The output looks like this:

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

```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
CSE 143X
Accelerated Computer Programming I/II
Hashing

"the eagle flies at midnight"

2886dba4
c8c519f1
e6e44416
9580f18b
Today, we will consider multiple new implementations of IntSet:

```java
defaulpublic interface IntSet {
    public void add(int value);
    public void remove(int value);
    public boolean contains(int value);
}
```
Design a class RangeSet that represents a set which only allows numbers inside a **fixed range**.

You should have a constructor:

| RangeSet(max) | This constructor initializes a new RangeSet which only allows elements between 0 (inclusive) and **max** (exclusive). |

And the following **public** methods:

| add(val) | Adds **val** to the RangeSet if it is a valid value and throws an IllegalArgumentException otherwise. |
| remove(val) | Removes **val** to the RangeSet if it is a valid value in the set and does nothing otherwise. |
| contains(val) | Returns true if **val** is in the RangeSet and false otherwise. |

**add**, **remove**, and **contains** must all be $O(1)$
public class RangeSet implements IntSet {
    private boolean[] data;

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

    public void add(int value) {
        if (value >= this.data.length || value < 0) {
            throw new IllegalArgumentException();
        }
        this.data[value] = true;
    }

    public boolean contains(int value) {
        if (value >= this.data.length || value < 0) {
            return false;
        }
        return this.data[value];
    }

    public void 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></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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];
    }

    private int hashCode(int value) {
        return value % data.length;
    }

    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;
    }
}
HashSet Attempt #1 Problem!

Consider the following insertions: 1, 21

First:

Then:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & \ldots \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & \ldots \\
\end{array}
\]

Uh oh! We’ve overwritten the one!

How can we fix this?

Instead of storing an integer, let’s store a list of integers

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & \ldots \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & \ldots \\
\end{array}
\]
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;
}