

CSE 143X

Accelerated Computer Programming I/II

Comparable



Storing Multiple Choice Quizzes

The text file:

- 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 class MCQuiz {
    private String classroom;
    private String section;
    private List<String> questions;
    private List<String> studentAnswers;

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

    public String getAnswer() { ... }
    public String getSection() { ... }
    public List<String> getQuestions() { ... }
}
```

We would like to do the two 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 Integers and Strings

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

```
Sorting An Integer List
public static void sortFromBestToWorstIntegerList (
    List<Integer> list) {
    for (int i = list.size() - 1; i > 0; i--)
        for (int j = i + 1; j < list.size(); j++)
            if (list.get(i).compareTo(list.get(j)) > 0)
                swap(i, j);
    }

    List temp = list.toArray();
    list.sort(Comparator.reverseOrder());
    list.addAll(temp);
}
```

```
Sorting A String List
public static void sortFromBestToWorstStringList (
    List<String> list) {
    for (int i = list.size() - 1; i > 0; i--)
        for (int j = i + 1; j < list.size(); j++)
            if (list.get(i).compareTo(list.get(j)) > 0)
                swap(i, j);
    }

    List temp = list.toArray();
    list.sort(Comparator.reverseOrder());
    list.addAll(temp);
}
```

Sorting Strings

```
Sorting A String List
if (list.get(i).compareTo(list.get(j)) > 0)
    swap(i, j);
}
```

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:

- 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(j)) > 0)
    swap(i, j);
}
```

Sorting Multiple Choice Quizzes

Sorting A MCQuiz List

```
public static void sortFromBestToWorstMCList (
    List<MCQuiz> list) {
    for (int i = list.size() - 1; i > 0; i--)
        for (int j = i + 1; j < list.size(); j++)
            if (list.get(i).compareTo(list.get(j)) > 0)
                swap(i, j);
    }

    List temp = list.toArray();
    list.sort(Comparator.reverseOrder());
    list.addAll(temp);
}
```

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

Implementing A compareTo

```
public int compareTo(MCQuiz other) {
    // From above: list.get(i).compareTo(list.get(j))
    // Meaning: this.numberCorrect() - other.numberCorrect()
    // Casting: this.numberCorrect() - other.numberCorrect() = 0
    return this.numberCorrect() - other.numberCorrect();
}
```

Sorting An MCQuiz List

```
if (list.get(i).compareTo(list.get(j)) > 0)
    swap(i, 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".

```
Comparable
The Comparable interface allows us to tell Java how to sort a type of object:
1 public interface Comparable<T> {
2     public int compareTo(T other);
3 }
This says, "to be Comparable, classes must define compareTo".
```

Printing The Quizzes in Order

```
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<T> {
2     public int compareTo(T other);
3 }
This says, "to be Comparable, classes must define compareTo".
```

MCQuiz: Defining compareTo

Attempt #1

```
1 public class MCQuiz implements Comparable<MCQuiz> {
2     ...
3     public int compareTo(MCQuiz other) {
4         return this.numberCorrect() - other.numberCorrect();
5     }
6 }
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 }
7 }
This won't even compile! We need to return an int.
```


```

Comparable: Tricks #1 & #2

Int Trick

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 Simple implements Comparable<Simple> {
2     public int compareTo(Simple other) {
3         return ((Integer)this.x).compareTo(other.x);
4     }
5 }

```

Object Trick

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 Simple implements Comparable<Simple> {
2     public int compareTo(Simple other) {
3         return this.x.compareTo(other.x);
4     }
5 }

```

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

MCQuiz: Defining compareTo

Attempt #3

```
1 public class MCQuiz implements Comparable<MCQuiz> {
2     ...
3     public int compareTo(MCQuiz other) {
4         double thisPr = (double)this.numberCorrect()/this.correctAnswers.size();
5         double otherPr = (double)other.numberCorrect()/other.correctAnswers.size();
6         return thisPr.compareTo(otherPr);
7     }
8 }
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 thisPr = (double)this.numberCorrect()/this.correctAnswers.size();
5 double otherPr = (double)other.numberCorrect()/other.correctAnswers.size();
6 int result = thisPr.compareTo(otherPr);
7 if (result == 0) { result = this.studentName.compareTo(other.studentName); }
8 return result;
9 }
10 }
This still doesn't work, but it's not as clear why. Let's try the second task.
```


```

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).

```
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 }
```

Grouping the Quizzes by Student

11

The output looks like this

```

1 // Student: [Student {quiz1= 1/1, quiz2= 4/10}]
2 // StudentTo: [StudentTo {quiz1= 2/1, quizScore: 4/10}]
3 // StudentTo: [StudentTo {quiz1= 2/1, quizScore: 4/10}]
4 // StudentTo: [StudentTo {quiz1= 1/1, quizScore: 2/1}]
5 // StudentTo: [StudentTo {quiz1= 4/10}]

```

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.quizScore() / this.getMaxScore();
5         double otherPer = (double) other.quizScore() / other.getMaxScore();
6         int result = (int) (other.compareTo(this));
7         if (result == 0) {
8             result = this.studentName.compareTo(other.studentName);
9         }
10        if (result == 0) {
11            result = this.quizScore().compareTo(other.quizScore());
12        }
13        return result;
14    }
15 }

```

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



IntSet

Today, we will consider multiple new implementations of IntSet:

```

1 public interface IntSet {
2     public void add(int value);
3     public void remove(int value);
4     public boolean contains(int value);
5 }

```

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:

<code>RangeSet(max)</code>	This constructor initializes a new <code>RangeSet</code> which only allows elements between 0 (inclusive) and <code>max</code> (exclusive).
----------------------------	---

And the following **public** methods:

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

`add`, `remove`, and `contains` must all be $O(1)$.

RangeSet

```

1 public class RangeSet implements IntSet {
2     private boolean[] data;
3
4     public RangeSet(int max) { this.data = new boolean[max]; }
5
6     public void add(int value) {
7         if (value >= this.data.length || value < 0) {
8             throw new IllegalArgumentException();
9         }
10        this.data[value] = true;
11    }
12
13    public boolean contains(int value) {
14        if (value >= this.data.length || value < 0) {
15            return false;
16        }
17        return this.data[value];
18    }
19
20    public void remove(int value) {
21        if (value < this.data.length && value >= 0) {
22            this.data[value] = false;
23        }
24    }
25 }

```

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 (Hash Code)

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 `R`, consider the following hashcode:

```

1 public int hashCode(int value) {
2     return value % R;
3 }

```

Now, let's insert the following data: 0, 4, 13

0	4	13
0	1	4
1	2	5
2	3	6
3	4	7
4	5	8
5	6	9
6	7	10
7	8	11
8	9	12
9	10	13

HashSet Attempt #1

5

```

1 public class IntHashSet implements Set<Int> {
2     public final int DEFAULT_SIZE = 20;
3     public Integer[] data;
4
5     public IntHashSet() {
6         this.data = new Integer[DEFAULT_SIZE];
7     }
8
9     private int hashCode(int value) {
10        return value % data.length;
11    }
12    public void add(int value) {
13        this.data[hashCode(value)] = value;
14    }
15
16    public boolean contains(int value) {
17        return this.data[hashCode(value)] == value;
18    }
19
20    public void remove(int value) {
21        this.data[hashCode(value)] = null;
22    }
23 }

```

String hashCode()

7

```

1 public int hashCode() {
2     int h = hash;
3     if (h == 0 && value.length > 0) {
4         char val[] = value;
5
6         for (int i = 0; i < value.length; i++)
7             h = 29 * h + val[i];
8     }
9     hash = h;
10    return h;
11 }
12 }

```

HashSet Attempt #1 Problem!

6



Consider the following insertions: 1, 21

First:

Then:

Uh oh! We've overwritten the one!

How can we fix this?

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

