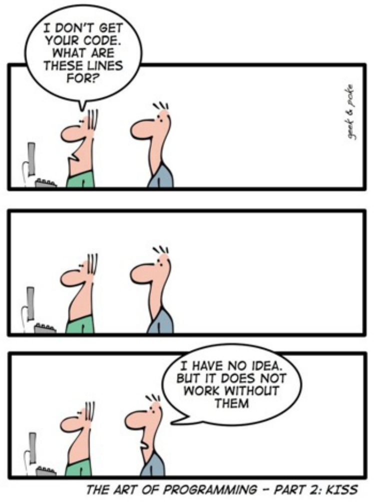
CSE 143

Computer Programming II

Interfaces & Comparable


I DON'T GET YOUR CODE. WHAT ARE THESE LINES FOR?
geek & poke
I HAVE NO IDEA. BUT IT DOES NOT WORK WITHOUT THEM
THE ART OF PROGRAMMING – PART 2: KISS

Outline

1. Understand How To Use Interfaces

2. Learn about the `Comparable` Interface

Interfaces

Interface

An **interface** specifies a group of behaviors and gives them a name. Classes can choose to **implement** interfaces which require them to implement all of the methods in the interface.

Interfaces answer the question:

> “To be an **X**, which methods does another class need to have?”

IntList Interface

For example: To be a **List**, which methods does another class need to have?

- Lists have an `add` method
- Lists have a `remove` method
- Lists have a `get` method
- Lists have a `set` method
- Lists have a `size` method
- . . .

Normally, we specify a method **and** its implementation. Java allows us to just specify the header:

“`public String toString();`”

is a valid line of code.

Interface Syntax

To **Specify** An Interface

```java
public interface IntList {
   public void add(int value);
   public int remove(int index);
   public int get(int index);
   public void set(int index, int element);
   public int size();
   public boolean isEmpty();
}
```

To **Use** An Interface

Edit the first line of a class (say `ArrayIntList` or `LinkedIntList`):

- `public class ArrayIntList implements IntList {...}`
- `public class LinkedIntList implements IntList {...}`

Also, make sure it actually has all the methods the interface is supposed to have. . .

Outline

1. Understand How To Use Interfaces

2. Learn about the `Comparable` Interface

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:

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

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

Thinking about compareTo

Think about the following transformation when using compareTo:

```
this.compareTo(that) < 0
this - that < 0
this < that
```

This works if you replace < with =, >, !=, . . . :

Normal		compareTo
a < b	is to	a.compareTo(b) < 0
a <= b	is to	a.compareTo(b) <= 0
a == b	is to	a.compareTo(b) == 0
a != b	is to	a.compareTo(b) != 0
a >= b	is to	a.compareTo(b) >= 0
a > b	is to	a.compareTo(b) > 0

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

```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
2. Collect all quizzes of each particular student together and display them (still from worst-to-best)

Printing The Quizzes in Order

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**.

`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:

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

```
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

```
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.

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: **String**s (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));
}
```

Grouping the Quizzes by Student

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

```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 Interface/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