Project 4:

In this project we're going to build our own word processor. In the past, when I've specified APIs for projects, some of you have gotten frustrated because you barely named a method wrong and it didn't work with my code tester.

Well, good news everyone! I've included a file with this project that will yell at you at compile-time if you mess up the method names or return types. This file is still a .java file, but it's called an interface. The interface specifies all the methods and their return types for the whole project. If your project doesn't match it exactly, java will politely inform you by throwing a bunch of error text in your face, how nice!

Our interface for this project looks like this:

```
public interface Ed
{
    String getFirst();
    String getRest();
    Ed rightArrow();
    Ed leftArrow();
    Ed delete();
    Ed insertString(String c);
    Ed homeKey();
    Ed endKey();
    String toString();
}
```

This format should look really familiar to something that we did before. I'll give you a minute to think about it. Seriously, don't move ahead until you've had a second to think…

If you were thinking an abstract superclass, you're right! An interface is the exact same as an abstract class with strictly abstract methods. Not all methods in an abstract class have to be abstract, but if they are, you're better off using an interface instead. Remember, abstract methods are the only time it's ever okay to end the method signature with a semicolon.

In order to use this interface, we need to tell the class we're working with that it should match this interface. In order to do this, you need to have your class signature say: `public class Editor implements Ed`. Note that while we `extend` superclasses, we `implement` interfaces.

So lets get to these actual methods. In Editor, the class we're writing, our text content is stored as two Strings: first and rest. The first String represents all the text that is before the cursor and the rest String represents all the text that is after the cursor. The getFirst() and getRest() methods are used for EditorPanel.java in order to test that your editor is working correctly. All they have to do is return the text that's before the cursor and return the text after the cursor respectively.

The rest of the methods, save one, all return a new Ed object. Since our Editor implements Ed, all we have to do is return a new Editor object to represent the current state of the Editor. Since we'll be creating so many Editor objects, we need to have a nice Constructor to create these objects. If we create new Editors by passing in the first and rest strings, then that's all the information we need.

As such, inside each method that returns an Ed, you'll create a new Editor through the constructor and return that. So for each of those, you'll take the first and rest Strings and modify them appropriately, and create a new Editor with the new first and rest values and return that. For examples, I'll use the convention (first, rest).

Method	Before	After
rightArrow()	("That's a nice ", "doggy")	("That's a nice d", "oggy")
leftArrow()	("That's a nice d", "oggy")	("That's a nice ", "doggy")
delete()	("That's a ", "nice doggy")	("That's a ", "ice doggy")
backspace()	("That's", " a nice doggy")	("That'", " a nice doggy")
insertString("F")	("Th", "at's a nice doggy")	("ThF", "at's a nice doggy")
homeKey()	("That's a", " nice doggy")	("", "That's a nice doggy")
endKey()	("That's a", " nice doggy")	("That's a nice doggy", "")

The last method, toString() should return a string that looks like first|rest. Where "|" is shift + backslash.

To submit your project, I need your whole project folder in an archive. Make sure you have 7zip or an appropriate archiving program installed, select your project folder and create an archive from that folder. Your archive must have the naming convention PeriodLastFirstProject7.7z (or .zip or .rar…).

Once you have the archive created, you can submit it through the class website.