Assignment #7: CellSpreadSheet

Due Dates:

7A: Sunday, Dec. 12, 11:59 PM ET

7B: Tuesday, Dec. 16, 11:59 PM ET

Overview

As we said in the past, in the real world programs are not written once and then forgotten, rather they are maintained, features are added, bugs are fixed, etc., over time.

In that spirit, in this assignment you will write a new version of the SpreadSheet program, this time using a more Object Oriented approach to the code. Most of your existing logic will still work and will not change much, but it will be organized differently. In other words, **there is very little new <u>logic</u> per se you have to write for this assignment**, rather this assignment is all about using Java's Object Oriented syntax/features. In this assignment you will practice:

1) Writing a program that is made up of multiple classes, with all that comes with that in terms of thinking in a more Object Oriented way regarding how work gets divided between classes
2) Writing, running, and testing a program that is spread across multiple .java files
3) Implementing interfaces
4) Writing and using constructors
5) Unit testing your code

What You Will Submit

You must submit 3 .java files:

1) `CellSpreadSheet.java`
2) `DoubleCell.java`
3) `FormulaCell.java`

Two other files will be needed to compile and run your program (`Cell.java` and `CellProvider.java`) but those make not be changed <u>at all</u> from the interfaces I provided you, so I will simply use my own copies of those files to compile and run your program.

Changes from Assignment 6

Changes to the Spread Sheet Public API

See the API section below. From the perspective of someone <u>using</u> the program, i.e. another programmer creating an instance of the class and making calls, there are only 3 changes:

1) The name of the class has been changed to `CellSpreadSheet`
2) The order of the constructor <u>arguments</u> have been flipped – columns comes first and rows comes second
3) We no longer deal with Object arrays (i.e. one or two dimensional arrays whose data type is `java.lang.Object`). Instead, the data is stored as arrays of `Cell` objects, and wherever an `Object[]` or `Object[][]` appeared in assignment #6, a `Cell[]` or `Cell[][]` is used instead in this assignment.

Look at the demo program at the bottom of this document and compare it to the one from assignment #6, and you will see very little difference.

Refactoring of the Code

The code of your solution, however, has to be reorganized /changed to meet the new requirements we will now describe. Such a reorganization of code is often referred to as "refactoring." The changes include the following:

1) As stated above, each cell in the spreadsheet is now stored as an instance of a class that implements the `Cell` Interface. The interface is copied below.

```
public interface Cell {
    double getNumericValue();
    String getStringValue();
}
```

2) You must write **two classes that implementation** the `Cell` interface, one called `DoubleCell` which will be used for cells in the spreadsheet whose value is a **`double`**, and one called `FormulaCell` which will be used for cells in the spreadsheet whose value is a spreadsheet formula. If you do not recall the details of how one implements an interface, review lecture #16 and/or the relevant readings in the book.
3) `DoubleCell` must:
    a. have one constructor, which takes one parameter, a **`double`**, which is the value of this cell. `DoubleCell` must store this value in a **private** instance variable
    b. `DoubleCell.getNumericValue()` must return the value that was passed in to its constructor
    c. `DoubleCell.getStringValue()` must return the result of calling `java.lang.Double.toString(double)` to convert the value that was passed in to its constructor to a String
4) `FormulaCell` must:
    a. Have one constructor which takes 2 parameters – the first parameter is a String which contains the formula, and the second parameter is an instance of a class that implements the `CellProvider` interface. This interface is implemented by your `CellSpreadSheet` class (more on that below), and it should be your `CellSpreadSheet` instance that is passed in as this second parameter. Both parameters must be stored as **private** instance variables.
    b. `FormulaCell.getStringValue()` simply returns the formula String
    c. `FormulaCell.getNumericValue()` calls the `CellProvider.getCell()` method twice to get the values of the formula's two operands, does the math specified by the formula, and returns the result of the math. `FormulaCell` must not store/keep the `Cell` objects returned by the `CellProvider` in instance variables to reuse later. Instead it must call `CellProvider.getCell()` twice every time `FormulaCell.getNumericValue()` is called, since in between two times it's called the `Cell` objects in the spread sheet might've been replaced by new `Cell` objects, thus rendering obsolete/wrong any old `Cell` objects that `FormulaCell` would've held on to as the value of the cells referenced in the formula.
5) `CellSpreadSheet` must implement the `CellProvider` interface (copied below). `CellSpreadSheet` does not figure out the numeric or String value of any `Cell`, rather it simply calls `Cell.getNumericValue()` and `Cell.getStringValue()` to get values of cells. The logic from your `ArrayBasedSpreadSheet` in assignment #6 that figured out the numeric value of a cell that had a formula in it must not exist in your `CellSpreadSheet` class. Instead, `CellSpreadSheet` will simply call `Cell.getNumericValue()` to get **all** numeric values, and if a given `Cell` is an instance of `FormulaCell`, then the `FormulaCell` instance will make calls back to `CellSpreadSheet` via the method defined in the `CellProvider` interface to get the formula's operands.

```
public interface CellProvider {
    Cell getCell(char column, int row);
}
```

6) When `CellSpreadSheet.setValue(char column, int row, String value)` is called, `CellSpreadSheet` must create an instance of either `DoubleCell` or `FormulaCell`, depending on whether `value` parses as a double or not, and save it as the value of `data[column][row]` in the spreadsheet.

7) Calling `CellSpreadSheet.evaluateFormula` must return the same value as calling `CellSpreadSheet.getValue` on the same column and row.

WARNINGS

1) You may not make **ANY** changes to the interfaces I have provided you
2) Instances of `DoubleCell` may not have any references to **any** other objects – all they have saved as an instance variable is one **`double`**
3) `FormulaCell` instances **must not** be aware of the fact, or take any advantage of the fact, that the `CellProvider` passed in to their constructor is an instance of `CellSpreadSheet`, nor should it even be aware that the `CellSpreadSheet` class exists at all. You should assume that in my tests I will create a separate implementation of `CellProvider` and test your `FormulaCell` implementation using my `CellProvider` implementation. Stick to the API/interfaces, and you will be fine.

API

In your 2D Array, the first dimension must be the columns and the second dimension must be the rows

```
    /**
     * Initializes the spreadsheet to be a given height and width
     * @param rows
     * @param columns
     */
    public CellSpreadSheet(int columns, int rows)


    /**
     * @param showFormulas if true, show the raw cell formulas. If false, show the value generated
by calculating the formulas.
     * @return a String, comma separated values representation of the entire spreadsheet. Each line
ends with "\n"

     */
    public String getSpreadSheetAsCSV(boolean showFormulas)


    /**
     *
     * Sets a cell to given value, where the value is either a double or a cell formula.
     * Formulas are written as two cell references separated by a basic math operation. There must
be a space
     * between each of the 3 elements, i.e. a space between the first cell reference and operation,
and then another
     * space between the operation and the second cell reference. Example formulas:
     * Addition: A1 + C3
     * Subtraction:  D7 - F6
     * Multiplication: F15 * Z9
     * Division: B6 / C2
     *
     * If the column or row is beyond the current bounds of the 2D array storing the spreadsheet,
the size of the 2D array must be expanded as needed to store the value at column, row.
     *
     * @param column column whose value to set, between 'A' and 'Z'
     * @param row row whose value to set. An integer >= 1
     * @param value must be a string representation of a double or a cell formula
     */
    public void setValue(char column, int row, String value)


    /**
     * Returns a complete copy of the spreadsheet data. Since it is a COPY, any edits to the copy
will have no effect on the spreadsheet itself.
     * @return a complete copy of the spreadsheet data
     */
    public Cell[][] getCopyOfData()



    /**
     * Expand the spreadhseet to extend it to the given column.
     * @param column
     */
    public void expandColumnRange(char column)


/**
 * Returns a copy of a given column through a given row.
 * If throughRow is larger than the current height of the column, all the values in the rows past
the current column height will be null.
 * @param c the column to get, between 'A' and 'Z'
 * @param throughRow the row to get a copy through.
 * @return
 */
public Cell[] getCopyOfColumnThroughRow(char c, int throughRow)



    /**
     *
     * @param column (first column is A)
     * @param row (first row is 1)
     * @return value stored in that cell. If it holds a double, return the double. If it holds a
formula, return the result of calculating the formula stored in the cell. If the cell is empty, it
returns 0.
     */
    public double getValue(char column, int row)



    /**
     *
     * @param cell the cell whose value should be returned. E.g. "A1" or "F9", etc.
     * @return
     */
    public double getValue(String cell)



    /**
     * Evaluate the formula held in the given cell
     * @param column 'A' through 'Z'
     * @param row row number, 1 or greater
     * @return
     */
    public double evaluateFormula(char column, int row)
```

Sample Program Which Uses the CellSpreadSheet

```
public class Assignment7Demo {
    public static void main(String[] args) {
        CellSpreadSheet cellsheet = new CellSpreadSheet(6,6);
        //set a few values, print it out
        cellsheet.setValue('A',1,"100");
        cellsheet.setValue('F',5,"55");
        System.out.println(cellsheet.getSpreadSheetAsCSV(true));
        //fill in some data on column C
        cellsheet.setValue('C',1,"10");
        cellsheet.setValue('C',2,"11");
        cellsheet.setValue('C',3,"12");
        cellsheet.setValue('C',4,"13");
        cellsheet.setValue('C',5,"14");
        cellsheet.setValue('C',6,"15");
        //print out again, showing new values
        System.out.println(cellsheet.getSpreadSheetAsCSV(true));
        cellsheet.expandColumnRange('H');
        //print out again, showing expanded spreadsheet
        System.out.println(cellsheet.getSpreadSheetAsCSV(true));
        //print out some values
        System.out.println("Value of C3 is: " + cellsheet.getValue('C',3));
        System.out.println("Value of F5 is: " + cellsheet.getValue('F',5));
        System.out.println("Value of B3 is: " + cellsheet.getValue('B',3) + "\n");
        //set some cells to formulas
        cellsheet.setValue('D',1,"C1 * F5");
        cellsheet.setValue('E',1,"D1 / C2");
        cellsheet.setValue('A',2,"A1 + C3");
        cellsheet.setValue('B',3,"C6 - C1");
        System.out.println("The value of the formula stored in D1 is: " +
cellsheet.evaluateFormula('D',1) + "\n");
        System.out.println(cellsheet.getSpreadSheetAsCSV(true));
        System.out.println(cellsheet.getSpreadSheetAsCSV(false));

    }
}
```

Output of Sample Program

```
A,B,C,D,E,F
100.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,55.0
0.0,0.0,0.0,0.0,0.0,0.0

A,B,C,D,E,F
100.0,0.0,10.0,0.0,0.0,0.0
0.0,0.0,11.0,0.0,0.0,0.0
0.0,0.0,12.0,0.0,0.0,0.0
0.0,0.0,13.0,0.0,0.0,0.0
0.0,0.0,14.0,0.0,0.0,55.0
0.0,0.0,15.0,0.0,0.0,0.0

A,B,C,D,E,F,G,H
100.0,0.0,10.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,11.0,0.0,0.0,0.0,0.0,0.0
```

```
0.0,0.0,12.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,13.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,14.0,0.0,0.0,55.0,0.0,0.0
0.0,0.0,15.0,0.0,0.0,0.0,0.0,0.0

Value of C3 is: 12.0
Value of F5 is: 55.0
Value of B3 is: 0.0

The value of the formula stored in D1 is: 550.0

A,B,C,D,E,F,G,H
100.0,0.0,10.0,C1 * F5,D1 / C2,0.0,0.0,0.0
A1 + C3,0.0,11.0,0.0,0.0,0.0,0.0,0.0
0.0,C6 - C1,12.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,13.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,14.0,0.0,0.0,55.0,0.0,0.0
0.0,0.0,15.0,0.0,0.0,0.0,0.0,0.0

A,B,C,D,E,F,G,H
100.0,0.0,10.0,550.0,50.0,0.0,0.0,0.0
112.0,0.0,11.0,0.0,0.0,0.0,0.0,0.0
0.0,5.0,12.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,13.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,14.0,0.0,0.0,55.0,0.0,0.0
0.0,0.0,15.0,0.0,0.0,0.0,0.0,0.0
```