CSE 11 **START EARLY!**
Spring 2015 Program Assignment #4 (100 points)

Due: 1 May 2014 at 1159pm (2359, Pacific Standard Time)

PROGRAM #4 : AsciiArt

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

In lecture we've discussed multidimensional arrays and `ArrayList`. In this homework, you will work with multiple classes to be able to create some ASCII-based (text-based) art. There are three public classes.

The first class is called `AsciiShape`. This class is being provided to you and may not be modified. It enables the user to define rectangle and triangle "shapes" using 2D `char` arrays. You should generate the Javadoc for this code to read the details of the constructors and methods.

A second class called `AsciiGrid` defines a 2D "grid" in which the user can place (and) clear shapes. You may think of this as a drawing canvas of characters. It has a fixed size (at construction). You are provided a skeleton of this class and should use Javadoc to create the full documentation for the public methods and constructors.

You will create a program called `AsciiArt` that allows you interactively create `AsciiShapes` (shapes), place them on an `AsciiGrid` (grid) them, clear areas from the grid, and print the grid. `AsciiArt` will support working with multiple shapes.

The goals of this assignment are 1) to become comfortable with using multiple classes and better understanding when Object references are useful, 2) Take advantage of the `ArrayList` class to keep track of user-defined shapes during an interactive `AsciiArt` session 3) become more comfortable with reading javadoc-created documentation and using it to guide implementation. This program builds on what you have learned in program #3.

In java, it object references are literally everywhere. This program will require you to better understand reference vs. copy

The `AsciiGrid` class

You are being provided a skeleton of the `AsciiGrid` class that has been commented using Javadoc-style comments. Your first step should be to generate the documentation using the javadoc command-line tool.

<u>When you Implement AsciiGrid, **You may NOT**</u>

1. Change the signature of any `public` method or constructor

2. Add any new `public` methods, you may (and probably will) add `private` methods
3. Define any `public` instance or class variables

In grading your assignment, we will write our test programs to utilize only your implemented `AsciiGrid` class to test its functionality. We will then test your `AsciiArt` program against our implementation of `AsciiGrid` and finally we will test your two classes together.

You should also note, that in the skeleton, `boolean` methods always return `false.` That is clearly incorrect in the full implementation. Those return values in the skeleton are present so that the initial code will compile. For clarity, we will list the constructors and methods defined in `Image11` with a brief description

Constructor Summary

Constructors

Constructor and Description
`AsciiGrid()` Constructor
`AsciiGrid(int row, int col)` Constructor

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
`boolean`	`clearShape(char[][] shape, int r, int c)` clear the blocks in the grid defined by the 2D shape starting at grid at location (r,c).
`char[][]`	`getChars()` return a row x col array of the current char array This should be a full/deep copy, not a re internal storage
`int[]`	`getSize()` Return the width and height of the grid

`boolean`	**`placeShape`**`(char[][] shape, int r, int c)` place the 2D shape in the grid at location (r,c) only if every row of the shape fits in the gri doesn't intersect any non-empty spaces.
`java.lang.String`	**`toString`**`()` create a nice, printable representation of the grid and filled coordinates

For methods that return `boolean`, they should return `true` if method was successful, `false` if an error would occur. For example, if you attempted to place a shape that either runs into non-empty cells or would not completely fit on the grid. you should return `false`**. Your `AsciiGrid` implementation should never generate a runtime error.** Nor should it print any error messages to the screen (though you will find such error messages useful during debugging)

You should note that `clearShape`() operates on the current state of the grid. It only uses the "extent" (dimensions of each row of its shape argument) to define which elements in the grid to reset to the empty character.

Other requirements of AsciiGrid

1. The EMPTY character is defined as a space (' '). You must define a constant for this
2. The default size of a grid when created with 0-argument constructor is 25 rows x 40 columns
3. The upper-left of the grid is considered to be (0,0). Just like graphics coordinates
4. `getChars()` is not needed to implement `AsciiArt,` it WILL be used for automated testing of your class implementation

The AsciiArt Program

This is a java program and you must therefore define a `main()` method. Its job is to take commands from the user and behave appropriately. Your program must keep track of every created shape. Each created shape has a numerical ID with it. The first shape created has ID 0 and the IDs are incremented by one for each new shape created. Shapes are never deleted. The user can list these shapes to remind him/her what shapes are available for placing on the grid. It is highly recommended that you use `ArrayList` to keep track of your list of defined shapes. The commands that AsciiArt must understand are

Command	**Arguments**	**Action**
triangle	*symbol row col*	Create a triangle shape (using `AsciiShape`) using symbol as the character for the chars in the shape. If the user types in more than 1 character for symbol, use the first character typed.
rectangle	*symbol row col*	Create a rectangle shape (using `AsciiShape`) using symbol as the character for the chars in the shape. If the user types in more than 1 character for symbol, use the first character typed.

list		List the shapes by printing the index of the shape and the shape itself. Each shape is assigned an index starting at 0.
new		Create a new AsciiGrid using the default constructor
new	*row column*	Create a new AsciiGrid that is row x column
print		Print the current state of the grid. See the Javadoc for framing
place	*idx row col*	Place the shape with ID idx on the grid starting at coordinate (row,col)
clear	*idx row col*	clear the area defined by shape with ID idx from the grid starting at coordinate (row,col)
set	*symbol idx*	Set the symbol of the shape with ID idx to the new symbol
exit	*0 or more args*	Exit the program

Initialization
When your program begins, there is no grid and no defined shapes, any operation, except exit, list, triangle, rectangle should result in a BAD INPUT message to the screen until at program initialization.

Command Loop

The program should print out a command prompt of "> ", (note space that follows the '>') and then wait for user input. If a command is successful, it should print OK, otherwise is should print BAD INPUT. Note, arguments to commands are not always integers. If a command is supposed to have integer arguments, we will not supply non-integer arguments. You will not, for example, have to properly handle a bad input string like "set eleven zero". The "type" of the first argument changes depending on the command issued. You should check if the number of arguments to a particular command is correct before attempting to convert the arguments.

The program should not create a new Scanner instance each time through the command loop. It should create a Scanner instance once. This is so the we can test your program using automated procedures.

Case insensitivity
Commands are insensitive to case. That is "new", "NEW", "New", "neW" are all valid forms of the new command. Be smart and use a String built in method to make this easy. We might even put spaces *before* a command is entered, your program should handle this condition. (look at String for a helpful method)

Whitespace insensitivity
Commands and their arguments can be separated by more than one space. For example, "place 4 18 24" and "   place   4   18   24   " are both valid input strings. See below in hints.

Definition of "BAD INPUT"

1. A command is given the wrong number of arguments.

2. If the form of the command is ok, then BAD INPUT should be displayed if the `AsciiGrid or AsciiShape` method that was invoked returned an indication of error (See the Javadoc-created documentation)
3. exit followed by any number of arguments is not BAD INPUT. It should exit the program
4. When defining a triangle, rectangle or setting the symbol for an existing shape, if the user types in more than one character for the symbol, just use the first character.
5. It's not possible with the definition of the program to set the symbol of any shape to a white-space. You are not expected to be able to accomplish this

Hints:

1. You will likely find it easier to read a complete line of input and then extract the command and any arguments. The `nextLine()` method in the `Scanner` class is likely useful to you.
2. We haven't covered regular expressions (and may not this quarter), but to split a String (e.g. input) into an array of Strings separated by arbitrary amounts of white space, you can do the following in your code

   ```
   String [] args = input.split("\\s+");
   ```
3. There are several ways to code the main command loop, but `if … else if .. else if … else`, might be very convenient for you.
4. Define helper functions when you need them.
5. Before placing a shape, there are two logical conditions that must be satisfied: 1) the shape must fit on the grid and 2) if it fits, it shouldn't try to set any non-empty coordinates in the grid. Only if both of those two conditions are met, can the shape be placed successfully. Those tests are moderately complicated, write private helper methods for each logical operation. clearShape needs to only check if the shape fits on the grid.
6. Think about handling command arguments. Sometimes the arguments are "`String int int`" sometimes "`int int int`" sometimes "`String int`", etc. Notice that the first argument might or might not be an int. Any arguments past the first one are always ints (for valid commands!).
7. The `getShape()` method of `AsciiShape` is useful for sending data about a particular `AsciiShape` instance to some `AsciiGrid` methods

Example Sessions of Running AsciiArt

1. Define and list some shapes

```
$ java AsciiArt
> triangle $ 3 5
OK
> rectangle @ 4 4
OK
> list
0:
$
$$
$$$$$
1:
@@@@
@@@@
@@@@
```

```
@@@@
OK
> set X 0
OK
> list
0:
X
XX
XXXXX
1:
@@@@
@@@@
@@@@
@@@@
OK
> exit
OK
$
```

2. Create a grid and place a shape in a few places

```
$ java AsciiArt
> rectangle & 4 4
OK
> triangle * 3 3
OK
> new 15 20
OK
> place 0 1 1
OK
> place 0 1 15
OK
> place 1 8 8
OK
> place 0 10 1
OK
> place 0 10 15
OK
> print
======================
|                    |
| &&&&          &&&& |
| &&&&          &&&& |
| &&&&          &&&& |
| &&&&          &&&& |
|                    |
|                    |
|                    |
|        *           |
|        **          |
| &&&&   ***    &&&& |
| &&&&          &&&& |
| &&&&          &&&& |
| &&&&          &&&& |
|                    |
======================
OK
> exit
```

```
    OK
    $
```

3. Some error output

```
$ java AsciiArt
> place 0 4 5
BAD INPUT
> rectangle 3 4
BAD INPUT
> rectangle *** 3 4
OK
> list
0:
****
****
****
OK
> place 0 4 5
BAD INPUT
> grid 10 10 12
BAD INPUT
> set 0
BAD INPUT
> exit now
OK
$
```

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundlePR4" program that will turn in the files

`AsciiArt.java AsciiGrid.java`

No other files will be turned in and and it **must be named exactly as above.** BundlePR4 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11s/public/bin/bundlePR4
```

The output of the turnin should be similar to what you saw in your first programming assignment.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

Can I add extra output? No. We attempt to autograde your programs to the extent possible. Having extra output can cause you to lose points.

What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

I'm not using getChars() or getSize() is that OK?. Yes. We will test your implementation of AsciiGrid without printing output, these method enable our testers to verify functionality

If the user generates too many arguments for a command is that "BAD INPUT"?. Yes. Except for the exit command

If the user generates too few arguments for a command is that "BAD INPUT"?. Yes.

Do I have to check for all kinds of crazy inputs? Check for what we've asked for and have your program respond properly. We will test with reasonable inputs in all other cases. In particular, we will test with commands that don't exist, but we won't ever give non-integer arguments your commands that only take integers.

Do I have to use the ArrayList class? No, it's not required. But you will likely find this class useful

Can my AsciiArt or AsciiGrid class extend an existing class (using the extends keyword)? No.

Can you give is more sample inputs and outputs?. Look in the file testInput. If your program is right, then running `"$ java AsciiArt < testInput"` should give you something very recognizable.

Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon. For example, the professor's implementation of AsciiArt.java with comments (but with whitespace lines for readability) is 169 lines. The amount of code he added (including any private methods) to AsciiGrid.java was 88 (including comments). Your code may be longer or shorter in both cases. If you find yourself writing many 100s of lines of code more than these numbers, you need to ask for advice. You are encouraged to write Javadoc-style comments for all your methods, public and private.

START EARLY! ASK QUESTIONS!