CSE 11 **START EARLY!**
Winter2016 Program Assignment #3 (100 points)

Due: 14 Feb 2016 at 1159pm (2359, Pacific Standard Time) (SUNDAY!)

PROGRAM #3 : AsciiArt

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

In lecture and reading 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 and three child 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 oval, rectangle, triangle "shapes" using 2D `Character` arrays as a way to record shape. There are three child classes defined in `AsciiShape.java`. You should generate the Javadoc (use the -package argument to Javadoc to generate information for the derived classes

A second class called `AsciiGrid` defines a 2D "grid" in which the user can place (and) clear `AsciiShapes`. 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 complete the implementation of this class

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

The goals of this assignment are 1) to become comfortable with using multiple classes and better understanding ways to have classes interact, 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 throughout the course.

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

This is a more challenging program. Please start before the due date.

Everything is in graphics coordinates. (0,0) is upper left. Column indices are positive going right. Row indices are positive going downwards.

The `AsciiShape` class

You are to create the Javadoc for this class. You should also look at its code. There is an abstract class, `AsciiShape`, and three derived concrete classes (`Triangle, Rectangle, Oval`). Each shape is represented as a rectangular array of `Character`. When a concrete shape is constructed, some of the elements in the rectangular array may remain as `null`. Let's look at the method

`java.lang.Character[][]` **`getShape`**`()`

return a 2D Character array of the shape This is a full/deep copy, not a reference to internal storage.

Returns:

`rectangular array of Character. Non-null array elements define what part of the array a specific shape occupies`

Triangles and Ovals have some of the elements of this Character array set to `null`, other parts contain the symbol that makes up the `AsciiShape`. You should download the file ShapeTest.java and run it. It prints out the elements of an `Oval` and then uses AsciiShape's `toString()` method. A sample is below. ShapeTest.java utilizes `getShape()`.

```
$ java ShapeTest
Enter height and width: 4 8
(0,0):null|(0,1):#|(0,2):#|(0,3):#|(0,4):#|(0,5):#|(0,6):#|(0,7):null|
(1,0):#|(1,1):#|(1,2):#|(1,3):#|(1,4):#|(1,5):#|(1,6):#|(1,7):#|
(2,0):#|(2,1):#|(2,2):#|(2,3):#|(2,4):#|(2,5):#|(2,6):#|(2,7):#|
(3,0):null|(3,1):#|(3,2):#|(3,3):#|(3,4):#|(3,5):#|(3,6):#|(3,7):null|

 === toString of shape ===
 ######
########
########
 ######
```

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.

When you Implement AsciiGrid, **You may NOT**

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 (you may, of course define as many private variables and methods as you desire)

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 `AsciiGrid` with a brief description

```
public class AsciiGrid
extends java.lang.Object
```

define a 2D array of chars as a way to make ascii art. can place and clear an arbitrary 2D array of chars in the grid if asked-for array fits.

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**(AsciiShape shape, int r, int c) clear the elements 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 reference to internal storage
`int[]`	**`getSize`**`()` Return the width and height of the grid
`boolean`	**`placeShape`**`(AsciiShape shape, int r, int c)` place the 2D shape in the grid at location (r,c).
`java.lang.String`	**`toString`**`()` create a nice, printable representation of the grid and filled coordinates the grid should be framed.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

AsciiGrid

```
public AsciiGrid()
```

Constructor

AsciiGrid

```
public AsciiGrid(int row,
                 int col)
```

Constructor

Parameters:

`row` - number of rows in the ascii grid

`col` - number of columns in the ascii grid

For methods that return `boolean`, they should return `true` if method was successful, `false` if an error would occur. **Your `AsciiGrid` implementation should never generate a runtime error. This means that your code should check that parameters passed to it are valid/sensible.** Do NOT print any error messages to the screen in your turned program (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.

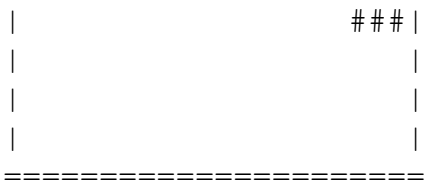
Other requirements of AsciiGrid

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

Special Clarification on AsciiGrid

When a shape is being placed on the grid (placeShape) or cleared from the grid (clearShape), you are translating the Shape to have it's upper-left corner start at coordinates (r,c). Call this the "translated shape". Think of this translated shape as a "paintbrush" only the parts of the paintbrush that are a) non-null and b) intersect the grid can affect the grid. In other words, the translated shape may only partially overlap the grid. This is not an error. This makes it possible to have a grid with a shape on it that is an oval but only partially overlaps. The program `GridTest` (when you have implemented AsciiGrid Properly) should give you following output.

```
$ java GridTest
Enter height, width
10 20
=== Grid with Centered Oval ===
======================
|                    |
|                    |
|       ######       |
|     ##########     |
|     ##########     |
|     ##########     |
|       ######       |
|                    |
|                    |
|                    |
======================
=== Grid with Partially Overlapped Oval ===
======================
|                    |
|                    |
|                  ###|
|                #####|
|                #####|
|                #####|
```



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 of added shapes are incremented by one. The first shape created has ID 0, the second shape has ID 1 and so on. *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**
grid	*height width*	Create a new AsciiGrid that is row x column
print		Print the current state of the grid. See the Javadoc for framing
oval	*height width*	Create an Oval shape (see classes in `AsciiShape`) that is is height x width using '#' as the character for the chars in the shape.
rectangle	*height width*	Create a Rectangle shape (see classes in `AsciiShape`) that is is height x width using '#' as the character for the chars in the shape.
triangle	*height width*	Create a Triangle shape (see classes in `AsciiShape`) that is is height x width using '#' as the character for the chars in the shape.
list		List the shapes by printing the index of the shape and the shape itself. Each shape is assigned an index starting at 0.
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)
symbol	*idx symbol*	Set the symbol of the shape with ID idx to the new symbol. If symbol has more than 1 character, use only the first character
exit	*0 or more args*	Exit the program

Initialization

When your program begins, there should be a defined grid that is 20 x 40 but no defined shapes.

Command Loop

The program should print out a command prompt of "> ", **(note space that follows the '>', you must include this so that the autograder will work properly!!!!)** and then wait for user input. If a command is successful, it should print OK, otherwise is should print BAD INPUT and some type of informational string on the same line.

Note, arguments to commands are not always integers. We will not test with the wrong types of arguments, but we will test with the wrong number of arguments. You will not, for example, have to properly handle a bad input string like "symbol eleven zero". You must first check if the number of arguments to a particular command is correct before attempting to convert the arguments. We might give you bad input that looks like "symbol eleven" (wrong number of arguments)

IMPORTANT: The program MUST 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 "grid", "GRID", "Grid", "griD" are all valid forms of the grid command. Be smart and use a String built in method to make this easy (how do you change to lower case?). We might even put spaces *before* a command is entered, your program should handle this condition. (look at String for a helpful method that trims or strips off leading and trailing whitespace)

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 for how to handle this easily..

Definition of "BAD INPUT"

1. A command is given the wrong number of arguments.
2. A command given illogical arguments. It is not sensible to build a grid with negative dimensions.
3. 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)
4. exit followed by any number of arguments is not BAD INPUT. It should exit the program .
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
6. Your code should NEVER generate an IndexOutOfBounds Exception or a NullPointerException (Or any other type of Exception)

Hints:

1. You will 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.trim().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. I suggest that you create helpers for each one of the commands above. For example, for a new AsciiGrid, you might have a helper called `private boolean addGrid(String [] args)`. So if the command is "grid", the helper does the work of interpreting the arguments and building a new grid.
5. Think about handling the input. The first word of the input line is the command itself (e.g. place, clear, list, …). The next are the arguments.

Example Sessions of Running AsciiArt

Define and list some shapes

```
[cse11@workstation PR3]$ java AsciiArt
> Triangle 4 4
OK
> Oval 4 8
OK
> rectangle 2 10
OK
> symbol 2 @
OK
> list
0:
#
##
###
####
1:
 ######
########
########
 ######
2:
@@@@@@@@@@
@@@@@@@@@@
OK
> exit
OK
[cse11@workstation PR3]$
```

Create a grid and place a shape in a few places

```
[cse11@workstation PR3]$ java AsciiArt
> grid 20 40
OK
```

```
> oval  5 5
OK
> rectangle 2 2
OK
> rectangle 1 10
OK
> place 0 5 5
OK
> place 0 5 25
OK
> place 1 12 18
OK
> place 2 19 10
OK
> print
=========================================
|                                       |
|                                       |
|                                       |
|                                       |
|                                       |
|      ###                  ###         |
|     #####                #####        |
|     #####                #####        |
|     #####                #####        |
|      ###                  ###         |
|                                       |
|                                       |
|                  ##                   |
|                  ##                   |
|                                       |
|                                       |
|                                       |
|                                       |
|                                       |
|          ##########                   |
=========================================
OK
> exit
OK
[cse11@workstation PR3]$
```

Some Error Output

```
[cse11@workstation PR3]$ java AsciiArt
> place 0 4 5
BAD INPUT: Invalid place parameters
> place 0
BAD INPUT: Invalid place parameters
> grid -1 10
BAD INPUT: Invalid grid parameters
> grid 10 20 30
BAD INPUT: Invalid grid parameters
> grid 10
BAD INPUT: Invalid grid parameters
> triangle -2 10
BAD INPUT: Invalid shape parameters
> triangle -2
BAD INPUT: Invalid shape parameters
```

```
> triangle 3 4 5
BAD INPUT: Invalid shape parameters
> symbol 4 5
BAD INPUT: Invalid set parameters
> exit
OK
[cse11@workstation PR3]$
```

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 "bundlePR3" 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.** BundlePR3 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/cs11wa/public/bin/bundlePR3
```

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. If we give a valid command the correct number of arguments, the arguments will be of the correct type.

How specific do my "informative" strings after BAD INPUT have to be? We will be looking for BAD INPUT. The informative string (see example) is to help you. Please keep BAD INPUT and your string on a single line.

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?. Yes. We will make some available.

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 212 lines. The amount of code he added (including any private methods) to AsciiGrid.java was 75 (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.

I'm really lost, how do I even start this project? This is the best question. You need to first understand AsciiShape. Try reading the code. It's an abstract class with three derived classes. You don't need to understand all the detail of which Characters are set to non-null (the oval is not hard math, just a little long to get it right).

Then look at GridTest.java and see what methods from AsciiGrid it invokes. Make GridTest.java work with your implementation of AsciiGrid.

Then Extend or add to GridTest.java (or create a new program) to test other parts of AsciiGrid (like clearing shape areas).

You are ready to tackle AsciiArt.
First, work out how you want to handle user input.

- Read a line of input from the standard input
- Extract the command from the input
- Extract the arguments (keep it as an array of strings)
- Call a specific helper method that processes the command and arguments
- DO NOT try to build all the commands at once. If you use helpers, they start to become pretty repetitive, getting the first few methods correct will make the rest of the code go more easily.
- Printing is your friend for debugging.

Can you give me an example of a helper method that you coded? Yes. Here's a helper (just the method) that creates a new AsciiGrid (called when "grid <h> <w>" is the command

```
private boolean addGrid(String [] args)
{
    errMessage = "Invalid grid parameters";
    if (args.length != 2)
        return false;
    int [] parms = convertArgs(args,0);
    if (parms[0] < 0 || parms[1] < 0)
        return false;
    theGrid = new AsciiGrid(parms[0],parms[1]);
    return true;
}
```

Look at it for a minute or two.

1. As an example, If the user typed "grid 20 40" then args is an array of Strings = {"20","40"}
2. If the wrong number of args is typed in, return that we failed
3. convertArgs is another helper – it takes an array of Strings that are integers and converts them to an array of ints.
4. If either of the argument is < 0, that's BAD INPUT (but I let the code that called this helper, process the output and the "errMessage" (errMessage is an instance var)
5. If all is good, create a new AsciiGrid Instance of the asked-for size. (theGrid is an instance var)

I'm still Lost.. So more guidance? In AsciiArt, think about what you need to have to make the program work. Think about the components (objects) you need first.

1. A grid to draw on
2. An ArrayList of shapes so that users can place shapes on the grid or set their symbols

3. A command processing loop.

   Don’t try to start writing code, make a list of things you need to make this program work. If you are clear on what should be doing, then the coding becomes much easier. Don’t GUESS!

START EARLY! ASK QUESTIONS!