CSE 142, Autumn 2008
Programming Assignment #2: ASCII Art (16 points)
Due Tuesday, July 7, 2009, 11:30 PM

This assignment focuses on `for` loops, expressions, variables, and constants. Turn in <u>two</u> Java files as described below.

Part A: ASCII Art Contest (2 points):

The first part of your assignment is to write a program that produces any text art (sometimes called "ASCII art") picture you like. Write a Java class named `AsciiArt` in a file named `AsciiArt.java`. The art can be posted on our Facebook application. Your program can produce any text picture you like, with the following restrictions and details:

- The picture should be <u>your own creation</u>, not an ASCII image you found on the Internet or elsewhere.
- The number of lines drawn should be at least 3 but no more than 200, and no more than 100 chars / line.
- The picture should not include hateful, offensive, or otherwise inappropriate images.
- The code should use at least one `for` loop or static method, but should not contain infinite loops.
- The picture must not be identical to your solution for Part B or consist entirely of reused Part B code.
- Your code should not use material beyond Ch. 3 of the book.
- If your Part A program compiles and runs successfully and meets the above constraints, it will receive the full 2 points. Part A will not be graded on style ("internal correctness").

```
      /**\
     //**\\
    ///**\\\
   ////**\\\\
  /////**\\\\\
+=*=*=*=*=*=*+
|..|\....../\..|
|./\|\../|/\.|
|/\|\|\|/|\|/\|
|\/|\|\|/|\|\/|
|.\|\|..|/|\|.|
|..\|/....\|/..|
+=*=*=*=*=*=*+
|\/|\|\|/|\|\/|
|.\|\|..|/|\|.|
|..\|/....\|/..|
|../|\....|/\..|
|./\|\../|/\|.|
|/\|\|\|/|\|/\|
+=*=*=*=*=*=*+
      /**\
     //**\\
    ///**\\\
   ////**\\\\
  /////**\\\\\
```

Part B: Rocket Ship (14 points):

The second part of your assignment is to produce a specific text figure that is supposed to look like a rocket ship. Turn in a class named `DrawRocket` in a file named `DrawRocket.java`. You should **exactly** reproduce the format of the output at left. This includes having identical characters and spacing.

One way to write a Java program to draw this figure would be to write a single `System.out.println` statement that prints each line of the figure. However, this solution would not receive full credit. A major part of this assignment is showing that you understand `for` loops.

In lines that have repeated patterns of characters that vary in number from line to line, represent the lines and character patterns using nested `for` loops. (See Chapter 2's case study.) It may help to write pseudocode and tables to understand the patterns, as described in the textbook and lecture.

Another significant component of this assignment is the task of generalizing the program using a single **class constant** that can be changed to adjust the size of the figure. See the next page for a description of this constant and how it should be used in your program.

The course web site will contain files that show you the expected output if your size constant is changed to various other values. You can use our Output Comparison Tool on the course web site to measure numbers of characters and to verify the correctness of your output for various values of the size constant.

Part A will not be graded for style except as specified on the first page. But Part B will be graded both on "external correctness" (whether the program compiles and produces exactly the expected output) and "internal correctness" (whether your source code follows the style guidelines in this document).

(continued on back)

Style Guidelines (for Part B):

As a reference, our solution to Part B has 5 methods besides `main` and occupies around 70-90 lines including comments and blank lines, though you do not have to match this exactly.

Use of `for` loops (nested as appropriate)

This program is intended to test your knowledge through Chapter 2, especially nested `for` loops. If you like, you may also use the Java features from Chapter 3 such as parameters, although you are not required to do so and will receive no extra credit for doing so. You may not use any Java constructs beyond Chapter 3.

Use of static methods for structure and elimination of redundancy

Continue to use static methods to structure your solution in such a way that the methods match the structure of the output itself. Avoid significant redundancy; use methods so that no substantial groups of identical statements appear in your code. No `println` statements should appear in your `main` method. You do not need to use methods to capture redundancy in partial lines, such as the two groups of "`ma`"s in the following line:

```
|.\/\/..\/\/.|
```

Source code aesthetics (commenting, indentation, spacing, identifier names)

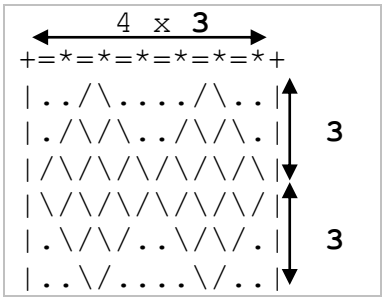
You are required to properly indent your code and will lose points if you make significant indentation mistakes. See the textbook for examples of proper indentation. No line of your code should be over 100 characters long.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of `ClassNames`, `methodAndVariableNames`, and `CONSTANT_NAMES`.

Include a comment header at the beginning of your program with basic information and a description of the program. **Also include a comment at the start of each method**, describing its behavior. Your comments should be in your own words.

Class constant for figure's size

You should create one (and only one) class constant to represent the size of the pieces of the figure. Use **3** as the value of your constant. Notice that the subfigures in the middle of the output have a height of 3, and the subfigures' height determines their width, so there is only one size variable. Your figure must be based on that exact value to receive full credit.



On any given execution your program will produce just one version of the figure. However, you should refer to the class constant throughout your code, so that by simply changing your constant's value and recompiling, your program would produce a figure of a different size. Your program should scale correctly for any constant value of 2 or greater.

Development Strategy (How to Get Started):

This program is best completed in stages. We strongly recommend the following development strategy:

1. **Tables**: Examine the output and write tables to discover the patterns of repeated characters on each line.
2. **Code w/o Constant**: Completely write the Java code to draw the rocket ship at its default size of 3.
3. **Code w/ Constant**: Add a constant to your code so that the rocket ship can scale to different sizes.

To summarize, you should not worry about the constant at first. Write an initial program without a constant, using loop tables or pseudocode to help you deduce the patterns in the output. After your figure looks correct at the default size, begin a second version with the constant. See Chapter 2's case study for a good example program.