Objectives – practice using these new concepts:

- for loops & switch statements
- formulas to determine counts (for controlling for loops) – i.e., a function of height and/or width
- treat a String as an array of char’s (using charAt(i) method)
- write a modular program so it’s easier to develop, debug, test and modify in the future
  o use incremental development (write some modules, test program, write more modules, test it again, . . .)
  o use static methods with multiple levels of calling (including parameter passing)

Project Description
This app produces a vertical sign of chunky characters (using the exact font shown below) based on a message supplied by the user (which includes words/numbers of letters/digits/space). The “sign” is printed to the IDE’s console (output) window. The user will specify height and width for the characters – and all characters in the message are the same specified size.

Project Structure
This is a modular program with 3 separate classes (each in its own java file – all in the same package) which contain static methods. This uses a hierarchical call structure, where main calls appropriate CharDes class methods (as well as the 3 local private methods in its own class), and those CharDes methods in turn call appropriate Line class methods. [Program MUST follow the structure specified below].

LEVEL 1 - MAIN PROGRAM (“top boss”): ChunkySign class (= project name) contains the main method (the overall controller of the program) and several private service methods (which main calls). main “runs the show”, calling other methods in CharDes class to manage what happens for a particular character. The methods include:

1. User input is done via input dialog windows in these “local methods” (in THIS class) – and there’s some data editing of the data, as needed, before sending it to main. (These methods CHANGE the data, as needed, rather than re-asking user to enter it again).
   a. getMessage – convert user’s message to all CAPS – characters might be printable or nonPritable:
      i. printable chars: a digit (0-9), a space, letters from this set: C E F H I L O P S T U
      ii. nonPrintable chars (i.e., other letters or any symbol) will be converted to a # in this method
   b. getWidth
     - width must be => 8 and <= 18. If not within range, use default of 10 (& tell user, using messagebox)
     - width must be an EVEN number. If ODD, add 1 to it to make it even (& tell user, using messagebox)
   c. getHeight
     - height must be => 10 and <= 20. If not within range, use default of 10 (& tell user, using messagebox)
     - height must be an EVEN number. If ODD, add 1 to it to make it even (& tell user, using messagebox)

2. main processes the message, char-by-char - by grabbing a char from the message [charAt(i)], then using a switch to call the appropriate method in CharDes class, then printing 1 blank line (so each big chunky char is separated from the next big char).
   IMPORTANT: DO NOT control how a char is printed HERE – that MUST be done in the appropriate CharDes class method.
   - DO NOT do any actual printing of a char HERE – that MUST only be done by Line class methods.

LEVEL 2 – CHARACTER DESIGN METHODS (“mid-level managers”): CharDes class contains 23 methods (10 digits, 11 legal letters, 1 for space, 1 for hash). The bodies of these methods specify HOW that individual char is drawn by calling specific Line class methods. These 23 methods DO NOT actually DO ANY printing – instead, they CALL various Line class methods to do ALL printing. The methods here must provide the Line class methods with the necessary counter(s) value(s) so the Line class methods know how tall/wide to draw the lines. All calculations of those counters’ values MUST be done HERE in the CharDes class methods – no hard-coding, but CALCULATE the values as FUNCTIONS of height and width. (Those 23 were provided to these 23 methods by main).
   IMPORTANT: Nonprintable characters entered by the user were already converted to #’s by getMessage method.
   METHOD NAMES: desZero, des1, ..., des9, desC, ..., desOh, desP, ..., desSpace, desHash

LEVEL 3 – LINE DRAWING METHODS (“workers”): Line class methods DO the actual line-printing to the console. There are 6 methods:

1. doHor prints 2 horizontal lines of the size specified by the caller – printing STARTS AT LEFT MARGIN.
2. doMixHor prints 2 horizontal lines of the size specified, which are a MIX of leading spaces, then some X’s. The caller supplies the 2 counts. This method first prints numSpaces spaces, then prints numXs X’s. (This type of line is needed for 3’s middle line, the top & bottom of the 0 (zero), the middle of the 2, etc.) [NOTE: numXs plus numSpaces is NOT necessarily the char’s full width – because the method does NOT print trailing spaces after the X’s].
3. doLVer prints XX at LEFT margin for the number of lines specified by the caller.
4. doRVer prints XX at RIGHT margin for the number of lines specified by the caller. Caller also supplies actual char’s full width so THIS METHOD can calculate how many spaces to print BEFORE the XX.
5. doLRVer prints XX at LEFT margin AND XX at RIGHT Margin for the number of lines specified by the caller. Caller also supplies actual char’s full width so THIS METHOD can calculate how many spaces to print BETWEEN the XX’s.
6. doMVer prints XX in MIDDLE for the number of lines specified by the caller. Caller also supplies actual char’s full width so THIS METHOD can calculate how many spaces to print BEFORE the XX.

Sample Output for user input 26 with width of 10 and height of 12:
The Character's "Font" - to show EXACTLY what each char prints as for input message “0123456789 CEFHILOPSTUW” with use-specified width of 10 & height of 12. But the sign is VERTICAL - chars do not “print across” as I show below – I’m just “saving paper” here.

<p>| | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx</td>
<td>xx</td>
<td>xxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>xxxxx</td>
<td>xx</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>xxx</td>
<td>xx</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>xx</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>xx</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>x</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>xx</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
<td>xx</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>xxxxx</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
<td>x</td>
<td>xxxxxx</td>
</tr>
</tbody>
</table>

The signs are printed vertically. Each character is represented in a 10-wide by 12-high grid.