CSE 142, Spring 2011
Programming Assignment #1: Song (10 points)
Due Tuesday, April 5, 2011, 9:00 PM

Program Description:

This program tests your understanding of static methods and `println` statements. Write a Java class called `JackBuilt` in a file named `JackBuilt.java`. (Use exactly this file name, including identical capitalization.)

A *cumulative song* is one where each verse builds upon previous verses. Examples of cumulative songs are "The Twelve Days of Christmas" and "There Was An Old Lady Who Swallowed A Fly." For this assignment, you will write a program that outputs the following cumulative song, a variation of a classic song called "The House That Jack Built":

```
This is the house that Jack built.

This is the malt
That lay in the house that Jack built.

This is the rat,
That ate the malt
That lay in the house that Jack built.

This is the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

This is the maiden all forlorn
That milked the cow with the crumpled horn,
That tossed the dog,
That worried the cat,
That killed the rat,
That ate the malt
That lay in the house that Jack built.

<< your custom 8th verse goes here >>
```

The first seven verses printed by your program must **exactly** reproduce the output at left. This includes identical wording, spelling, spacing, punctuation, and capitalization.

However, to encourage creativity, the last verse of your song (the final bold part in ***<< >>***) may print any text you like. Creative verses submitted may be shown in class anonymously at a later date. The only restrictions on your custom verse are the following:

- The verse must not be identical to another verse or consist entirely of text from earlier in the song.
- The number of total lines in the verse should be at least three (3) but no more than fifty (50).
- The text of the verse should not include hateful, offensive, or otherwise inappropriate speech.
- The code to produce the verse is still subject to the style guidelines on the next page.

One way to write this program would be to simply write a series of `println` statements that output each line of the song in order. But such a solution would not receive full credit. Part of the challenge of this assignment lies in recognizing the structure and redundancy of the song and improving the code using static methods.

(continued on back)

Style Guidelines:

You should not place any `println` statements in your `main` method. (It is okay for `main` to have empty `println` statements to print blank lines.) Instead of printing in `main`, use static methods for two reasons:

1. Structure

You should write static methods to capture the structure of the song. You should, for example, have a method for each of the verses of the song (including your custom verse) to print that verse's entire contents.

2. Eliminating redundancy

You should use only one `println` statement for each distinct line of the song (other than blank lines). For example, the following line appears several times in the output, but you should have only one `println` statement in your program that prints that line of the song:

```
That lay in the house that Jack built.
```

But it is not good style to have lots of methods that each print just one line. Instead, identify groups of lines that appear in multiple places in the song and create methods to represent those groups. There is a general cumulative structural redundancy to the song that you should eliminate with your methods. Recall that methods can call other methods (which can themselves call other methods, and so on). The key question to ask is whether you have repeated lines or groups of lines of code that could be eliminated if you structured your methods differently. This includes sequences of `println` statements and also repeated sequences of method calls.

You do *not* have to eliminate partial redundancy in lines that are similar but not identical, such as the following. We suggest ignoring the partial similarity and treating them as though they were entirely different lines.

```
This is the malt
That ate the malt
```

Include a comment at the beginning of your program with some basic information and a description of the program in your own words. For example:

```
// Suzy Student, CSE 142, Autumn 2049, Section XX
// Programming Assignment #1, 06/07/49
//
// This program's behavior is ...
```

For this assignment, you should limit yourself to the Java features covered in Chapter 1 of the textbook. Though we will cover Chapter 2 while you work on this assignment, please do not use Chapter 2 features on this program, such as mathematical expressions, or `for` loops.

As a point of reference, our solution to this program has roughly 14 methods other than `main` and is around 90-100 lines long including comments and blank lines (or 51 "substantive lines" when we paste it into our Indenter Tool on the course web site). But these are just rough guidelines; you do not have to match them exactly.

Submission and Grading:

Turn in your Java source code file electronically from the Homework page on the course web site.

Part of your program's score will come from its "external correctness." External correctness measures whether the output matches exactly what is expected. (We are very picky about the output matching exactly. Every character and space must match.) Programs that do not compile will receive no external correctness points.

The rest of your program's score will come from its "internal correctness." Internal correctness measures whether your source code follows the style guidelines specified in this document. This includes having an adequate comment header and capturing the structure and redundancy of the song as specified previously. You should also limit the length of each line in your program to fewer than 100 characters.