Take-home Assessment 1: Song

due April 6, 2021, 11:59pm

This assignment will assess your mastery of the following objectives:

- Write a functionally correct Java program to produce specified console output.
- Write static, void methods to provide structure to the code and eliminate redundancy.
- Write and call static methods to manage the flow of control in a program.
- Follow prescribed conventions for spacing, indentation, naming methods, and header comments.

Program Behavior

You will write a Java program that produces as output a cumulative song in which successive verses build on previous verses (as described on Wikipedia). In particular, we will produce the song "There Was an Old Woman Who Swallowed a Fly." Your program should produce ***exactly*** the output below, except for the custom verse (see below). You can use the Mark button in Ed to check if your output is correct.

You must **exactly** match the output here, including both content and format. Check your output carefully!

Expected Output

```
There was an old woman who swallowed a fly.
I don't know why she swallowed that fly,
Perhaps she'll die.

There was an old woman who swallowed a spider,
That wriggled and iggled and jiggled inside her.
She swallowed the spider to catch the fly,
I don't know why she swallowed that fly,
Perhaps she'll die.

There was an old woman who swallowed a bird,
How absurd to swallow a bird.
She swallowed the bird to catch the spider,
She swallowed the spider to catch the fly,
I don't know why she swallowed that fly,
Perhaps she'll die.

There was an old woman who swallowed a cat,
Imagine that to swallow a cat.
She swallowed the cat to catch the bird,
She swallowed the bird to catch the spider,
She swallowed the spider to catch the fly,
I don't know why she swallowed that fly,
Perhaps she'll die.

There was an old woman who swallowed a dog,
What a hog to swallow a dog.
She swallowed the dog to catch the cat,
She swallowed the cat to catch the bird,
She swallowed the bird to catch the spider,
She swallowed the spider to catch the fly,
I don't know why she swallowed that fly,
Perhaps she'll die.

<< Your custom sixth verse goes here >>

There was an old woman who swallowed a horse,
She died of course.
```

Custom Verse

As indicated above, you should include a custom sixth verse that matches the pattern of the first five verses. This custom verse should be printed in place of "*« Your custom sixth vese goes here »*" in the output above. (You should *not* print that placeholder text.) For example, some versions of the song have a sixth verse for swallowing a goat ("Just opened her throat to swallow a goat"). Notice that the first two lines should either end in the same word (fly/fly, bird/bird, cat/cat, etc.) or should end with rhyming words (spider/inside her). You should not simply copy one of the previous animals or to use the verses you'll find on the web (e.g., goat and cow); you should write your own custom verse. The text of the verse should not include hateful, offensive, or otherwise inappropriate speech.

Implementation Guidelines

In addition to producing the desired behavior, your code should be well-written and meet all expectations described in the grading guidelines and the Code Quality Guide. For this assessment, pay particular attention to the following elements:

Capturing Structure

This means you will have at least seven methods in your program (though you might want or need more).

You should use static methods to accurately capture the structure of the song in your code. You must, for example, have a separate method for each of the seven verses of the song (verses are separated by blank lines in the output). As a result, you will not have any `println` statements in main except perhaps a `println` that produces a blank line.

In addition, you should not not have any methods that include only a single `println` statement. (Calls to methods like these should be replaced by the direct call to `println`.) All methods should capture a meaningful portion of the program that is not already captured by another method.

Avoiding Redundancy

You should also use static methods to avoid "full-line" redundancy. In particular, you **must** make sure that you use only one `println` statement for each distinct line of the song. For example, the line:

```
Perhaps she'll die.
```

Count carefully–the number of non-blank `println` calls should equal the number of *unique* lines in the song (including your custom lines).

appears several times in the output. To receive full credit, you must have only one `println` statement in your program that produces this line.

On the other hand, you are *not* required to fix "partial-line" redundancy, as in pairs of lines like these:

```
There was an old woman who swallowed a horse,
There was an old woman who swallowed a dog,
```

or these:

```
She swallowed the dog to catch the cat,
She swallowed the dog to catch the bird,
```

It is not possible to avoid this type of redundancy using material we have covered so far (methods and `println` statements), so *you are not expected to do so*.

Permitted Java Features

The only statements in your program should be method calls and `println`s. There should be no `print`s.

You should not use any Java features that we have not covered in class. For this assessment, you should limit yourself to the Java features covered in chapter 1 of the textbook. You should also **not** use `System.out.print()` statements even though they are covered in chapter 1. All output (*including the last line*) should be produced using `System.out.println()`. You also may not use the `\n` escape sequence.

Indentation and Whitespace

Your program should be properly indented and make proper use of blank lines as shown in class and discussed in the Code Quality Guide.

Header Comments

You should include a comment at the beginning of your program with some basic identifying information and a description of the program. Your comment should look something like this:

```
// Grace Hopper            << replace with your name >>
// 4/6/2021
// CSE142
// TA:  Ada Lovelace       << replace with your TA's name >>
// Take-home Assessment #1
//
// This program will...    << add a brief description of the program >>
```

You can include additional information if you like, but make sure at least these details are present. Your program description should be specific to this assessment, but not include any implementation details.

Your code should also include a comment at the beginning of each method that describes that methods behavior. See the Code Quality Guide for more information.

Getting Help

If you find you are struggling with this assessment, make use of all the course resources that are available to you, such as:

- Reviewing relevant examples from lessons, section, and lab
- Reading the textbook
- Visiting support hours
- Posting a question on the message board

Collaboration Policy

Remember that, while you are encouraged to use all resources at your disposal, including your classmates, **all work you submit must be entirely your own**. In particular, you should **NEVER** look at a solution to this assessment from another source (a classmate, a former student, an online repository, etc.). Please review the full policy in the syllabus for more details and ask the course staff if you are unclear on whether or not a resource is OK to use.

Reflection

In addition to your code, you must submit answers to short reflection questions. These questions will help you think about what you learned, what you struggled with, and how you can improve next time. The questions are given in the file `SongReflection.txt` in the Ed lesson; type your responses directly into that file.

Running and Submitting

You can run your program by clicking the "Run" button in Ed. This will compile and execute your code and show you any errors, or the output of your program if it runs correctly. If you believe your output is correct, you can submit your work by clicking the "Mark" button in the Ed lesson. You will see the results

of some automated tests along with tentative grades. **This grade is not final until you have received feedback from your TA.**

You may submit your work as often as you like until the deadline; we will always grade your most recent submission. Note the due date and time carefully—**work submitted after the due time will not be accepted**.