Assignment 11: Comprehensive Project

- Due Tuesday by 11:59pm
- Points 100
- Submitting an external tool
- Available Jul 18 at 12am - Aug 1 at 11:59pm

The purpose of this assignment is to practice your knowledge of data structures and problem solving.

For this assignment you are required to practice pair programming and only one partner should submit the Java program files specified below. See **the guidelines and policies for pair programming (https://utah.instructure.com/courses/967061/pages/pair-programming-guide)** .

The Problem

In a video game, the characters often must say phrases throughout the play of the game. The phrases must be of a certain form (e.g., a noun followed by a verb followed by an object). However, the phrases must also be randomized, so that the character does not repeat exactly the same phrase again and again.

We have been asked to construct a program that randomly generates phrases. The form of each phrase is specified using an input grammar file.

As an example, consider the input grammar file **super_simple.g (https://utah.instructure.com/courses/967061/files/164019972?wrap=1)** **(https://utah.instructure.com/courses/967061/files/164019972/download?download_frd=1)** . A possible random phrase of this form is "The mouse stood on the dog." Another is "The cat sat on the mouse."

The formatting rules of the input grammar file are very strict, in order to simplify the task of parsing the input.

- The file contains the definitions of one or more non-terminals.
- Non-terminals are always denoted with angle brackets, and the name of a non-terminal may not contain any whitespace. For example, <noun> and <long-int> are correct non-terminals. <awesome word> and <noun > are not correct non-terminals.
- Terminals are denoted by the absence of angle brackets.
- Each non-terminal definition begins with an opening curly brace (on its own line with no extraneous spaces). The non-terminal being defined appears alone on the next line. The choice of one or more productions to which the non-terminal can be expanded appear next (one per line). A single production consists of one or more terminals and/or non-terminals. A

single blank space may appear between each terminal and non-terminal; however, no extraneous spaces appear before the first (non-)terminal or after the last (non-)terminal. Finally, the non-terminal definition ends with a closing curly brace (on its own line with no extraneous spaces).

- Every non-terminal used in a production is defined somewhere in the file, but not necessarily before (or after) the production in which it is used.
- The starting non-terminal is always identified as <start>.
- Comments may appear in between non-terminal definitions. Therefore, any lines outside of the curly braces should be ignored.
- Error-checking on the input grammar file is not required, and you may assume that all grammar file used for grading are correct.

The following are some sample input grammar files and possible random phrases for each.

- **poetic_sentence.g (https://utah.instructure.com/courses/967061/files/164019971?wrap=1) (https://utah.instructure.com/courses/967061/files/164019971/download?download_frd=1)**

  The waves portend like big yellow flowers tonight.

- **mathematical_expression.g (https://utah.instructure.com/courses/967061/files/164019970?wrap=1) (https://utah.instructure.com/courses/967061/files/164019970/download?download_frd=1)**

  (y + ((x - 2) * e))

- **assignment_extention_request.g (https://utah.instructure.com/courses/967061/files/164019968?wrap=1) (https://utah.instructure.com/courses/967061/files/164019968/download?download_frd=1)**

  I need an extension because I had to practice for an alligator wrestling meet, and as if that wasn't enough I just didn't feel like working, and then, just when my mojo was getting back on its feet, my dorm burned down.

Requirements

- Create a new class called *RandomPhraseGenerator* in a new package called *comprehensive*. The user starts your program by running this class (i.e., by calling *RandomPhraseGenerator*'s *main* method).
- You may create as many other new classes as needed. Make sure that all Java files required by your program are in the *comprehensive* package.
- The path and name of the input grammar file are given as input to your program via the command line (as the first command-line argument). Also given as input is the number of random phrases your program should generate (as the second command-line argument).

- The output of your program is simply the random phrase(s), printed one per line. The output must preserve any blank spaces that do or do not appear between terminials and non-terminals. For example, given input grammar file **super_simple.g (https://utah.instructure.com/courses/967061/files/164019972?wrap=1)** **(https://utah.instructure.com/courses/967061/files/164019972/download?download_frd=1)** , "The cat sat on the mouse." is a possible random phrase, but "The cat sat on the mouse ." is not.

- Because all grading occurs outside of Eclipse, you **must** make certain that your program runs correctly in the terminal of a CADE lab machine before submission. As an example, to run your program such that it generates five random phrases for the input grammar file **poetic_sentence.g (https://utah.instructure.com/courses/967061/files/164019971?wrap=1)** **(https://utah.instructure.com/courses/967061/files/164019971/download?download_frd=1)** , issue the following command.

  ```
  java comprehensive/RandomPhraseGenerator poetic_sentence.g 5
  ```

  In this example, your working directory must contain both the input grammar file and your *comprehensive* package. You may also give filenames that include absolute and relative paths as the first command-line argument.

- Command-line arguments can be specified in Eclipse, using the *Run Configurations* menu.

- You may assume that your program is not tested for input grammar files that are non-existent or incorrectly formatted.

- Take care to design your solution to be as efficient as possible. See the section below for details of how your random phrase generator is evaluated for running-time efficiency.

NOTE: It is intentional that you are being given no guidance as to how to solve the problem of generating random phrases. It is critical that you gain experience solving a problem "from scratch," designing the structure of your classes and methods, as well as, choosing the best data structures and algorithms for the problem. Because the readers of your code have no assumptions about how it is organized, you **must** document it well.

See **Assignment 1 (https://utah.instructure.com/courses/967061/assignments/14082494)** for style and design requirements, as well as the late policy for assignments. At a minimum, every class and method must be commented using Javadoc.

Submission

Submit all files required to run your *RandomPhraseGenerator*. All files should be in the *comprehensive* package. DO NOT submit any testing or timing files.

See **Assignment 2 (https://utah.instructure.com/courses/967061/assignments/14121576)** for instructions on how to designate a group in Gradescope.

The auto-grader is set to run a minimal set of "pre-tests" to ensure you have included all required classes and methods specified above. The remaining points are assessed after the assignment deadline and based on whether your code passes "post-tests", how good your style / design is, and how efficient your solution is (see below). Note that Assignment 11 is 100 points for programming and 50 points for analysis.

Some of the tests reference these input grammar files: **hello_world.g (https://utah.instructure.com/courses/967061/files/164019969?wrap=1)** ⬇ **(https://utah.instructure.com/courses/967061/files/164019969/download?download_frd=1)** , **super_simple.g (https://utah.instructure.com/courses/967061/files/164019972?wrap=1)** ⬇ **(https://utah.instructure.com/courses/967061/files/164019972/download?download_frd=1)** , **poetic_sentence.g (https://utah.instructure.com/courses/967061/files/164019971?wrap=1)** ⬇ **(https://utah.instructure.com/courses/967061/files/164019971/download?download_frd=1)** , **abc.g (https://utah.instructure.com/courses/967061/files/164019967?wrap=1)** ⬇ **(https://utah.instructure.com/courses/967061/files/164019967/download?download_frd=1)** , **abc_spaces.g (https://utah.instructure.com/courses/967061/files/164019966?wrap=1)** ⬇ **(https://utah.instructure.com/courses/967061/files/164019966/download?download_frd=1)** . These are not the only grammar files used by the autograder.

Running-time requirement

25 / 100 points for the Assignment 11 program (25 / 150 for Assignment 11 overall) are assessed based on the running time of your random phrase generator.

- Any submission that is within one standard deviation of the average running time of all submissions and correctly handles all input grammars used to assess running time earns 25 points.
- Any submission that is slower than the average of all submissions by more than one standard deviation and/or does not correctly handle all input grammars used to assess running time earns 0-24 points, with the exact amount based on distance from the average running time and correctness.

What does this mean? To achieve a good score, you should avoid using data structures and algorithms that are inefficient for solving our problem.

To assist you in determining how the efficiency of your random phrase generator compares to other submissions, an additional Gradescope auto-grader is provided: **Random Phrase Generator — Running time**.

NOTE:

- If your results are "The autograder failed to execute correctly...." this means your submission fails for the input grammars used to assess running time. The failure is likely due to one or

both of these reasons:

  - Your submission does not pass all pre-tests. Submit to **Assignment 11: Comprehensive Project** for details.
  - Your submission incurs an unexpected exception, such as NullPointerException or ArrayIndexOutOfBoundsException.
- This auto-grader does not exhaustively check for correctness of your submission. Having an execution time displayed on the leaderboard does not guarantee that your submission passes all post-tests.
- Results displayed on the Leaderboard are not official. You can expect small differences in running times when submitting the same solution at different times, depending on the load operating on the auto-grader. Therefore, official running times are collected for all submissions on a dedicated machine after the late-period deadline. BEWARE: The Leaderboard may indicate that you have the fastest submission when you actually do not. Regardless of the small variations in running times displayed, the Leaderboard is useful in indicating when you have a slow and inefficient submission and are at risk of earning less than 25 points.
- Submission to **Random Phrase Generator — Running time** is not required but strongly encouraged.
- Submission to **Assignment 11: Comprehensive Project** is required.

Analysis

The **Analysis Document (https://utah.instructure.com/courses/967061/assignments/14082293)** must be written and submitted by each programming partner individually.

This tool needs to be loaded in a new browser window

Load Assignment 11: Comprehensive Project in a new window