# **Team Project TP-1**

# **Phase 3 (TP-1.3) *Play A Game of Skunk***

**Software Analysis and Design**

**Fall Semester 2019**

**SEIS 635 (3 semester credits)**

**Section 01**

**out October 11, 2019**

**due by (suggested)**

**November 1, 2019 (3 weeks)**

**20 points**

***Overview***

For this third and final Phase (TP-1.3) of the first Team Project, you will finish your implementation of **SkunkApp** so you can play one **game** of Skunk with two or more players. A game is multiple Turns ending when one or more players score >= 100.

You will continue with the same team and remote GitHub host repository named **tp\_1\_skunk** containing Eclipse project **SkunkProject** you used for earlier TP-1 Phases.

I will grade your work by checking the final committed version of your host repo code as soon as you submit a text message to the **TP-1.3 Assignment** tool in Canvas. Your work will also include some non-code artifacts, added to your project as described below.

I will not hold you to a "hard-and-fast" due date, but TP-2 will be underway soon, so don't get too far behind.

***Process Steps***

* ***Continue team development using Git/EGit and your same GitHub repository.***  
  Continue with both team members working independently on the remote **tp\_1\_skunk** GitHub repository. Each member should commit regularly to their local Git repo, occasionally pushing to the remote repo on the host's GitHub account. Before doing work, remember to fetch and merge ("pull") the latest changes from the host repo, merging them into your local repo. This will help to avoid Git merge conflicts.

You should also continue creating JUnit unit tests for as much of your non-UI code as possible. In this Phase, you will measure your test coverage using Eclipse's ECLEmma (see below), ideally writing enough test cases to have total (100%) coverage of the class under test.

Also remember that a superior architectural design separates the user interaction code ("presentation logic") from the core non-UI game code ("domain logic") - and you should think about refactoring your TP-1.2 code to improve this separation before you implement the new game (multiple-round) features. Part of your grade for this assignment will reflect how completely you implement this separation.

* ***Extend your* SkunkApp *so it plays a complete Game of Skunk, consisting of one or more Turns and ending when one (or more) players reach 100.***  
  When you start your app, read the number of players following by the names of each. Initialize the attributes of each player, then offer the user an option to view the complete rules for your game and display them on request.  
    
  Now start a new Game of Skunk, consisting of a sequence of multiple Turns taken by players, one after another. Each Game ends with one or more players scoring >= 100, followed by a final set of Turns giving all non-100 players a final chance to increase their score. After this, do the final accounting that updates each player's chip count according to the final player scores.   
    
  You should have already implemented the "play one Turn" functionality in TP-1.2, so extend it here to "play one Game" functionality, keeping track of Turn scores for each player as well as their chip totals. If you haven't defined a Player software class, think about doing so. Each Player can track their different scores, perhaps also having responsibilities for scoring Rolls and/or Turns and/or Games.   
    
  Be sure to add tests for any new software classes you create, striving for the JUnit test coverage of your non-UI code to be as complete (close to 100%) as possible. As discussed above, try to separate your Presentation Logic and Domain Logic as much as you can.
* ***Measure your test coverage of non-UI using ECLEmma, an Eclipse plug-in.***  
    
  We'll see how to use Eclipse's ECLEmma, a code coverage tool that's included with the standard Eclipse Java installation. It allows you to measure and view how much of your app code is actually tested (covered) by your test code. 100% coverage is the goal, though it is difficult to attain. I will award **up to 1 Extra Credit point** for test coverage within non-UI code of beyond 90%. Here's the ECLEmma site URL: <http://www.eclemma.org/>
* ***Reverse-engineer your final non-test code, creating a UML Software Class Diagram.***Install and learn how to use ObjectAid UML, another Eclipse plug-in for creating reverse-engineered UML software class diagrams directly from your Java code. It's not included in the standard Eclipse installation, so you have to add it as an external tool and obtain a free external license to use it. Here's how: <http://www.objectaid.com/installation>   
    
  Use it to generate UML Class Diagrams for your **SkunkApp** non-test code. (We'll see how in class.) Then generate a .**png** or .**jpg** image of the diagram and include it as a document within your Eclipse project.
* ***Add extra features to your app if you wish, earning Extra Credit.***  
  You can earn up to **1-5 points of Extra Credit** for implementing one or more of the following features, as well as for implementing other features of your own invention. The professor will judge how much EC your extra features are worth, with his decision final. Here are some ideas:
  + Implement a complete Tournament, consisting of multiple Games with chips won and lost at the end of each.
  + Refactor your TP-1.2 implementation, improving its structure and ease of modification. Your commits should briefly note what and where you found some "code smell" that was improved with your subsequent refactoring.
  + Implement a GUI for your app's Presentation Logic. Strive to decouple your GUI code from your domain/business code as much as possible. Ideally, your domain software layer should not know if Think about using Eclipse's SWT (Simple Windowing Toolkit) to help implement your GUI.
  + Implement extra help within the app, as well as providing a clear and complete display of all aspects of game play and outcomes.
  + Implement a remote version of the app, where multiple players can run clients on remote PC's on a network, connecting to a central Skunk server which implements core functionality to each client.
  + Implement one or more "computer players", each of which generates their own moves without user input. You might examine how to play optimally, as described in some of the posted links.
  + Other rules that are variations of the original (see Wikipedia's entry on "Pig" for details).
  + Provide the ability to suspend a game and save its current state to external storage (file or database), as well as resume the game at a later time from where it left off.
  + Any other features you want.

When you submit your text message with SHA-1 id to be graded (see below), please describe what extra features you'd like to be considered for extra credit.

* ***Grading is based on code you've committed to GitHub, after you've submitted a text message to the TP-1.3 Assignment tool.***

When you are finished, submit a text message to the Canvas TP-1.3 assignment. It should contain the GitHub URI of your host repository, as well as provide the SHA-1 identifier of the commit you wish me to evaluate.

I will then grade your work by cloning the specified commit of your host repo (you did add **seis-635-fall2019** as a collaborator, didn't you?), then running all your tests, checking your test coverage, evaluating your code's "separation of logics", and playing your game to evaluate it for correctness and ease-of-use. I will also evaluate any Extra Credit that you submit, so be sure you describe this in your text submission.

In general, I will make sure you've satisfied all the requirements of this and earlier Phases, and grade you on the correctness and completeness of your submitted code.

* ***Each team member's grade for this Phase will be based on how closely you obey the above requirements. Also, large differences in team member's efforts will result in penalties.***

I will examine the relative effort of all team members, as measured by the number of commits of each under their own GitHub ids.

If one team member does all or most all of the project's Git commits, the grades for all members may be penalized for unequal effort. Members with few or no commits will be penalized the most.

Thus, strive for all team members to contribute.