# **Team Project TP-1:**

# **TP-1.1: Phase 1**

# **Beginning the**

# **Skunk Implementation**

**Software Analysis and Design**

**Fall Semester 2019**

**SEIS 635 (3 semester credits)**

**Section 01**

**out September 22, 2019**

**due October 4, 2019 \*\*\***

**10 points**

\*\*\* Try to meet this deadline, though it won't be strictly enforced.  
.\ ***Overview***

For this first Phase (TP-1.1) of the first Team Project (TP-1), you will form a team of two or three students.

You and your team member will then study a provided specification of the **Skunk** dice game. A separate document describes the detailed rules for this game.

By the end of this Project (not this phase!), you will have implemented a Java application that plays one game of Skunk, along with a collection of JUnit 5 tests that verify your implementation.   
  
For this first Phase, you are required only to implement "testable die and dice" classes along with JUnit 5 unit tests for them. Your only Canvas submissions are text messages described below; you will push your code to the "host" GitHub repo where I can access it.

You will work with your team partners in Eclipse, using EGit/Git and/or command-line Git, along with GitHub. I will grade your work by checking the final committed version of your code on the specified due date. You will continue working on this code across the remaining Phases of this Team Project TP-1.

***Process Steps***

* ***Form teams of two or three students to work on TP-1.***  
    
  You can choose your own teams but try to team up with others whose software skills are complementary to yours. Also try to choose teams so members can meet face-to-face outside of class or else via remote Zoom sessions, if needed. Avoid teams where all members are weak in Java. These teams will work together for the duration of Team Project 1, across multiple iterations.
* ***Submit to the TP-1 Canvas Assignment the names of all students on the team, as well as which student's GitHub account name will be the "host" for your project***.  
    
  Pick one of your team members and designate them as the **project host**. Submit the host's GitHub account name + names of all team members as the text submission for this TP1-1 Canvas assignment.  
    
  Only one student needs to do this Canvas submission.
* ***Download and import the starting non-Git* SkunkProject.zip *archive file into the host's Eclipse workspace.***   
    
  Have each team member start Eclipse on their own PC and configure it with their own Git credentials (GitHub name and email). Only the host is required to use their GitHub account, with host and non-hosts all committing to the host's remote GitHub repository.
* ***Meet with your team members and review the Skunk rules***.

Together, you should review the provided Skunk rules (specification), posted on our Canvas site within the TP-1 module. You will eventually implement all of its described behavior across the multiple phases of this project, but here you will first implement a version of the Skunk dice that make it easier to test your implementation.

* ***Download and import the starting (non-Git)* SkunkProject.zip *archive file into the host's Eclipse workspace.***   
    
  Download from Canvas and import the (non-Git) archive file **SkunkProject.zip** into the host's local Eclipse workspace (**Import...->General->Existing Projects into Workspace**, then select and OK). It contains starting code for the Skunk Eclipse project, named **SkunkProject**. Only do this for the host, initially. Note that the **SkunkProject.zip** file is posted as a link within the **TP-1.1 Assignment** on Canvas. We will review this operation in class.

This imported project has required starting structure, libraries, and several classes for your Skunk implementation. This includes **Dice** and **Die** within the default package within the **src** project folder. You can use them and/or modify them, as you wish. A separate project source folder named **test** is also provided: put all of your JUnit 5 test cases within a default or named package in this folder.

The project also contains the external library **lib/stdlib-package.jar**, which provides more convenient console I/O than standard Java's **System.out.println()** and such. The project folder **lib** contains this **.jar** as part of the project, thus allowing the project's **Referenced Libraries** to access it after importing. These libraries include classes **StdIn** and **StdOut** with convenient static methods for your console I/O. Access these by adding an **import** from the **.jar**'s library at the top of your source files: **import** **edu.princeton.cs.introcs.\***;

Please use **stdlib-package.jar** for your UI throughout this TP-1 project.

* ***Convert this local project into a Git-enabled one ("share the project").***

Share the host's Eclipse project (**Team->Share Project...**), creating a local repository named **tp\_1\_skunk** outside the Eclipse workspace on the host's PC. If working on a classroom PC, think about placing your repo in a folder on your **Desktop** so you can find it later. If working on your own PC, remember where you locate the repo.

* ***Create, stage, and commit a* main() *Java class* SkunkApp *to your local Git repo.***

Add the class **SkunkApp** with a **main()** method to your project. Throughout the project, this should be the starting point for running an interactive game of Skunk.

For now, have **main()** use **StdOut.println()** to print a welcome message (don't forget the **import** statement described above). Then create a **Dice** instance, invoke **roll()** against it, then print out the resulting value.

Stage these new files ("Git add"), along with others in the project, then do your first commit to your local repo. Don't forget to provide a commit message!

* ***Create an empty remote GitHub repository* tp\_1\_skunk *under the host's GitHub account.***

Add the other team members accounts as collaborators to the host's repo, along with my **seis-635-fall2019** GitHub account. Make sure that all collaborators accept the GitHub-generated email invitation. (You should be logged into your GitHub account when accepting via clicking on the email link.) Both host and collaborators should now see the **tp\_1\_skunk** repo listed within their own repositories within their GitHub account.

Note that **tp\_1\_skunk** is the name of the Git repository, not the name of your Eclipse project - which should be **SkunkProject**.

* ***Push the previous commit to this remote GitHub repo, and check your work.***

Push your commit to the empty **tp\_1\_skunk** GitHub repo under the host's account. Think about storing your GitHub logon credentials in Eclipse's secure storage, so you don't have to enter them over and over. Log onto the host's GitHub account, examine the **tp\_1\_skunk** repo and verify that this first commit was successful.

* ***Clone the GitHub repo to non-host local PC's.***

Now the other team member(s) should **clone** the host's GitHub **tp\_1\_skunk** repo to their own local PC, whether classroom or laptop. They should log onto their own GitHub account, then view and copy the shared host repo URI to the clipboard. (You did remember to add all team members as collaborators to the host's repo, didn't you?) Inside Eclipse, do **Import...->Git->Projects from Git->Clone URI** and the URI in the clipboard should be automatically entered into the dialog.   
  
Note that you don't need to log onto GitHub to clone (download a copy) the host repo, since it's a public repo - but entering and saving your own Authentication info (GitHub username and password) into Eclipse's secure store will allow you to skip logging on to GitHub for each repo access in the future.

* ***Begin your implementation of* SkunkApp *by implementing "testable" die and dice classes for use in both playing the game as well as testing it.***

The random nature of rolling dice makes it tricky to test your evolving **SkunkApp**. Thus, start by modifying **Die** to allow it to be initialized with a sequence of "pre-programmed" die values returned by repeatedly rolling such a "loaded" die. We'll discuss some possible designs for such loaded dice in class, but you are free to do this any way you'd like, including writing your own versions of die and dice classes.

Just remember you'll need to use your dice both in "random" mode in an actual game of Skunk, as well as in "pre-programmed" mode in your JUnit tests that verify the correct behavior of your game code methods.

* ***Implement at least your testable Skunk die/dice classes.***

By the due date, your minimum requirement is to implement the "testable" Skunk die and dice classes, along with complete JUnit tests for each them. You may continue with implementing more of the complete Skunk game if you wish, but this isn't required yet.

Another requirement: only use the Java console I/O using **lib/stdlib-package.jar**'s provided **StdIn** and **StdOut** when interacting with the user. No GUIs!

If you do implement actual interactive game behavior beyond just testable die/dice, I should be able to run **SkunkApp** as a Java application to see your game's behavior and play it myself. But it doesn't have to be a complete game - yet. For example, you might allow a user to play one "roll of the dice" or a complete turn (multiple rolls with turn ending with Skunk or user deciding not to roll again), or even an entire game.

In all cases, make sure you've committed and pushed all your code and tests to the host's GitHub repo. Your committed code should have no syntax errors, and all tests should succeed. If so, and if all team members have committed something to your project, and if you've done the text submission giving team members and specified host, then all team members will earn full credit. No other submissions are required.

* ***Add JUnit 5 tests demonstrating the correct behavior of your testable Skunk die/dice classes.***  
  A "best practice" for testing is to put your JUnit 5 test files in a different source folder than that containing the code under test. Suggestion: name this source folder **test**. Thus, within your Eclipse project add a new source folder (**File->New->Source Folder**) named **test**.   
    
  Within it, add your JUnit 5tests. Put them in the same package as the code under test, making it easier for the tests to access the code. (Different source folders can each contain code within the same package.)   
    
  Be sure that all of your tests succeed: 100% should pass.
* ***Experiment using Test-Driven Development (TDD).***

You are free to design your Java implementation in any way you wish but try TDD to start with. Ideally, using TDD results in better designs, but don't worry about this yet. We'll discuss this process in class, with support readings and tutorials posted on Canvas.

Remember the TDD procedure -- (a) Write a failing test, with not compiling counting as a failure; (b) Write just enough code to make the test succeed; (c) Refactor both code and tests, removing duplication and improving readability. Repeat!

* ***Work concurrently as a team, managing merge conflicts in your commits.***

Try working on your implementation with team members working independently, ideally using TDD. Before starting a coding session, each member should pull the latest changes from the host repo, thus downloading and merging them into the local repo. This helps to avoid Git merge conflicts.

If you encounter merge conflicts, resolve them as best you can; we'll see later how to deal with these situations that occur when both team members modify the same lines of code in the same files, and then try committing.

* ***Grading is based on code committed by this Phase's due date, as well as your Canvas text submission of team names and designated host.***

Soon after this Phase's due date, I will grade your work by cloning each of your host repos, running all your tests and **SkunkApp** itself, and making sure you've satisfied all the requirements. I will also check and record your Canvas text submission of team members and host account name.

Your team's grade will be based on how completely you satisfy the above requirements.