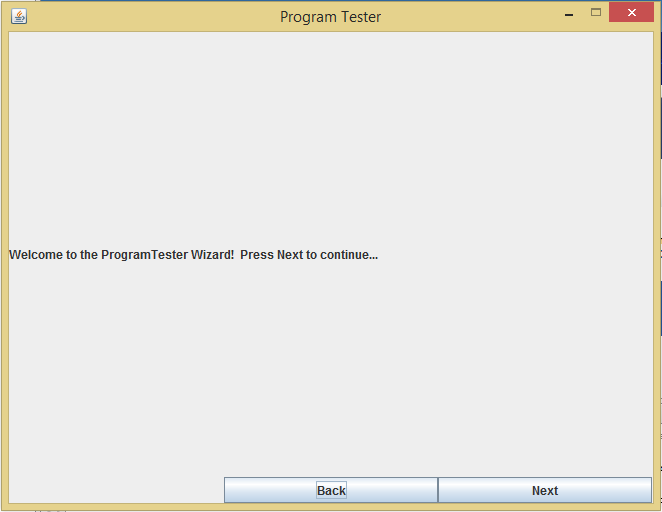
# System Description

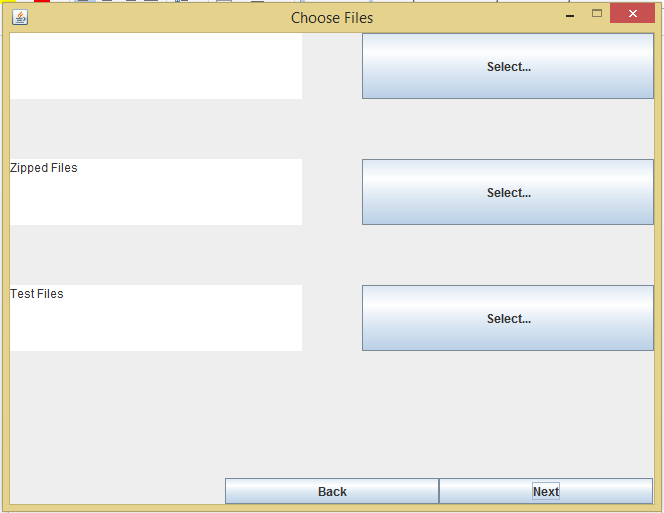
Our team took a wizard approach to creating our Program Tester UI. We wanted to enable our users to run the program correctly, out of the box, and creating a walk-through guide was just the way we wanted to do this. We took Dr. Mudgett’s working backend and added a UI with means to customize any tests to be run.

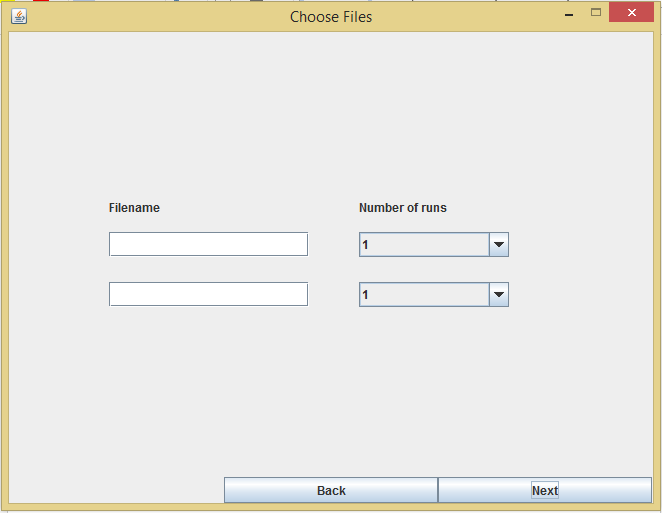
The final system has six steps:

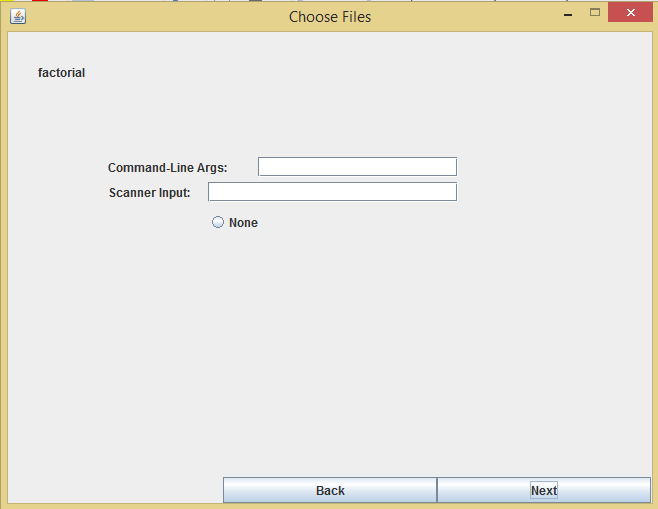
1. The user can set the directory of files, directory of zipped files, and the directory of test files.
2. The user can input the names of files to be run and the number of runs for each file.
3. The user can specify scanner and command-line inputs.
4. The user can specify the output folder and the output file name
5. The user can run the tests and view the output file.

# System Tutorial









# Grading Workflow

As it stands, our program uses the results-format created by Dr. Mudgett and could be used as a pass/fail tool, or a means to delve deeper into where students’ might be having problems in their programming. If the output file says something to the effect of, “compiler error,” one might interpret this as a failure, but other information included within the output file allows the user to potentially take a closer look as to what is causing the programmer their problem – a tool for growth.

# Functionality

Our original intention for this system included a set of requirements that included being able to edit the input files for scanner and command-line inputs, inputting the names of files you want to test and the number of runs for each file manually, selecting the location for the output file, and including a GUI in the form of a wizard. We wanted to create something usable by instructors out-of-the-box.

Currently, our system adheres to the following step criteria:

1. The user can run the Program Tester wizard GUI.
2. The user can set the directory of files, directory of zipped files, and the directory of test files.
3. The user can input the names of files to be run and the number of runs for each file.
4. The user can specify scanner and command-line inputs. – specify none for scanner
5. The user can specify the output folder and the output file name
6. The user can run the tests and view the output file.

From the beginning, there were several additional features we wanted to implement. These included allowing the user to choose a “master” project and specifying whether the output file would identify the student by student handle or student name. These are currently out-of-scope for us, but could be integrated with additional time. As it stands, we prioritized this for the distant future, since not all instructors have the same type of specific grading scheme (e.g. using versus not using student handles).

We also currently do not have functionality for viewing the students’ source code within the app, though this is a simple integration using Dr. Mudgett’s code. It simply remains to be seen where the best part of the interface would be to do this, and we prioritized this for later because it is straightforward enough to go into the downloaded folders and open java files by hand to view a particular student’s work more in-depth.

# Testing Summary

Currently in the process for writing tests. We found that the hardest part of this project was integrating the front end with the backend correctly, so this will be an area of focus as we create tests.

Our first step, which we are currently working on, is integrating our testing into the NetBeans project environment and making several sample tests run correctly. After this, the team will spend the next two weeks (sprint) creating tests on the sections of the project for which we are each responsible. This is how we will be integrating testing into our workflow.

# Do we have a basically functioning system?

Our project is nearly to a point of what Dr. Mudgett would consider “basic functionality.” We have our GUI and working backend, but we have not yet tested our system. Currently there is no built-in way to view source code, but output is provided.

# Critical Analysis

To make the system what we really want it to be, there are several more steps to take:

1. We would first want to review and fine-tune the interface, possibly running through the program with a potential user in order to clarify the wording and understand the user’s mental model for the system. This would allow us to better understand how we can make minor changes to better meet the user’s needs.
2. Second, we want to create a more cohesive grading scale or evaluation method. Instead of a simple pass/fail system, we would like to integrate feedback that could better explain errors, highlight errors within the source code for easier instructor review, or check answers against some master file.
3. Currently our system can test up to two programs with up to five runs of each. We’d like to allow the user more control over the number of files per test.

\*\*PDF AND WORD OF BOTH TUTORIAL AND THE ANSWERS TO OTHER QUESTIONS

Folder: 412-codereview3-summary-teamx, where x is your team number; then zip to 412-codereview3-summary-teamx.zip and upload to the server using the upload page by Tuesday 10/6/15 by 5pm so we can look them over before the code review on Wednesday 10/7.  
  
Dave