**Lab Assignment 6**

**Atlantis Entry Checkpoint System**

|  |  |
| --- | --- |
| **Lab Time** | **Due Date** |
| **Wednesday 10:30am** | **4/2 10:30am** |
| **Wednesday 12:30pm** | **4/2 12:30pm** |
| **Thursday 2:30pm** | **4/3 2:30pm** |

**Objectives**

* To practice making Advanced GUI application.
* To practice reading from and writing to a file.
* To practice using exceptions in Java
* To practice following the Pair Programming model.
* To practice using source control.

**Problem Specification**

Write a GUI application that works like a checkpoint system for a country named – “Atlantis”. Every time a person enters the country, an enrollment is made specifying his passport details by the immigration officer on duty. Officer should also be able to see all the enrollments made.

// gui and data have to be in separate classes

// set and get for filenames

// PROPERLY LABEL ALL GUI STUFF

**GUI Specifications**

Your Java application should:

* Allow the user to enter all the details of the passport through an enrollment form.
  + Please see Figure 2 for the minimum suggested fields, However you can use more fields than suggested.
    - Fields: Passport Type and Visa Type need to be an un-editable selectable box. Suggested values for Passport Type are “P” and “UNK” and Visa Type are “A”, “B” and “C”.
  + Allow the user to make multiple enrollments.
  + Display a message to confirm the data saved into the text file after each enrollment.
  + Clear all the fields after every successful enrollment, so that officer can enter new data.
  + Don’t allow user to save any enrollment data until all the fields are filled. Display a message to enter the appropriate data if some fields are incomplete. // use nullpointer exception
* Allow the user to go to menu form [see Figure 1] from the enrollment from.
* Allow the user to load the contents of the file to see all the enrollments. // cannot click “load” button unless text is in textfield. Only works if file is same as one you are writing to
* Allow the user to choose the data file containing data of the enrollments to view it in the form.
  + Don’t allow the user to load an incorrect file. Display a message if he does so.
  + Don’t allow the user to load a file without selecting one.
* Allow the user to make a new enrollment or load the enrollment data file any time in the application. Meaning he should be able to switch between windows seamlessly.
  + User shouldn’t only see one window during anytime of the application. Hiding a parent component behind a current component is not allowed.
* To simplify GUI, resizing of a window is not allowed.

Here is an example of what your windows might look like. This is only an example. You should put some thought into designing your window to be user-friendly and attractive.

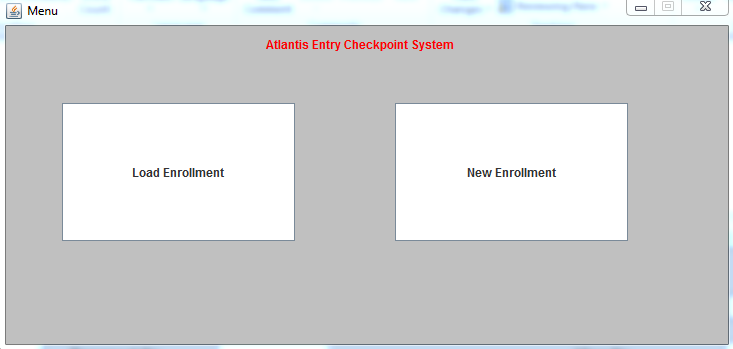


Figure 1 - Menu option

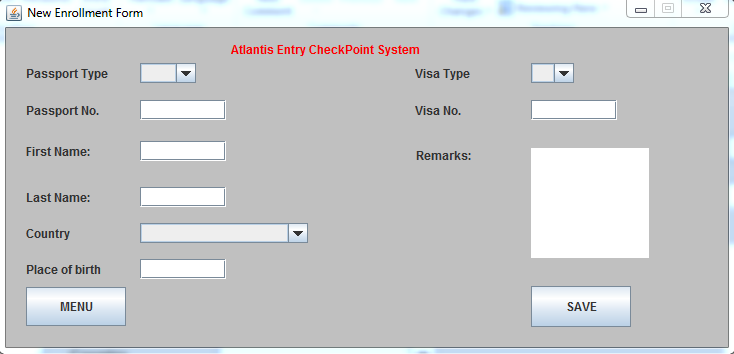


Figure 2 - Enrollment form

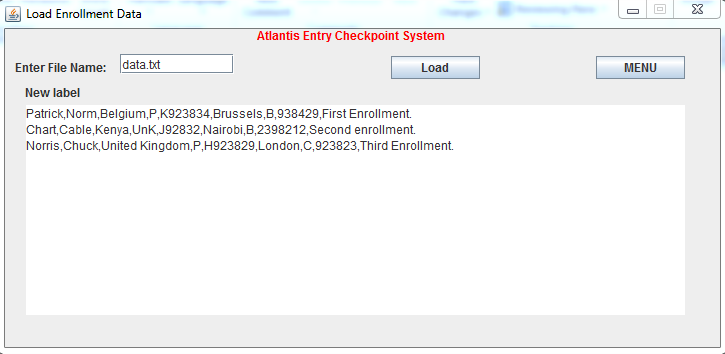


Figure 3 - Load Enrollment data

**Design Requirements**

* Your application **MUST** read from a text file while loading and write to a text file while saving enrollments.
* Use exceptions wisely to avoid application crashes. Some commonly used exception for this problem will be
  + FileNotFoundException
  + NullPointerException
* You should use drag and drop [Windows Builder] for this application.

### SVN Requirements

Your team **must use Subversion (SVN)** version control software for this assignment. Each person on the team must make multiple revisions to your repository, and every revision should have a meaningful comment. You must have **at least 2 revisions with meaningful comments from each team member** to get the credit for this part of the assignment. You have to submit the revision graph for your repository with the assignment. TA’s can show you the way to view revision log for your repository.

Each team member will *individually* submit the full assignment solution on Elearning, including the SLC report and JavaDoc documentation, as usual. However, all team members will receive exactly *the same grade*. Include the names of all team members on all submissions. You only need to submit **one** **paper copy** of the SLC report.

**Pair Programming**

**// submit one slc**

You will work in two-person teams for this assignment and **must utilize the pair programming model** to accomplish this task. Please refer to the additional pair programming explanations below as well as the instructions on pair programming received from your course and lab instructors to complete this lab assignment.

Two specific and well-defined roles, which require an equal amount of pair member contributions but have different responsibilities, are required:

* ***Driver:*** ‘Drives’ the keyboard and mouse. Sits in front of the keyboard and screen, changes any removable media but does not handle hardcopy documentation.
* ***Navigator:*** Does not touch the computer or any hardware. Handles documentation, makes observations and suggestions, points out potential problems, finds solutions to driver questions.

*Swapping roles*is required. This should be done equally and on a regular. Paired programming at a distance is possible (although restricted) through the use of communication software, email, cell phones, etc.

You must keep a *pair programming log* that describes how you followed the pair programming model. It should describe each session, including who took which role, how much time you spent during that session, and what you worked on as a pair. Each time you switch roles, create a new entry in the log.

Each pair member will submit the full assignment solution, SLC report, and *pair programming log* on E-learning *individually*, as usual; however, both partners will receive exactly *the same grade*. Include the names of both partners on all submissions. You only need to submit one paper copy of the SLC report.

**Code Documentation**

During the lab, we discussed how to use JavaDoc to generate documentation for your code. For this assignment, you must include documentation for your code as generated by JavaDoc. You should have JavaDoc comments for every class, constructor, and method. By default, JavaDoc should output html documentation to a subfolder within your project (/dist/javadoc). Make sure this folder is included when you zip your files for submission. You do not need to submit a hard copy of this documentation.

**Software Life Cycle Report**

You are required to write the SLC Report, explaining how you followed the Software Life Cycle as you completed this assignment for all the nine phases. A proper design (with stepwise pseudocode refinement), proper coding method (with stepwise code refinement starting from the most detailed pseudocode refinement), and proper testing are essential. For reference, please see the **Sample SLC Report** and **SLC Report Template** on course web pages.

*Note: Having* ***proper pseudocode*** *will be worth* ***20%*** *of the total LA grade.*

**Coding Standards**

You must adhere to all conventions in the CS 1120 Java coding standard. This includes the use of white spaces for readability and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions for naming classes, variables, method parameters and methods.

**Testing**

Make sure you test your application with several different values, to make sure it works.

**Assignment Submission**

* Generate a .zip file that contains all your files, including:
  + Program Files
    - Including any input or output files
  + The SLC Report file (a text with description of all nine phases of Software Life Cycle).
* Revision Graph of the repository.
* Javadocs
* Pair Programming Log