Project Design Example

1. Title

- Project Title: Individual Student Error Reports
- Program Name: exam_results

2. Author(s)

- Debra Deppeler (deppeler@cs.wisc.edu)

3. Description

A Java program that parses the exam results (.csv) from Testing and Evaluation and produces an error report that can be provided electronically to students without the need to print hundreds of pages and spend lecture time getting each page returned to students individually.

The error report shows the item number, correct answer, student answer, and brief description of what the question tested.

4. Stakeholders

Instructors

Will configure the exam information, run the program to generate reports and inform students of their availability. This will take about the same amount of time as printing the reports, but it will not require time spent during lecture to return the results. This time can be better used for discussing individual question results.

Students

Students will get the same information and a little more given that a description can be included for each question's results. Students will no longer need to wait while all other student's error reports are returned. Students can view the error report via a secured course file server.

NOTE: student prefer to get their scores via email, and not have to login to a CS file system.

5. OUTPUT

Individual Error Report (one for each student)

Use an error report (template.txt) file to configure the form of the error report for each student. Program will generate a report for each student and save at: /p/course/cs400-deppeler/handin/201901/studentCSlogin/e1_error_report.txt

```
  $NAME$
  $SUBSCORE1$
  $SUBSCORE2$
  $TOTALSCORE$

  Item Key Yours Description
  ---------------------------------------------------------------------
  1.    A    A
  2.    B    B
  3.    B    A    git push command
  4.    A    A
  5.    B    B
  6.    B    A     hashing extraction term
...
```

6. INPUT

config.txt

Contains the meta-data for each exam event: exam id, number of items in each exam part, max possible score for each exam part.

```
QUESTION_FILE = questions.csv
ROSTER_FILE = roster.csv
TEMPLATE_FILE = template.txt
RESULT_FILE = cp998-cp999.csv
RESULT_FILE_P = cp998.csv
RESULT_FILE_S = cp999.csv
OUTPUT_DIR = e1_results
```

TODO: implement the use of this file. Currently, the above values must be hard-coded into Main program.

template.txt

Contains the output that is common for all students. Individual student information will replace place-holder variables in the template.

```
Exam: %s

Results for: %s

Simple Choice:   %4s    / 12    (questions 1-12, 1pt each)
Multiple Choice: %4s    / 57    (question 13-30, 3pts each)

TOTAL SCORE:      %6s / 69    PCT: %s

ITEM  KEY  YOUR ANSWER (x means incorrect answer)
=========================================================================
```

roster.csv

Contains data for each student: name, photo_id, cslogin, netid, canvas_id, email@wisc.edu,

```
lec,lab,photo_id,canvas_ID,SIS_User_ID,campus_email,last_name,first_name,middle_name,login,netid,canvas_name
001,000,9012345678,123456,bucky@wisc.edu,bucky@wisc.edu,Badger,Bucky,,bucky,BUCKYB,BUCKY BADGER
001,000,9012345679,234567,becky@wisc.edu,becky@wisc.edu,Badger,Becky,B,becky,BECKYB,BECKY BADGER
```

questions.csv (key, and mapping from secondary to primary)

Contains the data for each question, the item's primary number, the correct answer, the secondary version item number, description

```
primary,answer,secondary,filename,description
1,A,4,TF_AVL_compare_RBT_lookup_s19.tex,Lookup operation RB vs AVL
2,B,5,TF_TESTING_whitebox_s19.tex,Testing white box vs black box
```

cp998.csv and cp999.csv

The output that is produced by Testing and Evaluation from scanning each student's scantron form. A separate file for each version as received from Testing and Evaluation. Columns include: last name, first name, uw ID number, special codes, item results, computed subscores, and total score.

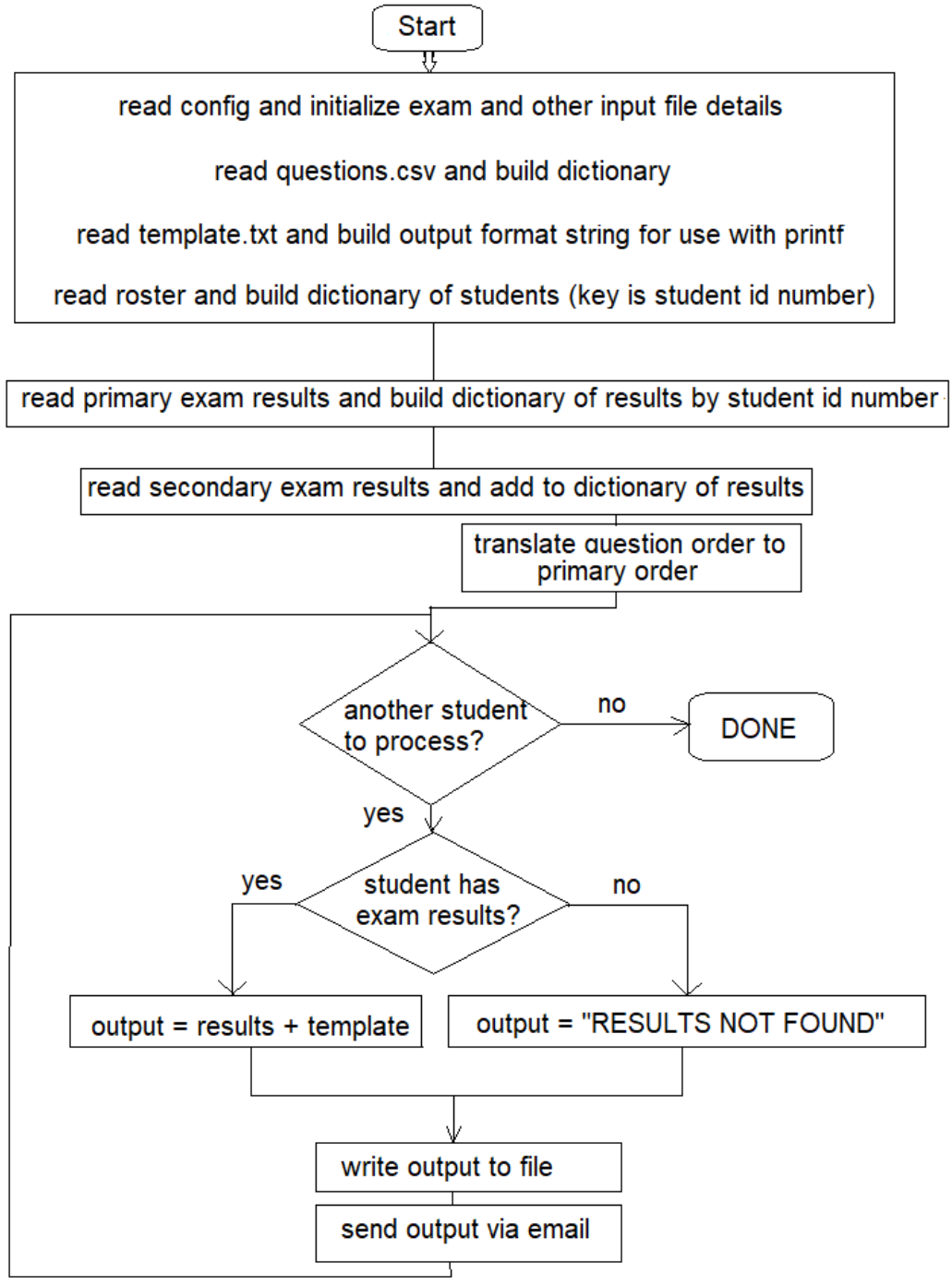
```
LASTNAME,FIRSTNAME,MI,ID,SPECIALCODES,TOTALSCORE,TOTALPCT,SUBSCORE1,SUBSCORE2,_1,_2
BADGER   ,BUCKY   , ,9012345678,001  2    ,66,95.65,12,54,2,2
BADGER   ,BECKY   , ,9012345679,001  1    ,61,88.41,10,51,1,2
```

Note: Reorder question item information to primary order For students who took the secondary version.

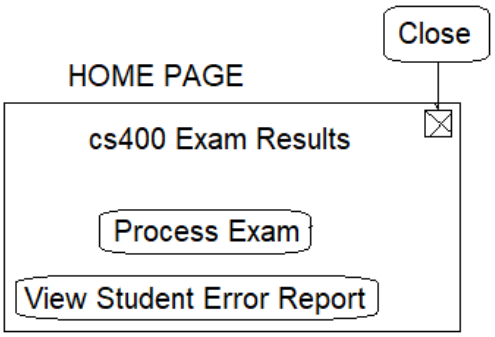
7. Diagrams

7a. Control Flow Diagram

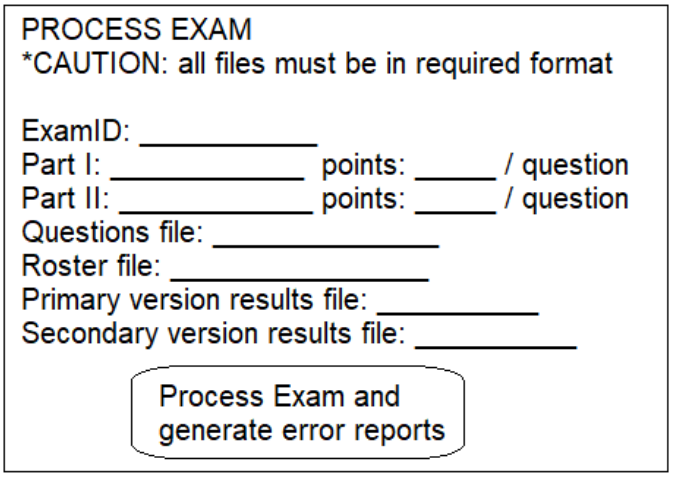
control-flow-example.png



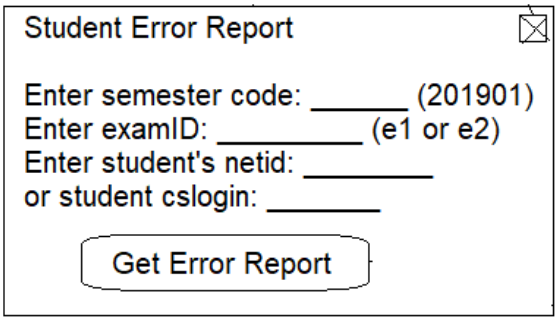
7b. Example Home Page



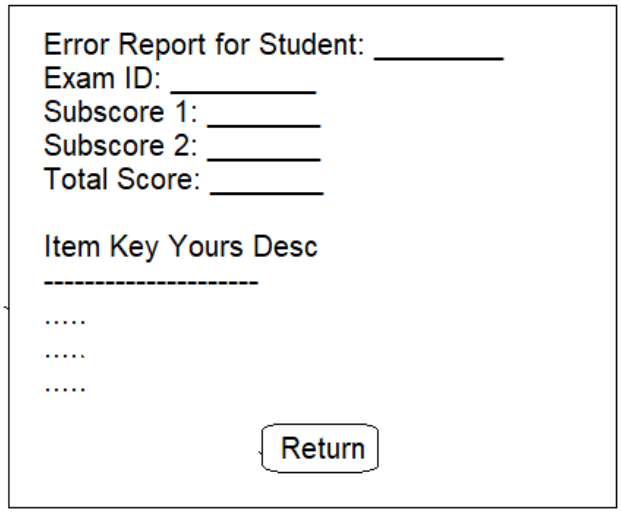
7c. Example Process Exam Results Page



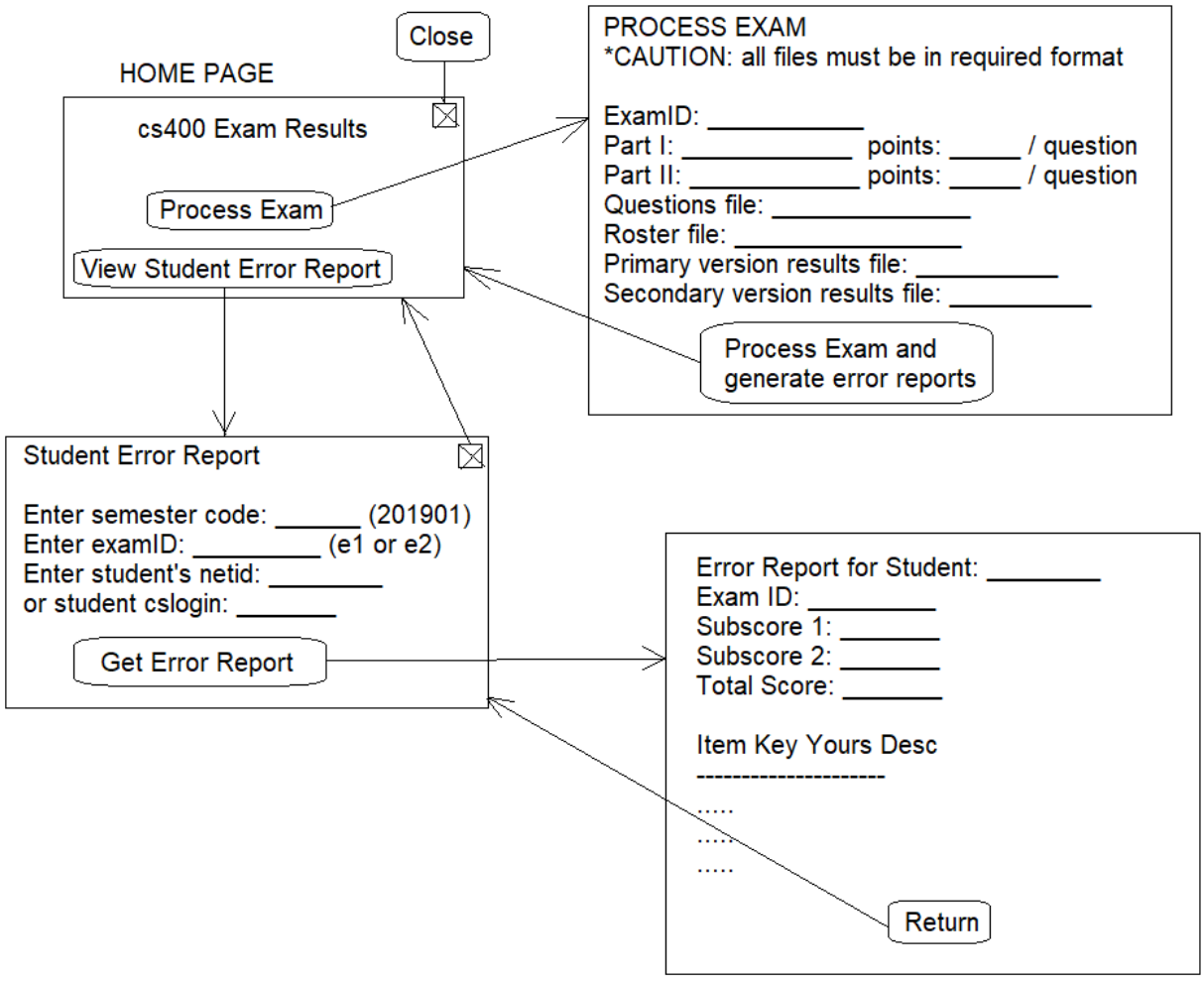
7d. Example Select Student Page



7e. Example View Student Page



8. Screen Graph



9. Class Summary

Question (one instance per question on exam)

- item - item's number on the primary version
- answer - A-E indicating the correct answer for this version
- secondary - item's number on secondary version
- points - the number of points for this question
- filename - the file that contains the question text
- description - 70 character or less description of the question topic

Student (one instance per student)

- name - preferred student name to be displayed on final report
- uwid - (primaryKey) - used to match student with exam result
- netid - student's unversity NetID
- cslogin - student's login on CS file system - used to store file and grant appropriate permissions
- canvasid - canvas's id - UNUSED, but may use in future to match with Canvas scores

Note: The above was replaced with the following:

- fieldList - the colunm names from the roster.csv file
- hashMap - hash table (dictionary) to find value that corresponds to a field name

Example: If student is a Student object

student.hashMap.get("netid") would return the netid for the student

ExamResult (one instance per student)

- hashMap - dictionary that maps each exam result field to its value for a given student

Note: also contains static methods for building a hashMap of all results by student ID

MainTest (JUnit 5 tests with sample "stub" data)

- read exam results data file and ensure that can get results for sample students as expected
- read student roster file and ensure that sample students are available as expected from collection of student
- read question scramble files and ensure that correct primary item number is returned for sample questions
- combine individual test results with output template file and confirm that values are posted as expected
- other tests ????

10. Implementation Phases

1. Milestone 1

- Get basic functionality working from the command-line.
- Pass unit tests and run through with sample data.
- Confirm format for input and output, and output template.
- Work on design, develop, and test the most independent pieces before combining to larger solution components.

2. Milestone 2

- Get non-functional GUI working, add buttons to see different UI forms and screens.
- Get stake-holder approval to proceed here. But, I am primary stake-holder, so if I like it, keep it (for now).

3. Milestone 3

- Add functionality to get output from sample data.
- Test and verify results are correct.
- Run with actual exam data and check against provided error reports for students across all lectures and exam versions.

4. FUTURE

- Add overall exam stats to the template.txt and thus final error reports for each student.
- Maybe even a histogram of some sort?
- Refine User Interface for instructor
- Port solution to a secure web application.
- Add ability for individual student access to their results through GUI.
- Send report via email (student preference).
- Get student preference for recieving report (via email or login to file system and distribute according to student preference).