1 Submission Instructions

Create a folder named <*asuriteid*> where *asuriteid* is your ASURITE user id (for example, since my ASURITE user id is *kburger2* my folder would be named *kburger2*) and copy all of your *.java* source code files to this folder. Do not copy the *.class* files or any other files. Next, compress the <*asuriteid*> folder creating a **zip archive** file named <*asuriteid*>*.zip* (mine would be named *kburger2.zip*). Upload <*asuriteid*>*.zip* to the Project 3 link by the project deadline. The deadline is in course schedule info . Consult the online syllabus for the late and academic integrity policies.

2 Learning Objectives

1. Complete all of the learning objects of the previous projects.
2. To implement a GUI interface and respond to action events.
3. To implement and use the binary search algorithm.
4. To implement and use the merge sort algorithm.
5. To implement the Comparable interface.

3 Software Requirements

Your program shall meet these requirements.

1. Student information for students in a course is stored in a text file named *gradebook.txt*. There is one student record per line, where the format of a student record is:

   last-name first-name hw1 hw2 hw2 hw3 exam1 exam2

   where:

last-name	The student's last name. A contiguous string of characters.
first-name	The student's first name. A contiguous string of characters.
hw1-hw4	The student's scores on four homework assignments, may be zeros.
exam1-exam2	The student's scores on two exams, may be zeros.

   Here is an example *gradebook.txt* file:

   Sample *gradebook.txt*
   ```
   Simpson Lisa 25 25 25 25 100 100
   Flintstone Fred 15 17 22 18 80 60
   Jetson George 20 21 22 23 70 83
   Explosion Nathan 5 4 3 2 1 0
   Muntz Nelson 20 15 10 5 60 70
   Terwilliger Robert 23 21 19 17 80 90
   Flanders Ned 12 14 17 23 85 95
   Bouvier Selma 16 16 16 16 16 16
   Spuckler Cletus 1 2 3 4 5 6
   Wiggum Clancy 6 5 4 3 2 1
   Skinner Seymour 19 23 21 24 78 83
   ```
2. When the program starts, it shall read the contents of *gradebook.txt* and sort the list of students into ascending order.
3. The program shall implement a GUI which permits the user to interact with the gradebook. Watch the Project 3 video lecture for a demonstration of how the GUI works.
4. When the enters a student's last name in the search text field and clicks the Search button, the homework and exam information for the student shall be displayed in the text fields.
5. When the user clicks the Search button and the search text field is empty an error message dialog shall be displayed.
6. When the enters a last name in the search text field and clicks the Search button, if the student is not found because the last name was entered incorrectly an error message dialog shall be displayed.

7. When the user is editing the information for a student and clicks the Save button, the student record shall be updated (these changes will be written to *gradebook.txt* when the program exits).
8. When no student record is being edited (the homework and exam text fields are empty) and the user clicks the Save button, nothing shall happen.
9. When the user is editing the information for a student and clicks the Clear button, the homework and exam text fields shall be set to empty and the student record shall not be updated.
10. When the user is editing the information for a student and clicks the Exit button, the student record shall be saved before exiting.
11. Whether the user is editing student information or not, when the Exit button is clicked the student records shall be written to *gradebook.txt* and the program shall terminate.

4 Software Design

Refer to the UML class diagram in Section 4.10. Your program shall implement this design.

4.1 Main Class

A template for *Main* is included in the zip archive. The *Main* class shall contain the *main*() method which shall instantiate an object of the *Main* class and call *run*() on that object. Complete the code by reading the comments and implementing the pseudocode.

4.2 CourseConstants Class

The complete CourseConstants class is included in the zip archive. This class simply declares some public static constants that are used in other classes.

4.3 GradebookReader

A class which reads the gradebook information from *gradebook.txt* and returns a *Roster* object which contains the student information. The complete code for this class is provided.

4.4 GradebookWriter

A class which writes the gradebook information to *gradebook.txt* before the program exits. This class inherits from *java.io.PrintWriter*. This class is very simple. Read the comments and implement the pseudocode.

4.5 Roster

Stores the student information in an *ArrayList<Student>* list. Read the comments and implement the pseudocode.

4.6 Searcher

This class shall implement one static method int *search*(*ArrayList<Student> pList*) which searches the list of *Roster* for a student with the specified last name. Since the roster is sorted into ascending order by last name, you shall implement either the iterative or recursive binary search algorithm. The method returns the index of the student in the list or -1 if the student is not found. Template not provided; use the UML class diagram.

4.7 Sorter

A class which implements the quicksort algorithm. All of the method are static and *sort*(*ArrayList<Student> pList*) is the only public method and calls private *quickSort*(*pList*, 0, *pList.size*() - 1) to sort the list. Template not provided; use the UML class diagram.

4.8 Student

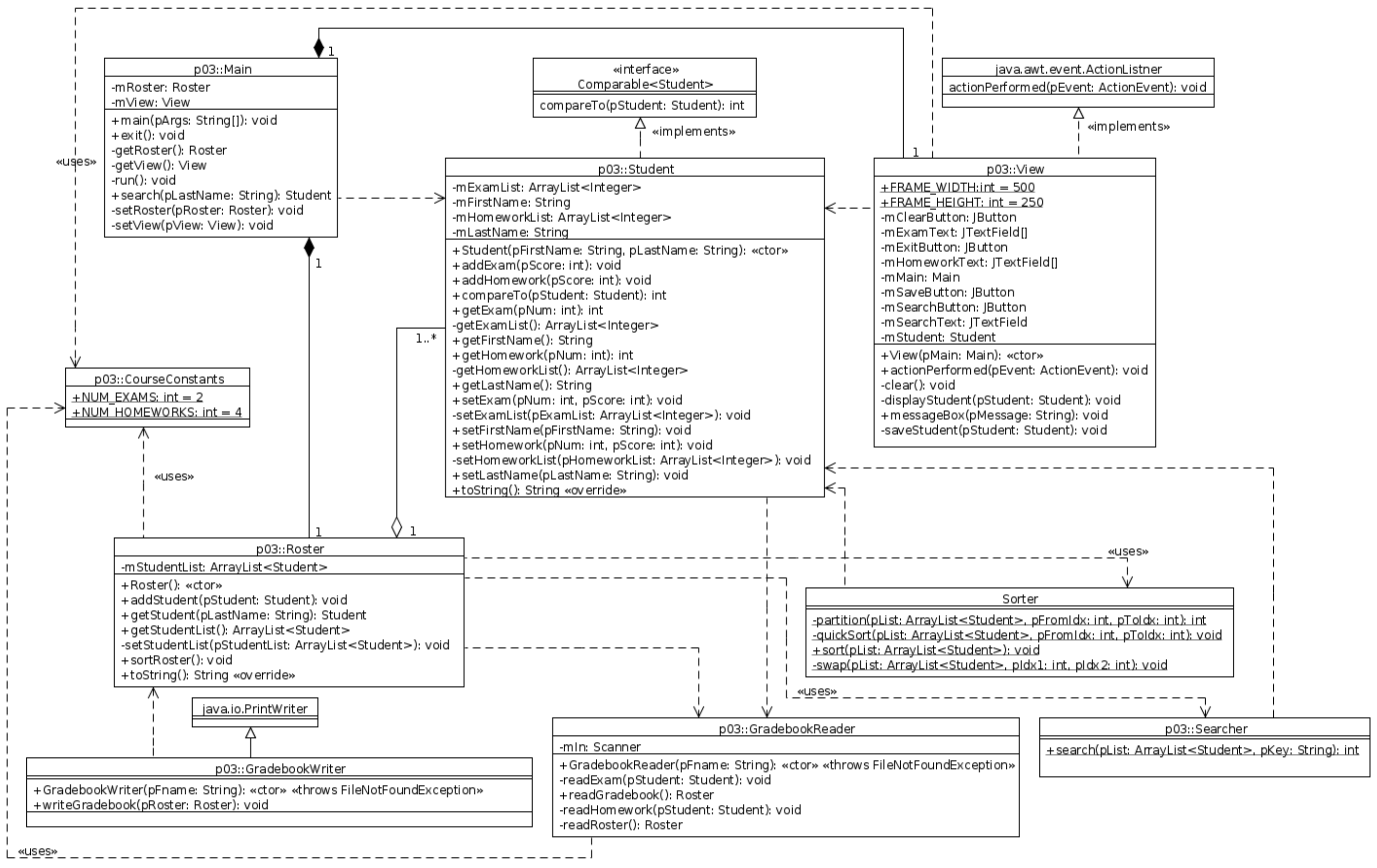
The *Student* class stores the information for one student. Read the comments and implement the pseudocode.

4.9 View

The *View* implements the GUI. Read the comments and implement the pseudocode.

4.10 UML Class Diagram

The UML class diagram is provided in the zip archive in UMLet format and as a PNG image. Your program shall implement this design.


p03::Main
-mRoster: Roster
-mView: View
+main(pArgs: String[]): void
+exit(): void
-getRoster(): Roster
-getView(): View
-run(): void
+search(pLastName: String): Student
-setRoster(pRoster: Roster): void
-setView(pView: View): void
«interface» Comparable<Student>
compareTo(pStudent: Student): int
java.awt.event.ActionListner
actionPerformed(pEvent: ActionEvent): void
«implements»
«implements»
«uses»
1
1
1..*
1
1
p03::Student
-mExamList: ArrayList<Integer>
-mFirstName: String
-mHomeworkList: ArrayList<Integer>
-mLastName: String
+Student(pFirstName: String, pLastName: String): «ctor»
+addExam(pScore: int): void
+addHomework(pScore: int): void
+compareTo(pStudent: Student): int
+getExam(pNum: int): int
-getExamList(): ArrayList<Integer>
+getFirstName(): String
+getHomework(pNum: int): int
-getHomeworkList(): ArrayList<Integer>
+getLastName(): String
+setExam(pNum: int, pScore: int): void
-setExamList(pExamList: ArrayList<Integer>): void
+setFirstName(pFirstName: String): void
+setHomework(pNum: int, pScore: int): void
-setHomeworkList(pHomeworkList: ArrayList<Integer>): void
+setLastName(pLastName: String): void
+toString(): String «override»
p03::View
+FRAME_WIDTH:int = 500
+FRAME_HEIGHT: int = 250
-mClearButton: JButton
-mExamText: JTextField[]
-mExitButton: JButton
-mHomeworkText: JTextField[]
-mMain: Main
-mSaveButton: JButton
-mSearchButton: JButton
-mSearchText: JTextField
-mStudent: Student
+View(pMain: Main): «ctor»
+actionPerformed(pEvent: ActionEvent): void
-clear(): void
-displayStudent(pStudent: Student): void
+messageBox(pMessage: String): void
-saveStudent(pStudent: Student): void
p03::CourseConstants
+NUM_EXAMS: int = 2
+NUM_HOMEWORKS: int = 4
«uses»
p03::Roster
-mStudentList: ArrayList<Student>
+Roster(): «ctor»
+addStudent(pStudent: Student): void
+getStudent(pLastName: String): Student
+getStudentList(): ArrayList<Student>
-setStudentList(pStudentList: ArrayList<Student>): void
+sortRoster(): void
+toString(): String «override»
«uses»
Sorter
-partition(pList: ArrayList<Student>, pFromIdx: int, pToIdx: int): int
-quickSort(pList: ArrayList<Student>, pFromIdx: int, pToIdx: int): void
+sort(pList: ArrayList<Student>): void
-swap(pList: ArrayList<Student>, pIdx1: int, pIdx2: int): void
java.io.PrintWriter
«uses»
p03::GradebookWriter
+GradebookWriter(pFname: String): «ctor» «throws FileNotFoundException»
+writeGradebook(pRoster: Roster): void
p03::GradebookReader
-mIn: Scanner
+GradebookReader(pFname: String): «ctor» «throws FileNotFoundException»
-readExam(pStudent: Student): void
+readGradebook(): Roster
-readHomework(pStudent: Student): void
-readRoster(): Roster
p03::Searcher
+search(pList: ArrayList<Student>, pKey: String): int
«uses»

5 Additional Project Requirements

1. Format your code neatly. Use proper indentation and spacing. Study the examples in the book and the examples the instructor presents in the lectures and posts on the course website.
2. Put a comment header block at the top of each method formatted thusly:

```
/**
 * A brief description of what the method does.
 */
```

3. Put a comment header block at the top of each source code file formatted thusly:

```
//**************************************************************************************************
// CLASS: classname (classname.java)
//
// COURSE AND PROJECT INFO
// CSE205 Object Oriented Programming and Data Structures, semester and year
// Project Number: project-number
//
// AUTHOR
// your-name (your-email-addr)
//**************************************************************************************************
```