Laboratory 11: Law of Demeter

Java API
Javadoc for EasyGridBag

In this lab, you will be designing and implementing the model portion of the Model-View-Controller design pattern. You will need a set of classes (a model) to allow the LabManager GUI to track the progress of lab students throughout the semester.

The number of sections that the LabManager will be working with is specified in the main method. In this lab, there will be two sections. Each section will have a different number of students, with the data for each section read in from and stored to a separate text file. The text file will store (in a comma delimited file) student ID, student name, lab grade and partner ID for each completed lab (14 total). A student is assigned a random partner each lab period. An odd number of students present for a given lab will result in one randomly chosen student as their own partner. If a student is present for the lab and works on the lab with their partner for the entire class, that student (and partner) receives a default 'D' for the lab, even if the lab is not complete or has several major issues. Completion of the lab before the final due date specified in the syllabus results in an A, B, C, or D depending on how many major or minor issues are still present in the final lab submission.

Interpreting a grade:

- A: lab completed with no issues (1.00 points)
- B: lab completed with a few minor issues (0.75 points)
- C: lab completed with a few major issues (0.50 points)
- D: lab not completed or several major issues (0.25 points)
- F: lab not attended or not working with partner (0.00 points)

Final Grade:

- A: >= 13.25 points
- B: >= 11.00 points
- C: >= 8.50 points
- D: >= 4.50 points

Lab:

- **Model**-View-Controller Design Pattern
- Class Cohesion and Separation of Responsibilities
- Complete and Consistent Protocol
- Law of Demeter
- Expert Pattern

Part I: Class Cohesion and Separation of Responsibilities

Most of the work for the actual GUI has been completed for you. Thus, your model must interface with the GUI using the protocol listed in Part II. This means that your model will include, at a minimum, a **Sections** class.

Try to determine what minimal set of classes that you need to perform the duties described at the top of the lab. You are only concerned with the classes required to store the data (the model), not the GUI. Remember that classes have knowing, doing, or controlling responsibilities (separation of responsibilities). The Expert pattern also suggests that classes responsible for knowing data should manipulate it (so some classes may have both knowing and doing responsibilities). Make sure that you adhere to the **Law of Demeter** in your design, and consider any additional classes that may be needed for proper class cohesion.

Download the following files:

- Random.java //no work
- Lab.java //no work
- Labs.java //no work
- Student.java
- Students.java
- Section.java
- Sections.java

Part II: Model and Law of Demeter

Using the classes provided in the previous part, implement the following public interface for a **Sections** class so that it can interact with the GUI (**some of these methods may have been completed for you**):

- Sections(int num_sections, double[] grade_constants, int total_num_labs) //constructor
- int getNumSections()
- int getNumStudents(int sectionID) //the number of students in a given section
- int getNumLabs(int sectionID) //the number of labs completed by the class for a given section
- int getNumLabs(int sectionID, int studentID) //the number of labs completed by an individual student
- int getPartnerID(int sectionID, int studentID, int labID) //a student's partner for a given lab (note consistent parameter ordering)
- String getStudentName(int sectionID, int studentID)
- boolean isPresent(int sectionID, int studentID, int labID)
- String studentInfo(int sectionID, int studentID) //all lab grades for a single student (used in top text box in GUI)
- String partnerList(int sectionID, int labID) //list of partners for the requested lab
- void computePartners(int sectionID) //randomly assign partners for the lab
- boolean isActive(int sectionID, int studentID) //is student still in the class?
- void setInactive(int sectionID, int studentID) //student has withdrawn
- void setNotPresent(int sectionID, int studentID, int labID) //student absent for lab (simply set the lab grade to F)
- void setGrade(int sectionID, int studentID, int labID, char grade) //update the student's grade for the lab (and partner's)
- void addLab(int sectionID) //starting a new lab
- void writeFiles()
- **add at least one convenience method (calls essential methods to perform a task)**

Note: IDs passed as parameters are 1-based!

Note that the method signatures (studentID and labID passed as parameters with a consistent ordering) imply that the Law of Demeter is followed by the GUI. That is, instead of obtaining a reference to a student and then obtaining a reference to a particular lab, the GUI passes the student and lab IDs as parameters, and lets the classes that control the relevant data handle the request.

Now, using the above interface as a guide, finish the **Section, Students,** and **Student** classes so that Sections calls methods from Section, Section calls methods from Students, Students calls methods from Student, and Student calls methods from Labs (Law of Demeter). Note that parameters in these classes may be different than the above interface, and not all methods will need to be repeated for every class (ex. getNumStudents() does not need to be included within the Student class, as Students will return the number of students in its getNumStudents() method).

Even though the GUI uses defensive programming before calling the above methods, your methods should check that the parameters are valid (preconditions are met) before proceeding. To save time, you may use simple error checking (error codes or default values) rather than exception handling. Methods that have several parameters should have their parameters verified in different classes.