[SOEN343] PROJECT MILESTONE 4

Laura BERNHARDT – Guillaume FORTIN – Magalie PERON – Derek RINGUETTE – Nicolas TRICHEUX

Project Cabra

Laura BERNHARDT - #7118112
Guillaume FORTIN - #6325300
Magalie PERON - #7129173
Derek RINGUETTE - #1947834
Nicolas TRICHEUX - #7129491

MILESTONE 1: PROJECT SELECTION

Project description

Cabra is a free open source program meant to enhance your studying skills. It revolves around the concept of flashcards. These come in the format of a question and an answer or multiple answers, which allow you to classify how well you understand the subject (know it, sort of, don't know). The program is only limited by the amount of cards or projects you create, which can be categorize by lesson and further grouped into parent subjects. For each project (set of cards) the program shows you statistics on how well you know it and shows you your progression into learning the new material.

This is an interesting project because it was conceptualized and is currently being developed by a single high school student. The most recent version is very popular (several thousand downloads) and it is cross platform (multi-os, mobile and browser based). To some extent teachers can even make their own study cards based on the material they want the students to learn and share it with them using a simple passcode. Moreover, the source code is very organized in terms of comments. This way, the algorithms being implemented are easier to understand and could be a great help to us in the refactoring phase of the project.

Project maturity

Cabra's maturity level is stable. The earliest posted version of the project was submitted on March 20, 2011. The project is currently active with its latest update submitted April 6, 2013. The update prior to that one was posted on October 29, 2013. Only one developer has worked on the project: a high school student named Neel Mehta. We have communicated with the developer since we have taken an interest in his application. He is always working on Cabra, he wants to add some features to the desktop version. He even asked us if we wanted to help him improve the project and give us Github access.

Project domains

Cabra's project domains are related to studies, as the project aims at helping students learn in a more efficient way. The primary users for this software are students who want to use a tool to take better course notes and prepare efficiently for exams by memorizing and practising. Teachers could also use Cabra to make students learn faster and more easily.

The project features are the following:

- ✓ Flashcards: created by the user or imported, they can be used to test your knowledge on a particular subject
- ✓ Notes: taken by the user, allow highlighting of important points and possibility to waste less paper
- ✓ Leitnerian Algorithm: cards are sorted into two groups depending on the user's success rate, and are displayed accordingly
- ✓ Interface: intuitive and easy to personalize, aimed at any type of student
- ✓ Portability: the application can be used either on a computer, an Internet browser or a smartphone based on Android or iOs

Project size and scope

Using SLOCcount program, we have found that the entire project is made of 5604 lines and 89 classes. Therefore, we have chosen to concentrate on specific features such as the flashcard and note modules. These two features are made of 22 classes and 2357 lines of code. We are interested in working on these modules because the flashcards and the notes represent the core of the software and are what truly make it interesting. Furthermore, because the software is based on these two modules, we think that it would be a great help for the developer if the core of his application is refactored. We have already studied the code of this project and we have found some smelling codes such as "switch cases", duplicated codes and long methods. That's why we assume this is a reasonable amount of work.

Groups members

Laura Bernhardt

Cabra is written in a language that I know : Java. I already did projects on Java and I have a course of Java this term. Moreover, I studied C++ and C#, so I know several concepts of OOP as : design patterns and several kinds of refactoring (pull-up method, consolidate conditional fragments, extract methods, …). I think I can contribute actively to Cabra refactoring. However, I think I can learn a lot too because there are lots of concepts I didn't used to implement in terms of refactoring and OOP to clear and improve the code. Finally, I'm very interesting about Cabra project because it implies several features as loading/creation of flash cards, notes, sharing with friends and there is a concept of events with GUI.

Guillaume Fortin

Cabra is a small and compact study aid program, it revolves around the use of virtual flash cards to organize your material. It is a very intuitive and fun to use application, proves that the simplest ideas are often the most effective. Cabra is a java written program, I have some a bit of experience in java, both android and swing. I don't have as much experience in OOP as in procedural programming, but I am trying to slowly transition and I believe this project will help me do so. I pick up new programming techniques very quickly and I believe that with what I learn throughout this semester I will be able to contribute in the cleanup and refactoring of this project. I have already noticed some files and classes that could use it.

Magalie Peron

I am really interested into Cabra project because first I find it is a really useful tool. The principle of testing yourself based on question cards is quite smart, and the fact that you can do it on your smartphone anywhere at anytime is time saving. I also like this idea of participating in a project aiming at helping people, here to learn efficiently, and promoting knowledge sharing between users.I think I could help on that project because I studied Java and Swing over the last year, which is the language that is being used to develop Cabra. Furthermore I am convinced that working on a real-world project and exchange with its creator would be a great and rewarding experience for me, in terms of both group work and Object Oriented programming.

Derek Ringuette:

First and foremost, I believe Cabra is a good application to work on for this project because it is written in the Java programming language, which is the programming language I am most familiar and proficient with. Having graduated from a three-year java-centered computer science program in CEGEP, I believe I have a lot to offer my team in terms of experience and ideas. The one setback this project seems to have is that it contains more classes in its architecture than is recommended in the specifications. But I think my team and I can work around this minor problem by restricting ourselves to a well designed scope to focus on. In summation, I think this project will really enhance my learning in subjects such as teamwork, refactoring and scope selection. It will also add to my Java and object-oriented programming knowledge as a whole.

Nicolas Tricheux:

Cabra is a useful flashcard program, written in Java, based on the Leitnerian Algorithm, that helps student to better organize their learning sessions.
I’m really interested in working on the Cabra project because it seems to be a pleasant application that works very well and is easy to use. I have spent a lot of time trying it; I have imported a project, created my own one and created flashcards, and I can say that it is a very intuitive application. Furthermore, the application interface can be customized. I think I may use it to organize my learning sessions.

I am pretty much familiar with the Java language since I have already made professional software using this language, such as an Android application. Obviously, working on this project would be a great opportunity to improve the way I design an application in an Object Oriented language.

MILESTONE 2: PROJECT DESIGN

Persona, actors & stakeholders

Our persona is Marc is a 22 year old high school student taking history. He is not very well organized and has a busy schedule. He sometimes fails to prepare for exams because he lacks the skills needed to cover all the important concepts in a timely and efficient manner. Furthermore, his history class is especially challenging because it contains many of dates and historical figures to memorize. Marc has a basic knowledge when it comes to computers. He uses them as a tool to go on Internet, create Excel sheets or Powerpoint presentations. He is looking for a user-friendly application to help him optimize his study sessions and achieve better results.

We have 2 extras personas. Joseph is a history teacher who's been wondering how he can improve the performance of his students on tests. Being that his subject is about memorization he is hoping to find a method that is good for association. Joseph is also very eco friendly and would like to be able to give his students practice quizzes that don't require any paper. Maria is a lawyer who has to get ready for a trial soon. The subject at which the law has recently changed. She needs to find a way to memorize all the changes quickly and efficiently before she is ready. She also likes to memorize some queues to which she has to responds and would like a method that would help her with that too.

The primary actor is the user of the application. Their goal using this application is to study efficiently and as fast as possible in order to learn and pass exams. We want to focus on this actor because he is the one with the greatest influence on system.
The secondary actor is the system under design. Their goal is to use logic, such as the Leitnerian algorithm, to allow the user to study more efficiently.

There are currently no stakeholders for this project. This being an open-source application developed by a single high-school student, nobody can be identified as an actual stakeholder at this point in time.

Informal use cases

Create notes

- ✓ *Precondition*

  A project has been created.
- ✓ *Main success scenario*
  1. The user **creates** a note under the Notes tab
  2. The system opens the new note window
  3. The user types the note name
  4. The system opens the note window
  5. The user types the content of the note
  6. The user selects save the note
  7. The system saves the note
- ✓ *Alternative scenarios*

  3a. The user enters a note name that already exists
  - a. The user enters the name
  - b. The system asks the user if he want to replace the ancient note by a blank one
  - c. The user chooses no
    - d. The system closes the new note windows

d. The user chooses yes

d. The system **erases** the ancient note and **replaces** it with a blank one.

5a. The user adds a picture to the note

a. The system shows the files manager

b. The users chooses the pictures he/she wants to add to the note

5b. The user adds a style to part of the note

6a. The user **deletes** the note

Create flashcard

- ✓ *Preconditions*

  At least one project has already been created
- ✓ *Main success scenario*
  1. The user selects create a flashcard
  2. The system opens the new flashcard window
  3. The user enters the question and the answer, selects add flashcard
  4. The system **adds** the flashcard to the deck as a Rank A flashcard
- ✓ *Alternative scenarios*

  3a. The user creates a picture question flashcard
    - a. The user selects add picture
    - b. The system opens windows explorer window
    - c. The user selects which picture to use
    - d. The system **adds** the picture to the flashcard
    - e. The user enters the answer, selects add flashcard
    - f. The system **adds** the flashcard to the deck as a Rank A flashcard

  3b. The user neglects to specify either a question or an answer
    - a. The system moves the cursor to the missing field
    - b. The user enters the required information, selects add flashcard
    - c. The system **adds** the flashcard to the deck as a Rank A flashcard

Remove Flashcard

- ✓ *Preconditions*

  At least one project has already been created, at least one flashcard created
- ✓ *Main success scenario*
  1. The user selects the Card Manager tab
  2. The system updates window to show all the flashcards in the deck
  3. The user scrolls down and finds the card which needs to be deleted, selects garbage can icon related to that card
  4. The system creates a confirmation window
  5. The user confirms the delete
  6. The system **removes** the flashcard from the deck

Answer question

- ✓ *Precondition*

  At least one created project, activate the project you want to work on (if there are more than 1 project), have at least one flashcard in the project
- ✓ *Main success scenario*
  1. The system shows the flashcard to the user.

2. The user can **answers** the question the answer box of the flashcard.
3. The system **records** the answer.
4. The user asks the system to show the answer of the flashcard.
5. The system queries the good answer of this flashcard.
6. The system shows it to the user.
7. The user **grades** himself (cf Grading use case).

✓ *Alternative scenario*

2.a The user **skips** the current flashcard.
b. The system picks up randomly a new flashcard.

Study

✓ *Preconditions*

At least one project has been created before.

✓ *Main success scenario*

1. The user **activates** the project he wants to study.
2. The user **starts** a new study session.
3. The user **answers** to one question. (cf answer use case)
4. The user **checks** validity of this answer of the flashcard (cf 3rd use case)
   (loop 3-4 until the end of the study session)

✓ *Alternative scenarios*

*a The user **resets** the deck.
1. The system **moves** all flashcards to rank A.

3a. The user **skips** the flashcard.
1. The system **picks up** a new flashcard.

Grading

✓ *Preconditions*

One flashcard answered by user.

✓ *Main success scenario*

1. The system supplies the user with 3 choices: got it, sort of and not.
2. The user validates his answer for this flashcard.
3. The system **upgrades** the flashcard to next rank (A to E).
4. The system rewards points to the user;
5. The system shows a real-time graphic with right/wrong answers.
6. The system **picks up** a new flashcard.

✓ *Alternative scenarios*

2a. The user doesn’t validate his answer for this flashcard.
a. The system **records** the wrong answer;
b. The system **downgrades** the flashcard to rank A.
c. The system shows a real-time graphic with right/wrong answers.
e. The system **picks up** a new flashcard.

2b. The user selects “sort of” for this flashcard.
a. The system **records** the user sort of knew the answer of the flashcard.
b. The system **downgrades** the flashcard to the previous rank (A to D)
c. The system shows a real-time graphic with right/wrong answers.
d. The system **picks up** a new flashcard.

Print flashcards

- ✓ *Preconditions*

  There is at least one project with at least one existing flashcard.
- ✓ *Main success scenario*

1. The user asks the system to **print** flashcards.
2. The user selects the project to print**.**
3. The system shows a preview of all flashcards to print.
4. The system launches the print manager.
5. The user can changes print options.
6. The system **prints** the flashcards.

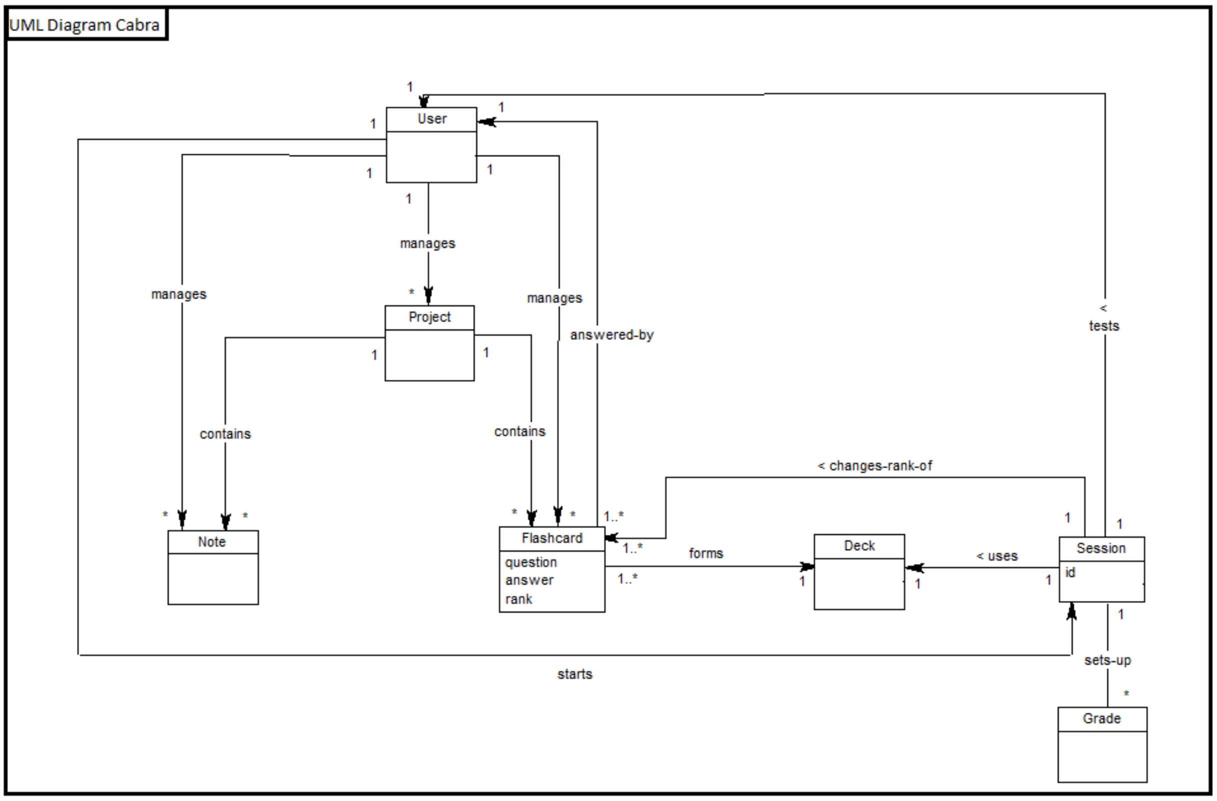
Modify flashcard

- ✓ *Preconditions*

  At least one flashcard created
- ✓ *Main success scenario*

1. The user selects the flashcard manager tab.
2. The user selects the flashcard icon he wants.
3. The system allows the editing the question and/or answer of the flashcard.
4. The user **edits** the question and/or answer of the flashcard.
5. The user saves the modifications of the flashcard.
6. The system records these modifications and change the flashcard content.

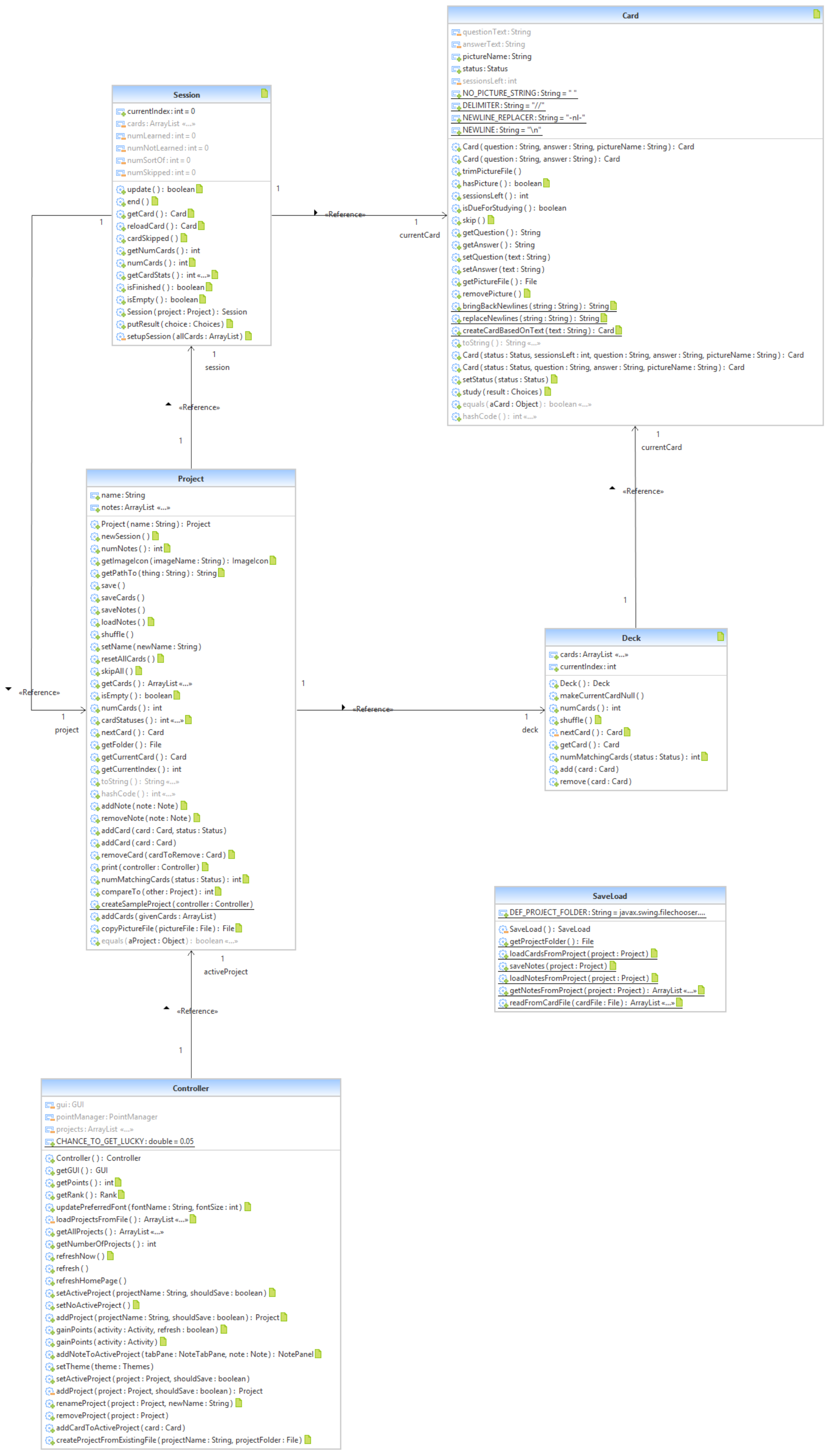
UML Diagram



Our UML diagram is very simple in design. It contains seven conceptual classes. The central class which is in control of the entire system is the User class. This user has the ability to create and manage a Project or several projects at a time. Once created, projects allow the instantiation of all useful aspects of the application such as Notes and Flashcards. In the event that the user wishes to test his knowledge of a subject, the user may start a study Session. This study session will create a Deck with a subset or all the flashcards in the project, and pick from these in order to test the user. The session keeps track of which flashcards the user is having trouble with in order to change their rank and ask those questions more frequently. Finally, the session also records the Grades associated with the flashcards in order to supply the user with information at the end of the study session.

MILESTONE 3: CODE SMELLS

This is the UML diagram of classes of interest (we focus on our domain: study). A pdf of this diagram is available in root folder.


Card
questionText : String
answerText : String
pictureName : String
status : Status
sessionsLeft : int
NO_PICTURE_STRING : String = " "
DELIMITER : String = "//"
NEWLINE_REPLACER : String = "-nl-"
NEWLINE : String = "\n"
Card (question : String, answer : String, pictureName : String) : Card
Card (question : String, answer : String) : Card
trimPictureFile ( )
hasPicture ( ) : boolean
sessionsLeft ( ) : int
isDueForStudying ( ) : boolean
skip ( )
getQuestion ( ) : String
getAnswer ( ) : String
setQuestion (text : String)
setAnswer (text : String)
getPictureFile ( ) : File
removePicture ( )
bringBackNewlines (string : String) : String
replaceNewlines (string : String) : String
createCardBasedOnText (text : String) : Card
toString ( ) : String «...»
Card (status : Status, sessionsLeft : int, question : String, answer : String, pictureName : String) : Card
Card (status : Status, question : String, answer : String, pictureName : String) : Card
setStatus (status : Status)
study (result : Choices)
equals (aCard : Object) : boolean «...»
hashCode ( ) : int «...»
Session
currentIndex : int = 0
cards : ArrayList «...»
numLearned : int = 0
numNotLearned : int = 0
numSortOf : int = 0
numSkipped : int = 0
update ( ) : boolean
end ( )
getCard ( ) : Card
reloadCard ( ) : Card
cardSkipped ( )
getNumCards ( ) : int
numCards ( ) : int
getCardStats ( ) : int «...»
isFinished ( ) : boolean
isEmpty ( ) : boolean
Session (project : Project) : Session
putResult (choice : Choices)
setupSession (allCards : ArrayList)
1
«Reference»
1
currentCard
1
1
session
«Reference»
1
Project
name : String
notes : ArrayList «...»
Project (name : String) : Project
newSession ( )
numNotes ( ) : int
getImageIcon (imageName : String) : ImageIcon
getPathTo (thing : String) : String
save ( )
saveCards ( )
saveNotes ( )
loadNotes ( )
shuffle ( )
setName (newName : String)
resetAllCards ( )
skipAll ( )
getCards ( ) : ArrayList «...»
isEmpty ( ) : boolean
numCards ( ) : int
cardStatuses ( ) : int «...»
nextCard ( ) : Card
getFolder ( ) : File
getCurrentCard ( ) : Card
getCurrentIndex ( ) : int
toString ( ) : String «...»
hashCode ( ) : int «...»
addNote (note : Note)
removeNote (note : Note)
addCard (card : Card, status : Status)
addCard (card : Card)
removeCard (cardToRemove : Card)
print (controller : Controller)
numMatchingCards (status : Status) : int
compareTo (other : Project) : int
createSampleProject (controller : Controller)
addCards (givenCards : ArrayList)
copyPictureFile (pictureFile : File) : File
equals (aProject : Object) : boolean «...»
«Reference»
1
project
1
«Reference»
1
deck
1
currentCard
«Reference»
1
Deck
cards : ArrayList «...»
currentIndex : int
Deck ( ) : Deck
makeCurrentCardNull ( )
numCards ( ) : int
shuffle ( )
nextCard ( ) : Card
getCard ( ) : Card
numMatchingCards (status : Status) : int
add (card : Card)
remove (card : Card)
SaveLoad
DEF_PROJECT_FOLDER : String = javax.swing.filechooser....
SaveLoad ( ) : SaveLoad
getProjectFolder ( ) : File
loadCardsFromProject (project : Project)
saveNotes (project : Project)
loadNotesFromProject (project : Project)
getNotesFromProject (project : Project) : ArrayList «...»
readFromCardFile (cardFile : File) : ArrayList «...»
1
activeProject
«Reference»
1
Controller
gui : GUI
pointManager : PointManager
projects : ArrayList «...»
CHANCE_TO_GET_LUCKY : double = 0.05
Controller ( ) : Controller
getGUI ( ) : GUI
getPoints ( ) : int
getRank ( ) : Rank
updatePreferredFont (fontName : String, fontSize : int)
loadProjectsFromFile ( ) : ArrayList «...»
getAllProjects ( ) : ArrayList «...»
getNumberOfProjects ( ) : int
refreshNow ( )
refresh ( )
refreshHomePage ( )
setActiveProject (projectName : String, shouldSave : boolean)
setNoActiveProject ( )
addProject (projectName : String, shouldSave : boolean) : Project
gainPoints (activity : Activity, refresh : boolean)
gainPoints (activity : Activity)
addNoteToActiveProject (tabPane : NoteTabPane, note : Note) : NotePanel
setTheme (theme : Themes)
setActiveProject (project : Project, shouldSave : boolean)
addProject (project : Project, shouldSave : boolean) : Project
renameProject (project : Project, newName : String)
removeProject (project : Project)
addCardToActiveProject (card : Card)
createProjectFromExistingFile (projectName : String, projectFolder : File)

The UML diagram shows us that the central class of the application's architecture is the Project class. This class contains methods to add and remove Notes as well as Flashcards to their respective collections. The Flashcard object, unlike the Notes object, contains an intermediary class (Deck) between itself and Project. Deck is a simple class which is instantiated immediately and stays instantiated as long as the project exists. A Deck acts in a similar manner as a collection object, meaning it supplies the methods related to adding and removing flashcards as well as shuffling and retrieving the next card during a study session. A Session, which is another class in the UML diagram, acts as a quiz in order to test the user on their knowledge of the flashcard material. During a study session, the user answers questions and specifies whether he got it right, wrong or sort of right. The Session class also contains the ability to skip the current flashcard. As we can see through the architectural diagram, all information regarding grades for a certain session is saved inside the Session object itself.

Next up, we have added the Controller and SaveLoad classes to our architectural focus. The Controller class acts as an intermediary between the view, which includes all the aspects related to the interface the user interact with, and the model, which encapsulates all the classes used to represent the data and concepts being manipulated. The vast majority of its methods contain calls to existing methods stored in either the Project class, or other classes related to GUI which we have scoped out of focus. The SaveLoad class supplies the application with the functions related to saving all progress made in regards to creating a project as well as the methods which load these previously saved states.

After comparing our diagram of conceptual classes to the UML diagram of actual relevant architecture, we can see that there is really not much of a difference between the two. The only major difference is that our conceptual architecture consisted of a Grade class. In our design, the Grade class would be instantiated by the Session class every time a new session was started in order to record and keep track of all the users' progress through a session as he or she is answering flashcards. As mentioned previously, the current system is designed to keep all of this information within the Session object. In our opinion, this information (information regarding how many flashcards were answered right, how many were answered wrong, how many were skipped, etc.) should be stored in a different class. We believe that the Session object should only focus on functionality associated with cycling through and retrieving flashcards. This would increase the cohesion of the Session object. We think that the current architecture design in respect to keeping track of grades can be greatly improved upon.

The second major difference between our conceptual diagram and the actual architecture is how the Deck object is utilized. In our conceptual architecture, the Deck class was directly used by the Session class. Its instantiation was for the sole reason of being used in conjunction with Session. In the actual design, the deck class is used as a collection of flashcards. Therefore, existence of a single flashcard object implies the existence of the Deck object. We think this is another aspect of the architecture which can be improved. A Deck of flashcards, when comparing it to its real-world counterpart, should contain persistent data from the point in time before a study session to the point in time when that study session is complete. We believe that the existence of another class, such as a Deck which is used with one specific paired Session would improve the application's design and make for a more coherent structure.

Reverse engineering tool

In order to help us have a good perspective of the Cabra architecture, we wanted to use a reverse engineering tool. First, we tried to find a tool for NetBeans but the market of NetBeans is very small compared to Eclipse market. So, finally, we used an Eclipse tool : UML Lab. It's a great util developed by Yatta Solutions which has lots of features. First of all, you can create your model and class diagrams with it, selecting the classes you want to have in the diagram. It shows you only these classes and links between them, if any. In the diagram itself, you are allowed (by a simple button click) to show parents, childs or any composition/aggregation links with other classes that the class has. Changing code in the project changes directly the diagram without regenerate it. Finally, one of the powerful tool that UML Lab has is refactoring tool. Indeed, you can refactor some code in the diagram itself and it changes directly the related code. We used this tool to do the class diagram. It allowed us to see first refactorings just looking for relation between the classes, like Inappropriate Intimacy between Project/Session for example.

Relations between 2 classes of interest

Session.java

```
public class Session extends Object{
  private Project project; //the project we're studying for


    public Session(Project project){
      this.project = project;
      project.setSession(this);
      cards = new ArrayList<Card>();
      setupSession(project.getCards());
  }

  public void end()
         {
      project.setSession(null);
  }
}
```

Project.java

```
public class Project implements Comparable<Project>{
  private Session session = null; //if a study session is going on, it's here



  public void newSession(){  // creates a new session  for this project
      do{
         setSession(new Session(this));
      }
      while(getSession().isEmpty());
  }
   public void addCard(Card card,Status status){
      card.setStatus(status);
      deck.add(card);
          //if the card has a picture, move the picture over here
      if(card.hasPicture()){
        File copiedFile = copyPictureFile(card.getPictureFile());
        card.setPictureName(copiedFile.getAbsolutePath());
        //and now trim the card's picture file... we won't need the full path any more
        card.trimPictureFile();
      }
      saveCards();
       if(session != null){  //since there's a new card, notify the session
        session.update();
      }
    }
  }
```

Code smells & Refactoring

There are several types of code smells in Cabra project. Some of them are easy to detect and fix. Others are more complex.

First of all, there are some dead code in Card, Note, Project and Session classes. Both of them contains commented code, even whole functions (such as removeCard or decreaseCurrentIndex in Session class). This code smell can be fix simply in remove this code.

Another code smell which is easy to fix is Long Method. We can see examples of it in SaveLoad and Controller classes. In SaveLoad, loadCardsFromProject and getNotesFromProject are refactorable. In both of these functions, the developer wrote comments to explain the code. In order to improve the clarity of the code and to remove this code smells, we can use the Extract Method each time the developer put a comment. For example, in getNotesFromProject we can extract 4 methods : getFiles, readNote, manageIncompatibleKindOfNote, addNoteToNotesList. In the Controller class, almost all methods are too long but especially the constructor that is so long that the code is very unclear. The method to fix this is Extract Method too.

Other code smells are very interesting because they imply architecture design, coupling and cohesion. There is Inappropriate Intimacy between Project and Session. Indeed, the Project class has an instance of Session (set in SetSession which is called in the constructor of Session) and the session has an instance of the project (parameter of the constructor). However, the only reason of that is because the session wants the list of cards of the project to get the cards studied for this session. To reduce coupling and increase cohesion, instead of set the project in session’s constructor and get the cards with getter in project, we can pass directly the deck which contains manage the list of the cards to the session constructor. When this is the end of the session, the StudyPanel calls end() function of current session. This function just sets the session instance to null in project. Instead, StudyPanel can call a function end in the Project class that set directly instance of session to null. This way, the project has a session instance but the session doesn’t deal with project anymore.

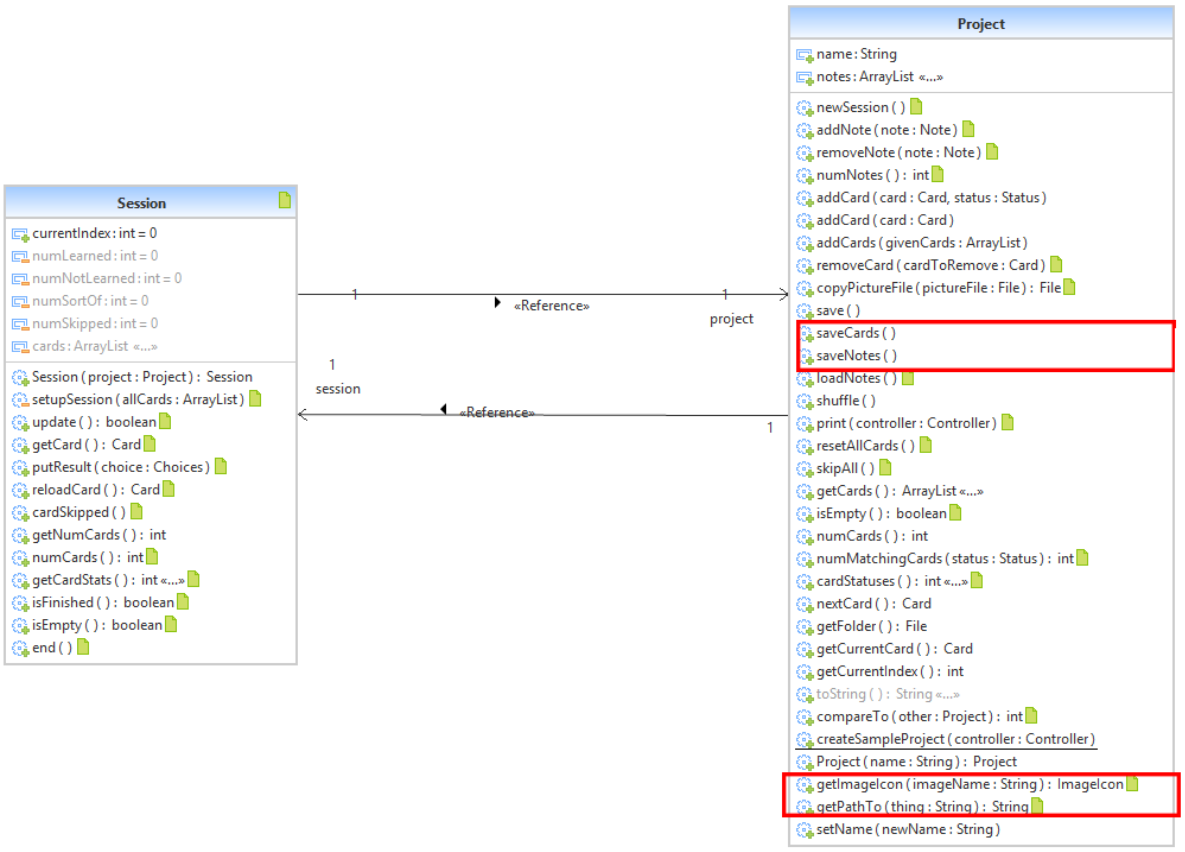
Some classes that we focus on have the Divergent Change code smells. Indeed, they have responsibilities that other classes should have. In order to increase cohesion, that each class have only one responsibility (that is the main concept of Object Oriented Programming).

It concerns Card, Project and Controller classes. In Card class, they are functions that deal with picture: trimPictureFile, getPictureName, getPictureFile, setPictureName, removePicture. However, the Card has to deal only with question and answer. It can have a picture but all functions that get the picture from file and so on should not be in Card class but an util class like PictureManager, which will be called by Card. So, we can use the Extract Class to do the class PictureManager and Move Methods dealing with Picture into this class. ReplaceNewLines, BringBackNewLines and constant attributes of class Card shouldn't be there because it deals with string stuff. The code smells is also divergent change. Extract a class StringUtils for example and move these methods into this class will increase cohesion. All of these changes are fixing another code smell : large class. Indeed, Card class has too much responsibilities and is too large. Combining extract class and move methods refactorings should increase cohesion and increase reusability and clarity of the code.

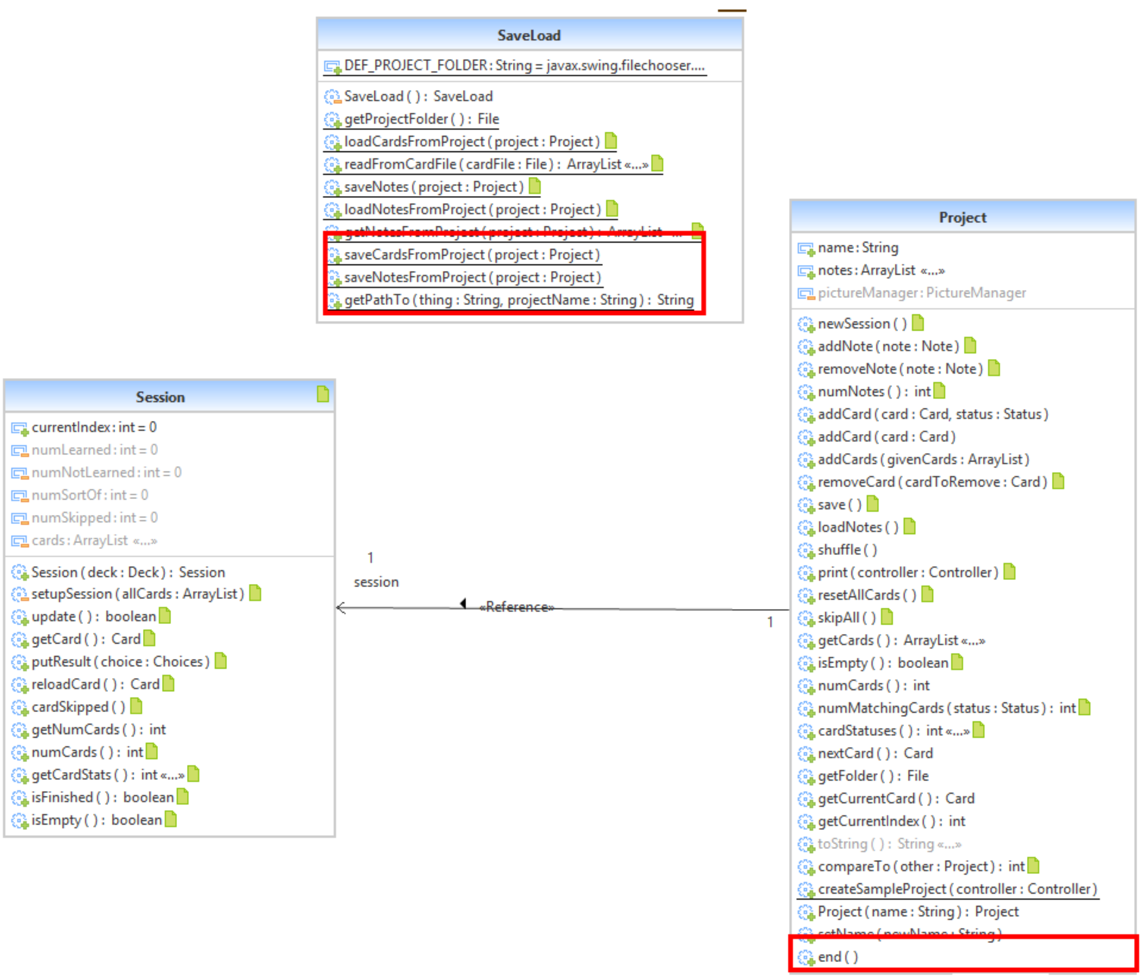
Divergent change exists in Project class too. The main purpose of this class is to manage cards of one project, containing in a deck and starts/ends sessions of study for this project. However, it contains functions dealing with picture (copyPictureFile), icon (getImageIcon), save/load (getPathTo, saveCards, saveNotes). copyPictureFile doesn't need Extract class refactoring because it can be moved in PictureManager class that we extract from Card. Likewise, getImageIcon can be moved in ImageManager and finally save/load functions should be moved in SaveLoad class, which is a class precisely managing the save and load of a project. All these changes are logical and increase cohesion of the Project class. All calls of these functions should be changed. Now, static class SaveLoad is called instead of Project and we pass in parameter to those functions the name of the project or the project itself when needed. With this architecture, it has only one responsibility and combining with previous refactorings fixing Inappropriate Intimacy and Dead Code, it fixes another code smell in this class : Large Class.

Controller is the class linking user interactions with the "real" study code. It's interesting because this is the class which call lots of functions of the classes we've seen previously. Maybe that's why there are some divergent changes in this class. Indeed, because it's the link between GUI and study algorithm, the developer did a Large Class and put in one place all code calling the algorithm according to events. However, in some cases such as addProject, removeProject and loadProjectsFromFile, it seems to be in the wrong place because of their purpose. For example, addProject and removeProject create and remove a directory for the project itself. However, it should be logical that it'd be in SaveLoad class (that deals with all events about saving/loading a project, find it in the files explorer, and so on). So, first we should move these methods into SaveLoad class. Then, we should rename the class SaveLoad because now it create and remove a class too. For example, SaveLoad can be renamed as……… . Finally, we can move loadProjectsFromFile that searches itself all projects created by user in the old class SaveLoad renamed ……. These refactorings increase cohesion of the Controller class but reduce its size (not a Large Class anymore). It removes too the feature envy with SaveLoad class because instead of calling Controller functions to call SaveLoad functions, all the related code is directly moved into the SaveLoad class.

There is an example of refactorings that we can do with just two classes : Session and Project. The class diagram below represents the classes before the refactorings.


Session
currentIndex : int = 0
numLearned : int = 0
numNotLearned : int = 0
numSortOf : int = 0
numSkipped : int = 0
cards : ArrayList «...»
Session (project : Project) : Session
setupSession (allCards : ArrayList)
update ( ) : boolean
getCard ( ) : Card
putResult (choice : Choices)
reloadCard ( ) : Card
cardSkipped ( )
getNumCards ( ) : int
numCards ( ) : int
getCardStats ( ) : int «...»
isFinished ( ) : boolean
isEmpty ( ) : boolean
end ( )
1
«Reference»
1
project
1
session
«Reference»
1
Project
name : String
notes : ArrayList «...»
newSession ( )
addNote (note : Note)
removeNote (note : Note)
numNotes ( ) : int
addCard (card : Card, status : Status)
addCard (card : Card)
addCards (givenCards : ArrayList)
removeCard (cardToRemove : Card)
copyPictureFile (pictureFile : File) : File
save ( )
saveCards ( )
saveNotes ( )
loadNotes ( )
shuffle ( )
print (controller : Controller)
resetAllCards ( )
skipAll ( )
getCards ( ) : ArrayList «...»
isEmpty ( ) : boolean
numCards ( ) : int
numMatchingCards (status : Status) : int
cardStatuses ( ) : int «...»
nextCard ( ) : Card
getFolder ( ) : File
getCurrentCard ( ) : Card
getCurrentIndex ( ) : int
toString ( ) : String «...»
compareTo (other : Project) : int
createSampleProject (controller : Controller)
Project (name : String) : Project
getImageIcon (imageName : String) : ImageIcon
getPathTo (thing : String) : String
setName (newName : String)

We can see the Inappropriate Intimacy between those two classes and looking at the Project class, we can see that there are Large Class and Divergent change (methods' name) code smells too. If we do the refactorings suggested before, the diagram class becomes as below:


SaveLoad
DEF_PROJECT_FOLDER : String = javax.swing.filechooser....
SaveLoad ( ) : SaveLoad
getProjectFolder ( ) : File
loadCardsFromProject (project : Project)
readFromCardFile (cardFile : File) : ArrayList «...»
saveNotes (project : Project)
loadNotesFromProject (project : Project)
~~getNotesFromProject (project : Project) : ArrayList «...»~~
saveCardsFromProject (project : Project)
saveNotesFromProject (project : Project)
getPathTo (thing : String, projectName : String) : String
Project
name : String
notes : ArrayList «...»
pictureManager : PictureManager
newSession ( )
addNote (note : Note)
removeNote (note : Note)
numNotes ( ) : int
addCard (card : Card, status : Status)
addCard (card : Card)
addCards (givenCards : ArrayList)
removeCard (cardToRemove : Card)
save ( )
loadNotes ( )
shuffle ( )
print (controller : Controller)
resetAllCards ( )
skipAll ( )
getCards ( ) : ArrayList «...»
isEmpty ( ) : boolean
numCards ( ) : int
numMatchingCards (status : Status) : int
cardStatuses ( ) : int «...»
nextCard ( ) : Card
getFolder ( ) : File
getCurrentCard ( ) : Card
getCurrentIndex ( ) : int
toString ( ) : String «...»
compareTo (other : Project) : int
createSampleProject (controller : Controller)
Project (name : String) : Project
~~setName (newName : String)~~
end ( )
Session
currentIndex : int = 0
numLearned : int = 0
numNotLearned : int = 0
numSortOf : int = 0
numSkipped : int = 0
cards : ArrayList «...»
Session (deck : Deck) : Session
setupSession (allCards : ArrayList)
update ( ) : boolean
getCard ( ) : Card
putResult (choice : Choices)
reloadCard ( ) : Card
cardSkipped ( )
getNumCards ( ) : int
numCards ( ) : int
getCardStats ( ) : int «...»
isFinished ( ) : boolean
isEmpty ( ) : boolean
1
session
«Reference»
1

Code of one suggested refactoring

Suggested Refactoring: Using deck in the session.

The Deck class extends objects and is essentially an array list of Cards. Currently the deck is used by the project to maintain all of it's cards. When you time comes to study the project makes a session which then gets all the cards back from the project and stores them in an independent array list of Cards. Rather than doing this we would consider passing the whole deck during the session creation and use a local deck to manipulate the needed Cards of the project deck during the study session. This will give a more logical structure to the current code closer to our conceptual diagram and eliminate some duplicate code.

Code example:

Deck.java
```
public class Deck extends Object{
  private ArrayList<Card> cards; //flash cards of owner project
  private Card currentCard = null;
  private int currentIndex;


  public Card getCard(){
     if(numCards() == 0)
        return null; //no cards here
     currentCard = nextCard()
     return currentCard;
  }
  public int getCurrentIndex(){
     return currentIndex;
  }
  public int numCards(){
     return cards.size();
  }
...
```
Session.java
```
public class Session extends Object{
  ...
  private int currentIndex = 0; //how many cards we have studied
  private ArrayList<Card> cards; //cards we'll study
  private Card currentCard; //current card we're studying
  ...
  public Session(Project project){
     ...
     cards = new ArrayList<Card>();
     setupSession(project.getCards());
  }
  public Card getCard(){
     if(currentIndex >= numCards()){
        return null;
     }
     currentCard = cards.get(currentIndex);
     currentIndex++;
     return currentCard;
  }
  public int getCurrentIndex(){
     return currentIndex;
  }
  public int getNumCards(){
     return cards.size();   }
```

MILESTONE 4: REFACTORING

Identification of one substantial pattern (individual)

Mediator Pattern: Controller.java

Nicolas TRICHEUX #7129491

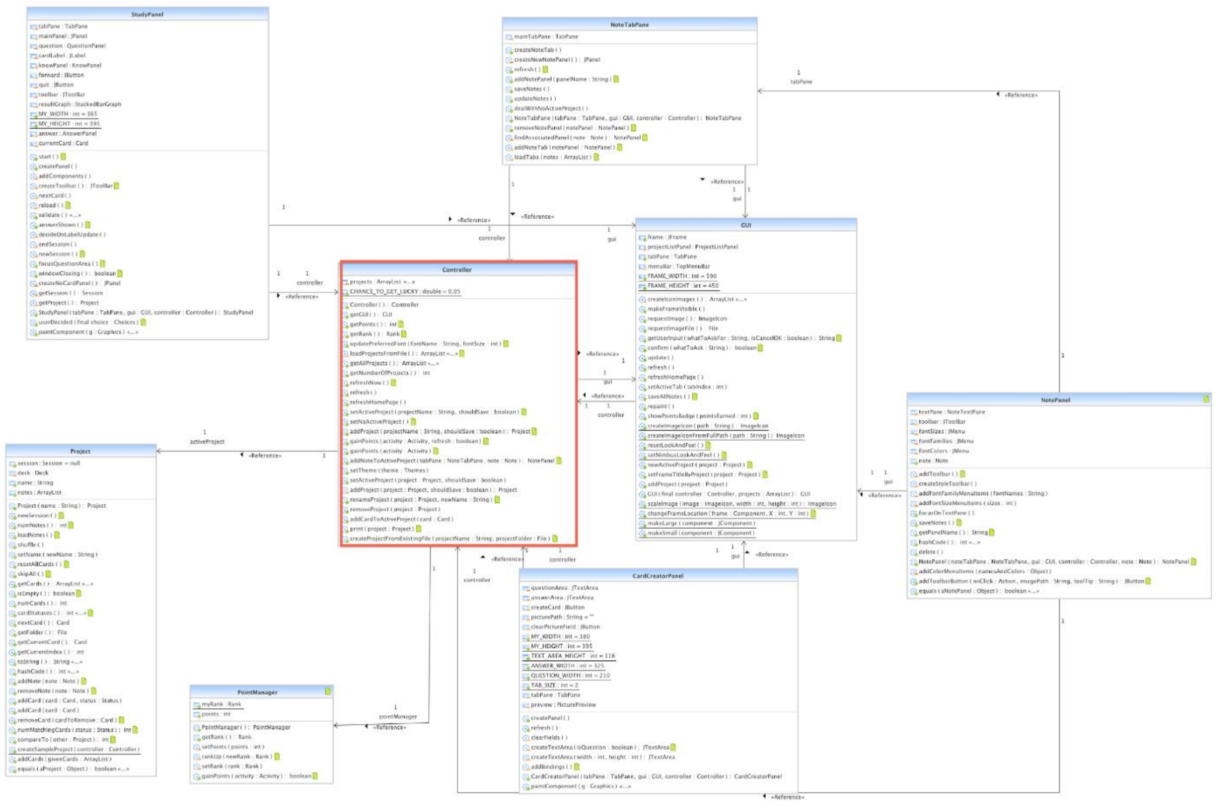
Reference : http://www.blackwasp.co.uk/Mediator.aspx

The Controller knows about the GUI, all the projects, the PointManager class, the active project and an array list with all the different projects.

The controller is referenced in 16 objects. It is linked to the GUI and handles the distribution of work based on the decisions of the user. The controller needs to know about all the projects because it will set the active project among one in its array list. It needs to know about the project, because inputs from the user modify the project. Indeed, the cards and the deck are accessible through the project. After a work has been done, it notifies the controller to refresh the user interface.

Unlike the Mediator pattern, there is only one concrete mediator in this architecture. Therefore there is no mediator interface. This way of implementing this pattern is not as efficient as the real mediator pattern in terms of coupling and cohesion, because there is only one concrete mediator. In fact, the mediator is coupled to every single object that needs to talk to it. It also has a low cohesion, because it is responsible for many different tasks. We actually pointed out, in the last Milestone that there is a lot of divergent change in this class. Maybe the developer should have created different concrete mediators that would have limited the coupling and the cohesion.
However it was definitely wise to use the mediator pattern, even if this is a simplified version of it, since the logic is separated from the GUI and every single user's request is handled in the same way.

Here is the UML class diagram showing how the main classes we have studied are related to the controller.



```
public final class Controller extends Object {
        //communicates with the GUI and object classes to get stuff done
        private GUI gui;
        private PointManager pointManager;
        private ArrayList<Project> projects;
        private Project activeProject;

 public void gainPoints(PointEnums.Activity activity, boolean refresh){
        pointManager.gainPoints(activity);
        //show how many points were earned
        gui.showPointsBadge(activity.getPoints());

        if(refresh)                gui.refresh();
        }

 public void updatePreferredFont(String fontName, int fontSize){
        FontManager.updatePreferredFont(fontName, fontSize);
        //validate frame so the changes take effect
        if(gui != null)
        gui.update();
        }

 public NotePanel addNoteToActiveProject(NoteTabPane tabPane,Note note){
        activeProject.addNote(note);
        //save while we're at it
        SaveLoad.saveNotes(activeProject); //no need to save cards too
```

```
          return new NotePanel(tabPane,gui,this,note);
          }
public void createProjectFromExistingFile(String projectName,File projectFolder){
          Project project = new Project(projectName);
          //add cards
          SaveLoad.loadCardsFromProject(project);
          //add notes
          SaveLoad.loadNotesFromProject(project);
          addProject(project,true);
          }
}
```

Example of the Project notifying the Controller after an input by the user on one of the user interfaces:

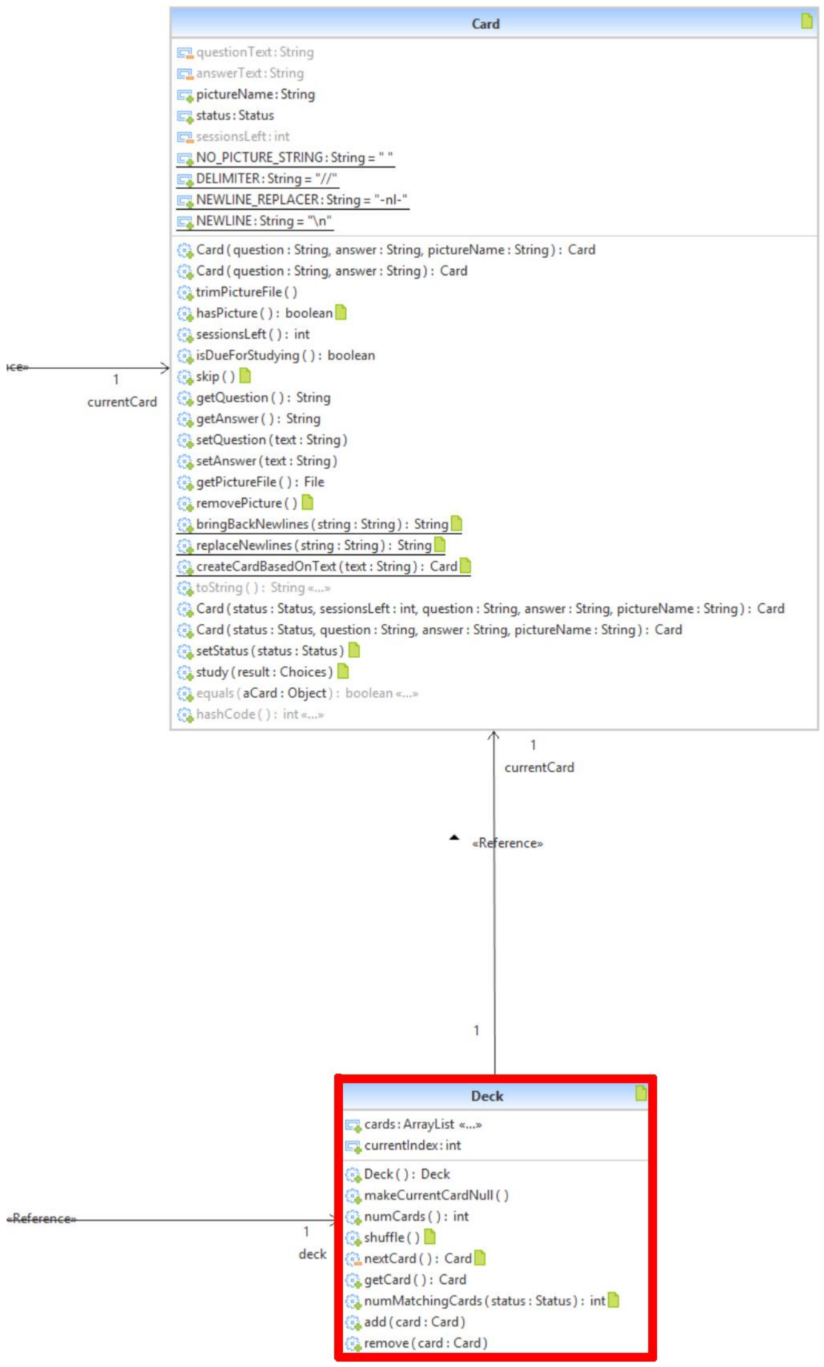
```
 public static void createSampleProject(Controller controller){
          Project project = controller.addProject("Sample", true);
      ArrayList<Card> cards = new ArrayList<Card>();
          cards.add(new Card("What is the ultimate answer to life, the universe, and
everything?", "42"));
      project.addCards(cards);
          //return project;
      controller.refresh();
          }
```

Iterator Pattern: Deck.java

Magalie PERON #7129173

The Deck class holds the cards of the project, and iterate through them as needed. In this way, it behaves like the Iterator pattern, with methods to get the current and next card. However, it is not separated in a specific Iterator class which has only this role, but is mixed up with the class which actually holds, adds and removes cards from the deck. Therefore the internal structure is still exposed in the Deck class, which is should not in a proper Iterator.

Here is the UML class diagram showing how the Deck class is related to the Card class.

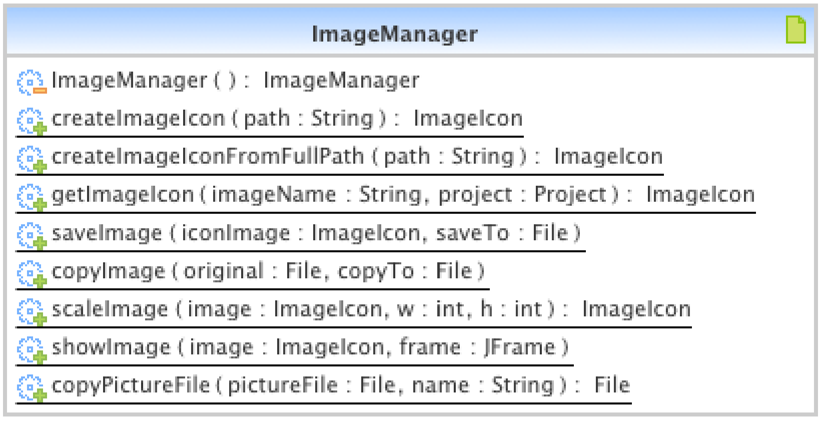

Card
questionText : String
answerText : String
pictureName : String
status : Status
sessionsLeft : int
NO_PICTURE_STRING : String = " "
DELIMITER : String = "//"
NEWLINE_REPLACER : String = "-nl-"
NEWLINE : String = "\n"
Card (question : String, answer : String, pictureName : String) : Card
Card (question : String, answer : String) : Card
trimPictureFile ( )
hasPicture ( ) : boolean
sessionsLeft ( ) : int
isDueForStudying ( ) : boolean
skip ( )
getQuestion ( ) : String
getAnswer ( ) : String
setQuestion (text : String)
setAnswer (text : String)
getPictureFile ( ) : File
removePicture ( )
bringBackNewlines (string : String) : String
replaceNewlines (string : String) : String
createCardBasedOnText (text : String) : Card
toString ( ) : String «...»
Card (status : Status, sessionsLeft : int, question : String, answer : String, pictureName : String) : Card
Card (status : Status, question : String, answer : String, pictureName : String) : Card
setStatus (status : Status)
study (result : Choices)
equals (aCard : Object) : boolean «...»
hashCode ( ) : int «...»
ice»
1
currentCard
1
currentCard
«Reference»
1
Deck
cards : ArrayList «...»
currentIndex : int
Deck ( ) : Deck
makeCurrentCardNull ( )
numCards ( ) : int
shuffle ( )
nextCard ( ) : Card
getCard ( ) : Card
numMatchingCards (status : Status) : int
add (card : Card)
remove (card : Card)
«Reference»
1
deck

```
public class Deck extends Object{


        private ArrayList<Card> cards; //flash cards of owner project
        private Card currentCard = null;
        private int currentIndex; //the index of the current card being viewed. Between 0 and length
of cards

        public Card getCurrentCard(){
                return currentCard;
        }

        public void add(Card card){
                cards.add(card);
        }

        public void remove(Card card){
                cards.remove(card);
        }

        private Card nextCard(){
                Card card = null;
                try{
                        card = cards.get(currentIndex);
                }
                catch(IndexOutOfBoundsException e){
                        //tried to access a bad location, so shuffle and try again
                        shuffle();
                        return nextCard(); //return a new card
                }
                currentIndex++;
                if(currentIndex >= cards.size()){
                        //we've run out of cards
                        shuffle(); //for next time
                }
                return card;
        }
}
```

Proxy Pattern: ImageManager.java

Guillaume FORTIN #6325300

In the ImageManager class you have class getting and image icon for the majority of the project and only when needed will it go get the full image to display it. This acts like the proxy pattern since it is only loading a truncated version of the file and when the time comes it will load the real file. This reduces wasted data when unecessary. As we can see from the code extract the ImageManager returns an ImageIcon java type for the most and then it would delegate to showImage for displaying the whole thing.


ImageManager
ImageManager ( ) : ImageManager
createImageIcon (path : String) : ImageIcon
createImageIconFromFullPath (path : String) : ImageIcon
getImageIcon (imageName : String, project : Project) : ImageIcon
saveImage (iconImage : ImageIcon, saveTo : File)
copyImage (original : File, copyTo : File)
scaleImage (image : ImageIcon, w : int, h : int) : ImageIcon
showImage (image : ImageIcon, frame : JFrame)
copyPictureFile (pictureFile : File, name : String) : File

```
/** Creates an image icon if you know its full path (C:\Users\...)
   *
   * @param path the absolute path to the image (.png, .jpg, .gif)
   * @return the created icon
   */
  public static ImageIcon createImageIconFromFullPath(String path){
     return new ImageIcon(path);
  }
/**     * Creates a dialog containing the image (at full size) and shows it to the user.
   * Best used to show an image full size when the thumbnail is clicked on.
   * @param image the image to show full size.
   * @param frame the parent frame of the dialog that will be shown
   */
  public static void showImage(ImageIcon image, JFrame frame){
          int width = image.getIconWidth();
     int height = image.getIconHeight();

     JPanel imagePanel = new JPanel(new GridLayout(1,1));
     imagePanel.add(new JLabel(image));
     imagePanel.setPreferredSize(new Dimension(width, height));

     JDialog dialog = Utils.putPanelInDialog(
           imagePanel,
           frame,
           "Full size image (" + width + "x" + height + ")", "pics.png", -1, -1 );
     dialog.setVisible(true);
  }
```

Composite Pattern

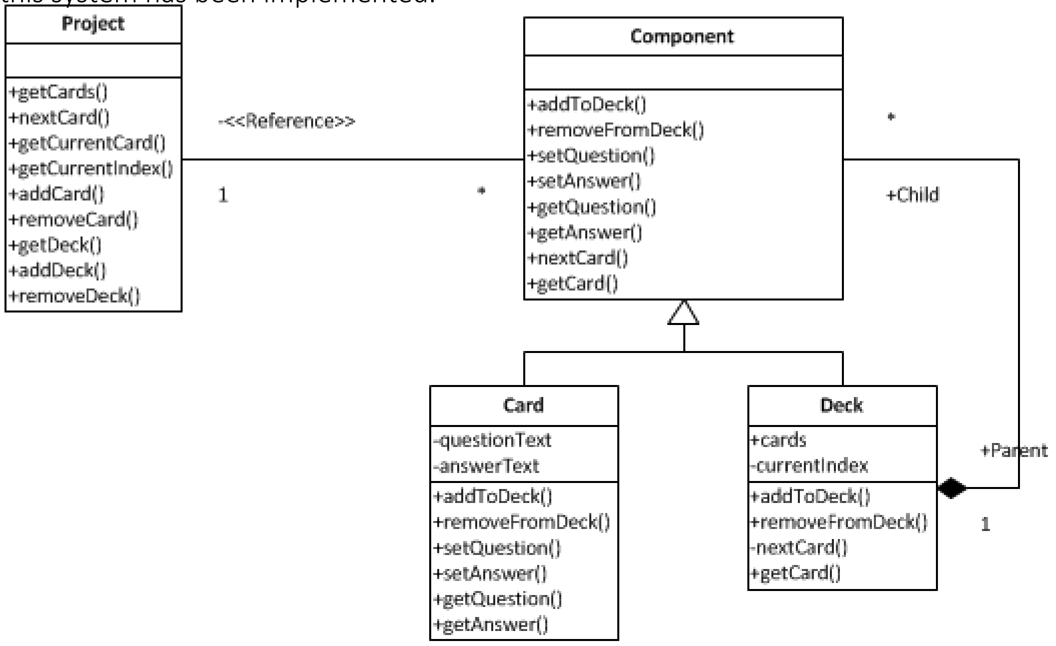
Derek RINGUETTE #1947834

References: http://www.oodesign.com/composite-pattern.html
http://java.dzone.com/articles/composite-design-pattern-java-0

After reviewing the structure of the Cabra application, I have come to the conclusion that significant design patterns are non-existent within the current application. Due to the lack of design patterns, I have decided to take a different approach for this section. I will describe how the implementation of a design pattern could help add a functionality which I think is missing from the Cabra application. I will do this by describing the new functionality. Then, I will explain how it could be realized through the application of the composite design pattern.

Let us analyze the following scenario: John is a student enrolled in a course which has three tests throughout the semester and one cumulative final exam at the end of the session. He has used the Cabra application as a study tool throughout the semester and has created three study projects, one for each test. Now, he wishes to study for his final and has no way of combining his 3 projects into one project he could study from. I propose that Cabra should allow the addition and removal of Decks to and from larger Decks. This would permit a Cabra user (i.e. John) to create large, robust decks of flashcards composed of previously created smaller decks. Let us look at how the Composite pattern would allow us to achieve this.

The composite pattern's main purpose is to allow functionalities to be shared among parent and child classes. In other words, provide the same methods to both a composite and its leaves. In this case, a Deck would be given the same functionalities as a flashcard currently has (i.e. adding and removing to and from a deck). The key concept of the Composite pattern is the implementation of a component interface. This interface provides the outline for functionalities you wish to share between parents and children as well as provide the interface for outside classes to interact with. I have added a very simplified conceptual diagram to depict the structure of the relevant classes after this system has been implemented.



I believe that the implementation of this structure could add a very useful feature to the Cabra application which, in my opinion, should have been present much earlier in this product's lifecycle.

Observer Pattern

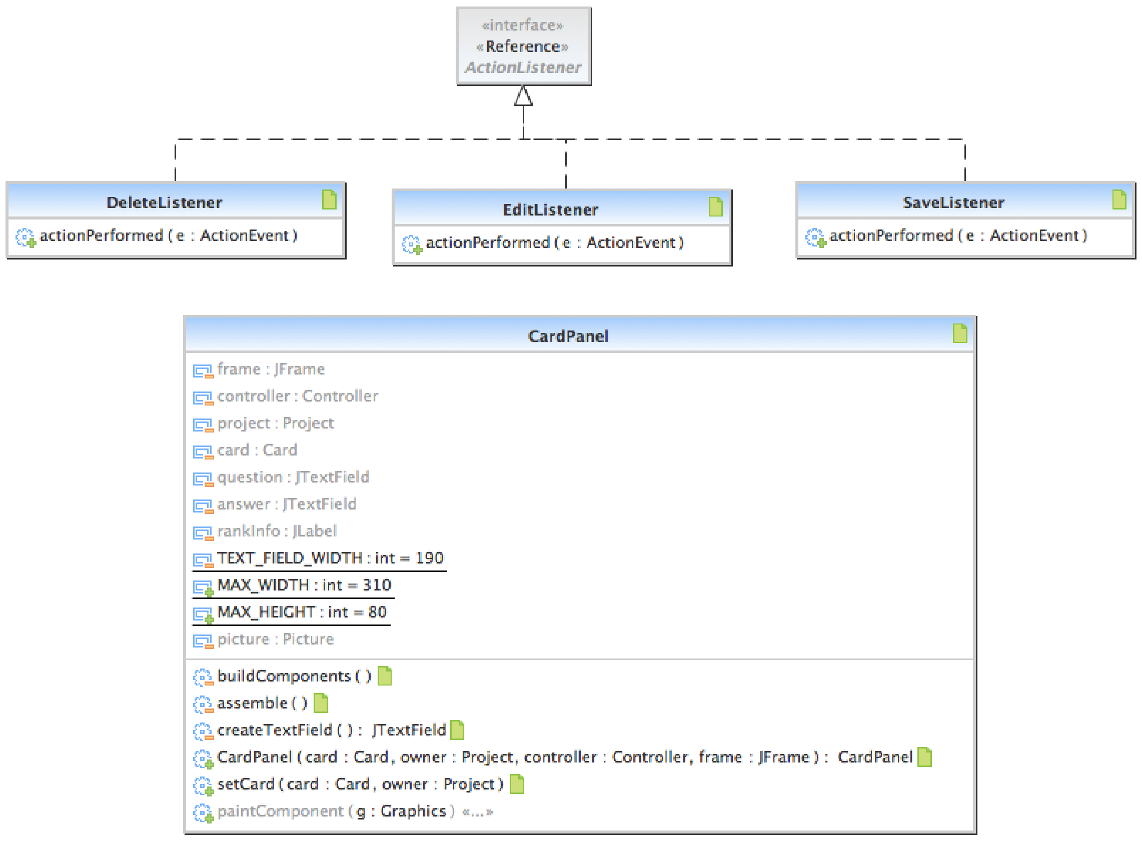
Laura BERNHARDT #7118112

Reference: http://sourcemaking.com/design_patterns/observer

In Cabra, the developer uses lots of Java components and features. For example, he uses the ActionListener of Java to implement the observer pattern for his GUI. The GUI is divided into panels : CardPanel, TabPanel, ProjectListPanel and so on. Each panel contains different kinds of listeners. I would like to talk about CardPanel. Into the card panel, two buttons are created: delete and edit button. Moreover, three custom listeners are implemented: DeleteListener, EditListener and SaveListener, which all inherit from ActionListener. Each button is link to a listener. So, when the button is clicked, the function actionPerformed (kind of onEvent function) is called on the good listener. The observer pattern is a good choice when you have a GUI like Cabra. Each panel is actually a view and they can have their own behavior, the only thing to do is override actionPerformed method. When user clicks on a button, the good method is "automatically" called if the link between the button and the listener was created before. Moreover, on this GUI, you can have more than one listener for the same GUI object. The Observer pattern allows this kind of situation.

I used UML Lab reverse engineering tool for this part. Indeed, when you drag and drop the panel classes, you can see all listeners that inherit from ActionListener in those classes.

Here we can see the Card Panel class. All listeners are nested classes of CardPanel.


«interface»
«Reference»
ActionListener
DeleteListener
actionPerformed (e : ActionEvent)
EditListener
actionPerformed (e : ActionEvent)
SaveListener
actionPerformed (e : ActionEvent)
CardPanel
frame : JFrame
controller : Controller
project : Project
card : Card
question : JTextField
answer : JTextField
rankInfo : JLabel
TEXT_FIELD_WIDTH : int = 190
MAX_WIDTH : int = 310
MAX_HEIGHT : int = 80
picture : Picture
buildComponents ( )
assemble ( )
createTextField ( ) : JTextField
CardPanel (card : Card, owner : Project, controller : Controller, frame : JFrame) : CardPanel
setCard (card : Card, owner : Project)
paintComponent (g : Graphics) «...»

```
public class CardPanel extends JPanel{

   public CardPanel(Card card,Project owner,Controller controller,JFrame frame){
      super(new BorderLayout());
           ....
      assemble();
           ....
  }

   private void assemble(){
           ....

      JButton delete = new JButton(GUI.createImageIcon("trash.png"));
         delete.setToolTipText("Delete this card");
         delete.addActionListener(new DeleteListener());
         toolbar.add(delete);
      JButton edit = new JButton(GUI.createImageIcon("pencil.png"));
         edit.setToolTipText("Edit this card");
         edit.addActionListener(new EditListener());
         toolbar.add(edit);
           .....
   }

   private class DeleteListener implements ActionListener{
      public void actionPerformed(ActionEvent e){
         //get confirmation
         if(InputManager.confirm("Are you sure you want to delete this card?", frame)){
            project.removeCard(card);
            controller.refresh();
         }
      }
   }

  private class EditListener implements ActionListener{
      public void actionPerformed(ActionEvent e){
         question.setEditable(true);
         answer.setEditable(true);
           ....
         question.requestFocus();
           ....
      }
   }
```

Implementation of a refactoring

Patchset 0/5: Remove divergent changes in Project class

For this refactoring part, we decided to clean the Project class which was very large, with low cohesion. In order to do this, we moved some methods dealing with SaveLoad, print and image management. Some patchsets are made of sub-patchsets because the refactoring was a little bit more complex and we wanted to do small and independent changes. We removed some message chains and we also applied the principle of preserving whole object.

Patchset 1/5: Move Method copyPictureFile from Project to ImageManager

The project was responsible for saving the picture associated to a project. That was a divergent change code smell because the project had to load this file. Indeed, it had to get the path of the picture and load it. It wasn't good at all in terms of cohesion. In order to increase the cohesion of the Project class we decided to move this function in the ImageManager class which already deals with this kind of stuff.

1.1 Copy the code from the source method to the target

ImageManager.java

```
public static File copyPictureFile(File pictureFile, String name)
  {
          String fileName = pictureFile.getName();
      File newFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" + fileName);
      ImageManager.copyImage(pictureFile,newFile);
      return newFile;
  }
```

Project.java

```
public static File copyPictureFile(File pictureFile, String name)
  {
          String fileName = pictureFile.getName();
           File newFile = new File(SaveLoad.getProjectFolder() + "/" + name + "/" + fileName);
          ImageManager.copyImage(pictureFile,newFile);
          return ImageManager.copyPictureFile(pictureFile, name);
  }
```

1.2 Remove source method and fix references

Project.java

```
public void addCard(Card card,Status status){
        // ....
        File copiedFile = copyPictureFile(card.getPictureFile());
        File copiedFile = ImageManager.copyPictureFile(card.getPictureFile(), name);
        // ....
}
```

Patchset 2/5: Move getImageIcon and its related methods from Project to ImageManager

As copyPictureFile, getImageIcon deals with image and calls GUI and SaveLoad methods to do that. It basically creates and returns an image icon based on the image name and folder path. For this refactoring, we had some steps to follow because the original method was:

```
public ImageIcon getImageIcon(String imageName){
        return GUI.createImageIconFromFullPath(getPathTo(imageName));
}
```

However, we can see that getPathTo is a method of Project but it's not logical because it returns an absolute path of a file. So, it would be logical to move it into SaveLoad class which deal with all file management, saving and loading. First step was a move method of getPathTo from Project to SaveLoad (which was divided into 2 steps: copy the source code from the source method to the target and remove source method and fix references):

SaveLoad.java

```
        // We pass the project in parameter
          public static String getPathTo(String thing, Project project){
              String folderPath = getProjectFolder().getAbsolutePath() + "/" +
                          project.getName(); //to the folder of the image
              String absolutePath = folderPath + "/" + thing; //the absolute path to the image
              return absolutePath;
          }
```

Project.java

```
        public ImageIcon getImageIcon(String imageName){
                return GUI.createImageIconFromFullPath(SaveLoad.getPathTo(imageName, this));
        }
```

Second step was to move getImageIcon from Project into ImageManager. First, we copy the line of the return statement into a new method getImageIcon in ImageManager. Finally, we delete the original method and fix all references that called the getImageIcon in Project to the ImageManager's one. In ImageManager, now we have:

```
public static ImageIcon getImageIcon(String imageName, Project project){
        return GUI.createImageIconFromFullPath(SaveLoad.getPathTo(imageName, project));
}
```

Patchset 3/5: Move Save from Project to SaveLoad

In the same way, we want to move save method into SaveLoad in order to increase cohesion of the Project class. We have to do this in three different steps: move saveNotes, move saveCards and finally move overall save method.

First, we copy the source code of saveCards from Project to SaveLoad. Then, we remove the source code in Project and fix references.

Secondly, we do the exact same thing for saveNotes (from Project to SaveLoad). Finally, we do the same thing for the overall save method.

We copy the source code from Project to SaveLoad.

Project.java

```
        public void save()
```

```
{
            SaveLoad.save(this);
}
```

SaveLoad.java

```
public static void save(Project project){
        //the methods are split up for convenience
        SaveLoad.saveCards(project);
        SaveLoad.saveNotes(project);
}
```

Finally we remove source code and fix references, so the save method in Project doesn't exist anymore.

Patchset 4/5: Move print from Project to Controller

The print method in Project allows the user to print flashcards and to gain points for printing cards. In order to increase cohesion, we want to move print feature into Controller because this class deals with points and management of features. This way, in print function, we will call Printer.print and make gain points with Controller method.

First, we copy the method of Print method into another Print method in Controller. We remove the body of Project.print and replace it with

```
public void print(Controller controller)
{
            controller.print(this); // we pass the Project instance
}
```

Then, we preserve whole object in method Controller.print and introduce variable in print, realPrint functions in Printer class. Indeed, in print method into Printer, the project and the cards are passed in parameter. It's exactly the same thing with realPrint whereas project contains cards. So, we now have:

Controller.java

```
public void print(Project project)
{
            -    Printer.print(project, project.getCards());
            +    Printer.print(project);
}

public static void print(final Project project){
        SwingUtilities.invokeLater(new Runnable(){
            public void run(){
-               Printer.realPrint(project,cards);
+               Printer.realPrint(project);
            }
        });

private static void realPrint(final Project project){
  // introduce variable
```

```
 +     final ArrayList<Card> cards = project.getCards();
...
}
```

Finally, we remove the original method and we fix references with new Controller.print method.

Patchset 5/5: Remove message chains in Project.getFolder method

Message chains increase coupling and reduce clarity of the code. So we wanted to clear the code in getFolder function.

First, we create a method getFolderPath in SaveLoad, then we copy the code from the source to the target and finally fix reference in getFolder.

Project.java

```
public File getFolder(){
 -      return new File(SaveLoad.getProjectFolder().getAbsolutePath() + "/" + name);
 +      return new File(SaveLoad.getFolderPath() + "/" + name);
}
```

SaveLoad.java

```
+          public static String getFolderPath(){
 +       return getProjectFolder().getAbsolutePath();
 +    }
```