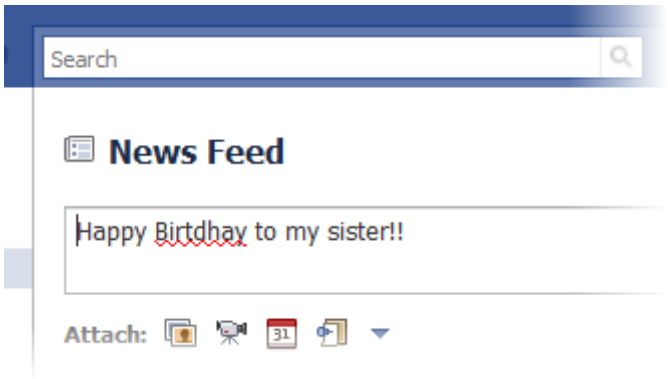
COP 3502
Project 5 (Due Dec 6, 2016)



Background

The University of Florida English department has asked you to create an autograder for its introductory class. There are many types of students who are taking this class. Some are high-achieving **High School** students, some are University of Florida **Undergraduate** students, and some are **Graduate** students who need a refresher. They would like to create a parent (base) Class that can be extended to provide functionality for various kinds of students. The department has specifically asked for the ability to supply the name of the file containing a student's essay (the student's' ID number.txt) and have the program grade the essay, by listing all the misspelled.

Objective

- Read in the file given
- Store each misspelled word
- List all misspelled words
- Generate an output file that contains the student's graded feedback

Your Assignment

You will need to create **five** files complete this project and name them **exactly** as follows: `Project5.java`, `Student.java`, `HighSchoolStudent.java`, `UndergraduateStudent.java`, `GraduateStudent.java`, `Dictionary.java`, `Grader.java`.

The **`Project5.java`** and **`Student.java`** files have been provided, and it's your responsibility not to modify them. If you modify them, you will most likely fail the grading

program. Each of the .java files and their associated classes are described as below:

Project5.java (Do not modify, or you will fail the grading)

The Project5 Class will hold the **main method** of your program. We have already implemented that for you. Generally, it will:

- ask you to input the dictionary name
- load the dictionary
- Grade the students' essays (in a loop) based on the student ID (input from the keyboard).
- Terminate when `"exit"` is typed in for student ID.

Do not modify this class. It provides you an entry point and a skeleton from which to start your project. If the other five required classes have not been correctly implemented, it's highly possible that your program will fail to compile.

Also, you **can** add your own methods in addition to the mandatory methods and fields in the five required classes. Please do not change the method signature to handle exceptions. You should surround any code that may throw an exception in a try/catch block. The exceptions should not be raised to the main method.

Dictionary.java

The dictionary class will load the dictionary from a dictionary file, and be used to check misspellings. The class **must** have the following members with the **exact same signatures:**

- A data field to hold the words from the dictionary
  - `private ArrayList<String> dictionary = new ArrayList<>();`
- A method that returns the number of words in the loaded dictionary
  - `public int getVocabularySize()`
- A method which will load the dictionary file based on the file name. You may assume the dictionary file is in the same folder as the Dictionary class. If loading the dictionary succeeds (meaning that the file was found), return true. Otherwise return false.
  - `public boolean loadDictionaryFromFile(String filePath)`
- A method which being used to check whether a given word is in the dictionary. If the word is in the dictionary, return `true`, otherwise return `false`. In the essay, a word is considered a series of **letters,** separated by spaces. **As long as the word, exactly as given, is found in the dictionary, we will consider it to be correct. If a word is not in the dictionary, it will be considered as a misspelling. For example, if you pass in the word *"cat's"*, you will see that *"cat"* is in the dictionary, however *"cat's"* is not. In this case, "*cat's*" will be considered as misspelling. Please follow this rule. DO not make up your own rule.**
  - `public boolean isWord(String word)`

Grader.java

The Grader class will contain a Dictionary object and use it to grade a student's essay from the essay file. The class **must** have the following members with the **exact same signatures**:

● Fields for the following attributes

- `private boolean available`
  - This variable indicates whether the student variable in the Grader class is holding a Student instance
- `private Student student`
  - Variable which holds the graded student instance
- `private Dictionary dict`
  - Variable which holds the dictionary instance

● A constructor which will pass in a created dictionary instance, then initialize the local dict field

- `public Grader(Dictionary dict)`

● A method which will grade the given student essay based on passed in student file name. The grading criteria is explained in the **Dictionary class**. You should use the *`isWord()`* method in the **Dictionary class** to decide whether a word is misspelled. After parsing the original essay file, the **you should instantiate the local** `student` **variable to an appropriate Class Instance.**

- E.g. after parsing the essay file, if you find out that the student is a Graduate Student, the student's name is *FName LName*, id is *12345678,* school major is *Computer Science*, advisor is *Kyla McMullen.* If you have stored all the errors in the variable `errorList`, Then you should instantiate the appropriate Student Object for the student variable. For example:

`student=new GraduateStudent(name,id,essay.toString(),errorList,major,advisor).`

After grading the file and creating the appropriate Student object, the `available` variable should be set to false and the method will return **true**. Otherwise (e.g. IO error/exception ...), the method should return **false** and reset all local variables except the `dict` field by calling `public void reset()` method. **Important: you should handle all exceptions inside the method. DO NOT throw any exception throw the method, or the mandatory signature will be changed.**

- `public boolean gradeStudent(String fileName)`

● A method to check whether the grader is available to grade next student. "**available"** means that the *`student`* **field has been set to** *`null`* **and the** *`available`* **variable is** *`true`*.

- `public boolean isAvailable()`

● A "getter" method to get the graded/generated student instance

- `public Student getStudent()`

● A method to reset the local variables, excluding the `dict` field. When the reset method is called, the student variable should be set to null and the available variable should be set to true

- `public void reset()`

Student.java (Do not modify, or you will have compilation errors in grading)

The Student class will act as your **parent** class from which the different types of students will be derived. This class contains the following members:

- Fields for the following attributes
  - `private String name  \\ first last`
  - `private String id`
  - `private String essay`
  - `private ArrayList<String> errorList`
- "Getter"/"Setter" methods for each attribute
  - `public String getEssay()`
  - `public void setEssay(String essay)`
  - `public String getId()`
  - `public void setId(String id)`
  - `public String getName()`
  - `public void setName(String n)`
  - `public ArrayList<String> getErrorList()`
  - `public void setErrorList(ArrayList<String> errorList)`
- A method that will format and return all misspelled words in the error list, to be used for printing
  - `public String getPrintableErrorList()`
- An abstract method being used to write the graded files to file system, which you are supposed to implement in all the subclasses
  - `public abstract void writeToFile()`
- A constructor
  - `public Student(String name, String id, String essay, ArrayList<String> errorList)`

HighSchoolStudent.java

The HighSchoolStudent class is a **child class** of the Student Class. The class **must** have the following members with the **exact same signatures**:

- Fields for the following attributes
  - `private String nameOfSchool`
- "getter" and "setter" methods
  - `public String getSchoolName()`
  - `public void setSchoolName(String schoolName)`
- A method implements the abstract method in parent class, which will write the student's grading information to the file.
  - `public void writeToFile()` -
    - The file should be in the same path as all class files.
    - The file name should be named `id+"_graded.txt"`. e.g. if the student's id is "`12345678`", the generated file name should be "`12345678_graded.txt`"
    - In the file, the content should be written as following sequence: (suppose the student's name is *FName LName*, id is *12345678,* school name is *Gainesville High School*)
      - `High School Student FName LName`
      - `Student ID: 12345678`
      - `School Name: Gainesville High School`
      - `(1)errwrod1`

- `(2)erword2`

■ How to define error words will be discussed in Grader class.
■ **Hint**: use `public String getPrintableErrorList()` method in parent Student class to format the error words output.
■ **Important: you should handle all exceptions inside the method. DO NOT throw any exception throw the method, or the mandatory signature will be changed. You should enclose the code that may throw the exception within a try/catch block.**

- A constructor
  - The constructor must initialize all fields of the Student object, using the constructor's parameters
    - `HighSchoolStudent(String name, String id, String essay, ArrayList<String> errorList, String nameOfSchool)`

UndergraduateStudent.java

The UndergraduateStudent Class is a **child class** of the Student Class. The class **must** have the following members with the **exact same signatures**:

- Fields for the following attributes
  - `private String major`
- "getter" and "setter" methods
  - `public String getMajor()`
  - `public void setMajor(String major)`
- A method implements the abstract method in parent class, which will write the student's grading information to the file.
  - `public void writeToFile()`
    - The file should be at the same path as all class files.
    - The file name should be `id+"_graded.txt"`. e.g. if the student's id is `"12345678"`, the generated file name should be `"12345678_graded.txt"`
    - In the file, the content should be written as following sequence: (suppose the student's name is *FName LName*, id is *12345678,* school major is *Computer Science*)
      - `Undergraduate Student FName LName`
      - `Student ID: 12345678`
      - `Major: Computer Science`
      - `(1)errwrod1`
      - `(2)erword2`

    - How to define error words will be discussed in Grader class.
    - **Hint**: use `public String getPrintableErrorList()` method in parent Student class to format the error words output.
    - **Important: you should handle all exceptions inside the method. DO NOT throw any exception throw the method, or the mandatory signature will be changed. You should enclose the code that may throw the exception within a try/catch block.**

- A constructor
  - The constructor must initialize all fields of the Student object, using the constructor's parameters

■ `UndergraduateStudent(String name, String id, String essay, ArrayList<String> errorList, String major)`

GraduateStudent.java

The GraduateStudent Class is a **child class** of the Student Class. The class **must** have the following members with the **exact same signatures**:

- Fields for the following attributes
  - `private String major`
  - `private String advisor`
- "getter" and "setter" methods
  - `public String getMajor()`
  - `public void setMajor(String major)`
  - `public String getAdvisor()`
  - `public void setAdvisor(String advisor)`
- A method implements the abstract method in parent class, which will write the student's grading information to the file.
  - `public void writeToFile()`
    - ■ The file should be at the same path as all class files.
    - ■ The file name should be `id+"_graded.txt"`. e.g. if the student's id is `"12345678"`, the generated file name should be `"12345678_graded.txt"`
    - ■ In the file, the content should be written as following sequence: (suppose the student's name is *FName LName*, id is *12345678,* school major is *Computer Science*, advisor is *Kyla McMullen*)
      - `Graduate Student FName LName`
      - `Student ID: 12345678`
      - `Major: Computer Science`
      - `Advisor: Kyla McMullen`
      - `(1)errwrod1`
      - `(2)erword2`
    - ■ How to define error words will be discussed in Grader class.
    - ■ **Hint**: use `public String getPrintableErrorList()` method in parent Student class to format the error words output.
    - ■ **Important: you should handle all exceptions inside the method. DO NOT throw any exception throw the method, or the mandatory signature will be changed. You should enclose the code that may throw the exception within a try/catch block.**
- A constructor
  - The constructor must initialize all fields of the Student object using the constructor's parameters
    - ■ `GraduateStudent(String name, String id, String essay, ArrayList<String> errorList, String major, String advisor)`

Sample Essay Files

(saved as 11111111.txt)

```
Graduate Student

John Smith
```

```
11111111

Computer Science

Kyla McMullen

This esasy is not very long. I hope my advisor won’t be agnry
with me.
```

Sample Essay File

(saved as 12345678.txt)

```
Undergraduate Student

Jane Doe

12345678

Mechanical Engineering

Writing an essay is hard. If I had a nickel for every misspeled
word in this essay I would have exactly one nickel.
```

Sample Essay File

(saved as 99999999.txt)

```
HighSchool Student

Allen Anderson

99999999

Gainesville High School

Shall I compear thee to a summer day?

Thou art more lovely and more temparate:

Rough winds do shake the darling buds of May,

And summer's leese hath all too short a date
```

Sample Output File (saved as 11111111_graded.txt)

```
Graduate Student John Smith

Student ID: 11111111

Major: Computer Science

Advisor: Kyla McMullen

(1)esasy

(2)won’t

(3)agnry
```

Sample Output File (saved as 12345678_graded.txt)

```
Undergraduate Student Jane Doe
Student ID: 12345678
Major: Mechanical Engineering
(1)misspeled
```

Sample Output File (saved as 99999999_graded.txt)

```
High School Student Allen Anderson
Student ID: 99999999
School Name: Gainesville High School
(1)temparate
(2)winds
(3)buds
(4)summer's
(5)leese
```

Sample Run (input in red)

```
Please specify the file name (without extension) of the dictionary.
pineapple
Failed to load dictionary. Please try again.
Please specify the file name (without extension) of the dictionary.
dictionary
Dictionary load successfully.
Please specify the ID of the student whose essay will be graded.
11111111
Student 11111111 has been graded.
Please specify the ID of the student whose essay will be graded.
12345678
Student 12345678 has been graded.
```

```
Please specify the ID of the student whose essay will be
graded.

pokemon

Grading student pokemon failed. Please try another one.

Please specify the ID of the student whose essay will be
graded.

exit
```

Submission Requirements

- Zip all seven Class files `Project5.java`, `Student.java`, `HighSchoolStudent.java`, `UndergraduateStudent.java`, `GraduateStudent.java`, `Dictionary.java`, `Grader.java` and your extra classes into one single zip file **without any folder hierarchy**. For verbal and pictorial instructions on how to do that, please refer Project 4 submission requirements.
- The zip fileshould be named `project5_ulfid.zip` where uflID is the alphanumeric portion of your ufl email account that comes before the **@ufl.edu** part.
- Submit on time using Canvas

Notes

- **When inputting essay files and generating graded output files, make sure they are in teh same folder as all your class files.**
- If your zip file is not named `project5_ulfid.zip`, your project will not be graded.
- If you do not name the items above exactly as specified, where specified, your project will not be graded correctly.
- Please convert both the dictionary words and essay words to **lowercase** and **remove all punctuation except the apostrophe (') and hyphen (-)** during word comparison.
- It is highly recommended that you test your program piece by piece before assembling your final code. You will have a much easier time if you build your program piece by piece rather than trying to write the entire program. If you have

more questions about the project, it is highly recommended to consult your TA at office hours.

❖ Please make sure to mimic the output in the Sample Runs, and your output should be the same as the one in the Sample Runs. **More importantly, your file output should follow the format described previously. The grades will be decided by those output files and the implementation of mandatory methods/classes.**

❖ **Some, unit testing will be made available for this project. You will still need to compare your output file to the correct output.**

Grading

● 100% of your grade will be based on your program's ability to generate proper output (files) and using all of the mandatory methods.