CIS 200: Project 8 (50 points)
Due Monday, Apr 18th by midnight
(Syllabus originally listed the due date as Fri, Apr. 15th … given an extra weekend to complete)

Reminder: Programs submitted after the due date/time will be penalized 10% for each day the project is late (not accepted after 3 days, i.e. Midnight, Thu, Apr 21st)

Reminder: ALL projects are **intended to be done individually. Don't share your code with anyone.** *If it is not your work, don't submit it as yours!* Refer to the policy within the course syllabus which states, ***"If two (or more) students are involved in ANY violation of this policy, ALL* students involved receive a *zero*** for the assignment and the offense is officially reported to the *KSU Honor Council*. The second offense results in a failing grade for the course and possible suspension from the university (this decision is made by the *K-State Honor Council*)."

Assignment Description: Design and develop a MVC solution to calculate a student's grade based on *weighted averages*.

Points Possible					*Example Student*		
Category	**Pts**	**Weight**	**Adjusted**		**Pts**	**Adjusted**	**%**
Lab	200	15%	157.5		194	152.8	97.0%
Projects	400	15%	157.5		295.5	116.4	73.9%
3 Exams	150	30%	315.0		82.5	173.3	55.0%
CodeLab	200	10%	105.0		180	94.5	90.0%
Final Exam	100	30%	315.0		62.5	196.9	62.5
Total:	1050	100%	1050	**Total:**	814.5	733.8	
Overall %	**69.9%**	***← Student Adj Ttl (733.8) / Adj Ttl (1050) = 69.9%***					

Within points possible, the ***adjusted points for each category*** = *total points possible* x *Weight %*.
For example, *Lab Adjusted Points* = 1,050 * 0.15 = 157.5

Within each student, the ***adjusted points for each category*** = *(points earned / points possible)* x *Adjust Points possible*. For example, *Lab Adjusted Points* = (194/200) * 157.5 = 152.8.

Note: Last column for '*Example Student*' (%) on each category shown simply for reference.

Implementation Requirements:

Your project will contain **three** classes: ***Student (Model), View*** and ***StudentApp (Controller)***.

Student (*Model Class*)

Do not read in input from user nor write output within this class. Data validation should be done in the Model class – however, *data validation is NOT required for this project*.

- Create a class called ***Student*** that contains (*at a minimum*) the following 9 ***private*** data properties – student's *first name, last name, WID* and student's total score on *labs*, *projects*, *exams*, *codelab*, and *final exam* (all doubles) and the student's *overall % score* (double).
- User will enter in the ***total points possible*** for each category in the *StudentApp* class, but store the ***weight %*** as *private constants (15%, 15%, 30%, 10%, 30%, respectively)* in the *Student* class

Include, at a minimum, the following constructors/methods **within the *Student* class** (your class can contain *more* methods than these but must contain at least these methods)

1. A default no-argument constructor to initialize a *Student* object to the following default values: first/last name = "*no name entered*", WID = "*no WID*" / all numeric values ***explicitly*** to *zero*.
2. An 8-argument constructor to initialize a *Student* object to values passed-in for *first & last name, WID* and student's ***total, adjusted or adjusted %*** scores on *labs*, *projects*, *exams*, *codelab*, and *final exam*. (*Overall %* is calculated)

(Note: Both constructors might not be used for this project, but are still required to be defined in this class)

3. A ***private*** method that ONLY calculates the ***weighted overall %*** for the student and stores in the proper data property for a student object.
4. A ***toString*** method that ***returns a String*** that can be used to display a String representation of a Student – specifically, *lastname, firstname* of the student, their weighted overall % (one number after the decimal), and their final grade (based on the grading scale below). Method must NOT contain any print statements and must be declared using the following method signature: ***public String toString()***

89.5 or above	***A***
79.5 or above	***B***
68.5 or above	***C***
Above 58.5	***D***
58.5 and below	***F***

*Sample Output using the String returned by the **toString** method…*

Student Name: Jones, Bob
WID: 123456789
Overall Pct: 79.5%
Final Grade: B

- Again, the **Student** class does not contain any I/O (no println's, nextLine's, etc.) Only *data properties*, *constructors*, *methods* that manipulate and/or use the data properties, and the *toString* method.

View (*View Class*)

- The **View** class will contain **only** the *Input-Output* portion of the program. I will leave it up to you how this class is designed, but I would suggest that you follow the *40-20-40* rule discussed in class and sketch out a design of your class (i.e. what methods it will contain, what each method will do, the method signature of each, etc.) before you actually start coding. You will *not* submit this design but it will help the coding process go quicker in the long run.

StudentApp (*Application/Controller Class*)

- The StudentApp class will should contain ***only*** a main method with **NO print statements, data validation** or **input statements** *(i.e. s.nextLine)*. Communicate between the *Model* and *View* classes by creating objects of each in the *StudentApp* class. You must call the constructors/methods defined in the *Student* and *View* classes to perform the "*work*" of the application.
- Optional methods are acceptable, if they are not a required part of the *Student* class, but keep to a minimum. Within main, allow the user to enter information for as many students as they desire (up to a maximum of 20 – do not allow more). Store the information of each student in a *Student* object and store the object in an *array of Student* objects. *YOU* can decide how to end the input. Once information has been read in for all desired students, display the info as shown for each student, pausing the output for each student until 'enter' is pressed (*see next page*).

Sample Run (yours can vary from the output shown below, as long as all info is displayed)

```
Please enter the total number of points possible for Labs: 200
Please enter the total number of points possible for Projects: 400
Please enter the total number of points possible for Exams: 150
Please enter the total number of points possible for CodeLab: 200
Please enter the total number of points possible for Final Exam: 100

Enter the student's first name: Bob
Enter the student's last name: Smith
Enter the student's WID: 123456789

Enter the student's Total Labs score: 182.5
Enter the student's Total Projects score: 305
Enter the student's Total Exams score: 82.5
Enter the student's CodeLab score: 182
Enter the student's Final Exam score: 96

 Would you like to enter another student?
        1 Student(s) entered so far.
        (20 students can be entered.)
Press Enter for another student or any other key to quit.

Enter the student's first name: Carol
Enter the student's last name: Jones
Enter the student's WID: 987654321

Enter the student's Total Labs score: 190
Enter the student's Total Projects score: 390
Enter the student's Total Exams score: 140
Enter the student's CodeLab score: 200
Enter the student's Final Exam score: 95

 Would you like to enter another student?
        2 Student(s) entered so far.
        (20 students can be entered.)
Press Enter for another student or any other key to quit. n

Student Name: Smith, Bob
WID: 123456789
Overall Pct: 79.5%
Final Grade: B
        Press enter to display next student...

Student Name: Jones, Carol
WID: 987654321
Overall Pct: 95.4%
Final Grade: A
        Press enter to display next student...

All students displayed...
```

Extra Credit (+5): (This should NOT be difficult, if the rest of the appl. was developed correctly)

Include an additional class (***View_GUI.java***) that preforms the same function as the *View* class, except that it uses a *GUI-based* interface vs. a *Console-based (text-based)*. Modify your ***StudentApp*** class so that it reads the desired interface requested from the command line.

For example: **java StudentApp console** → Run *Console-based* interface

java StudentApp gui → Run *GUI-based* interface

Documentation:
You must put a description of the project at the top of the file **and at the top of each method**. Please use this template for the **top of the file**:

```
/**
 * (description of the project)
 *
 * @author (your name)
 * @version (which number project this is)
 */
```

Please use this template for the **top of each method**:

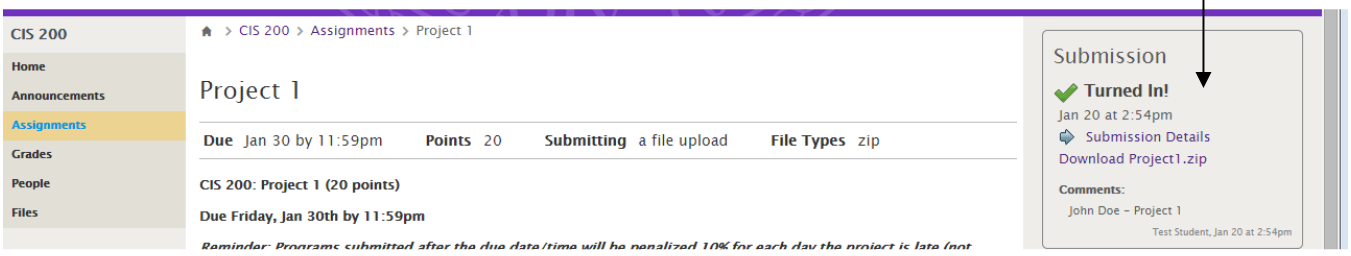
```
/**
 * (description of the method)
 *
 * @param (describe first parameter)
 * @param (describe second parameter)
 * (list all parameters, one per line)
 * @return (describe what is being returned)
 */
```

Submission – *read these instructions carefully*

To submit your project, first create a folder called **Proj8** and copy your completed *Student.java, View.java (and View_GUI.java if doing the extra credit)* **and** *StudentApp.java* files into that folder. **Make sure all files are included and compile first**! Then, right-click on that folder and select "*Send To->Compressed (zipped) folder*". This will create file ***Proj8.zip***.

Log-in to Canvas and upload your ***Proj8.zip*** file. Only a .zip file will be accepted for this assignment in Canvas. **Put your name and Project 8 in the comments box. If you did the extra credit,** include this in your comments box when submitting.

Reminder: **It is the student's responsibility to verify that the correct files are properly submitted. If you don't properly submit the correct file, it will not be accepted after the 3-day late period.**



Grading: You must submit **all 3 classes** and ***ALL*** must compile to be considered for partial credit. For example, submitting only the *Student* class would not earn you any points. **Programs that do not compile will receive a grade of 0**. Programs that *do* compile will be graded according to the following rubric:

Requirement	**Points**
StudentApp class (CODE) ***(9 points)***	
Should contain ONLY a main method. Contains NO *print* or *input* statements or data validation; Properly creates objects of the *Student* and *View* classes and properly utilizes the methods defined in these TWO classes. Allows no more than 20 students to be entered.	**9**
Student class (CODE) ***(10 points)***	
Contains NO *print* or *input* statements. (Usually contains data validation, but none required for this project). Contains no-arg and 8-arg constructor, PRIVATE calc method, and toString	**10**
View class (CODE) ***(7 points)***	
Contains **only** the *Input-Output* portion of the program. No data validation	**7**
Execution (20 points)	
Produces correct output using the input data given on p.3 of the instruction sheet	**9**
Produces correct output using additional input data *not* provided on p.3 (i.e. test your program thoroughly!)	**10**
Allows a way for user to stop or quit entering students	**1**
Documentation – includes documentation on ***EACH file*** and above ***EACH method***	**3**
Project Submission (zip file with name and Project # in description box)	**1**
Extra Credit – View_GUI class (+5 pts.)	
Reads the I/O desired from the command line / Works properly using both a Console-based interface and a GUI-based interface	**+5**
Minus Late Penalty (10% per day)	
Total	**50+5**