PROG 1400 - ASSIGNMENT 2

INHERITANCE

Assignment Value: *15%* of overall course mark.

Due Date: **See due date designated on the Assignment 3 dropbox on Brightspace.**

Nov 12 @ 11;59pm.

Late submissions will receive the standard late submission penalty as stated in the course outline.

Assignment Instructions:

Use IntelliJ to create a Java Swing UI application.

Submissions:

Submit your complete folder (project) in zipped folder on Bright space (Assignment 2 Dropbox). Make the project's name (folder name) is "Assignment2_Yourname". Don't forget to use the link for GitHub repository to save another copy of this assignment up into the cloud. Submit your recording to me through a private chat on Teams.

Evaluation:

To ensure the greatest chance of success on this assignment, be sure to check the marking rubric contained at the end of this document or in Brightspace. The rubric contains the criteria your instructor will be assessing when marking your assignment.

Program – College Accounting Application

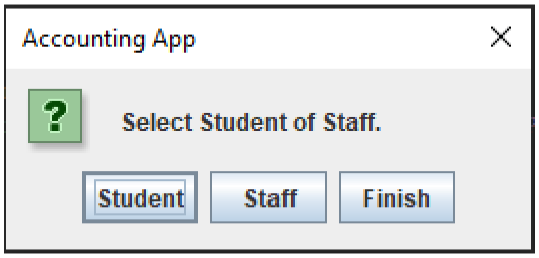
DESIGN AND CODING

Each Person in the system (either Student or Staff) shares common attributes of **Name** and **Address**. Make sure your classes do not duplicate any code by using Java inheritance to share these common attributes. So, your program might need at least 4 classes (Main, Person, Student, Staff).

PROGRAM REQUIREMENTS

You have been tasked with creating an accounting application for a college. The Accounting department wants to track Student fees and Staff payroll in the same application. After all data is entered, the application will run a financial process that will either invoice students for that semester's fees or pay staff their bi-weekly salary.

The application will start up with a Java Swing dialog that asks the user to enter Student, Staff or Finish.



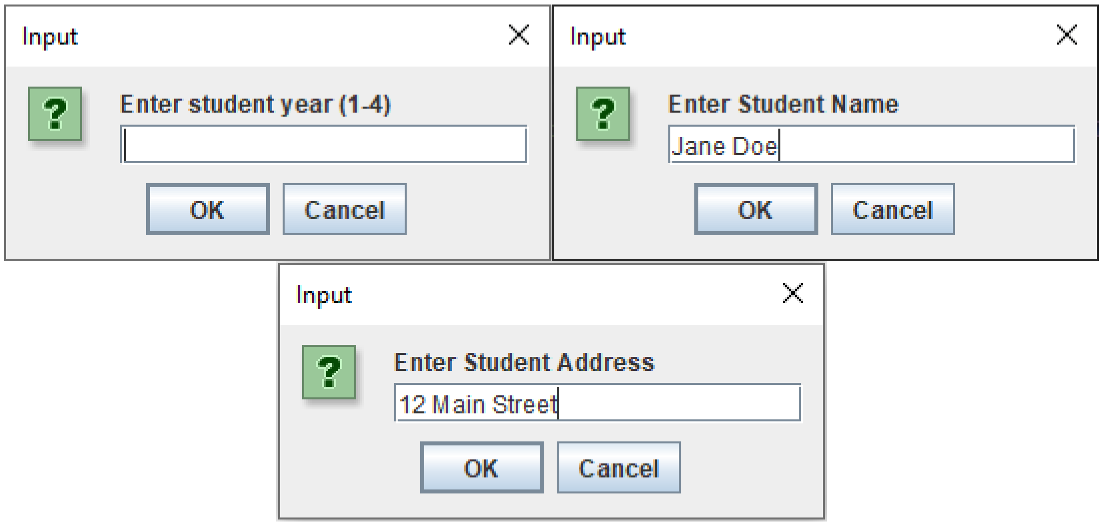
If the user selects Student, they will be asked to enter the following information for the student:

- Name
- Address
- Year (1,2,3,4)

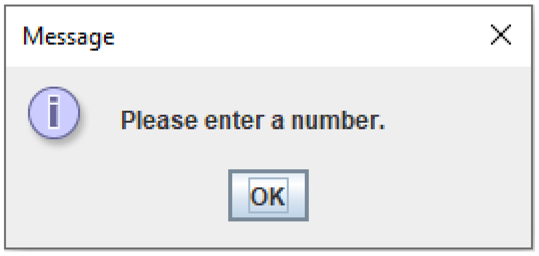
If the user selects Staff, they will be asked to enter the following information for the staff member:

- Name
- Address
- Years of Service

The data entry dialog for Student information looks like the following:

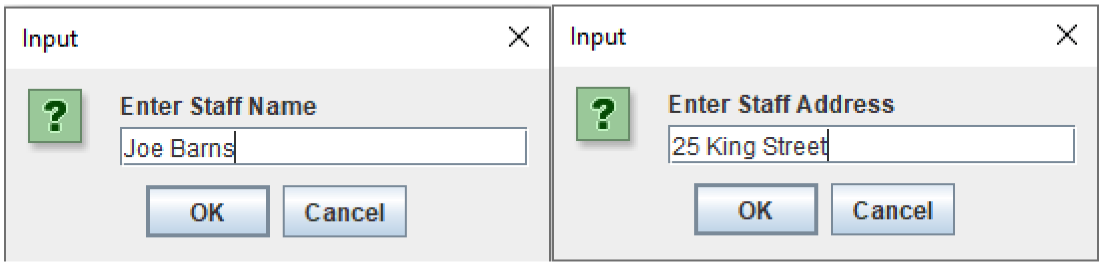


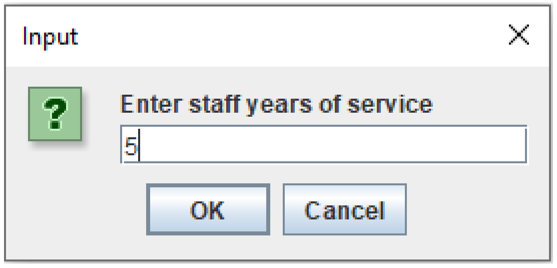
Note that there is validation on these fields. If you enter blank, you should get a message dialog saying, "Please enter valid information". If you enter a non-numeric value for student year, you should see the following:



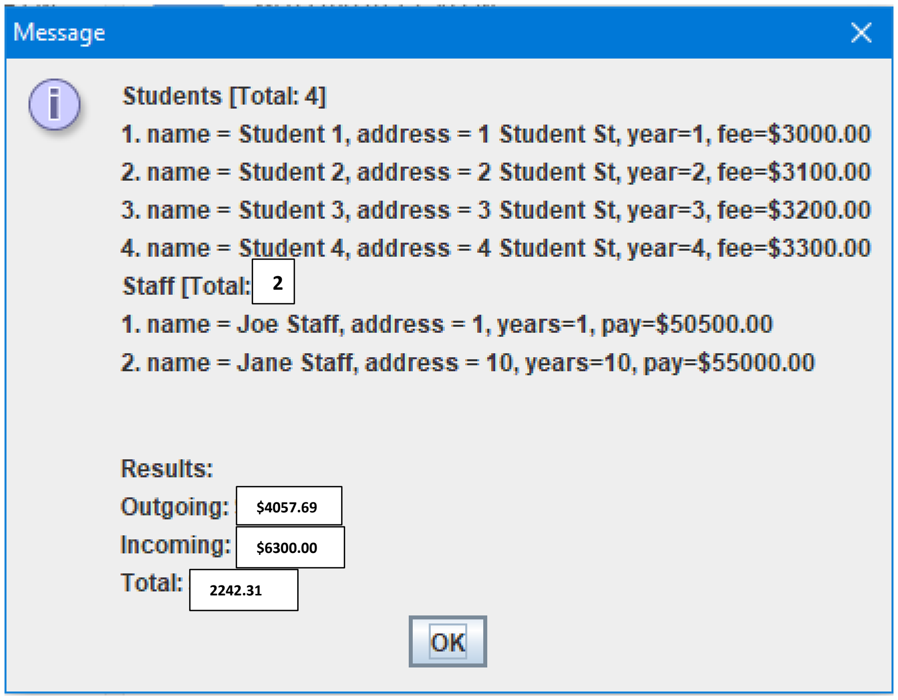
Also validate that the student year is between 1 and 4. Inform the user with a dialog if it is outside this range.

If the user selects "Staff", they will also be presented with dialogs to enter Name and Address but now the dialog should prompt "Enter Staff Name" and "Enter Staff Address". The same validation applies as with Student Name and Address. The third dialog asks the number of years of service. This must be validated as a whole number greater than zero and less than 30.





Finally, once the students and staff information has been entered, the final payment and invoicing process will be run. The results will be presented in a dialog that looks like the following:



```
Students [Total: 4]
1. name = Student 1, address = 1 Student St, year=1, fee=$3000.00
2. name = Student 2, address = 2 Student St, year=2, fee=$3100.00
3. name = Student 3, address = 3 Student St, year=3, fee=$3200.00
4. name = Student 4, address = 4 Student St, year=4, fee=$3300.00
Staff [Total: 2
1. name = Joe Staff, address = 1, years=1, pay=$50500.00
2. name = Jane Staff, address = 10, years=10, pay=$55000.00


Results:
Outgoing: $4057.69
Incoming: $6300.00
Total: 2242.31
```

Some requirements of the final accounting process:

- You should output the students and staff details separately. Output the total of each list first, followed by each person with a numeric counter beforehand. **Note**: the details of each staff and student should be reported by over-riding the toString() method.
- Students: Fees are calculated in the class constructor based on which year they are a student. The amount of fees goes up by $100 dollars depending on which year you are. So first years pay $3000, second years pay $3100, and so on. A student pays fees in 2 installments, so you only invoice for half the fee amount.
- Staff: Salary is calculated from a base pay of $50,000. Staff are paid step increases of $500 for each full year of service. Staff get paid every 2 weeks (bi-weekly) so the result of running the accounting process should be to pay the yearly salary divided by 26.
- Results: At the end of the dialog, display what money is going out and coming in after invoicing each student and paying each staff member. Finally, display the total as the difference of outgoing and incoming. This number can be negative or positive to reflect if the college is in deficit or not.
- All dollar amounts should be formatted properly.

College Accounting App

Name: ________________________________

Criteria	Insufficient (0 pts)	Needs Development (1-2 pts)	Sufficient (3-4 pts)	Excellent (5 pts)	Mark	X
Input / Output	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	All Staff and Student inputs can be successfully entered, and use descriptive prompts. The report output lines are well-formatted and contain all expected information All output values are formatted, where appropriate, using proper currency formatting (e.g. preceded by a $ symbol, two decimal places)		
Inheritance/OOP	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Solution displays strong understanding of Inheritance. There should be a parent class that cannot be instantiated and two sub-classes that have appropriate attributes and methods for their type. All variables should be private and exposed via methods. Fees and Pay should be created in the class constructor		2
Data Validation	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Data input is validated properly: integers must be equal to zero or above and in the correct range. Strings must not be empty.	0	0
Functions (Methods)	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Each class has functions that are called to calculate class specific code. I.e. student invoicing and staff biweekly pay calculation is done in the correct class.		
Override	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Each class should override parent code where necessary and this should be used. For example, toString() to report on each person's details.		
Final Report	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	The final report is present and displays expected report data. Output is well-formatted, clearly labeled and has an easily readable layout.		
Calculation	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	All values are calculated properly: student fees, staff overall pay, incoming funds, outgoing funds and final total		
Array & Object Usage	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Dynamic arrays are used to store objects as expected. Proper interaction with object arrays is demonstrated.		
GitHub Classroom	No submission to GitHub Repository	Submitted with one or two commits only.	-	Organized commits and several commits done starting from the assignment published date till submission date		2
Comments & Best Coding Practices (At least 60% of the requirements must be complete)	Little to no effort was made or contains too many errors / omissions.	A reasonable effort was made, but there are multiple areas for improvement.	A good effort was made, but at least one error or omission exists.	Organizational or explanatory comments are used extensively, most are meaningful and easily understood. A consistent naming convention was used for most of the program and deviated very little. Source code was clean, consistently well-formatted and easy to read.		

Video Recording Demonstration	No Effort made.	A reasonable effort was made, but there are 3 missing parts.	A good effort was made, but there are 2 missing parts.	. Excellent explanation for the following: 1. The classes, attributes, constructors, and methods. 2. The objects that are created, i.e., the instantiating process. 3. How did you create your final report message? 4. While recording, create a new report message (dialog box) that prints out your name and w-number.		4
				Total:		**/75**