CIS 200: Project 9 (60 points + 25% Extra Credit!)
Due FRI, Apr 19th by 11:59pm
LAST JAVA Project!

Reminder: Help not available after GTA Office hours Fri 12:15-3:15pm, so get questions answered before!

Programs submitted after the due date/time will be penalized 10% for each day the project is late. Projects are not accepted after 2 days after the due date – no exceptions.

Students are expected to follow the academic integrity expectations stated on p.1 of the syllabus and on the Canvas course homepage as well as the Academic Integrity Policy stated on p.7 of the syllabus. While other students and sources are allowed, citation of those sources are REQUIRED within the source code submitted. ***If you didn't write it, you must cite it!***

If considering doing the extra credit, please read the requirements for the extra credit first (p.2), since program development would be different for the extra credit .vs the required project.

Assignment Description:

1) Create an **Employee** class using the UML. Below is just the minimum for the class. You may add to this class however you see fit.

Employee
-name: String -idNumber: String -employeeType: char -payRate: double -yearlySalary: int -WEEKS_IN_YEAR = 52 (int constant - static)
+Employee (String, String, char, double) +Employee (String, String, char, int) +getGrossPay(double): double +getGrossPay(): double +toString(): String +equals(Employee): boolean

Specifics:

1) Employees can either be *Salaried* or *Hourly* employees. Store an 'S' for *Salaried* or 'H' for *Hourly* in *employeeType*.
2) If they are *hourly*, you will use constructor one to create the object and pass in the employee's *name*, *idNumber*, *employeeType* ('S' or 'H'), and their *hourly pay rate (double)*. Set *yearlySalary* to -1.
3) If they are *salaried*, you will use constructor two to create the object and pass in the employee's *name*, *idNumber*, *employeeType* ('S' or 'H'), and their *yearly salary (int)*. Set *hourly pay rate* to -1.
4) *getGrossPay* is an *overloaded method*.

- Method one is used for *hourly* employees. Pass in the *hours* the employee worked and return their *gross pay* for the week. If they work over 40 hours, they are paid *time and half* for time over 40 hours.
- Method two is used for *salaried* employees and simply returns their weekly salary based on the constant stored in WEEKS_IN_YEAR.

5) *toString* returns a String that can be used to display the employee's *name* and *idNumber* only.
6) *equals* tests two *Employee* objects for equality, based on the *idNumber*.

---

2) Create a **Proj9** class that does the following:

a) Create and properly use an **ArrayList of *Employee* objects**. You will allow the user to enter the information for as many employees as they would like. Validate first (see below) then add each object to the *ArrayList* once it is valid. Input will include all info needed to create an *Employee* object.

Since one of the primary objectives of this project is to work with *ArrayLists*, ½ credit maximum will be given for a program that does not utilize an *ArrayList*

YOU will decide where to do the following data validation (*Employee* or *Proj9* class)

b) **Data Validation:** Use Exception Handling (unless otherwise indicated) to check for and *properly handle* the exceptions below. To properly handle an exception (*if it occurs*), display the error message shown on the next page and allow the user to re-enter input until the exception is no longer thrown. You must validate each piece of data input BEFORE moving on to the next piece of data.

Since a secondary objective of this project is to work with exceptions, you must use *exception handling* (unless otherwise indicated) to get points for this portion of the assignment.

- An *empty string* (i.e., enter is pressed) for ANY of the data fields (*throw* and *catch* an exception within a loop until valid)
- Only 'S' or 'H' (upper or lower case) is allowed for *employeeType* (a loop for this is OK, exception handling not required)
- A *character* or *double* is entered for *yearlySalary* (int only – use exception handling)
- A *character* is entered for *payRate or hoursWorked* (double only – use exception handling)

Once all input for an object has been read in and validated, use the *equals* method to verify that employee does not already exist in your ArrayList (case-specific). If it already exists, display the error message "*Duplicate employee – not added to the list*" and do NOT add that employee to the ArrayList.

For maximum points, you must invoke your *equals* method to perform this comparison.

While it is not required, you might want to add additional methods to read and return a valid name, ID, employee type, pay rate or yearly salary, etc. (more modular, so easier to develop, debug, and update.) If doing the extra credit, data validation usually goes in the model class, but I will leave that up to you for this last Java project.

c) Once all data is validated and all input has been read in from the user and stored in the *ArrayList*, use the *toString* method on each object in the *ArrayList* to display the name and ID number followed by the weekly pay formatted to two decimals. If the employee is *Hourly*, you will need to read in the number of hours worked for the week. Validate hours is a valid double value.

For maximum points, you must invoke your overloaded *getGrossPay* method to perform this calculation.

d) Once all output is displayed, prompt the user to enter an *idNumber* – if found (case-specific), *delete* that employee from your *ArrayList* and indicate that the employee has been deleted from the *ArrayList*. If *not* found, display an appropriate error message and allow user to re-enter until a valid id number is entered.

e) Lastly, display the final contents of the *ArrayList* after deleting the employee. Display only the name and ID number. (Hint: *toString*!)

Sample run below… (Input shown in red. Error messages shown in purple.)

Enter employee's name: <enter pressed>
Name is required. Please re-enter
Enter employee's name: <enter pressed>
Name is required. Please re-enter
Enter employee's name: **John Doe**
Enter John Doe's ID #: <enter pressed>
ID is required. Please re-enter
Enter John Doe's ID #: <enter pressed>
ID is required. Please re-enter
Enter employee's ID #: **12345g**
Is employee S)alaried or H)ourly? <enter pressed>
Employee Type is required. Please re-enter
Is employee S)alaried or H)ourly? <enter pressed>
Employee Type is required. Please re-enter
Is employee S)alaried or H)ourly? a
Must be an 'H' or 'S'. Please re-enter
Is employee S)alaried or H)ourly? a
Must be an 'H' or 'S'. Please re-enter
Is employee S)alaried or H)ourly? **s**
Enter John Doe's yearly salary: $52,500
No chars or decimals allowed. Enter integers only
Enter John Doe's yearly salary: 52500.50
No chars or decimals allowed. Enter integers only
Enter John Doe's yearly salary: **52500**
Would you like to enter another employee? (Y/N): y

Enter employee's name: Harold Jones
Enter Harold Jones's ID #: 123asd
Is employee S)alaried or H)ourly? h
Enter Harold Jones's pay rate: 12.50
Would you like to enter another employee? (Y/N): y

Enter employee's name: John Carpenter
Enter Harold Jones's ID #: 123asd
Is employee S)alaried or H)ourly? h
Enter Harold Jones's pay rate: 15.50
Duplicate employee – not added to the list
Would you like to enter another employee? (Y/N): y

Enter employee's name: Bill Smith
Enter Bill Smith 's ID #: 2345b
Is employee S)alaried or H)ourly? H
Enter Bill Smith's pay rate: $12.50
No chars allowed. Enter doubles only
Enter Bill Smith's pay rate: $12.50
No chars allowed. Enter doubles only
Enter Bill Smith's pay rate: 12.50
Would you like to enter another employee? (Y/N): n

Current contents of ArrayList:
John Doe
ID #12345g
Weekly Gross Pay - $1,009.62 (*Comma optional*)

Harold Jones
ID #123asd
Enter Harold Jones's hours worked: 39.5
Weekly Gross Pay - $493.75

Bill Smith
ID #23456
Enter Bill Smith's hours worked: 41.2 hours
No chars allowed. Enter doubles only
Enter Bill Smith's hours worked: 41.2
Weekly Gross Pay - $522.50

Enter an ID number to delete employee: 123sd
ERROR - ID # not found. Please try again
Enter an ID number to delete employee: 123ASD
ERROR - ID # not found. Please try again
Enter an ID number to delete employee: 123asd
Employee with ID#123asd removed from ArrayList

Final contents of ArrayList:
John Doe
ID #12345g

Bill Smith
ID #23456

OPTIONAL Extra Credit Challenge: (25% extra credit)

** To be considered for extra credit, you MUST add a comment in your top documentation "EXTRA CREDIT INCLUDED" and indicate which extra credit was done (#1 only, #1 and #2, ALL, etc.) so that the GTA knows to test your program for these options. Otherwise, the GTA will NOT test for extra credit.

Make the following modifications to *add* the functionality to Project 9. You will submit a SINGLE version of this Project to Canvas. Simply *add* the functionality. You can do *any* or *all* of the extra credit below.

1) (+5%) For the search and delete portion, add all Employees in the *ArrayList* to a *HashMap*, using the ID # as the key value. (Hint: Use a loop to add the ID of each employee as the key along with that employee object as the value). Then use the HashMap to *search* for as many employees as they desire (one at a time) by requesting an *ID Number*. If found, delete that employee from the ArrayList. If not found, display message shown on p.3. Have user enter 'exit' (*upper* or *lowercase* or a *mixture*) to quit

2) (+10%) Use *MVC architecture* by creating a separate class to handle the VIEW using *Console I/O*. Then use the VIEW class to handle all I/O within the Project. Follow the guidelines of MVC Architecture – no input statements (*nextLine*, *nextInt*, *nextDouble*, etc.) or output statements (*println*) in the *Controller* (Proj9) or *Model* (Employee) classes.

3) (+10%) Using *MVC architecture*, create *another* separate class to handle the VIEW using *GUI* – this will give you a total of 4 classes. Since this is a GUI, you can take some flexibility in the final output, as long as it is close and is complete. Again, *no I/O statements* in the *Controller* or *Model* classes.

   Include a line in your controller class initially commented out that creates a *View_GUI* object rather than a *View_Console* object. GTA should be able to comment out one line and uncomment the other and the view should change from Console to GUI.

```
//SWITCH BETWEEN THE VIEWS BELOW
        View_Console view = new View_Console();
        // View_GUI view = new View_GUI();
```

Documentation: At the top of EACH CLASS, add the following comment block (each class description will differ, since the purpose of each class differs.)

```
/**
* <Full Filename>
* <Student Name / Lab Section Day and Time>
* <COMPLETE description of the project (at least 3 or 4 sentences) – i.e. What does the program do?
    Must be detailed enough so outside reader of your code can determine the specifics of the program>
*/
```

At the top of *EACH* method, excluding *main*, add the following comment block, filling in the needed info:

```
/**
 * (description of the method)
 * @param (describe first parameter)      …if no parameters, don't include
 * @param (describe second parameter)
 * …(list all parameters, one per line)
 * @return (describe what is being returned)      …if nothing is return, don't include
 * @throws IOException for File I/O      …if method doesn't throw an exception, don't include
*/
```

-You WON'T generate a JavaDoc for Project 9, but still include JavaDoc Comments on ALL methods.

Requirements

This program will contain TWO separate class files (3 or 4 if doing extra credit #2/#3). EACH submitted files must compile (by command-line) with the statement: **javac <filename>.java**

It must then RUN by command-line with the command: **java Proj9**

***Make sure and test this before submitting, so points are not lost unnecessarily.**

Submission – read these instructions carefully or you may lose points

Programs that do not compile will receive a grade of ZERO, regardless of the simplicity or complexity of the error, so make sure you submit the *correct* file that *properly compiles*.

To submit your project, first create a folder called *Proj9* and copy ALL *.java* files into that folder. Then, right-click on that folder and select "*Send To → Compressed (zipped) folder*" to create the file *Proj9.zip*. Log-in to Canvas and upload the *Proj9.zip* file. Only a .zip file accepted for this assignment in Canvas.

Grading: Since the grader will be testing your program using this data as well as different data, make sure and test your solution thoroughly with both the given data and other possible data.

Programs that don't compile/run at the command line will receive a grade of ZERO, regardless of the simplicity or complexity of the error, submit the *correct* file that verify that it *properly compiles/runs*.

Programs that *do* compile/run will be graded according to the following rubric:

Requirement	Points
**** MAXIMUM of HALF-CREDIT if program does not utilize an ArrayList throughout****	
*****Employee.java (Code) (10 pts.)**	**-**
Documentation – includes correct doc heading with specific purpose and Javadoc comments above EACH method (including Constructors)	**2**
All data properties included, static constant and methods properly declared (2 Constructors, Overloaded *getGrossPay* methods, *toString*, and *equals*	**8**
*****Proj9.java (Code) (12 pts.)**	**-**
Documentation – includes correct doc heading with specific purpose and Javadoc comments above EACH method (including Constructors)	**2**
Create and properly use an **ARRAYLIST** of Employee Objects. Adds each object to ArrayList.	**3**
Properly call methods in the Employee class through Employee objects – uses toString to display	**3**
Calls equals method to check for duplicates AFTER all values are verified	**2**
Call overloaded getGrossPay when displaying weekly pay	**2**
*****Within Employee.java or Proj9.java (Code) (8 pts.)**	**-**
EXCEPTION HANDLING added to handle an *empty string* entered for ANY data field	**2**
Exception Handling or a loop added to handle values other than 'S' or 'H' for *employeeType*	**2**
EXCEPTION HANDLING added to handle a *char* or *double* entered for *yearlySalary* (int)	**2**
EXCEPTION HANDLING added to handle a *char* entered for *payRate or hoursWorked* (double)	**2**
Execution (30 points)	**-30-**
Run 1: Properly error checks and produces identical output using the input data given on the instruction sheet. Search works correct if not found and if found.	**15**
Run 2: Properly error checks and produces identical output using DIFFERENT input data (and constant value) *not* provided (so test program thoroughly!)	**15**
Extra Credit #1: Use a HashMap for the search and delete portion of the program.	**+3**
Extra Credit #2: MVC architecture is used…Console VIEW class created. No I/O statements in the controller or model classes.	**+6**
Extra Credit #3: Additional GUI VIEW class created. No I/O statements in the controller or model classes. GTA comment outs creation of View_Console and uncomments View_GUI.	**+6**
Minus Late Penalty (10% per day)	
Total	**60+15**