Lab 10 (20 points)
There are several learning objectives to this assignment

- Creating Classes that use Inheritance
- Creating Interfaces
- Creating Classes that implement Interfaces
- Creating advanced algorithms
- Using printf and String.format to format currency

Coders will create a Person Interface based on the following UML.

PersonInterface
CURRENT_YEAR: int //set to 2023 (Is final and static since an interface)
setName (name:String): void getName (): String toString(): String // Prints out Name: <name field>

Next, create a Person class that implements PersonInterface per the below UML.

Person
- name: String
+ Person () //name set to “No name yet” + Person (name:String) + <methods per PersonInterface>

Use the Person class to create an Employee class that extends Person. Employee has two private instance variables: (1) an integer called hireYear and (2) a String called idNum. It will be necessary to create two constructors. The default constructor calls super() and sets hireYear to 0 and idNum to Onboarding. The other constructor accepts three arguments - a String for name, an int for hireYear, and a String for idNum. You will need to create two public methods equals(Object o) and getServiceYears(). getServiceYears() subtracts hireYire from CURRENT_YEAR. NOTE-Since CURRENT_YEAR was created in PersonInterface, it is final, static, and accessible to Person and all classes with an is-a relationship with Person (all the other classes).

Employee Methods (**with appropriate javadocs**)
1) equals() based on idNum - a unique entry in a fictional database (see below).
2) getServiceYears() returns the difference between CURRENT_YEAR and hireYear.
3) getter and setter methods for hireDate and idNum.
4) toString() (line1) “Name: “ with name, (line2)“ID Number:” with idNum, (line3) “Year Hired: “ with year employee was hired and “Years of Service:” with years of service.

HINT-In the Employee.toString(), you should incorporate super.toString() to pull toString() info from Person.

```
public boolean equals (Object o){
    boolean isEqual = false;
    if (o != null && getClass()==o.getClass()){
        Employee copy = (Employee)o;
        if (idNum.equalsIgnoreCase(copy.idNum))
            isEqual = true;
    }
    return isEqual;
}
```

Additionally, there are two classes that inherent (extend) Employee class called FullTime and Adjunct. FullTime has a double instance field var called salary. Adjunct has two double instance field vars – hours, hrRate. You will need to create two public methods for FullTime and Adjunct called .toString() and .getSalary() – details below. FullTime also has setSalary() with a double input parameter.

FullTime toString()
(line1) Name: <name field>, (line2) ID Number: <idNum field>, (line3) Year Hired: <year hired> <space> Years of Service: <years of service>, (line4) Salary: <salary>. Limit salary to 2 decimal places

Adjunct toString()
(line1) Name: <name field>, (line2)ID Number: <idNum field>, (line3) Year Hired: <year hired><space>Years of Service: <years of service>, (line4) Hours: <hours field>, Hourly Rate: <hourly rate>, and Salary: <result of hours * hourly rate> Limit hourly rate and salary to 2 decimal places.

getSalary() returns salary for FullTime and hours multiplied by hourly rate to obtain salary for Adjunct.

NOTE: when using .toString(), you do not need to use dot-notation. For example, if you have an Employee object called fred, the statement System.out.println(fred); prints out the fred object (in this case an Employee object), per the Employee.toString().

You will need to do the following:

(1) Create Person Interface and then Person, Employee, FullTime, and Adjunct classes including appropriate constructors, getters and setters as well as a driver (HRDemo) that tests your classes. HRDemo details are below.

HRDemo Details

1) Create a FullTime object called fred passing in the following arguments to an appropriate FullTime constructor - "Flinstone, Fred", 2013, "BR-1", 75000.1234.
2) Create an Adjunct object called barney passing in the following arguments to an appropriate Adjunct constructor – "Rubble, Barney", 2014, "BR-2", 320, 60.55. *320 represents the hours for teaching 2 courses for 2 semesters per year, 60.55 is the hourly rate.*
3) Create a default FullTime object called wilma.
4) Use an appropriate setter to update the wilma object's name to "Flintstone, Wilma"

5) Use an appropriate setter to update the wilma object's idNum to "BR-3"
6) Use an appropriate setter to update the wilma object's hireYear to 2015
7) Use an appropriate setter to update the wilma object's salary to 78123.2468
8) Create a new Employee object called betty passing in the following arguments to an appropriate Employee constructor - "Rubble, Betty", 2017, "BR-4"
9) Create a FullTime object called wilma2 passing in the following arguments to an appropriate FullTime constructor - "Slate, Wilma", 2015, "BR-3", 78123.2468
10) Create a new ArrayList called `staff` that is a collection of ??. HINT: What is the most basic superclass between Person, Employee, FullTime, and Adjunct? Create staff based on this <type> as all other objects will have a is-a relationship to the top most superclass. Polymorphism will be able to take care of knowing how to appropriately print an object based on its type even in there are multiple types in `staff`. 😊
11) Add the following objects to `staff` - fred, barney, wilma, betty, wilma2.
12) Print out each of the staff objects (created above). HINT – Review Ch7 ArrayLists. You can either use a for each loop or you can use a standard for loop. If using a for loop, print out Employee (i+1) before each entry. As a reminder, you cannot access indices in for each loops ☹️ And therefore cannot print out Employee (i+1) using the for each approach.
13) Use the provided equals() to determine if wilma and wilma2 are the same.
14) After confirming that wilma and wilma2 are the same objects, use an appropriate setter to update the wilma object name to "Slate, Wilma"
15) Remove the wilma2 object from staff. Hint – remove subscript four [4].
16) Repeat step 12 above.

EXTRA CREDIT (10pts, that's right 50%!!!) – See Canvas for separate XC Assignment

Submitting your work

For all labs you will need to provide a copy of all .java files. **No need to provide .class files. I cannot read these.** ***NOTE – For Replit, please update Main.java to another name such as TempProb.java, ProChall3.java, etc.*** In addition to your .java files, you will need to provide output files of your console. The name of the output file should match the class name and have the .txt extension such as TempProbOut.txt, ProChall3Output.txt. For GUIs such as JOptionPane, you will instead need to create screenshots. For Windows users, Snipping Tool is a great way to do this. Chromebook - Shift+Ctrl+Show Windows. Mac OS users, you can see how to take screenshots using the following url - https://support.apple.com/en-us/HT201361.