Due Oct 12 by 11:59pm **Points** 100 **Available** Oct 2 at 9am - Oct 13 at 11:59pm

This assignment was locked Oct 13 at 11:59pm.

Module 4 — Composition and Inheritance

Home

Modules

(https://

Homework 4 Refactoring, Composition, Delegation, and Inheritance

1 Employees and Paychecks Redesign

1.1 Introduction

We will revisit your Employee and Paychecks homework assignment to apply some of the key design principles you have learned since homework #1. While familiar, this homework will require you to:

- do a thorough redesign of your previously working solution to meet new requirements
- refactor of your code to meet the new requirements while preserving relevant existing behavior
- consider crisp boundaries for your classes' behaviors

1.2 What to do

We're expanding our payment system to handle payroll for salaried and hourly workers. Redesign your solution from homework 1 and refactor your code to do the following:

EMPLOYEE

You must implement a single `Employee` class that can represent both **salaried** and **hourly** employees. Your Employee class should have two constructors that allows us to distinguish between the kind of employee we're creating. One constructor allows us to create employees that we will pay as "salaried" workers. It takes 5 parameters: name, id, pay rate, pay, and a boolean flag to indicate if the employee is a manager or not. The pay rate for salaried employees is their **yearly salary**. The pay interval determines how frequently the employee is paid: weekly (1 time per week), bi-weekly (1 time every two weeks) or quad-weekly (1 time every four weeks). Assume that there are 52 weeks in the year.

```
Employee(String name, String id, double payRate, int payInterval, boolean isManager )
```

The second constructor takes four parameters: name, id, par rate and hours worked. This constructor creates "hourly" employees similar to those from homework 1. In this case, the pay rate represents the hourly rate, and hours worked represents the number of hours the employee has worked. Hourly employees are paid at their rate * hours if the number of hours worked is 40 or less. If the hours worked exceeds 40, the employee is paid at an overtime rate of 1.5x for all of the hours in excess of 40.

Hourly employees cannot be managers in our system.

```
Employee(String name, String id, double payRate, double hoursWorked)
```

Throw an **`IllegalArgumentException`** if either the name or ID are null OR if they are empty strings (length = 0) when creating employees. It is also illegal for Pay rate and hours worked to be negative.

Employees must implement the following two methods: one to return a Paycheck instance (thereby "getting paid" for the current pay period) and one to answer if the employee is a manager or not:

```
public boolean isManager()

public IPaycheck getPaycheck()
```

PAYCHECK

You must redesign and refactor your **`Paycheck`** class such that it implements the **`IPaycheck`** protocol (interface) below:

```
public interface IPaycheck {
  double getTotalPay();
  double getPayAfterTaxes();
  double getPayRate();
}
```

As part of our redesign our payment company requires that you implement two variations of **`Paycheck`**: an **`HourlyPaycheck`** class and **`SalariedPaycheck`** class that both adhere to the **`IPaycheck`** protocol. You are free to implement the concrete **`HourlyPaycheck`** and **`SalariedPaycheck`** any way you wish, but you must implement the contract as specified by the Java interface given above.

The **constructor signatures** for your Paycheck classes should conform to the following:

```
HourlyPaycheck(double payRate, double hoursWorked) throws IllegalArgumentException
```

```
SalariedPaycheck(double payRate, int payInterval) throws IllegalArgumentException
```

Additionally, take note of these special considerations:

`HourlyPaycheck` has a few more features than `SalariedPaycheck`:

- `addHoursWorked(double hoursAdded)` which takes a parameter (double) and adds the value of that parameter to the current number of hours worked this week. Note: for this system, it is legal to add "negative" hours for the week, however the total number of hours worked for the week cannot drop below 0.
- `resetHoursWorked()` that resets the Paycheck to zero hours for payment.
- `getHoursWorked()` that answers the number of hours an Hourly employee worked for the current paycheck

As with the previous version of this assignment, there should be a suitable `.toString()` method for Employees and Paychecks:

- Paychecks return a String representing the current payment AFTER taxes are assessed. The string representation should be in the form of US dollars with text as illustrated below:
  Payment after taxes: $ ###.##
- Employee returns a String, allowing Employee objects to be represented by the employee name, ID, and current week's payment after taxes are assessed.
  A visual representation of the String (if we were to print it) would be like this (note the space between the $ and the numeric value - you optionally might want to explore the java.text.DecimalFormat class to help your formatting):
  Name: Clark Kent
  ID: SUPS-111
  Payment after taxes: $ 416.50

We're using US dollars here. Also note that we don't pay employees in fractions of cents either. Our company policy is that if an employee earned more than 0 but less than 1 cent, we'll pay them the 1 cent.

As part of your submission, remember:

- Create JUnit tests to validate your solution.
- All of your solution classes should be placed in a package named `employee`. Note: Your test classes should **not** be placed in the `employee` package.

- Create a UML class diagram that describes your solution and include it with your submission

How to submit

1. You can upload your homework to Gradescope by create a zip file that contains the src and test directories (with the appropriate package directories as specified in this assignment)

   OR

2. You can submit to Gradescope via your GitHub account if you have one. As part of your academic honesty agreement, ensure your GitHub repo is **private**, not public!