# ICSI 213 Assignment 1 – Start the payroll system

**This assignment is extremely important – the next assignment after this one uses this one!**

**If you have bugs or missing features in this, you will need to fix them before you can continue on to new assignments. This is very typical in software development outside of school.**

**You must submit .java files. Any other file type will be ignored. Especially “.class” files.**

**You must not zip or otherwise compress your assignment. Blackboard will allow you to submit multiple files.**

***You must submit buildable .java files for credit.***

## Introduction

In this assignment, we are going to start working on a payroll system. We will build classes that represent the various ways different groups of employees might be paid. This will ensure that you have a solid understanding of classes, inheritance, and polymorphism.

## Details

### Employee

Create an abstract base class: Employee

In most companies, you are assigned an employee number, just like UAlbany has student numbers (starts with 00). We want to do the same thing. Create a private static integer on the Employee class for the next employee number. Create a private member integer for the current employee’s employee number. Create an accessor for the current employee’s employee number.

Every employee has a first name and a last name. Add private members for those. Create accessors and mutators for them.

Create a constructor for Employee that requires first and last name as parameters. Set the private first and last name members. Set the employee number and increment the next employee number.

Override “toString” so that the employee’s name and employee number are formatted:  
Id:3 - Phipps, Michael

Create a “GetPaycheck” abstract method. It takes no parameters and returns a float.

### Salaried Employee (SUBCLASS)

~~Create a class for SalariedEmployee that is a sub class of Employee.~~

Salaried employees are paid a set sum per year. Add a protected member for their yearly pay.

Implement GetPaycheck for salaried employees. Pay is calculated 26 times per year, so their paycheck should be 1/26 of their salary.

Implement accessors and mutators for salary.

Create a constructor for this class that sets names, and yearly salary.

Override ToString(). Use the base class’s ToString method and add to it:

Salaried, Base : $40,000; Id:3 - Phipps, Michael

### Hourly Employee (SUBCLASS)

~~Create a class for HourlyEmployee that is a subclass of Employee.~~

Hourly employees are paid by the fractional hour at a set rate. For example – an employee might have worked 25.5 hours and he might have a pay rate of $15.25.

Add two private members – hours worked this pay period and pay rate. Implement GetPaycheck for hourly employees (hours worked \* pay rate).

Implement accessors and mutators for these private members.

Create a constructor for this class that sets names and pay rate. Do not set hours worked.

Override ToString(). Use the base class’s ToString method and add to it:

Hourly: $15.25; Id:3 - Phipps, Michael

### Commission Employee (KIND OF LIKE HOURLY)

Commissions are more complex than salaried or hourly payment. Each employee has a commission schedule. A commission schedule is a 2D array. Each row holds 2 items – a minimum number of units sold and a value per unit.

Example: DON’T HARD CODE THIS AS A REF

|  |  |
| --- | --- |
| Minimum Units | Value Per Unit |
| 0.0 | 2.5 |
| 12.0 | 3.2 |
| 25.0 | 4.1 |
| 33.0 | 4.9 |
| 42.0 | 5.2 |

Each employee has a base pay rate (like salaried employees).

Each employee has a commission schedule and a number of units sold.

To calculate payroll, divide the base pay rate by 26 (like salaried employees).

Then find the highest value per unit where the number of units sold is greater than the minimum units.

Multiply units sold by the value per unit and add that to the base pay.

In our example above, if a person with this commission schedule sold 20 units, they would meet the 0.0 and 12.0 minimum unit qualifications. 12.0’s Value per Unit (3.2) is greater than 0.0’s Value per Unit (2.5), so we would multiple 20 units \* 3.2 and get $60.40. We would add that to the base pay divided by 26.

Create a CommissionedEmployee class that is a subclass of SalariedEmployee.

Add two private members – commission schedule and units sold.

Implement accessors and mutators for these private members.

Implement GetPaycheck using the algorithm described above.

Create a constructor for this class that sets names, commission schedule and base rate. Do not set units sold.

Override ToString(). Use the base class’s ToString method and add to it:

Commission: Base : $40,000; Id:3 - Phipps, Michael

Testing

In labs, we have been using junit to test the classes that we have built. You should design and built your own tests for this assignment so that you **know** that it works correctly.

BUILD YOU OWN TEST CASES

CAN EXCHANGE JUNIT TESTS WITH OTHERS BUT NOTHING ELSE

**HINTS**

Do not wait until the assignment is nearly due to begin. **Start early** so that you can ask questions.

**Test your work thoroughly before handing it in. Trade test cases with your friends to torture each other’s code.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Comments | None/Excessive (0) | “What” not “Why”, few (4) | Some “what” comments or missing some (7) | Anything not obvious has reasoning (10) |
| Variable/Function naming | Single letters everywhere (0) | Lots of abbreviations (4) | Full words most of the time (8) | Full words, descriptive (10) |
| Employee – class | None (0) |  |  | Class is public, in a filename that matches the class name and has appropriate inheritance (4) |
| Employee – max employee number | None (0) |  |  | Is static, is an integer, is incremented when a new employee is added (4) |
| Employee – ToString | None (0) |  |  | Formatted correctly (3) |
| Employee – GetPaycheck | None (0) |  |  | Is abstract with correct return type (4) |
| Employee – employee number | None (0) |  |  | Assigned in the constructor (4) |
| Employee – name | None (0) |  |  | Private, correct data type, accessors and mutators (4) |
| Salaried – class | None (0) |  |  | Class is public, in a filename that matches the class name and has appropriate inheritance (4) |
| Salaried – constructor | None (0) |  |  | Correct parameters and sets private members (4) |
| Salaried - toString | None (0) |  |  | Formatted correctly and calls super class (3) |
| Salaried – yearly pay | None (0) |  |  | Private, correct data type, accessors and mutators (4) |
| Salaried - GetPaycheck | None (0) |  |  | Calculates correctly(4) |
| Hourly – class | None (0) |  |  | Class is public, in a filename that matches the class name and has appropriate inheritance (4) |
| Hourly – constructor | None (0) |  |  | Correct parameters and sets private members (4) |
| Hourly - toString | None (0) |  |  | Formatted correctly and calls super class (3) |
| Hourly – pay values | None (0) |  |  | Private, correct data type, accessors and mutators (4) |
| Hourly - GetPaycheck | None (0) |  |  | Calculates correctly(4) |
| Commission– class | None (0) |  |  | Class is public, in a filename that matches the class name and has appropriate inheritance (4) |
| Commission– constructor | None (0) |  |  | Correct parameters and sets private members (4) |
| Commission- toString | None (0) |  |  | Formatted correctly and calls appropriate super class (3) |
| Commission– pay values | None (0) |  |  | Private, correct data type, accessors and mutators (4) |
| Commission- GetPaycheck | None (0) |  |  | Calculates correctly(4) |