UNIVERSITY OF ALBERTA

# AUCSC 112 LAB

## Assignment #3

**(Due before midnight on day before the next lab)**

### Goals:

- Become proficient at object-oriented design and programming in Java, by creating several Abstract Data Types (ADTs) with a hierarchical relationship.
- Understand that avoiding repetition is important in programming and how to use inheritance in that avoidance.
- Be able to readily create and use objects.

### Reference:

Heise, Chapter 3.

### Instructions:

For this week's program you may work as an individual, or a group of two. Do not share any lines of code between groups. If working in a group, each person must write the code together with the other person – do not split the work up and assign parts to each person. One person types, the other thinks, and then switch roles. Your partner must come from your lab section.

Document ALL of your code! Make sure that you add file headers (with your name(s), id #, date, and a brief summary). Also document each of the main steps in your solutions. Ensure that all your variables are named descriptively. All methods should have a header (i.e. documentation explaining the parameters and what each does – use Javadoc format). Avoid documentation that is simply a translation of Java code into English.

### The Program to Create:

You have been hired by Human Resources of Augustana Associated (HRAA) to provide the software to manage their students' and employees' information., including payroll, with the prospect of adding their student management system at a later date.

The main goal of this assignment is that you write the code to be able to print out an employee's paycheck receipt, including the amount of net pay and all deductions that have been made and a student's course report, including courses, marks, and grade point average.

### Classes

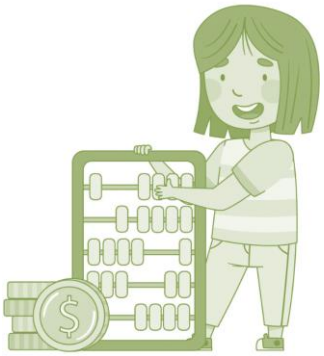
You must create the following ADTs, that is, classes, in appropriate hierarchical relationship:

1) **`Person`** – contains the data and methods common to all people, including all employees and all students. Do not allow instances of type `Person` to be created.
2) **`Employee`** – contains the data and methods common to all employees.
3) **`Student`** – contains the data and methods common to all students.
1) **`SalaryEmp`** – contains the data and methods for salaried employees. In particular, salaried employees get paid a fixed amount per month, no matter how many hours they work.
4) **`HourlyEmp`** – contains the data and methods for hourly employees. In particular, hourly employees get paid per hour worked, with the possibility of overtime pay.
5) **`Course`** – contains the data and methods for a course.

## Data

The following ***data fields = attributes*** should be present somewhere in your project and you may include extra data (but do not use a data field when you should use a local variable):

1) Each person, whether student, employee, salaried employee or hourly employee has a unique **ID number**, an integer, which should be generated by the program whenever a new person is created. ID numbers must differ by at least a value of 3, so for example, if one ID number is 5 then 3, 4, 6, and 7 cannot be ID numbers.
2) A static **ID number tracking field** to ensure unique ID numbers. All ID numbers must be unique and differ from all other ID numbers by 3, so you should track what your last ID number was and generate a new one. This can be done by using the "`static`" modifier on the ID number tracking field.
3) Each employee or student has a **name**, represented as a String.
4) Each employee or student has an **address**, represented as String
5) Each person has a **birthdate**, stored as three ints for day, month and year.
6) The number of **hours worked** by an employee in a month.
7) Every salaried employee has a **normal number of work hours** recorded for one month (189 hours at the moment), though they may work more or less.



8) The **maximum amount of CPP** (Canada Pension Plan) contribution that can be paid is $266.67 per month
9) The **maximum amount of EI** (Employment Insurance) premium that can be paid is $74.17 per month.
10) The **pay rate** for each employee, either as a yearly amount for salaried employees or as an hourly wage to hourly employees.
11) The **gross pay** the employee earned this month.
12) The **tax** the employee pays this month.
13) The **CPP contribution** the employee pays this month.
14) The **EI contribution** the employee pays this month.
15) The **net pay** the employee is paid this month.
16) A **course list**, with room for up to 12 courses per student, organized as an array.
17) Each course has an **identifier** (a String) such as "AUCSC 112",
18) Each course has a **final mark**.
19) Each student has a **GPA**.

## Methods

The following ***methods*** should be present somewhere in your project, though in as few locations as possible (that is, cover the requirement with inheritance wherever reasonable):

1) Constructors for each class, including, but not limited to:
   - For Salaried Employees:
     **`SalaryEmp`**(name, address, salary, birthDay, birthMonth, birthYear)
   - For Hourly Employees:
     **`HourlyEmp`**(name, address, hourlyRate, birthDay, birthMonth, birthYear)
   - For Students:
     **`Student`**(name, address, birthDay, birthMonth, birthYear)
   - For Courses:
     **`Course`**(name, mark)
   - Default contructors in every file. All default Strings should be "Test" along with the field name, for example, "Test Name" for a default name field. Default birthdays should be Jan. 1, 2000. Default marks should be 0%. Default pay rates should be $15.00 for hourly employees and $31,200.00 for salaried employees.

2) Getters for private or protected data, for example, **`getGrossPay()`**, **`getTax()`**, **`getCpp()`**, **`getEi()`**, **`getNetPay()`**.

3) **`toString()`** for every class.

4) **`setHoursWorked()`**—Sets the number of hours that an hourly employee has worked this month.

5) **`calcGrossPay()`** – A method that determines an employee's gross pay for one month. This method does not take any parameters and it returns a double that is the gross pay. Furthermore, the gross pay data field is set to this amount.
   - For Salaried Employees: the gross pay is the salary divided by 12.
   - For Hourly Employees: ensure that the hours worked for the month are set and then calculate gross pay based on these hours worked. The wage earned is the hours worked multiplied by the pay rate. However, if the hours worked are over 140, then there is overtime pay added. For the first 7 hours of overtime, the wage is at time-and-a-half, that is, (hours over 140 but under or equal to 147) * 1.5 * pay rate. For any additional hours, that is, for hours over 147, the wage is double-time, so multiply by 2.

6) **`calcTax()`** – This method calculates how much tax the employee must pay upfront. The tax rate is a percentage of the gross pay, with the percentage varying depending on the pay bracket of the employee. Here is the table of tax rates:

| Yearly Expected Salary | Tax Rate |
|---|---|
| < $14,000 | 0% |
| $14,000 - $49,999.99 | 15% |
| $50,000 - $99,999.99 | 21% |
| $100,000 - $149,999.99 | 26% |
| $150,000 + | 29% |


A salaried employee can be slotted into the appropriate tax rate easily, e.g. if the employee's salary is $65,000 in a year, the tax rate is 21%. Hourly employees are more difficult. Once their gross pay is calculated, multiply that number by 12 to find their tax bracket, e.g. if an employee's pay rate is $75.00 per hour and they worked 130 hours, then their gross pay is $9750 (and 12(9750) = $117,000) putting them at a tax rate of 26%. In a different month, if they only worked 100 hours, then their gross pay is $7500 (and 12(7500) = $90,000) and their tax rate is 21% for that month.

7) **`calcCpp()`** – Canada Pension Plan premium is 5.45% of gross pay, up to a maximum of $266.67 in one month. This method should return the cpp amount and set the data value.

8) **`calcEi()`** – Employment Insurance premium is 1.58% of gross pay, up to a maximum of $74.17 in one month. This method should return the EI amount and set the data value.

9) **`calcNetPay()`** – Net pay is simply gross pay with tax, CPP, and EI deducted. This method should return the net pay amount and set the data value.

10) **`printPayCheque()`** – This method prints out a pay cheque for one employee for one month. It contains successive calls to `calcGrossPay`, `calcTax`, `calcCpp`, `calcEi`, `calcNetPay` and then displays the information. If a salaried employee's record is being printed, `printPayCheque` takes no parameters. However, if an hourly

employee's record is being printed, then `printPayCheque` takes a parameter of the number of hours worked.

Here is a sample of making a salaried employee, and calling `printPayCheque()`:

```
SalaryEmp x = new SalaryEmp("Meagain Minkle", "23 Japer Ave\r\nRed "
                            + "Deer, AB", 13999.99, 4, 8, 1981);
x.printPayCheque();
```

And here is the output (ID numbers would vary):


```
================================================
1000008
Meagain Minkle
23 Japer Ave
Red Deer, AB
------------------------------------------------
   Hours worked: 189.00   Rate: 13999.99
------------------------------------------------
   Gross Pay:    1166.67
   Tax:             0.00
   CPP:            63.58
   EI:             18.43

   Net Pay:      1084.65
================================================
```

Here is a sample of making an hourly employee, and calling `printPayCheque(185.0)`:

```
HourlyEmp y = new HourlyEmp("Ashley Laben", "18 Parkview Dr\r\nOhaton,"
                            + " AB", 15.67, 5, 6, 2001);
y.printPayCheque(185.0);
```

And here is the output (ID numbers would vary):

```
================================================
1000030
Ashley Laben
18 Parkview Dr
Ohaton, AB
------------------------------------------------
   Hours worked: 185.00   Rate: 15.67
------------------------------------------------
   Gross Pay:    3549.26
   Tax:           532.39
   CPP:           193.43
   EI:             56.08

   Net Pay:      2767.35
================================================
```

11) **`setMark(newMark)`** – Sets the mark for a course.

12) **`addCourse(newCourseIdentifier, mark)`** – adds a course to a student, as long as there is room left. The mark for the course should be set to appropriately to the mark.

13) **`calcGpa()`** – calculate a student's GPA, sets it to the data field and returns the value.

14) **`printReport()`** – prints a student's report, all courses and their marks with a final message about GPA.
Here is a sample of code to create a Student:

```
Student z = new Student("Mildred Mouse", "Box 56\r\nEdmonton, AB",
                        3, 12, 2002);
z.addCourse("AUSCI 235", 4);
z.addCourse("AUMAT 116", 3.3);
z.printReport();
```

And here is the printed report:

```
================================================
1000010
Mildred Mouse
Box 56
Edmonton, AB
------------------------------------------------
   AUSCI 235:         4.00
   AUMAT 116:         3.30

          GPA:        3.65
================================================
```

Use `double` for any amounts of money, but be aware that it would not be considered accurate enough in real systems. Normally, there are standards for dealing with money (ISO 4217 implemented in Java as `BigDecimal`), but we are not using them, just to keep our project simpler. For our assignment, all money amounts should be doubles, rounded to 2 decimal places, when printed.

HRAA thanks you for your hard work!

### Testing:

Three testing files are provided:

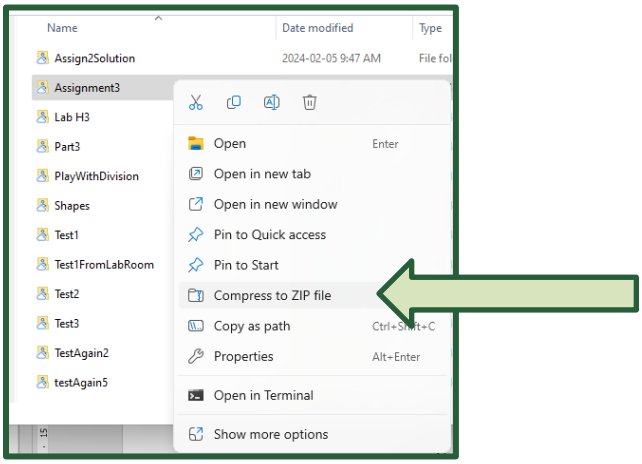
1) `SalaryEmpTest.java`
2) `HourlyEmpTest.java`
3) `StudentTest.java`

If you have forgotten how to add testing files to your project, please see the instructions under Part 3 of Assignment 1.

### How to Hand In Your Work:

If you worked with a partner, please only have one person submit the program files (1 – 5), and then both people submit the file in 6. If you worked as an individual, only submit the first 5 items.

1) A **`.zip`** file of your project. Do this by finding your folder, right-click, and select "Compress to ZIP file."




**2)** A **`.pdf`** of all of your Java files. You will need to make this manually. Open a new (plain) file in IntelliJ, and then cut and paste all your code into it, leaving 10 lines of space between files, in the following order:

a) your file with main
b) Person
c) Employee
d) Student
e) SalaryEmp
f) HourlyEmp
g) Course.

Print your file to pdf.

**3)** Only if you worked with a partner: a pdf of the group evaluation form (found on eClass or GitHub).

## Practice Questions – Assignment 3 Concepts:

These are self-check questions; do not hand in solutions. These questions are provided so that you can check whether you have learned the concepts we are expecting you to learn. No solutions are provided, and we encourage you to discuss these questions with other students.

**1)** How many hierarchies do you have in your project?
**2)** What does each hierarchy look like? Draw each hierarchy. (Go right up to `Object`.)
**3)** Did you minimize the amount of code you had to write?
**4)** Do you have any abstract classes? What are abstract classes used for?
**5)** Do you have any abstract methods? Which ones, and why?
**6)** Did you put any variables that should be local to a method, into your data fields?
**7)** Are all of your data fields either private or protected?
**8)** How do you access private data fields?
**9)** How many constructors did you make?
**10)** Can you identify the default constructors?

**11)** Did you use `this()` in your default constructors? If you did not, check your work again because you should have. Why?
**12)** Why are constructors public?
**13)** What does it mean for a data field to be static, inside of an abstract data type?
**14)** Do you have a `toString` method in each class? If not, why not?
**15)** Did you include any checks for whether numbers are reasonable? E.g. negative hours worked? Why are such checks important?
**16)** Explain why rounding to decimal places is not accurate enough for money.
**17)** A personal question: are you a programmer who is motivated by providing productivity software (e.g. this human resources system) or do you prefer to provide software for entertainment (e.g. the dice game)?
**18)** For what did you use your main method?
**19)** Comment on the modularity of your program – classes and methods.
**20)** Can you see encapsulation in your program?

Image Credits:
Watermelon: https://www.vecteezy.com/vector-art/23476534-watermelon-cartoon-mascot-or-character-carries-a-sack-of-money-a-paycheck-from-his-business
Salary: https://www.freepik.com/free-photos-vectors/number-60/2#uuid=017a1b6d-39e5-43f8-9ae8-99f554f6f183
Question Mark: https://www.freepik.com/free-photos-vectors/question-mark-cartoon
Hands in: https://www.shutterstock.com/search/cartoon-high-five-hand
Elephant: https://elements.envato.com/cute-elephant-employee-with-salary-cartoon-MHCRNJP?utm_source=graphicriver.net&utm_medium=promos&utm_campaign=elements_mkt-search-apiblock&utm_content=elements_api_block&_ga=2.60742574.2096482593.1707766950-1100610231.1707766950
Abacus: https://www.vectorstock.com/royalty-free-vector/little-girl-standing-near-dollar-coins-vector-37574175
Tax:
https://www.bing.com/images/search?view=detailV2&ccid=cn4aqzmb&id=540DA9F4C8145ABA96AB4EF9AA0EEBE3ABBB48E7&thid=OIP.cn4aqzmbEb_rwE_qOp3VngHaEK&mediaurl=https%3A%2F%2Fimages.livemint.com%2Fimg%2F2019%2F08%2F13%2F600x338%2Ftax_1562350731610_1565720251750.jpg&cdnurl=https%3A%2F%2Fth.bing.com%2Fth%2Fid%2FR.727e1aab399b11bfebc04fea3a9dd59e%3Frik%3D50i7q%252bPrDqr5Tg%26pid%3DImgRaw%26r%3D0&exph=337&expw=600&q=tax+cartoon+images&simid=608019403848510005&form=IRPRST&ck=D8E689B3C8A38DD40BDFC97BADB680C5&selectedindex=11&itb=0&ajaxhist=0&ajaxserp=0&pivotparams=insightsToken%3Dccid_12cv64v0*cp_1C445D382666716347077B490A3BCC93*mid_FE15250F049317557D04402B22742CDA63D86D83*simid_608008219766112067*thid_OIP.12cv64v02YuwqhToTPCn0QHaFG&vt=0&sim=11&iss=VSI&ajaxhist=0&ajaxserp=0