CS: 114 **OPPDA** **Fall, 2025**

Lab1 (50 points) **Due 2/17/2025**

Submit your solutions to canvas. Do not send the entire project. All that is needed are the files ending in .java. Each class will be defined in its own separate .java file. All driver code should be put in one .java file. Please make sure your name is included at the top of each .java file. Make one zip file that includes all the .java files and submit that one zip file to canvas.

Project Goals

1. Practice with polymorphism and polymorphic methods.
2. Practice implementing the DRY principle, and object decoupling.
3. Practice implementing abstract methods, and abstract classes.
4. Practice implementing concrete methods, and concrete classes.
5. Practice using `super`
6. Practice overriding `toString()`
7. Practice writing Javadoc complaint code

Problem 1

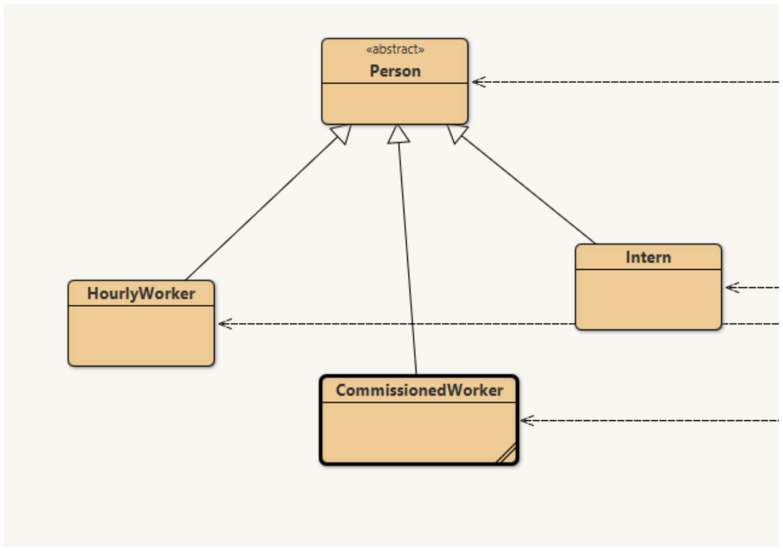
In Java you are to implement an employee payroll system. The Acme Gaming company consists of people. Each `Person` has an `id` and a `lastName`. The three types of employees are `HourlyWorker`, `CommissionedWorker`, and `Intern`. In addition to `id` and `lastName` an `HourlyWorker`, will keep track of the numbers of hours worked and the hourly pay rate. For this employee the `ComputeSalary()` method will compute the `regularPay` (the first 40 hour) as payRate * `hoursWorked`. For `overtimePay` (hours beyond 40) the pay rate will be time and half. The total pay will be the sum of the `regularPay` and the `overtimePay`.

In addition to `id` and `lastName` a `CommissionedWorker` will keep track of the total amount of sales, and the commission rate. The total pay will be total sales * commission rate. For this employee the `ComputeSalary()` method will compute the `totalPay` = `totalSales` * `commissionRate`.

In addition to `id` and `lastName` an Intern will keep track of the weekly pay amount for this employee the `ComputeSalary()` method will compute the total pay as simply the weekly pay amount.

It should be clear that `ComputeSalary()` is a polymorphic method. Use the DRY principle when designing your classes.

Your class diagram might look like in BlueJ:



Note that Person is an abstract class.

Two Person objects are considered equal if they have the same id. The natural ordering for a List of type Person is alphabetically by last name. The first lines of the driver should be:

```
List<Person> people = new ArrayList<>();
//List<Person> people = new LinkedList<>();
```

The driver should produce the same exact output if it is an `ArrayList` or a `LinkedList`. The one thing I do want you to add to the above declaration is the default capacity for the `ArrayList`. Examine the amount of data in the output and set the initial List capacity appropriately. You can only set the initial capacity for the `ArrayList`.

.

Create main code block that calls the driver P1 and produces this exact output below. The one exception is the shuffle will produce different orderings randomly each time you run your program. Use enhanced for-loops where you can. You do not need to turn in any UML diagrams, but your code should be Javadoc compliant. You may want to refer to slide 24 of lect4, and the lect4 example code for reference.

```
id                Name    Salary
101              Smith  $  563.75
123              Jones  $ 1000.00
120             Wilson  $  120.00
103           Williams  $ 1113.75
140             Decker  $ 3500.00
129              Brown  $  105.00
113             Miller  $ 1890.63
150              Davis  $ 3000.00
180              Adams  $  100.00
```

```
119          Murphy  $ 1906.25
Shuffle the List

 After shuffle
id              Name    Salary
180            Adams  $  100.00
119           Murphy  $ 1906.25
103         Williams  $ 1113.75
123            Jones  $ 1000.00
150            Davis  $ 3000.00
129            Brown  $  105.00
120           Wilson  $  120.00
101            Smith  $  563.75
113           Miller  $ 1890.63
140           Decker  $ 3500.00

Does List contain Davis : true
remove Davis : true
Does List contain Davis : false
remove Davis : false

indexOf Williams  is 2
use index 2 to get Williams reference variable and change
hours worked to 55 with a setter method.
Notice new salary 103 Williams 1265.625

First object in list is 180 Adams 100.0
Last object in list is 140 Decker 3500.0

Sort list by its natural ordering (alphabetically by last
name)

First object in list is 180 Adams 100.0
Last object in list is 120 Wilson 120.0

Iterate in order alphabetically by name
180 Adams 100.0
129 Brown 105.0
140 Decker 3500.0
123 Jones 1000.0
113 Miller 1890.625
119 Murphy 1906.25
101 Smith 563.75
103 Williams 1265.625
120 Wilson 120.0

Iterate in reverse order
```

```
120 Wilson 120.0
103 Williams 1265.625
101 Smith 563.75
119 Murphy 1906.25
113 Miller 1890.625
123 Jones 1000.0
140 Decker 3500.0
129 Brown 105.0
180 Adams 100.0
```