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

Project1 (100 points) **Due 2/3/2025**

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. For this project that file is on canvas and is called `Project1_driver.java` 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.

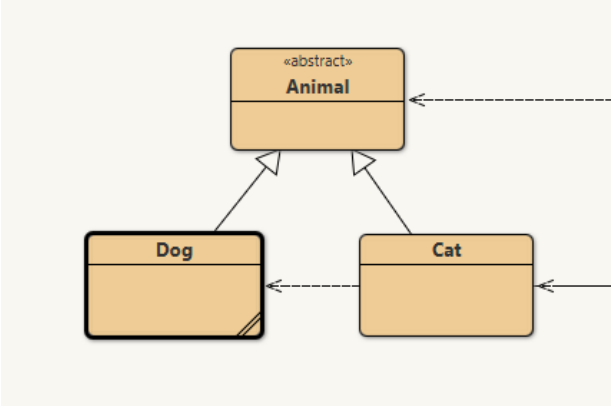
Project Goals

1. Practice with polymorphism and polymorphic methods.
2. Practice implementing the DRY principal, 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
8. Practice creating UML class diagrams and UML class relationships

Problem 1

Examine P1 in the Project1 driver code. Implement the Animal class. All animals will have `a name (String)` data member, and a `weight (double)` data member. The animal class will have two abstract methods. They are `Eat`, and `Sound`. Implement the `Dog`, `Cat`, and `Duck` classes which all inherit from the `Animal` class. Design your classes using the DRY principle. After you add the `Duck` class add one animal type of your choice. With your new Animal class be sure to include concrete methods for `Sound()` and `Eat()`, and also be sure to modify the driver code to instantiate objects of the new animal type. Be sure to implement all getters and setters for the `Animal` class.

Your class diagram might look like this in BlueJ:



Output should look something like this:

```
The pet's name is Rover
It weighs 12.0
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food

The pet's name is Felix
It weighs 7.0
Meow!, Meow!!
Eating Friskies Seafood Sensations cat food

The pet's name is Garfield
It weighs 6.5
Meow!, Meow!!
Eating Friskies Seafood Sensations cat food

The pet's name is Ren
It weighs 15.0
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food

The pet's name is Astro
It weighs 14.0
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food

----------------------------------------

After the pets have been eating the great pet food for a
month!!!
The pet's name is Rover
It weighs 12.5
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food

The pet's name is Felix
It weighs 7.5
Meow!, Meow!!
Eating Friskies Seafood Sensations cat food

The pet's name is Garfield
It weighs 7.0
Meow!, Meow!!
Eating Friskies Seafood Sensations cat food

The pet's name is Ren
```

```
It weighs 15.5
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food

The pet's name is Astro
It weighs 14.5
Bark!, Bark!!
Eating Purina Pro Plan High Protein Dog Food
```

Create a UML diagram for each class and show the class relations.

Problem 2

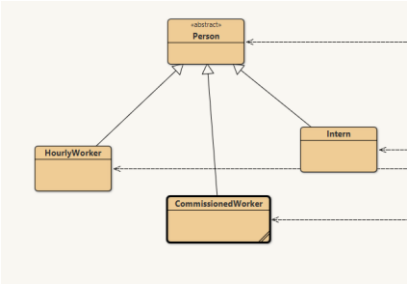
In Java you are to implement an employee payroll system. The Acme Gaming company consists of people. Each `Person` has an `id` and a `lastName`. The three types of employees are `HourlyWorker`, `CommissionedWorker`, and `Intern`. In addition to `id` and `lastName` an `HourlyWorker`, will keep track of 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.`

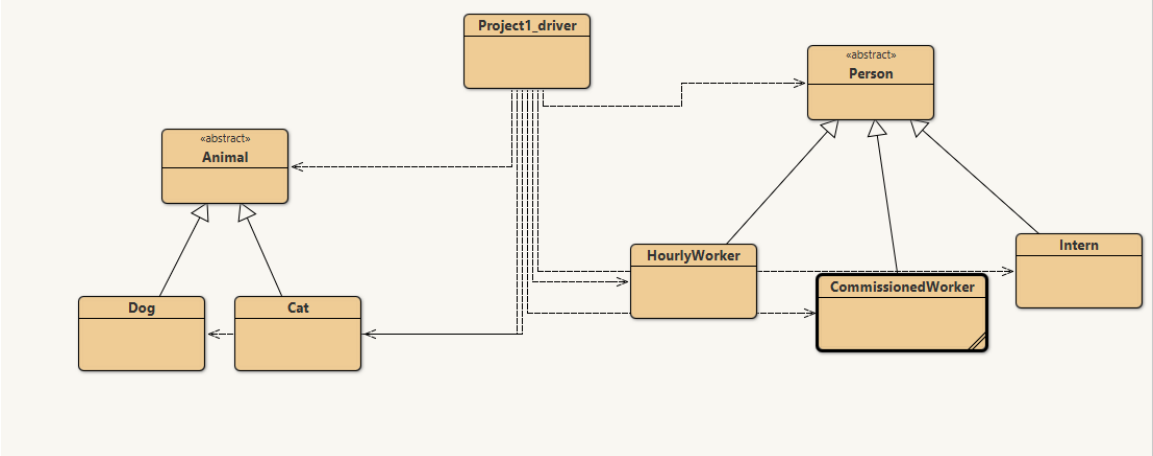
In addition to `id` and `lastName` a `CommissionedWorker` will keep track of the total amount of sales, and the commission rate. The total pay will be total sales * commission rate. For this employee the `ComputeSalary()` method will compute the `totalPay` = `totalSales` * `commissionRate`.

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.

There is one last thing you need to do so that both output loops in `P2()` work. You should override the `toString` method in the `Person` class. The implementation might look like this:

```
@Override
public String toString()
{
     return (this.id + " " + this.lastName);
}
```

You should also override in the other classes. For example, in the `HourlyWorker` class it might look like:

```
@Override
public String toString()
{
   return (super.toString() + " " + this.ComputeSalary());
}
```

Notice the use of `super` to invoke the `toString` method in the superclass `Person`.

Output from your `Project5_driver.java` should look like this:

```
101 Smith 563.75
123 Jones 1000.0
120 Wilson 120.0
103 Williams 1113.75
140 Decker 3500.0
129 Brown 105.0
113 Miller 1890.625
150 Davis 3000.0
180 Adams 100.0
119 Murphy 1906.25
```

```
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
```

UML Documentation

Generate a set of UML class diagrams for each class in Problem1. Include in this diagram the class relationship between each class if they exist. Do the same for Problem2.

Javadoc code compliance

Be sure that your java code is Javadoc compliant. **You do not have to create the Javadoc HTML files, but the instructor should be able to.** Javadoc is part of the JDK. It takes java source code and generates Java code documentation in HTML.

Javadoc comments are written with a special comment symbol:

```
/**
   This is a Javadoc comment
 */
```

There are several key tags that are used for formatting documentation. They are:

```
@version
@author
@param
@return
```

The documentation of a class should include:

- the class name
- a comment describing the overall purpose of and characteristics of a class
- a version number
- the author's name
- documentation for constructor and each method

Each method and constructor should include the following documentation:

- method name
- return type
- parameter names and types
- the purpose and function of the method
- a description of each parameter
- a description of the method return value

In BlueJ these documentation comments are automatically created when you invoke the New Class button. BlueJ allows several ways for you to generate the actual java doc html files.

VS Code offers a way to generate the Javadoc html files. Please visit:
https://marketplace.visualstudio.com/items?itemName=KeeganBruer.javadoc-generator

For additional Javadoc suggestions please visit:
https://www3.cs.stonybrook.edu/~cse214/sec01/Javadoc.htm