Polymorphism - Pairs Exercise

The purpose of this exercise is to provide you the opportunity to practice writing code using the principle of polymorphism.

Learning Objectives

After completing this exercise, students will understand:

- How to design and create super and sub-classes.
- How to use classes polymorphically.

Evaluation Criteria & Functional Requirements

- The project must not have any build errors.
- Appropriate variable names and data types are being used.
- Code is presented in a clean, organized format.
- The principle of polymorphism is being applied appropriately.
- The code meets the specifications defined below.
- The output for each program matches what is defined in the requirements.

In order to practice applying polymorphism, your task is to create class implementations for the exercises defined below. In any of the cases, you may add attributes (i.e. properties) and other supporting methods to the classes in order to fully implement the classes.

Worker Class

Attribute Name	Type	Get	Set
`firstName`	`string`	X	
`lastName`	`string`	X	

Method	Return Type
`calculateWeeklyPay(int hoursWorked)`	`double`

SalaryWorker

Create a `SalaryWorker` class is a `Worker` class.

Attribute Name	Type	Get	Set
`annualSalary`	`double`	X	

Constructor
`SalaryWorker(String firstName, String lastName, double annualSalary)`

A salaried employee "is-a" worker. hoursWorked expresses the number of hours for which the salaried employee worked in a given week. Since employee salaries are based on a 40 hour week, any hours above or below 40 are ignored and the following formula is used to calculate the weekly salary:

```
pay = annual salary / 52
```

HourlyWorker

Create an `HourlyWorker` class that is a `Worker` worker.

Attribute Name	Type	Get	Set
`hourlyRate`	`double`	X	

Constructor
`HourlyWorker(String firstName, String lastName, double hourlyRate)`

An hourly employee "is-a" worker whose hoursWorked express the number of hours the employee should be paid for.

```
pay = hourly rate * hoursWorked
overtime = hoursWorked - 40
pay = pay + (hourly rate * overtime * .5)
```

VolunteerWorker

Create a `VolunteerWorker` class that is a `Worker` class.

Constructor

`VolunteerWorker(String firstName, String lastName)`

A volunteer "is-a" worker whose hoursWorked express the number of hours volunteered. There is no monetary amount associated with volunteers, but the hours are recorded as pay.

```
pay = hoursWorked * 0
```

Program

Following the approach discussed in the lecture, create a List that represents a company's payroll and holds a collection of workers in it. The objective will be to:

- output each employee, the number of hours they've worked (you can use a random number generator), and the amount they will be paid for the week
- at the end show the sum of total hours worked and total weekly payroll

Expected Output

```
Employee             Hours Worked           Pay
=================================================
Mouse, Mickey        90                     $750.00
Geef, George (Goofy) 90                     $0.00
Duck, Daisy          110                    $750.00
Mouse, Minnie        20                     $2100.00

Total Hours: 310
Total Pay: $3,600.00
```

Getting Started

- Import the polymorphism-exercises project into Eclipse.
- Add code to the `src/main/java/com/techelevator` directory to complete the exercise.
- Verify your command line application is working as expected.

Tips and Tricks

- When writing your program, a question to ask yourself to ensure you are writing your code polymorphically is, "If a new requirement were provided, how much of my code will I need to change?" If the answer to this question is, "quite a bit", then you may have the opportunity to improve the quality of your code. Your code should be "open to extension, and closed for modification.open-closed-principle." In other words, what will happen if the Postal Service adds a new Economy delivery type? What would happen to the Toll Booth calculator if a new requirement were added for another type of vehicle?
- As you work on this assignment, you will need to verify that the application is working as expected for all of the scenarios outlined in the requirements. To do this, you will be practicing a form of testing referred to as "Manual Testing". Ideally, you will write automated unit tests to verify the applications you work on behave as expected. However, manual testing is equally important, and is a skill that is important on software development teams.
- Be sure you are verifying all of the execution paths for the application you are working on. For instance, verify that the postage calculator determines the correct rate when the distance is greater than 500 miles, and when the distance is less than 500 miles. Verify that the rates for each of the classes is being calculated correctly as well. Use the rate tables specified in the requirements for guidance.
