Lab 09: Inheritance and Polymorphism

Goals for this lab:

- ☐ Create a class that is inherited by another class
- ☐ Call a superclass's construction from a subclass's constructor
- ☐ Override a superclass's functions in a subclass
- ☐ Use polymorphism to allow the same function to be called by different classes
- ☐ Use `isinstance()` to distinguish between classes
- ☐ Write and execute Python programs using the given instructions

General Guidelines:

- ☐ Use meaningful variable names
- ☐ Use appropriate indentation
- ☐ Use comments, especially for the header, variables, and blocks of code.

Example Header:

```
# Author: Your name
# Date: Today's date
# Description: A small description in your own words that describes what
the program does. Any additional information required for the program to
execute should also be provided.
```

Example Function Docstring:

```
"""
General function description
:param p1: DESCRIPTION
:type p1: TYPE
:param p2: DESCRIPTION
:type p2: TYPE
:return: DESCRIPTION.
"""
```

1. Inheritance and Polymorphism

Inheritance and polymorphism allow software developers to reuse code in multiple different ways. By defining superclasses, subclasses can reuse functions and variables that have been previously defined, and override other functions that may need more specific functionality for a particular subclass. We just need to be careful about how data is stored and accessed when using data structures that store objects of multiple different classes.

For this exercise you will be writing a program to help a small business keep track of its employees. Specifically you will be inputting employee information, and then determining what the total amount of payroll the business will need to cover at the end of the week.

To complete this program:

- In a file named **Employee.py**, define a class named `Employee` which will store general employee data. The class should contain:
  - A protected member variable to store the employee's name
  - A protected member variable to store the employee's ID number
  - A protected member variable to store the employee's pay rate
  - A constructor that takes in a name, id, and pay rate as parameters, and uses them to set the class's member variables
  - A function named `calcPay`, which takes in the number of hours the employee worked and returns number of hours times their pay rate, representing their pay
  - Appropriate getter/setter methods for the member variables
- In a file named **Worker.py**, define a class named `Worker` which inherits `Employee` and will store information regarding a worker type employee. The class should contain:
  - A private member variable to store the worker's shift: 1 indicates the day shift, 2 indicates the night shift
  - A constructor that takes in a name, id, pay rate, and shift as parameters. The name, id, and pay rate should be provided to the superclass's constructor, and the shift should be used to set the `Worker` class's member variable
  - Appropriate getter/setter functions for the class variables
- In a file named **Supervisor.py**, define a class named `Supervisor` which inherits `Employee` and will store information regarding a supervisor type employee. The class should contain:
  - A private member variable to store the supervisor's level
  - A constructor that takes in a name, id, pay rate, and level as parameters. The name, id, and pay rate should be provided to the superclass's constructor, and the level should be used to set the `Supervisor` class's member variable
  - An overridden version of the `Employee` class's `calcPay` function, such that it calculates pay by multiplying the supervisor's pay rate by the number of hours worked and adds a bonus of 1000.00 multiplied by the supervisor's level
    - Be sure to use the protected member variable the `Supervisor` class has access to
  - Appropriate getter/setter functions for the class variables

- In a file named **hrManager.py**, define the functionality which will manage the employees of a small company. The file should contain:
  - A function named `calcTotalPay()`, which takes in one `List` containing both `Worker` and `Supervisor` objects and calculates and returns the total pay for all employees of the business
    - Be sure to use the appropriate function from the `Worker` and `Supervisor` objects to calculate an employee's pay
    - Assume each employee has worked 40 hours for the week
  - A function named `listEmployees()`, which in one `List` containing both `Worker` and `Supervisor` objects, returns nothing, and outputs all of the employee information with appropriate labels using a loop
    - Be sure to use `isinstance()` to output the appropriate information for the appropriate object type
    - For `Worker` objects, you should output whether they are working the day or night shift with an appropriate message
  - A `main` function, which should:
    - Create a single `List` to store both `Worker` and `Supervisor` objects
    - Prompt the user for how many employees they wish to add
    - For each employee, prompt the user if they wish to add a `Supervisor` or a `Worker`, and based on the decision:
      - Prompt the user for all of the information regarding a `Supervisor`, create a new `Supervisor` object using that information, and store that in the List of `Supervisors` and `Workers`
      - Prompt the user for all of the information regarding a `Worker`, create a new `Worker` object using that information, and store that in the List of `Supervisors` and `Workers`
      - Or, if the user did not select a `Supervisor` or `Worker`, inform the user that their input was incorrect and to try again
    - Call the `listEmployees()` function, and pass it the List of `Supervisors` and `Workers`
    - Call the `calcTotalPay()` function, pass it the List of `Supervisors` and `Workers`, and then output the total pay for all employees with an appropriate message

Complete the program, making sure to execute it to verify that it produces the correct results. Note that you will submit the **Employee.py, Worker.py**, **Supervisor.py**, and **hrManager.py** files to Canvas.

Submission

Once you have completed this lab it is time to turn in your work. Please submit the following files to the Lab 09 assignment in Canvas:

- **Employee.py, Worker.py**, **Supervisor.py**, and **hrManager.py**

Sample Output

Employee.py, Worker.py, Supervisor.py, and hrManager py

```
How many employees would you like to add: 3
Would you like to add a worker or a supervisor: super
super is not a worker or supervisor. Try again!

Would you like to add a worker or a supervisor: supervisor
Please enter the name of the supervisor: Sarah Roberts
Please enter the id of the supervisor: 12
Please enter the pay rate of the supervisor: 40.00
Please enter the level of the supervisor: 2

Would you like to add a worker or a supervisor: worker
Please enter the name of the worker: Paul Peterson
Please enter the id of the worker: 45
Please enter the pay rate of the worker: 20.00
Please enter the shift of the worker (1 for day, 2 for night): 1

Would you like to add a worker or a supervisor: worker
Please enter the name of the worker: Tanah AlSamri
Please enter the id of the worker: 36
Please enter the pay rate of the worker: 25.50
Please enter the shift of the worker (1 for day, 2 for night): 2

Name: Sarah Roberts
ID: 12
Pay Rate: $40.00
Level: 2
Name: Paul Peterson
ID: 45
Pay Rate: $20.00
Shift: Day Shift
Name: Tanah AlSamri
ID: 36
Pay Rate: $25.50
Shift: Night Shift
The total cost of all of the worker's pay is $5420.00
```

Rubric

For each program in this lab, you are expected to make a good faith effort on all requested functionality. Each submitted program will receive a score of either 100, 75, 50, or 0 out of 100 based upon how much of the program you attempted, regardless of whether the final output is correct (See table below). The scores for all of your submitted programs for this lab will be averaged together to calculate your grade for this lab. Keep in mind that labs are practice both for the exams in this course and for coding professionally, so make sure you take the assignments seriously.

Score	**Attempted Functionality**
100	100% of the functionality was attempted
75	<100% and >=75% of the functionality was attempted
50	<75% and >=50% of the functionality was attempted
0	<50% of the functionality was attempted