**UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS**

**Lab 7: Inheritance**

**Lab 8: Multi-Level Inheritance**

**Lab 9: Multiple Inheritance**

**Lab Title:** **EE-271** “**OOP & Data Structures Lab**”

Time: 20 min/ Task

**Lab report task:**

**Problem 1:**

1. Take the example of the rectangle class in the last lab and inherit a square class from the rectangle class.
2. Create two instances of the square class and call the instance methods.

**Lab Work Task:**

**Problem 1:**

1. Take the example of the rectangle class in the last lab and inherit a square class from the rectangle class.
2. Create two instances of the square class and call the instance methods.

**Problem 2:**

1. Take the example of the square class in the last lab and inherit a cube class with its own volume and surface area instance method that utilizes the base class method effectively.
2. Create two instances of the cube class and call both the instance methods. Verify manually or with an online calculator.

**Problem 3:**

1. Take the example of the circle class in the last lab and inherit a sphere class with its own volume and surface area instance method.
2. Create two instances of the sphere class and call both the instance methods.

**Problem 4:**

1. Take the example of the Point2D class in the last lab and inherit a Point3D class with its appropriate instance methods.
2. Create two instances and call the instance methods.

**Problem 5:**

The **HR system** needs to process payroll for the company’s employees, but there are different types of employees depending on how their payroll is calculated.

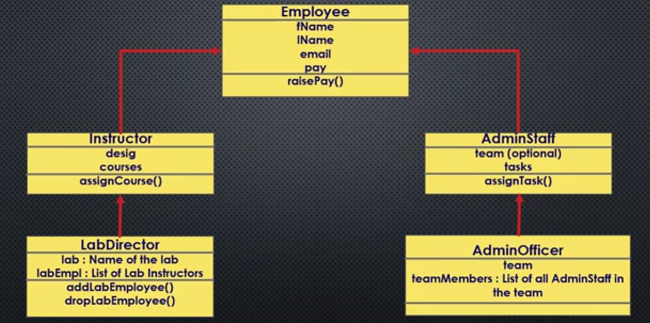
1. Implement a base class, Employee, that handles the common interface for every employee.
2. Administrative workers have a fixed salary, so every week they get paid the same amount.
   1. You create a derived class, SalaryEmployee that inherits from Employee. The class initializes with the .id and .name required by the base class, and you use super() to initialize the members of the base class.
   2. SalaryEmployee also requires a weekly\_salary initialization parameter that represents the amount that the employee makes per week.
   3. The class provides the required .calculate\_payroll() method that the HR system uses. The implementation just returns the amount stored in weekly\_salary.
3. The company also employs manufacturing workers who are paid by the hour, so you add HourlyEmployee to the HR system
   1. You create a derived class, HourlyEmployee, that inherits from Employee. The class initializes with the .id and .name required by the base class, and you use super() to initialize the members of the base class.
   2. HourlyEmployee also requires hours\_worked and the hourly\_rate required to calculate the payroll.
   3. The class provides the required .calculate\_payroll() method by returning the hours worked times the hourly rate.
4. the company employs sales associates who are paid a fixed salary plus a commission based on their sales, so you create a CommissionEmployee class
   1. Derive CommissionEmployee from SalaryEmployee because both classes have a weekly\_salary to consider. At the same time, you initialize CommissionEmployee with a commission value that’s based on the sales for the employee.
   2. In .calculate\_payroll(), you leverage the implementation of the base class to retrieve the fixed salary, and you add the commission value.
5. Implementing a PayrollSystem class that processes payroll.
   1. PayrollSystem implements a .calculate\_payroll() method that takes a collection of employees and [prints](https://realpython.com/python-print/) their .id, .name, and check amount using the .calculate\_payroll() method exposed on each employee object.

Task: creates the employees and passes them to the payroll system to process payroll

**Problem 6:**

Design a class for the employees of the electrical department.

1. Base class on the name of staff.
2. There are admin and faculty staff.
3. The faculty contains Lecturers, Assistant professors, Associate Professors, and Professors.
4. The admin staff contains Lab Engineers, Lab technicians, Lab assistants, and lab attendants.



**LAB 8**

**Lab Report Task:**

**Problem 1:**

1. Construct a rectangle class
2. Inherit a square class from the rectangle class.
3. Create a class Cube that inherits from Square and extends the functionality of .area () (inherited from the Rectangle class through Square) to calculate the surface area and volume of Cube.
4. Create two instances of the cube class and call both the instance methods.

**Lab work Task:**

**Problem 1:**

1. Construct a rectangle class
2. Inherit a square class from the rectangle class.
3. Create a class Cube that inherits from Square and extends the functionality of .area () (inherited from the Rectangle class through Square) to calculate the surface area and volume of Cube.
4. Create two instances of the cube class and call both the instance methods.

**Problem 2:**

1. Start adding the new classes to the existing class.

ProductivitySystem tracks productivity based on employee roles. There are different employee roles:

* **Managers:** They walk around yelling at people, telling them what to do. They’re salaried employees and make more money.
* **Secretaries:** They do all the paperwork for managers and ensure that everything gets billed and paid on time. They’re also salaried employees but make less money.
* **Sales employees:** They make a lot of phone calls to sell products. They have a salary, but they also get commissions for sales.
* **Factory workers:** They manufacture the products for the company. They’re paid by the hour.

First, you add a Manager class that derives from SalaryEmployee. The class exposes a .work () method that the productivity system will use. The method takes the hours that the employee worked.

Then you add the Secretary, SalesPerson, and FactoryWorker and then implement the .work () interface, so they can be used by the productivity system

**Problem 3:**

Design a class for the employees of the electrical department.

1. Base class on the name of staff.
2. There are admin and faculty staff.
3. The faculty contains Lecturers, Assistant professors, Associate Professors, and Professors.
4. The admin staff contains Lab Engineers, Lab technicians, Lab assistants, and lab attendants.

**Lab 9**

**Problem 1:**

1. It turns out that sometimes temporary secretaries are hired when there’s too much paperwork to do. The TemporarySecretary class performs the role of a Secretary in the context of the ProductivitySystem, but for payroll purposes, it’s an HourlyEmployee.
   1. Derive it from both Secretary and HourlyEmployee:

|  |
| --- |
| class TemporarySecretary(Secretary, HourlyEmployee):  pass |

Run the code

temporary\_secretary = TemporarySecretary(5, "Robin Williams", 40, 9)

Understand the code and error and check TemporarySecretary.\_\_mro\_\_

* + 1. Modify the above class as

|  |
| --- |
| class TemporarySecretary(HourlyEmployee, Secretary):  pass |

Run the code

temporary\_secretary = TemporarySecretary(5, "Robin Williams", 40, 9)

Understand the code and error and check TemporarySecretary.\_\_mro\_\_

* + 1. Modify the above class as

|  |
| --- |
| class TemporarySecretary(HourlyEmployee, Secretary):  def \_\_init\_\_(self, id, name, hours\_worked, hourly\_rate):  super().\_\_init\_\_(id, name, hours\_worked, hourly\_rate) |

Run the code

temporary\_secretary = TemporarySecretary(5, "Robin Williams", 40, 9)

Understand the code and error and check TemporarySecretary.\_\_mro\_\_

* + 1. Modify the above class as

|  |
| --- |
| class TemporarySecretary(Secretary, HourlyEmployee):  def \_\_init\_\_(self, id, name, hours\_worked, hourly\_rate):  HourlyEmployee.\_\_init\_\_(self, id, name, hours\_worked, hourly\_rate)) |

Run the code

|  |
| --- |
| temporary\_secretary = TemporarySecretary(5, "Robin Williams", 40, 9)  company\_employees = [temporary\_secretary]  productivity\_system = productivity.ProductivitySystem()  productivity\_system.track(company\_employees, 40)  payroll\_system = hr.PayrollSystem()  payroll\_system.calculate\_payroll(company\_employees) |

Understand the code and error and check TemporarySecretary.\_\_mro\_\_

* + 1. Modify the above class as

|  |
| --- |
| class TemporarySecretary(Secretary, HourlyEmployee):  def \_\_init\_\_(self, id, name, hours\_worked, hourly\_rate):  HourlyEmployee.\_\_init\_\_(self, id, name, hours\_worked, hourly\_rate)  def calculate\_payroll(self):  return HourlyEmployee.calculate\_payroll(self) |

Run the code

|  |
| --- |
| temporary\_secretary = TemporarySecretary(5, "Robin Williams", 40, 9)  company\_employees = [temporary\_secretary]  productivity\_system = productivity.ProductivitySystem()  productivity\_system.track(company\_employees, 40)  payroll\_system = hr.PayrollSystem()  payroll\_system.calculate\_payroll(company\_employees) |

Understand the code and error and check TemporarySecretary.\_\_mro\_

**Problem 2:**

**Understand the following code with the help of the concepts of multiple Inheritance in Python**

class Rectangle:

    def \_\_init\_\_(self, length, width, \*\*kwargs):

        self.length = length

        self.width = width

        super().\_\_init\_\_(\*\*kwargs)

    def area(self):

        return self.length \* self.width

    def perimeter(self):

        return 2 \* self.length + 2 \* self.width

class Square(Rectangle):

    def \_\_init\_\_(self, length, \*\*kwargs):

        super().\_\_init\_\_(length=length, width=length, \*\*kwargs)

class Triangle:

    def \_\_init\_\_(self, base, height, \*\*kwargs):

        self.base = base

        self.height = height

        super().\_\_init\_\_(\*\*kwargs)

    def tri\_area(self):

        return 0.5 \* self.base \* self.height

class RightPyramid(Square, Triangle):

    def \_\_init\_\_(self, base, slant\_height, \*\*kwargs):

        self.base = base

        self.slant\_height = slant\_height

        kwargs["height"] = slant\_height

        kwargs["length"] = base

        super().\_\_init\_\_(base=base, \*\*kwargs)

    def area(self):

        base\_area = super().area()

        perimeter = super().perimeter()

        return 0.5 \* perimeter \* self.slant\_height + base\_area

    def area\_2(self):

        base\_area = super().area()

        triangle\_area = super().tri\_area()

        return triangle\_area \* 4 + base\_area

class SurfaceAreaMixin:

    def surface\_area(self):

        surface\_area = 0

        for surface in self.surfaces:

            surface\_area += surface.area(self)

        return surface\_area

class Cube(Square, SurfaceAreaMixin):

    def \_\_init\_\_(self, length):

        super().\_\_init\_\_(length)

        self.surfaces = [Square, Square, Square, Square, Square, Square]

class RightPyramid(Square, Triangle, SurfaceAreaMixin):

    def \_\_init\_\_(self, base, slant\_height):

        self.base = base

        self.slant\_height = slant\_height

        self.height = slant\_height

        self.length = base

        self.width = base

        self.surfaces = [Square, Triangle, Triangle, Triangle, Triangle]

* 1. Run the following code and track the code flow.

cube = Cube(3)

cube.surface\_area()

Note: The **Method Resolution Order** (MRO) determines where Python looks for a method when there is a hierarchy of classes. Using super() accesses the next class in the MRO:

**Problem 3:**

Design a class for the employees of the electrical department.

1. Base class on the name of staff.
2. There are admin and faculty staff.
3. The faculty contains Lecturers, Assistant professors, Associate Professors, and Professors.
4. The admin staff contains Lab Engineers, Lab technicians, Lab assistants, and lab attendants.
5. Chairman and semester coordinator have both roles.