UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, JALOZAI CAMPUS

Lab 8: Multilevel Multiple Inheritance

Lab Title: EE-271 "OOP & Data Structures Lab"

Time: 10 min/ Task

Multilevel Inheritance in Python is a type of Inheritance that involves inheriting a class that has already inherited some other class.
This is a simple relationship between a child and a grandfather.

- Syntax

```
class base:
  // Members and Functions
  Pass
class derived1(base):
  // Members and functions of both base and derived1 classes
  pass
class derived2(derived1):
  // Members and functions of base class, derived1 class,
and derived2 class.
  Pass
```

Lab report Task

1. Define 3 objects of different classes in task 2 and run different methods on them.

Lab Work tasks

1. Consider the following classes.

```
class Manager:
  def final_review(self):
    print("Final Review")
class Reviewer(Manager):
  def review(self):
    print("Reviewing...")
class Writer(Reviewer):
  def writes(self):
    print("Writes the code")
```

a. Run the following code and observe.

```
o = Writer()
o.final_review()
o.review()
o.writes()
```

b. Run the following code and observe.

```
or1 = Reviewer()
```

```
or1.review()
```

c. Run the following code and observe.

```
or1.final_review()
```

d. Run the following code and observe.

```
or1.writes()
```

1. 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]
```

a. 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:

Reading Section

a. https://realpython.com/lessons/multiple-inheritance-python/
b. https://www.codingninjas.com/studio/library/multilevel-inheritance-in-python
c.