...

Unit 2: Design Principles

Session 3

Lesson Outcome:

Understanding Inheritance, Encapsulation and Polymorphism in Object Oriented Programming

Object Oriented Design

Topic 1: Inheritance

Inheritance

- Inheritance is a way of creating a new class by using details of an existing class without modifying it.
- The newly formed class is called a child class. Similarly, the existing class is a base class (or parent class).

Inheritance

```
[ ]  # parent class
     class Bird:

         def __init__(self):
             print("Bird is ready")

         def whoisThis(self):
             print("Bird")

         def swim(self):
             print("Swim faster")

     # child class
     class Penguin(Bird):

         def __init__(self):
             # call super() function
             super().__init__()
             print("Penguin is ready")

         def whoisThis(self):
             print("Penguin")

         def run(self):
             print("Run faster")

     peggy = Penguin()
     peggy.whoisThis()
     peggy.swim()
     peggy.run()
```

Inheritance

- In this example, we create two classes i.e. Bird (parent class) and Penguin (child class).
- The child class inherits the functions of parent class. We can see this from the swim() method.

```
[ ]  # parent class
     class Bird:

         def __init__(self):
             print("Bird is ready")

         def whoisThis(self):
             print("Bird")

         def swim(self):
             print("Swim faster")

     # child class
     class Penguin(Bird):

         def __init__(self):
             # call super() function
             super().__init__()
             print("Penguin is ready")

         def whoisThis(self):
             print("Penguin")

         def run(self):
             print("Run faster")

     peggy = Penguin()
     peggy.whoisThis()
     peggy.swim()
     peggy.run()
```

Inheritance

- Again, the child class modified the behavior of the parent class.
- We can see this from the whoisThis() method.
- Furthermore, we extend the functions of the parent class, by creating a new run() method.

```
[ ]  # parent class
     class Bird:

         def __init__(self):
             print("Bird is ready")

         def whoisThis(self):
             print("Bird")

         def swim(self):
             print("Swim faster")

     # child class
     class Penguin(Bird):

         def __init__(self):
             # call super() function
             super().__init__()
             print("Penguin is ready")

         def whoisThis(self):
             print("Penguin")

         def run(self):
             print("Run faster")

     peggy = Penguin()
     peggy.whoisThis()
     peggy.swim()
     peggy.run()
```

Inheritance

- Additionally, we use the super() function inside the __init__() method. This allows us to run the __init__() method of the parent class inside the child class.

```
[ ]  # parent class
     class Bird:

         def __init__(self):
             print("Bird is ready")

         def whoisThis(self):
             print("Bird")

         def swim(self):
             print("Swim faster")

     # child class
     class Penguin(Bird):

         def __init__(self):
             # call super() function
             super().__init__()
             print("Penguin is ready")

         def whoisThis(self):
             print("Penguin")

         def run(self):
             print("Run faster")

     peggy = Penguin()
     peggy.whoisThis()
     peggy.swim()
     peggy.run()
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Topic 2: Encapsulation

Encapsulation

- Here, we define a Computer class.
- We used __init__() method to store the maximum selling price of Computer.
- What do you notice in the code?
- What do you see in the output?

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Encapsulation

Here, we have tried to modify the value of __maxprice outside of the class.

However, since __maxprice is a private variable, this modification is not seen on the output.

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Encapsulation

- To change the value, we have to use a setter function i.e setMaxPrice() which takes price as a parameter.

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

Encapsulation

- What do you think the point of Encapsulation is?
- What type of program do you think needs encapsulation?

```python
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Topic 3: Polymorphism

Polymorphism

- Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).
- Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle).
- However we could use the same method to color any shape. This concept is called Polymorphism.

Example

```
[ ]  class Parrot:

        def fly(self):
            print("Parrot can fly")

        def swim(self):
            print("Parrot can't swim")

    class Penguin:

        def fly(self):
            print("Penguin can't fly")

        def swim(self):
            print("Penguin can swim")

    # common interface
    def flying_test(bird):
        bird.fly()

    #instantiate objects
    blu = Parrot()
    peggy = Penguin()

    # passing the object
    flying_test(blu)
    flying_test(peggy)
```

Polymorphism

- We define two classes Parrot and Penguin.
- Each of them have a common fly() method. However, their functions are different.

```
[ ]  class Parrot:

        def fly(self):
            print("Parrot can fly")

        def swim(self):
            print("Parrot can't swim")

    class Penguin:

        def fly(self):
            print("Penguin can't fly")

        def swim(self):
            print("Penguin can swim")

    # common interface
    def flying_test(bird):
        bird.fly()

    #instantiate objects
    blu = Parrot()
    peggy = Penguin()

    # passing the object
    flying_test(blu)
    flying_test(peggy)
```

Polymorphism

- To use polymorphism, we created a common interface i.e flying_test() function that takes any object and calls the object's fly() method.
- Thus, when we passed the blu and peggy objects in the flying_test() function, it ran effectively.

```
[ ]  class Parrot:

         def fly(self):
             print("Parrot can fly")

         def swim(self):
             print("Parrot can't swim")

     class Penguin:

         def fly(self):
             print("Penguin can't fly")

         def swim(self):
             print("Penguin can swim")

     # common interface
     def flying_test(bird):
         bird.fly()

     #instantiate objects
     blu = Parrot()
     peggy = Penguin()

     # passing the object
     flying_test(blu)
     flying_test(peggy)
```

Polymorphism

- What is the point of polymorphism?
- Why is it needed?
- How does it help?

```
[ ]  class Parrot:

         def fly(self):
             print("Parrot can fly")

         def swim(self):
             print("Parrot can't swim")

     class Penguin:

         def fly(self):
             print("Penguin can't fly")

         def swim(self):
             print("Penguin can swim")

     # common interface
     def flying_test(bird):
         bird.fly()

     #instantiate objects
     blu = Parrot()
     peggy = Penguin()

     # passing the object
     flying_test(blu)
     flying_test(peggy)
```

How are you feeling?

RED
I have no idea what you're talking about

YELLOW
I have some questions but feel like I understand some things

GREEN
I feel comfortable with everything you've said

Session 4

Lesson Outcome:

Practice Inheritance in Object Oriented Programming

Task:

With help from the following examples, implement Inheritance in your code.

Inheritance Recap

- Inheritance is a powerful feature in object oriented programming.
- It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Inheritance Example

- Recall: the derived class inherits features from the base class where new features can be added to it.
- This results in re-usability of code.

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Inheritance


```python
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

Inheritance

A polygon is a closed figure with 3 or more sides. Say,

we have a class called Polygon defined as follows:

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

Inheritance

The inputSides() method takes in the magnitude of each side and dispSides() displays these side lengths.

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

Inheritance

A triangle is a polygon with 3 sides. So, we can create a class called Triangle which inherits from Polygon. This makes all the attributes of Polygon class available to the Triangle class.

```
[ ]  class Triangle(Polygon):
         def __init__(self):
             Polygon.__init__(self,3)

         def findArea(self):
             a, b, c = self.sides
             # calculate the semi-perimeter
             s = (a + b + c) / 2
             area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
             print('The area of the triangle is %0.2f' %area)
```

Inheritance

However, class Triangle has a new method findArea() to find and print the area of the triangle.

```
[ ]  class Triangle(Polygon):
         def __init__(self):
             Polygon.__init__(self,3)

         def findArea(self):
             a, b, c = self.sides
             # calculate the semi-perimeter
             s = (a + b + c) / 2
             area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
             print('The area of the triangle is %0.2f' %area)
```

Inheritance

We can see that even though we did not define methods like inputSides() or dispSides() for class Triangle separately, we were able to use them.

If an attribute is not found in the class itself, the search continues to the base class. This repeats recursively, if the base class is itself derived from other classes.

Inheritance

Try it out for yourself!

Create a child class from the class you made last week

Thank You