Lecture 05: Inheritance and More

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

February 18, 2024

Relationship Between Classes:

is-kind-of,is-type-of,is-a

Inheritance

Let A and B be classes. To determine if A should be the parent class of B ask: Is B an A (or is B a type of A, or is B a kind of A)? If the answer is yes, then B is in is-a relationship to A.

1. **Example 1:** Manager is a kind of Employee.
2. **Example 2:** CreditCard Account is a type of BankAccount.
3. **Example 3:** Rectangle is a type of Shape2D.

Relationship Between Classes:

is-analogous-to

Generalization

If class X is-analogous-to class Y then look for superclass

1. **Example 1:** Rectangle and Triangle. They are same type, analogous. So look for a super type: Shape2D.
2. **Example 2:** Savings Account and Checking Account: both are analogous. Look for a parent type: BankAccount.

Relationship Between Classes:

is-part-of

No Inheritance

If class A is-part-of class B then there is no inheritance. Some negotiation between A and B for responsibilities may be needed

1. **Example 1:** A Student has many courses registered in a trimester. Registration is a class with many courses. Each course has got some different information, .i.e., title, id, credit hours, sections.
2. **Example 2:** A Direction class has Point has parts. Both are separate classes.

Inheritance

- **Inheritance** is a way to form new classes using classes that have already been defined.
- The new classes, known as **derived classes**, take over (or inherit) attributes and behavior of the pre-existing classes, which are referred to as **base classes** (or **ancestor classes**).
- It is intended to help reuse existing code with little or no modification.
- A class can be extended or subclassed
- The class that is extended is a super class / parent class.
- Some people use the words parent class or base class to mean a **super class**
- The extended class is a **sub class** or extended class or child class of its super class.
- An object created from the sub class has its own copy of all the non-static fields defined in its super class

Basic Inheritance

```
class A:
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    def printMe(self):
        print(f"x:{self.x},y:{self.y}")

class B(A):
    pass

b = B(10,15)
b.printMe()
```

Basic Inheritance

```
class Rectangle:
    def __init__(self,w=0,h=0):
        self.width=w
        self.height=h
    def area(self):
        return self.width*self.height

class ColoredRectangle(Rectangle):
    def __init__(self,w=0,h=0,c=""):
        super().__init__(w,h)
        self.color=c
    def periphery(self):
        return 2*(self.width+self.height)
```

Basic Inheritance

```
c = ColoredRectangle(8,4,"blue")
print("Area:",c.area())
print("Periphery:",c.periphery())
print("Color",c.color)
print("Height",c.height)
```

```
Area: 32
Periphery: 24
Color blue
Height 4
```

Method Overriding

```
class Rectangle:
    def __init__(self,w=0,h=0):
        self.width=w
        self.height=h
    def asString(self):
        return f"width: {self.width}, height: {self.height}"
class ColoredRectangle(Rectangle):
    def __init__(self,w=0,h=0,c=""):
        super().__init__(w,h)
        self.color=c
    def asString(self):
        return f"""width: {self.width}, height: {self.height}
color: {self.color}"""
c = ColoredRectangle(8,4,"blue")
print(c.asString())
```

Another use of super()

```
class Rectangle:
    def __init__(self,w=0,h=0):
        self.width=w
        self.height=h
    def asString(self):
        return f"width: {self.width}, height: {self.height}"
class ColoredRectangle(Rectangle):
    def __init__(self,w=0,h=0,c=""):
        super().__init__(w,h)
        self.color=c
    def asString(self):
        return super().asString()+f", color: {self.color}"
c = ColoredRectangle(8,4,"blue")
print(c.asString())
```

if super().__int__() is not called?

```
class A:
    def __init__(self,x=0):
        self.x=x
    def printMe(self):
        print(self.x)
class B(A):
    def __init__(self,y):
        self.y=y
b = B(100)
b.printMe()
```

```
AttributeError: 'B' object has no attribute 'x'
```

- When you add the __init__() function, the child class will no longer inherit the parent's __init__() function.

Access to attributes!

- All methods and attributes on a class are publicly available.
- Name mangling: to strongly suggest that outside objects don't access a property or method: prefix it with a double underscore.
- When we use a double underscore, the property is prefixed with _<classname>

```
class A:
    def __init__(self):
        self.__x=15
    def printMe(self):
        print(self.__x)
a = A()
a.printMe()
#print(a.__x) - this is not valid
print(a._A__x)
```

Access to attributes

- Any child class will have similar access to attributes with double underscores as prefix.

```
class A:
    def __init__(self,x=0):
        self.__x=x
class B(A):
    def __init__(self,x):
        super().__init__(x)
    def change(self,n):
        self._A__x=n
b = B(15)
b.change(25)
print(b._A__x)
```

Class Variable

```
class A:
    counter = 0
    def __init__(self,x=0):
        self.x=x
        A.counter = A.counter + 1
a1 = A()
a2 = A()
a3 = A()

print(a1.counter)
print(a2.counter)
print(a3.counter)
print(A.counter)
```

Class Variable - inheritance

```
class A:
    counter = 0
    def __init__(self,x=0):
        self.x=x
        A.counter = A.counter + 1
class B(A):
    def __init__(self,x=0,y=0):
        self.y=y
        super().__init__(x)
b1=B()
b2=B()
print(B.counter)
print(A.counter)
print(b1.counter)
print(b2.counter)
```