CSC148-Section:L0301
Week#2-Wednesday

Instructed by
AbdulAziz Al-Helali
a.alhelali@mail.utoronto.ca
Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap and Jacqueline Smith slides winter17
Outline

• Inheritance vs Composition
  • Review Father/Son example
  • Designing Square and RightAngleTriangle without inheritance
• Using inheritance to build
  • Shape – Super class
  • Square and RightAngleTriangle – as subclasses
Intro: Inheritance vs Composition

Composition
• Making use of other data types or objects of other classes

Inheritance
• A subclass inherits all attributes and methods (behavior) from superclass. Why?
  • to reuse code of existing class
• A subclass can extend, overload attributes and methods for a superclass
• Subclass can be called child class
• Supper class can be called parent class
Example:

class Father:
    x: int = 10
    y: int = 20

def m1(self) -> None:
    print('Father m1')

def m2(self) -> None:
    print('Father m2')

class Son(Father):
    z: int = 30

def m1(self) -> None:
    print('Son m1')

class Daughter(Father):
    z: int = 30

def m1(self) -> None:
    print('Daughter m1')

f = Father()
s = Son()
f.m1()
s.m1()
s.m2()
Design Square class

Squares have four vertices (corners) have a perimeter, an area, can move themselves by adding an offset point to each corner, and can draw themselves.
Use composition to build Square has_a relationship

• Why use composition?
  • We want to get abilities of existing classes (reusing our code=> save our time)
  • For drawing capabilities;
    • Use instance of Point to represent vertices and use distance method
    • Use instance (object) of Turtle to use its drawing capabilities (see next slide)

• Therefore:
  • Square has a Turtle.
  • Square has a Point.
Point class, we need to add two methods

- The Point class implementation from last week does not implement:
  - `def distance` and `__add__` which we will need while implementing Square
  - We need to add them to the Point class

```python
def distance(self, other: 'Point') -> float:
    """
    Return the distance between Point self and Point other.
    """
    return ((self.x - other.x) ** 2 + (self.y - other.y) ** 2) ** (1 / 2)

def __add__(self, other: 'Point') -> 'Point':
    """
    Return sum of Point self and Point other.
    """
    return Point(self.x + other.x, self.y + other.y)
```
Turtle class

- Turtle is a built-in class in Python
- Can be used to draw things:
- Try to run the code next in the console

```python
>>> from turtle import Turtle
>>> t = Turtle()
>>> t.penup()
>>> t.goto(0,0)
>>> t.goto(10,0)
>>> t.goto(100,0)
>>> t.goto(100,100)
>>> t.goto(-100,100)
>>> t.goto(-100,-100)
>>> t.pendown()
>>> t.goto(-100,0)
>>> t.goto(0,0)
>>> t.goto(0,-100)
>>> t.goto(-100,-100)
```
from turtle import Turtle
from point import Point

class Square:
    def __init__(self, corners: [Point]) -> None:
        self.corners = corners[:]
        self._turtle = Turtle()
        self._set_perimeter()
        self._set_area()

    def _set_perimeter(self) -> None:
        distance_list = []
        for i in range(len(self.corners)):
            distance_list.append(self.corners[i].distance(self.corners[i-1]))
        self._perimeter = sum(distance_list)

    def _set_area(self) -> None:
        self._area = self.corners[0].distance(self.corners[1]) ** 2

    def _get_perimeter(self) -> float:
        return self._perimeter

    def _get_area(self) -> float:
        return self._area

    def move_by(self, offset_point: Point) -> None:
        self.corners = [c + offset_point for c in self.corners]

    def draw(self):
        self._turtle.penup()
        self._turtle.goto(self.corners[-1].x, self.corners[-1].y)
        self._turtle.pendown()
        for i in range(len(self.corners)):
            self._turtle.goto(self.corners[i].x, self.corners[i].y)
        self._turtle.penup()
        self._turtle.goto(0, 0)
List comprehension example:

```python
>>> mylist = [1, 2, 3]
>>> result = [c+3 for c in mylist]
>>> mylist
[1, 2, 3]
>>> result
[4, 5, 6]
```

The for loop will go through `mylist` every round putting a new value for `c`. Then `c+3` will be evaluated and the value will be put in the result list as follows:

`[1+3, 2+3, 3+3]`

The result will be

`[4, 5, 6]`

```python
self.corners = [c + offset_point for c in self.corners]
```

Calls `__add__` in `Point` class

The resulting is a list of Points
more Square-like classes

What if we decided to devise a RightAngleTriangle class with similar characteristics to Square? There is an implementation of RightAngleTriangle, but it has a problem:

There's a lot of duplicate code. What do you suggest?
Where Can I find the code presented in class

• You can find the full code for Square, and RightAngleTriangle with NO inheritance in the course website under section MWF2 (L0301)

• with the following file names:
  • square_no_inh.py
  • right_angle_triangle_no_inh.py
  • test_no_inh_classes
    • This is a client code to test the above classes

• Download them Try different things with them and practice
  • Do not be afraid of doing mistakes
we could try:

• 1. cut-paste-modify Square $\rightarrow$ RightAngleTriangle?

• 2. include a Square in the new class to get at its attributes
• and services??

• We really need a general Shape with the features that are common to both Square and RightAngleTriangle, and perhaps other shapes that may come along
Shape

Super Class

All common attributes and methods of subclasses goes in the super class

RightAngleTriangle

subclass

Put only specific attributes and methods that applies only to this subclass

Square

subclass

Put only specific attributes and methods that applies only to this subclass

Future classes can extend shape easily
```python
from turtle import Turtle
from point import Point

class Square:
    corners: [Point]
    perimeter: float
    area: float

    def __init__(self, corners: [Point]) -> None:
        pass

    def _set_perimeter(self) -> None:
        pass

    def _set_area(self) -> None:
        self._area =
        self.corners[0].distance(self.corners[1]) ** 2

    def _get_perimeter(self) -> float:
        pass

    def _get_area(self) -> float:
        pass

    def move_by(self, offset_point: Point) -> None:
        pass

    def draw(self):
```

```python
from turtle import Turtle
from point import Point

class RightAngleTriangle:
    corners: [Point]
    perimeter: float
    area: float

    def __init__(self, corners: [Point]) -> None:
        pass

    def _set_perimeter(self) -> None:
        pass

    def _set_area(self) -> None:
        leg1 = self.corners[-1].distance(self.corners[0])
        leg2 = self.corners[0].distance(self.corners[1])
        self._area = (leg1 * leg2) / 2.0

    def _get_perimeter(self) -> float:
        pass

    def _get_area(self) -> float:
        pass

    def move_by(self, offset_point: Point) -> None:
        pass

    def draw(self):
```
abstract class Shape

• most of the features of Square are identical to RightAngleTriangle. Indeed I (blush) cut-and-pasted a lot...

• the differences are the class names (Square, RightAngleTriangle) and the code to calculate the area.

• put the common features into Shape, with unimplemented set area as a place-holder...

• declare Square and RightAngleTriangle as subclasses of Shape, inheriting the identical features by declaring:
  
  class Square(Shape): ...
from point import Point
from turtle import Turtle

class Shape:
    """
    A Shape shape that can draw itself, move, and report area and perimeter.
    """
    corners - corners of this Shape
    perimeter - length to traverse corners
    area - area of this Shape
    """
    corners: [Point]
    perimeter: float
    area: float
    #TODO implement the __eq__ and __str__

def __init__(self, corners: [Point]) -> None:
    """
    Create a new Shape self with corners.
    Assume that the corners are traversed in order, that the sides are equal
    and the sides are of equal length, and the vertices are right angles.
    """
    # we want to copy of the corners list using [:] not reference to them
    # so that if we change them we do not change the original list user
    # provided
    self.corners = corners[:]
    # a private attribute _turtle
    self._turtle = Turtle()
    self._set_perimeter()
    self._set_area()
Shape
Super Class Cont.

def __str__(self) -> str:
    """
    returns string representation
    """
    str_points = ",".join(str(c) for c in self.corners)

    return type(self).__name__+
    

def __set_perimeter(self) -> None:
    """
    Set Shape self's perimeter to the sum of the distances between
corners.
    """

distance_list = []
for i in range(len(self.corners)):
    distance_list.append(self.corners[i].distance(self.corners[i-1]))
self._perimeter = sum(distance_list)


def __set_area(self) -> None:
    """
    Set the area of Shape self to the Shape of its sides.
    """

    self._area = -1.0
    raise NotImplementedError("Set area in subclass!!!")


def get_perimeter(self) -> float:
    """
    Return the perimeter of this Shape
    """

    return self._perimeter

def get_area(self) -> float:
    """
    Return the area of this Shape
    """

    return self._area
def move_by(self, offset_point: Point) -> None:
    
    """
    Move Shape self to a new position by adding Point offset_point to each corner.
    """
    self.corners = [c + offset_point for c in self.corners]
    # print('corners')
    # for x in self.corners:
    #     print(x)

def draw(self) -> None:
    """
    Draw Shape self.
    """
    self._turtle.penup()
    self._turtle.goto(self.corners[-1].x, self.corners[-1].y)
    # print('-1 stuff')
    # -1 in self.corners[-1] refers to the last element
    # print(self.corners[-1].x, self.corners[-1].y)
    self._turtle.pendown()
    for i in range(len(self.corners)):
        self._turtle.goto(self.corners[i].x, self.corners[i].y)
    self._turtle.penup()
    self._turtle.goto(0, 0)

if __name__ == "__main__":
    import doctest
doctest.testmod()
    # s = Shape([Point(0, 0)])
from point import Point
from shape import Shape

class Square(Shape):
    """
    A Square Shape.
    """
    def __init__(self, corners: [Point]) -> None:
        """
        Create Square self with vertices corners.
        Assume all sides are equal and corners are square.
        Extended from Shape.
        >>> s = Square([Point(0, 0), Point(1, 0), Point(1, 1), Point(0, 0)])
        """
        Shape.__init__(self, corners)

    def _set_area(self) -> None:
        """
        Set Square self's area.
        Overrides Shape._set_area
        >>> s = Square([Point(0,0), Point(10,0), Point(10,10), Point(0,10)])
        >>> s.area
        100.0
        """
        self.area = self.corners[-1].distance(self.corners[0]) ** 2
from shape import Shape
from point import Point

class RightAngleTriangle(Shape):
    """
    A RightAngleTriangle Shape.
    """
    def __init__(self, corners):
        """ Create RightAngleTriangle self with vertices corners.
        Assume corners[0] is the 90 degree angle.
        Extended from Shape.
        """
        Shape.__init__(self, corners)

    def _set_area(self) -> None:
        """
        Set RightAngleTriangle self's area.
        Overrides Shape.set_area
        """
        >>> s = RightAngleTriangle([Point(0, 0), Point(1, 0), Point(0, 2)])
        """
        Shape.__init__(self, corners)
        
        def _set_area(self) -> None:
            leg1 = self.corners[-1].distance(self.corners[0])
            leg2 = self.corners[0].distance(self.corners[1])
            self.area = (leg1 * leg2) / 2.0

        >>> s = RightAngleTriangle([Point(0,0), Point(10,0), Point(0,20)])
        >>> s.area
        100.0
Where Can I find the code presented in class

• You can find the full code for Shape, Square, and RightAngleTriangle with inheritance in the course website under section **MWF2 (L0301)**

• with the following file names:
  • shape.py
  • square.py
  • right_angle_triangle.py
  • shape_simple_client
    • This is a simple client code to test the above classes
  • shape_adv_client
    • This is a simple client code to test the above classes

• Download them Try different things with them and practice
  • Do not be afraid of doing mistakes
inherit, override, or extend?

• subclasses use three approaches to recycling the code from their superclass, using the same name

• methods and attributes that are used as-is from the superclass are inherited -- examples?

• methods and attributes that replace what's in the superclass overridden -- example?

• methods and attributes that add to what is in the superclass are extended -- example?
write general code

• client code written to use Shape will now work with subclasses of Shape -- even those written in the future.

• The client code can rely on these subclasses having methods such as move by and draw

• Here is some client code that takes a list objects from subclasses of Shape, moves each object around, and then draws it.