Topics

• Why define subclasses?
  ▪ Understand the resulting hierarchy
  ▪ Design considerations

• How to define a subclass
  ▪ Initializer
  ▪ New methods
  ▪ Write modified versions of inherited methods
  ▪ Access parent’s version using super()

Goal: Make a drawing app

Rectangles, Stars, Circles, and Triangles have a lot in common, but they are also different in very fundamental ways....

Sharing Work

Problem: Redundant code.
(Any time you copy-and-paste code, you are likely doing something wrong.)

Solution: Create a parent class with shared code
  ▪ Then, create subclasses of the parent class
  ▪ A subclass deals with specific details different from the parent class
Extending Classes

class <name>(<superclass>):
    """Class specification""
    <class variables>
    <initializer>
    <methods>

So far, classes have implicitly extended object

Subclassing creates a hierarchy of classes
- Each class has its own super class or parent
- Until object at the "top"

Object has many features
- Default operators:
  __init__, __str__, __eq__

Which of these need to be replaced?

Want to use the original version of the method?
- New method = original+more
- Don't repeat code from the original
- Call old method explicitly

Can override methods; can access parent's version

Want to compare equality of the values [data] of two instances, not the id of the two instances!
Understanding Method Overriding

```python
import Shape

c1 = Circle(1,2,4.0)
print(str(c1))
```

- Which `__str__` do we use?
  - Start at bottom class folder
  - Find first method with name
  - Use that definition
- Each subclass automatically inherits methods of parent.
- New method definitions override those of parent.

The following questions will be addressed in the lecture that follows.

### Q1: Name Resolution and Inheritance

```python
class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18
```

- Execute the following:
  ```python
  >>> a = A()
  >>> b = B()
  ```
- What is value of `a.f()`?
  - A: 10
  - B: 14
  - C: 5
  - D: ERROR
  - E: I don’t know

### Q2: Name Resolution and Inheritance

```python
class A:
    def f(self):
        return self.g()
    def g(self):
        return 10

class B(A):
    def g(self):
        return 14
    def h(self):
        return 18
```

- Execute the following:
  ```python
  >>> a = A()
  >>> b = B()
  ```
- What is value of `b.f()`?
  - A: 10
  - B: 14
  - C: 5
  - D: ERROR
  - E: I don’t know

Name Resolution Revisited

- To look up attribute/method name
  1. Look first in instance (object folder)
  2. Then look in the class (folder)
- Subclasses add two more rules:
  3. Look in the superclass
  4. Repeat 3. until reach object

Often called the **Bottom-Up Rule**

```python
c1 = Circle(1,2,4.0)
r = c1.radius
c1.draw()
```