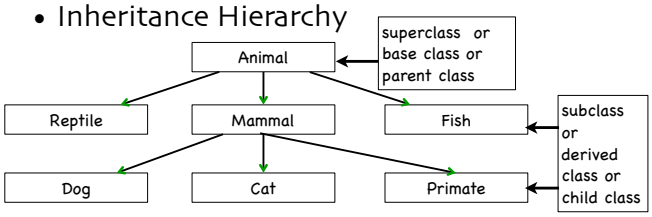
Inheritance

WTP 2010
Computer Science
Day 8: Thursday, July 8

Inheritance

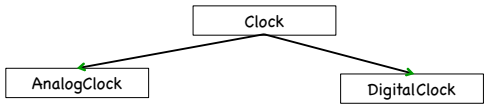
- Models "is a" relationships
- Uses similarities and differences to model groups of related objects
- Inheritance Hierarchy



Superclass and Subclasses

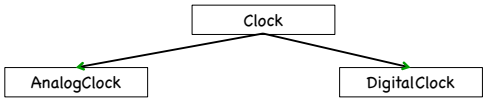
- Inheritance is a way of:
  - organizing information
  - grouping similar classes
  - classifying objects
- Superclass factors out capabilities common among classes
- Subclass specializes its superclass by:
  - adding new methods
  - overriding existing methods

Factor Out Common Functionality




```
class DigitalClock(object):

    def __init__(self, hour, min, sec, pos): …
    def __str__(self): …
    def draw(self, win): …
    def draw_face(self, win): …
    def draw_time(self, win): …
```

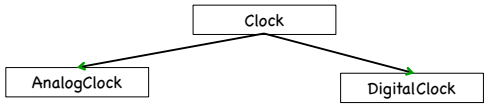
Superclass




```
class Clock(object):

    def __init__(self, hour, min, sec, pos):
            self.hour = hour
            self.min = min
            self.sec = sec
            self.pos = pos

    def __str__(self):
            time_str = '%2d:%2d:%2d' % (self.hour, self.min, self.sec)
               return time_str

    def draw_time(self, win):
            print self

    ...
```

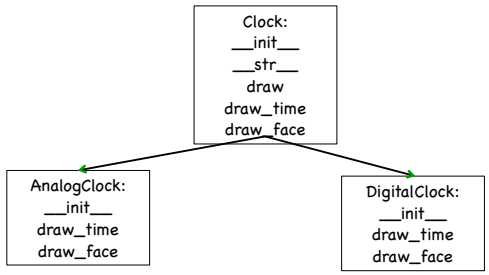
special method prints object as a string

Superclass (continued)

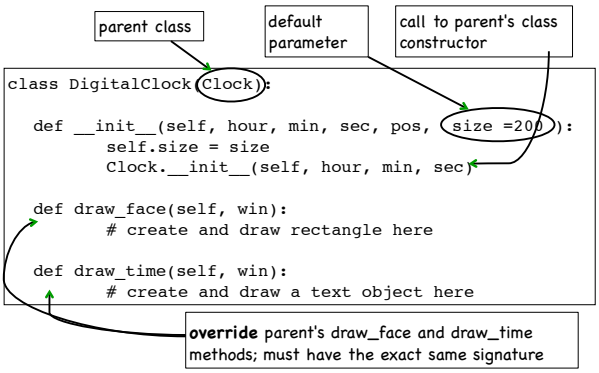



```
class Clock(object):

    ...

    def draw_face(self, win):
         # parent class has no implementation for draw_face
            return

    def draw(self, win):
        self.draw_time(win)
        self.draw_face(win)
```

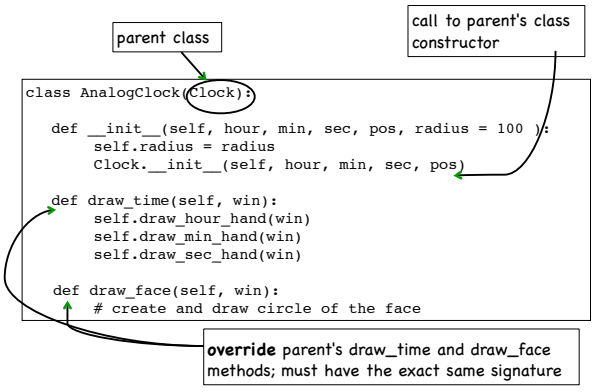
Clock Specialization



Subclass (Child Class)



Subclass (Child Class)



Parent Attributes / Methods

- Parent attributes/methods can be accessed the same way as the current object's attributes, e.g.

```
class DigitalClock(Clock):
  …
  def draw_time(self, win):
    time = Text(Point(self.pos.x + self.size/2,
                      self.pos.y + self.size/4),
                self.__str__())
    time.setSize(24)
    time.setStyle('bold')
    time.draw(win)
    self.text = time
```

Overriding methods

- Completely replace the functionality in the child class

```
class DigitalClock(Clock):
  …
  def draw_time(self, win):
    self.time = Text(Point(self.pos.x + self.size/2,
                           self.pos.y + self.size/4),
                     self.__str__())
    self.time.draw(win)
```

Overriding methods

- Extend the functionality in the child class

```
class DigitalClock(Clock):
  …
  def draw_time(self, win):
    Clock.draw_time(self, win)
    self.time = Text(Point(self.pos.x + self.size/2,
                           self.pos.y + self.size/4),
                     self.__str__())
    self.time.draw(win)
```

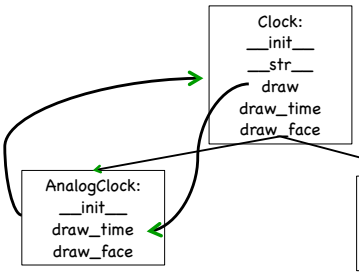
Method Resolution

default implementation of the draw_time and draw_face methods; will be overridden in child classes

```
class Clock(object):

   ...
   def draw_time(self, win):
        print self

   def draw_face(self, win):
        # parent class has no implementation for draw_face
        return

   def draw(self, win):
       self.draw_time(win)
       self.draw_face(win)
```

the draw method implementation is the same for all clocks; python will call the correct draw_time and draw_face methods depending on which clock object was created.

Method resolution




```
... # imports and clock definitions omitted

win = GraphWin('Clock', 400, 300)
clock = AnalogClock(5, 20, 30)
clock.draw(win)
win.mainloop()
```

Another Example

- What if we want to create a triangle graphics object?
- Is triangle a special kind of the ones we have available?
  - Point
  - Line
  - Circle
  - Oval
  - Rectangle
  - Polygon

Triangle Class

```
class Triangle(Polygon):

     def __init__(self, p1, p2, p3):
         Polygon.__init__(p1, p2, p3)
```

- What if we want all our triangles to have black outline?

```
class Triangle(Polygon):

     def __init__(self, p1, p2, p3):
         Polygon.__init__(p1, p2, p3)
         self.setOutline('black')
         self.setWidth(2)
```

setOutline and setWidth are capabilities of the Polygon class and are inherited by the Triangle class.

Today's Exercises

- Understanding inheritance
- Project preview
- Practice using inheritance
- Fill in the project group preference form
- Reading for tomorrow: Sections 9.14 (reread), 12