Object Oriented Programming

INHERITANCE

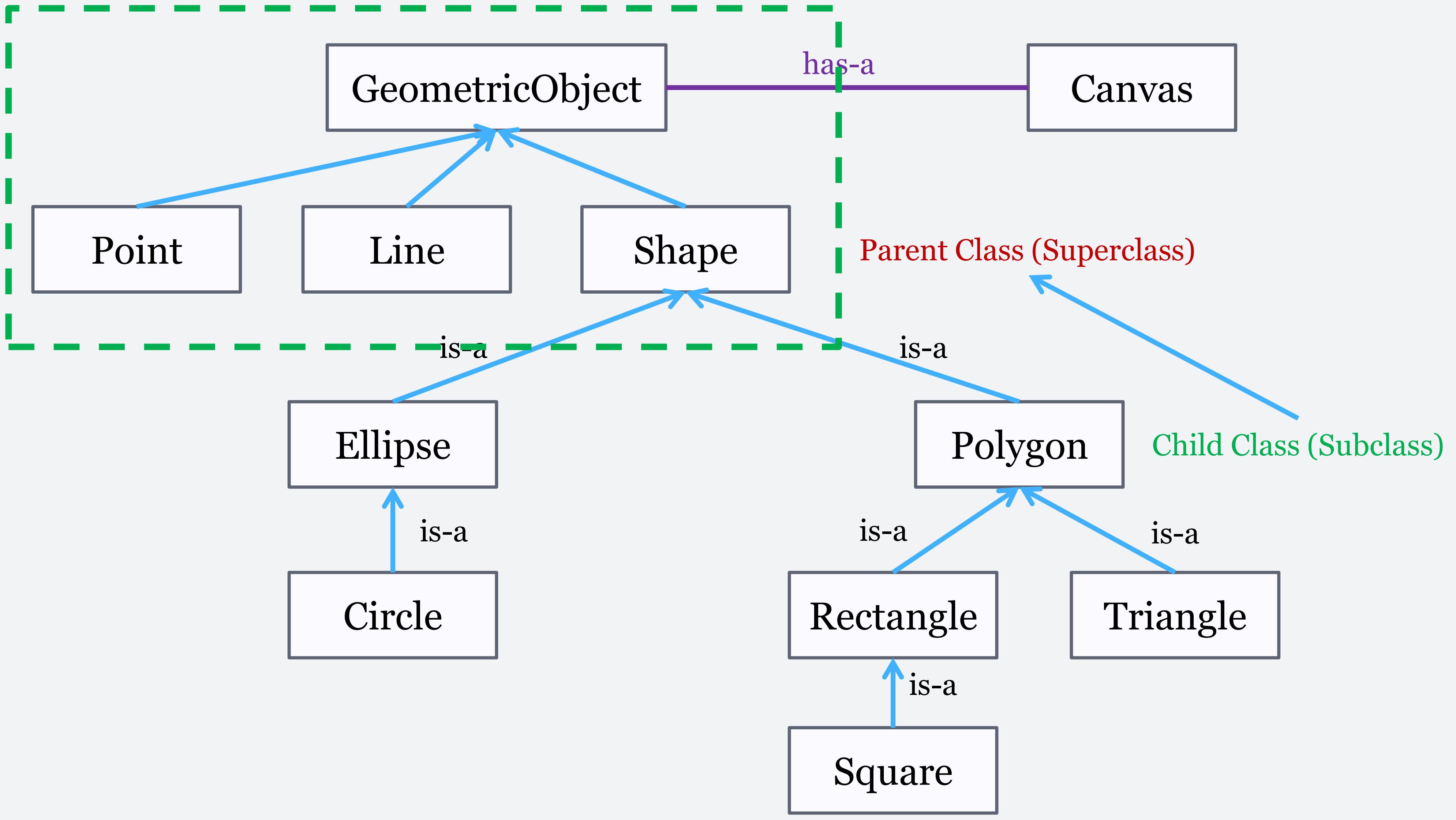
&

MORE ON CLASSES

Overview

- Inheritance continued
- Method Overriding
- Abstract Classes and Interfaces
- Static and Class methods

Inheritance Hierarchy



Calling the superclass constructor

- Use of the keyword **super**

Construct: super(classname, self).__init__(…)

Class definition

```
class Point(geometricObject):


   def __init__(self, x, y):
     super(Point,self).__init__()
     self._x = x
     self._y = y

   def draw(self):
       ...
```

Calling a superclass method

- Use of the keyword **`super`**

```
Construct: super(classname, self).methodName(…)
```

- You must be careful of the Method Resolution Order.

Private vs Public Attributes

- There is no completely private attributes in Python
- Can be simulated using __attributename (two underscores)
- If using private/protected attribute, you should provide adequate
  - Accessors method (read/get value)
  - Mutators method (change/set value)
- Which attribute must be public/protected/private is a design decision
- Which mutator/accessor to provide is also a design decision

Abstract Classes and Concrete Classes

- Abstract class `(see inheritance_abstract_shape.py)`
  - Objects never instantiated
  - Intended as a base class in an inheritance hierarchy

- Concrete class : class from which an object can be created

- Example : Concrete class `Point` derived from abstract base class `GeometricObject`

Abstract Classes and Concrete Classes

First Attempt:

Class definition

```
class GeometricObject(object):
   """Abstract base class GeometricObject""`

   def __init__(self):
       self._lineColor = 'black`
       self._lineWidth = 1

   def draw(self):
      """Abstract method; derived classes must override"""
      raise NotImplementedError("Cannot call abstract method")

   def setColor(self, colour):
       self._lineColor = colour
```

Not Satisfactory as instances of GeometricObject can still be created. Error only when calling draw.

Cannot enforce subclasses implementation of the method draw()

Abstract Classes and Concrete Classes

Second Attempt:

Class definition

```
import abc ## module for abstract classes
class GeometricObject(object):
   """Abstract base class GeometricObject"""
   __metaclass__ = abc.ABCMeta

   def __init__(self):
       self._lineColor = 'black'
       self._lineWidth = 1

   @abc.abstractmethod
   def draw(self):
      """Abstract method; derived classes must override"""
```

Enforce subclasses implementation of the method draw()

Method Overriding

- The class Point must implement the method draw(self).

- Rewriting the method of a superclass is called method **overriding**

- It should be done when the method need to be more specialised in the subclass

- Not all methods should be overridden in a subclass

Polymorphism

- Polymorphism : ability of objects of different classes related by inheritance to respond differently to same messages
- Python language inherently polymorphic because of dynamically typing
- Dynamically typed : Python determines at runtime whether an object defines a method or contains an attribute

Static & Class Methods

- Let look at a class representing Date
  - (See `StaticClassMethods.py`)
  - We can build an object given 3 int numbers for day, Month and Year
  - What if we want to create an object using a string “dd/mm/yyyy” ?
  - No **overloading** mechanism in Python

Class definition

```
class Date (object):

   def __init__(self):
        self.day = day
         self.month = month
         self.year = year

   def __repr__():
        return str(self.day)+"/"+str(self.month)+"/"+str(self.year)
```

Static & Class Methods

- A solution is using class method

Class definition

```
class Date (object):

    def __init__(self):
         self.day = day
          self.month = month
          self.year = year

    def __repr__):
         return str(self.day)+"/"+str(self.month)+"/"+str(self.year)

    @classmethod
    def from_string(cls, date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
        date1 = cls(day, month, year)
        return date1
```

Static & Class Methods

- A Static method provide facilities for a class without the need of an instance.
- doesn't take any <u>obligatory</u> parameters such as `self` or `cls`.

Class definition

```
 class Date (object):

    def __init__(self):
        self.day = day
         self.month = month
         self.year = year
...
    @staticmethod
    def is_date_valid(date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
        return day <= 31 and month <= 12 and year <= 3999
```

Static & Class Methods

- @classmethod means: when this method is called, we pass the class as the first argument instead of the instance of that class (as we normally do with methods). This means you can use the class and its properties inside that method rather than a particular instance.

- @staticmethod means: when this method is called, we don't pass an instance of the class to it (as we normally do with methods). This means you can put a function inside a class but you can't access the instance of that class (this is useful when your method does not use the instance).

- See `StaticClassMethods2.py`

Pitfalls

Data attributes override method attributes with the same name

- To avoid accidental name conflicts, it is wise to use some kind of convention that minimizes the chance of conflicts.
- Possible conventions include
  - capitalizing method names,
  - prefixing data attribute names with a small unique string
  - using verbs for methods and nouns for data attributes.

Summary

- You should have understood the principle of inheritance
- Call to the superclass Constructor
- The notion of method overriding
- The use of abstract classes and interfaces
- Static and class methods

Further Reading

- http://learnpythonthehardway.org/book/ex44.html