Object Oriented Programming

2

INHERITANCE

Overview

- Last term review
- is-a & has-a Relationship
- Inheritance
- Method Overriding

What are Classes?

- Classes are composed from structural and behavioural constituents.
- Data field members (member variables or instance variables) enable a class instance to maintain state.
  - a.k.a *properties*, *fields*, *data members*, or ***attributes***
- **Methods**, enable the behaviour of class **instances**.
- **Classes** define the type of their instances

Python and OOP

- Modules
- Classes : class keyword
- Attributes
- Methods: __repr__(self) for example
- Constructor: __init__(self,...)

Classes in Python

Class definition

```
class QueueOOP:
    def __init__(self):
        self._head = None ## Pointer to front of queue, _Node object
        self._tail = None ## Pointer to back of the queue , _Node object
        self._size = 0

    def dequeue(self):…

    def enqueue(self,element):…
```

Method definition

```
def dequeue(self):
        if self.isempty():
            raise IndexError('dequeue from empty queue')
        else:
            element = self._head.datum
            self._head = self._head.next
            if self._head is None: self._tail = None
            self._size -= 1
            return element
```

Encapsulation as Information hiding

- Data belonging to one **object** is hidden from other objects.
- Know what an object can do, not how it does it.
- Information hiding increases the level of *independence*.
- Independence of modules is important for large systems and maintenance.

Cohesion of methods

A method should be responsible for **one and only one** well defined task.

Cohesion of classes

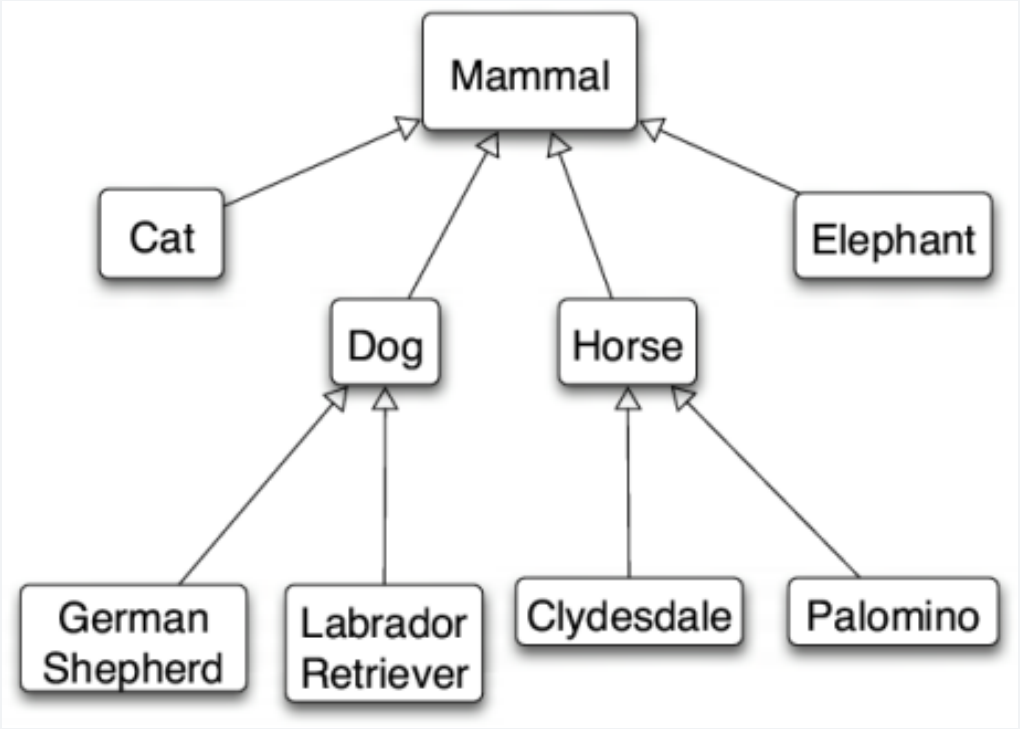
Classes should represent **one single**, well defined entity.

High Cohesion

- The class QueueOOP contains only method necessary for this type of ADT.
  - does NOT implement add_first, insert_at, remove, etc.
- If the user need such methods, he/she **must** use a different ADT (not a queue).
- Several ADTs could/should be implemented in the same module

Cohesion of classes

It should be represented in the classes hierarchy as well



Inheritance

12

IS-A RELATIONSHIP

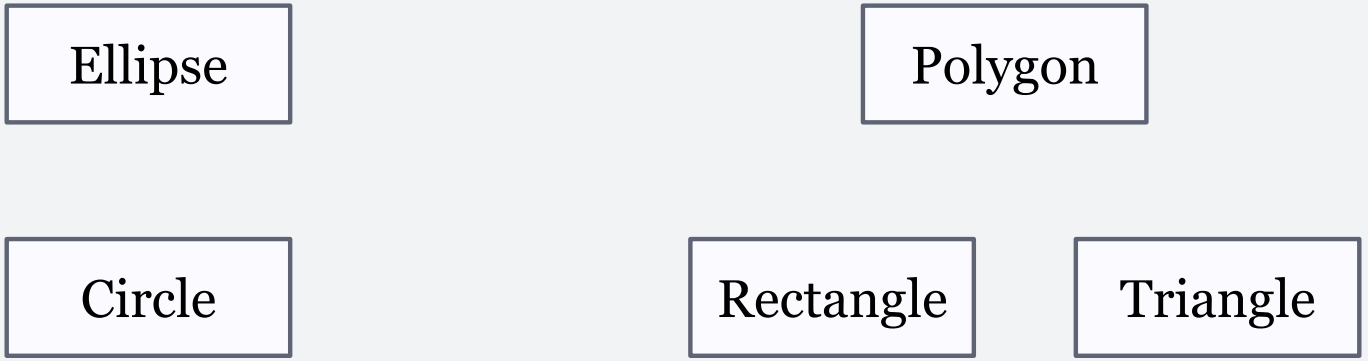
is-a & has-a Relationship

- The **is-a** relationship describes two objects where one object is more specific instance of the other (inheritance)
- The **has-a** relationship describes two object where one object use another object (composition)
- Example:
  - A Rectangle is-a more specific instance of a Polygon
  - A Circle has-a centre Point

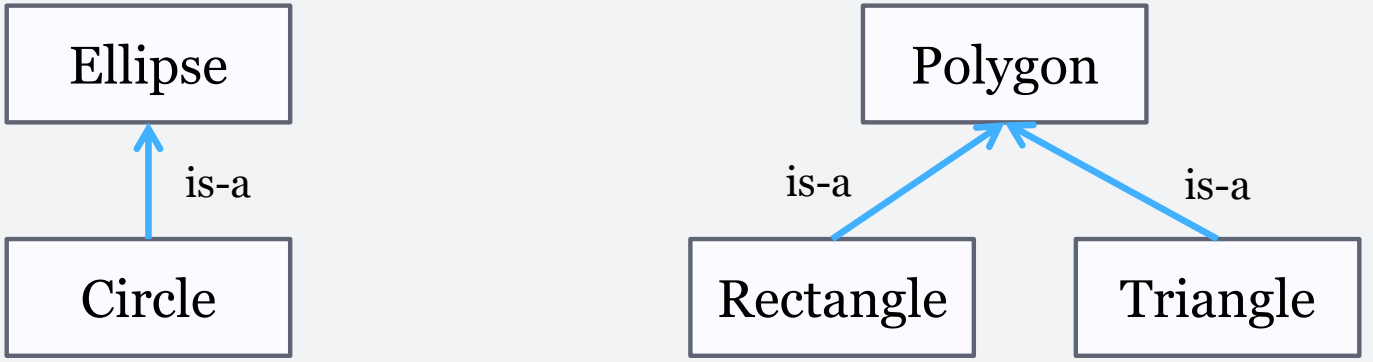
Building a Graphic Module

- list of objects:
  - Rectangle
  - Ellipse
  - Circle
  - Triangle
  - Polygon
  - Line
  - Point
  - Canvas

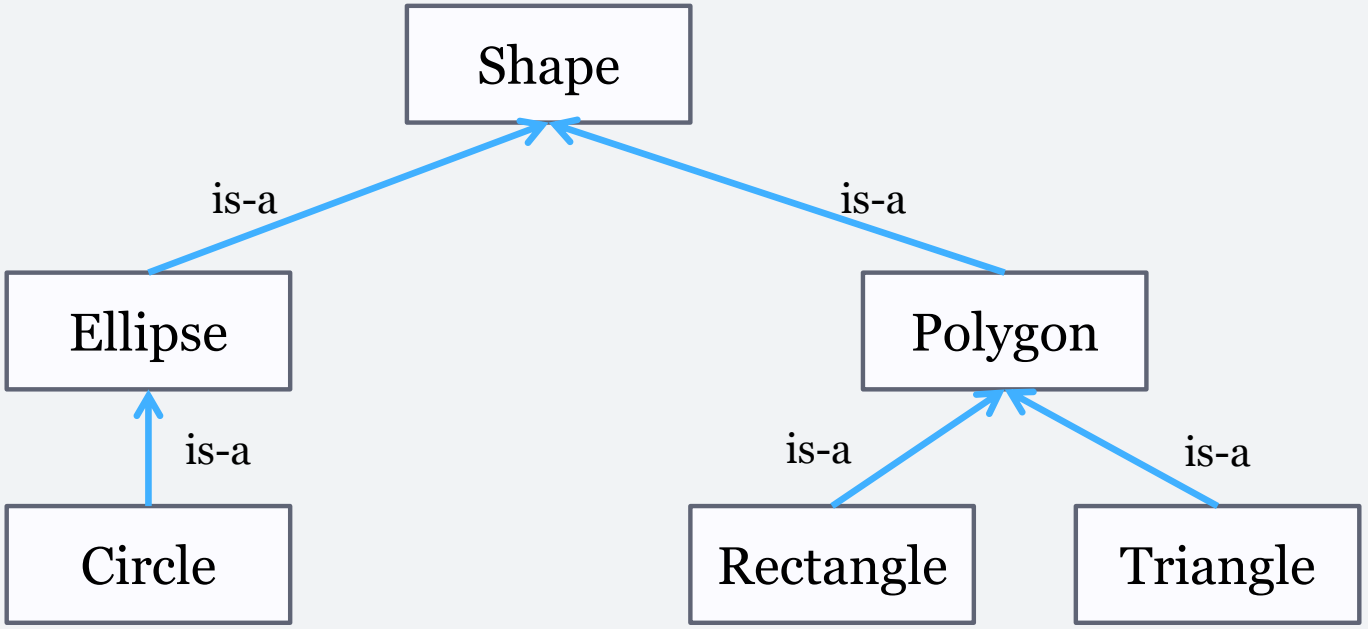
Building a Graphic Module



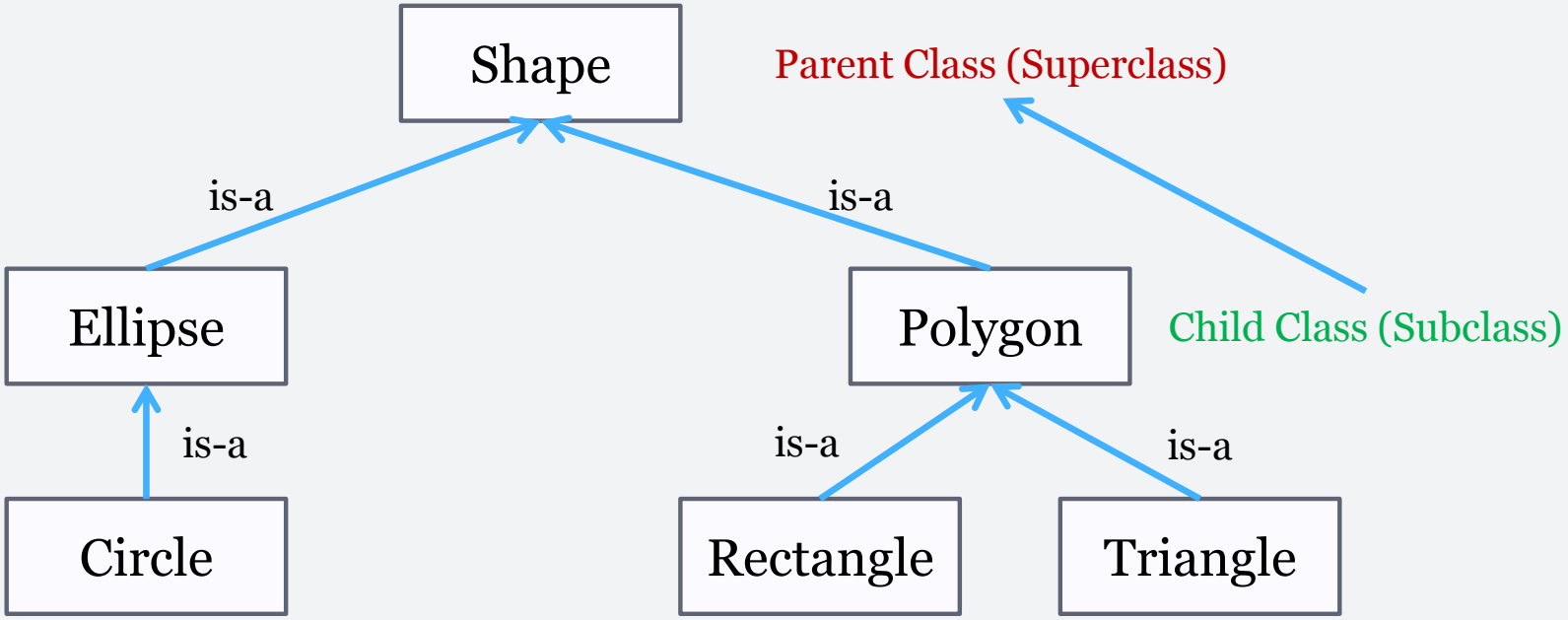
Inheritance Hierarchy



Inheritance Hierarchy



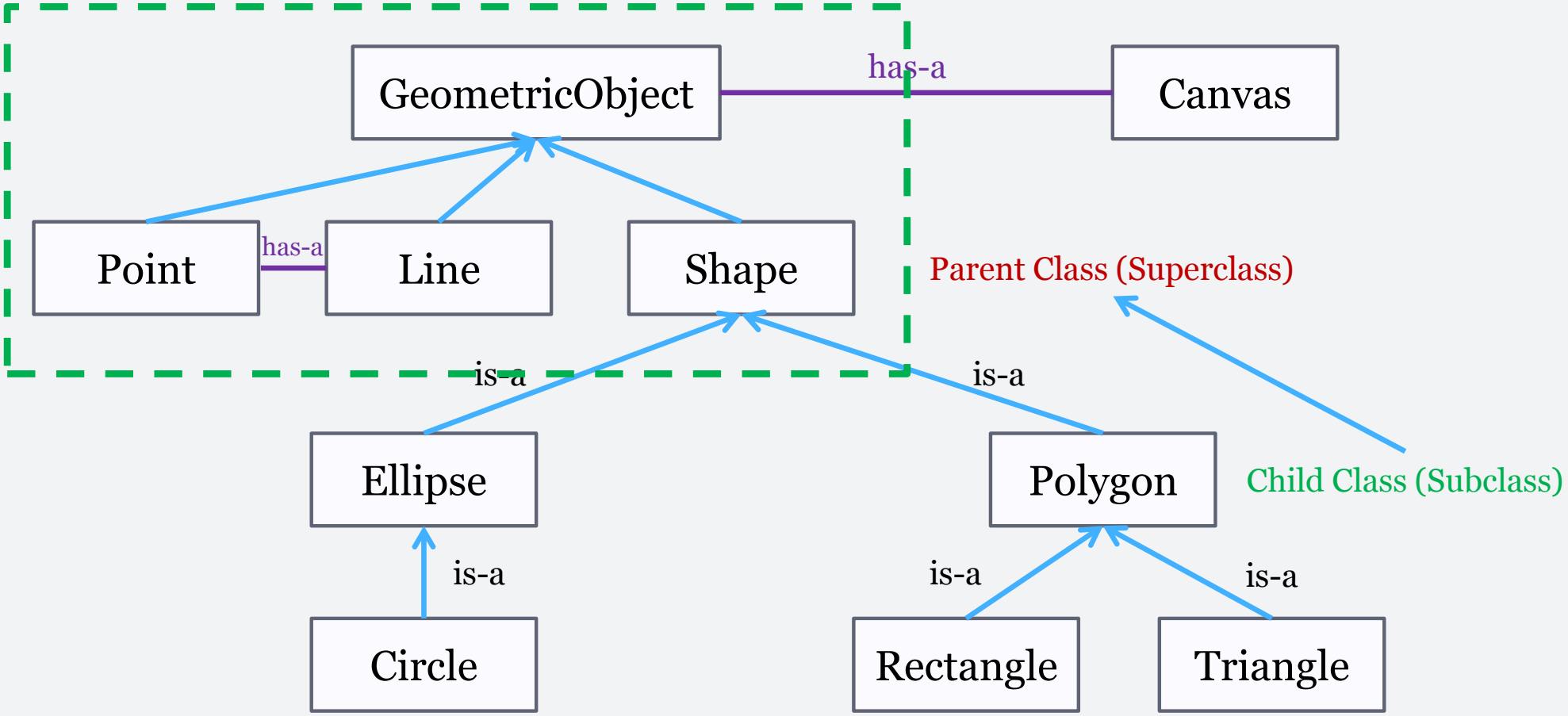
Inheritance Hierarchy



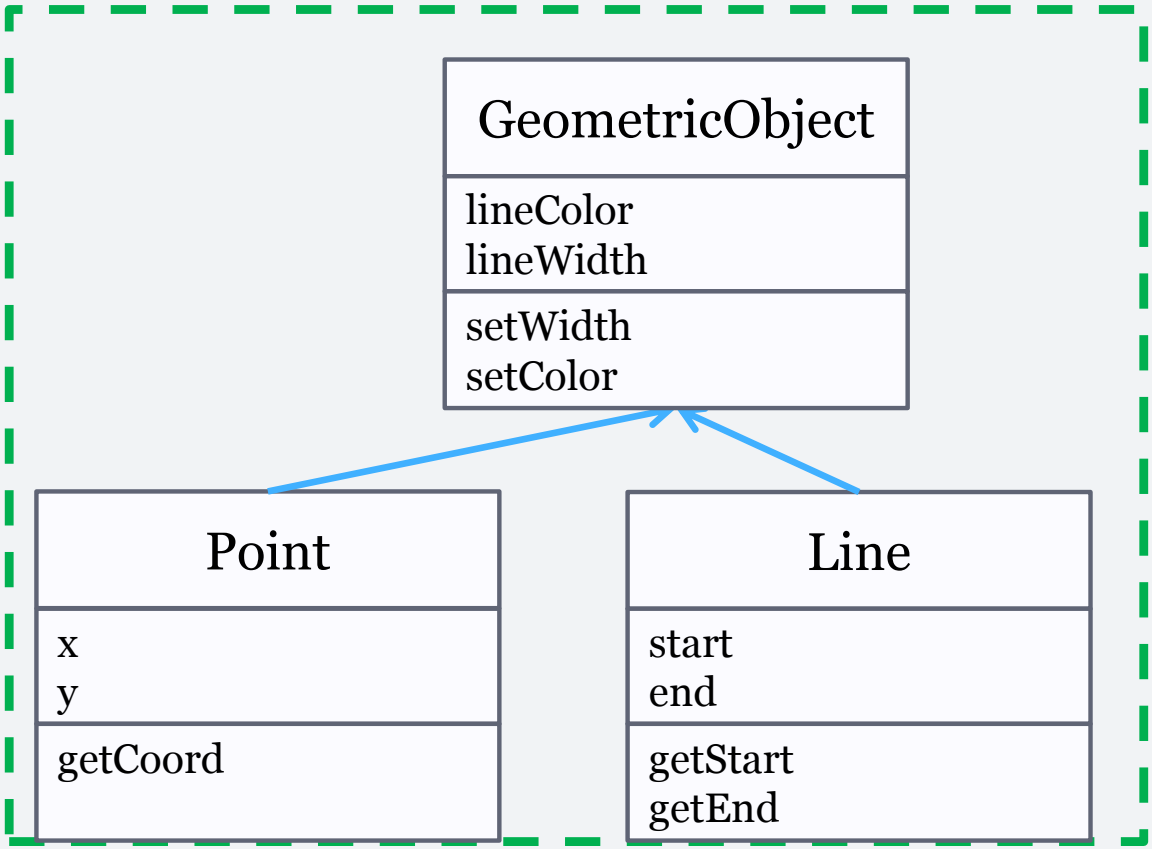
The Next Step in Design

- What things should each object know?
  - e.g. Instance variables
    - fill colour
    - Outline colour
    - Position in canvas
    - Line width
- What things should each object be able to do?
  - e.g. Methods
    - Change the fill colour: setFill(colour)
    - ....
- What are the commonalities between objects?

Inheritance Hierarchy



Instance Variables and Methods



Implementation

- see code

Summary

- You should have understood the principle of inheritance
- The notion of method overriding
- Call to the superclass Constructor

Exercises

- Write the remaining subclasses of shape.
- What other shape can you devise?
- What other attribute could be added to shape or its subclasses
- Change the draw method, such as given a turtle graphic object in its parameter, it draw the shape on the canvas. You can use the code from the Battleship program to learn how to draw using turtle.