Object-Orientation

(Class & Objects)

Saurabh Tiwari, PhD
DAIICT Gandhinagar

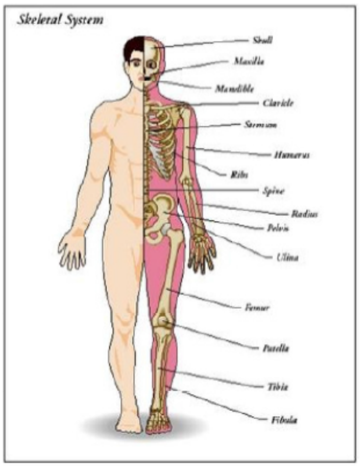
Real World Objects

Real World Objects

- Have identity and are distinguishable
- Share two characteristics
  - State
  - Behavior

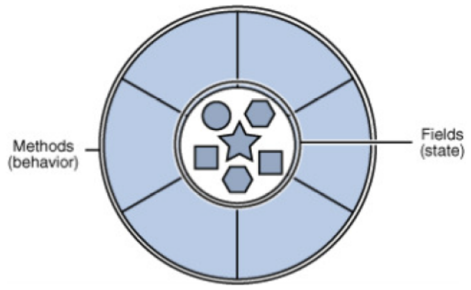
Real World Objects

- Some Object contains other Objects



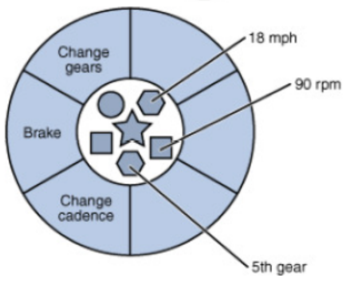
Software Objects

- Conceptually similar to real-world objects:
- they too consist of state and related behavior



Software Object: Cycle

```
class Bicycle {
      int cadence = 0;
      int speed = 0;
      int gear = 1;
      void changeCadence(int newValue) {
          cadence = newValue;
      }
      void changeGear(int newValue) {
          gear = newValue;
      }
      void speedUp(int increment) {
          speed = speed + increment;
      }
      void applyBrakes(int decrement) {
          speed = speed - decrement;
      }
      void printStates() {
          System.out.println("cadence:"+cadence+" speed:"+speed+" gear:"+gear);
      }
}
```



Software Objects

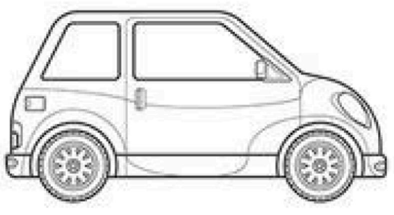
7

- Advantages
  - Abstraction
  - Encapsulation (Modularity)
  - Information-hiding
  - Code re-use
  - Pluggability and debugging ease

Class & Object

8

Class



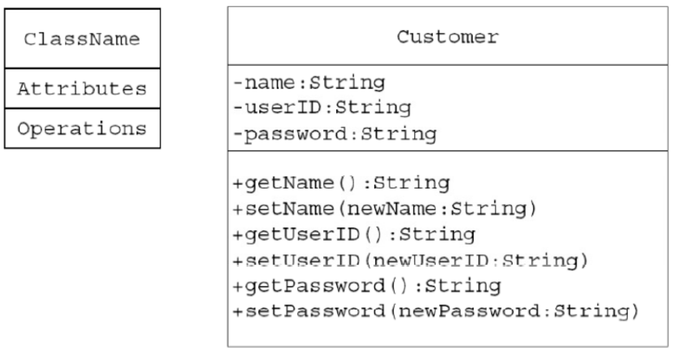
Blueprint of a car

Object

Car

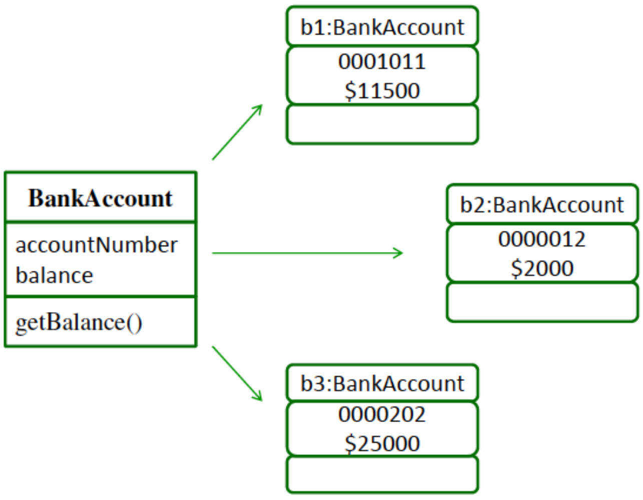
Class

9



Class and Objects

10

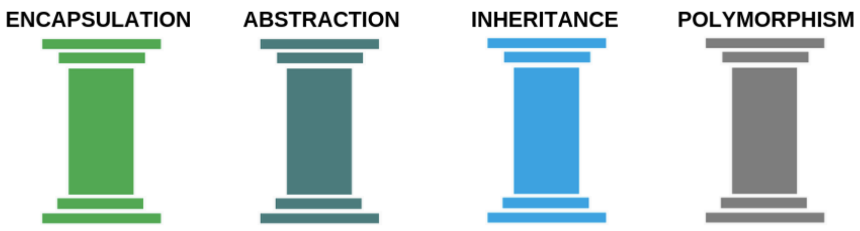


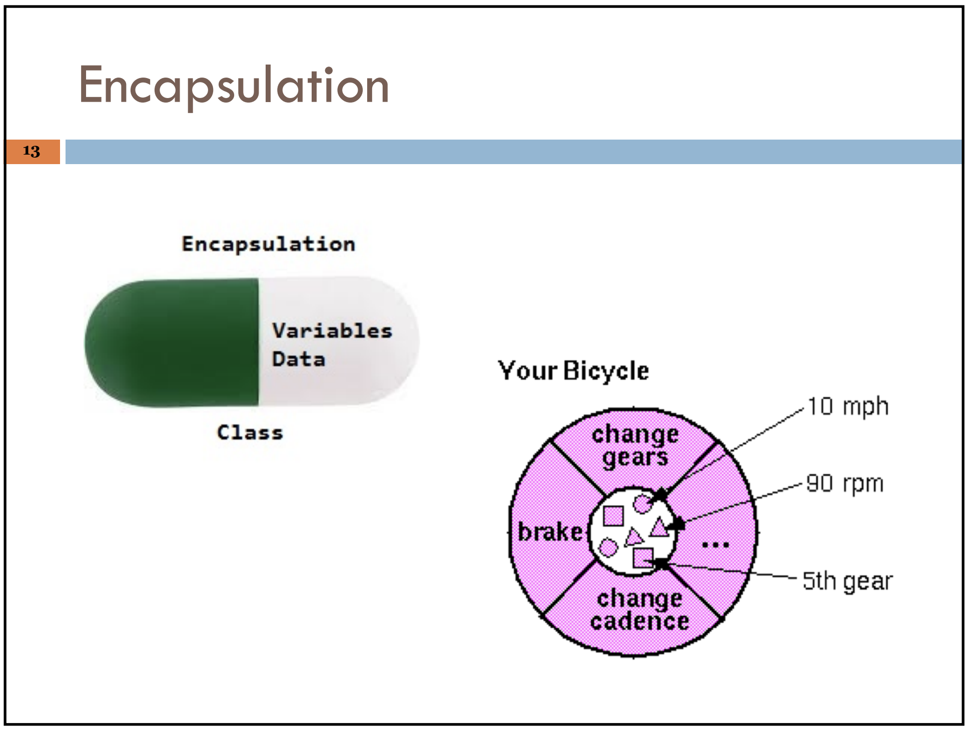
OO- Based on the Objects

11

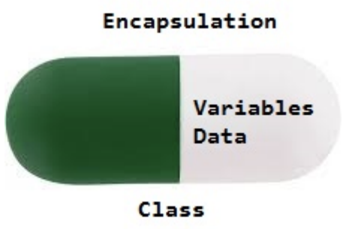
Four Pillars of OO

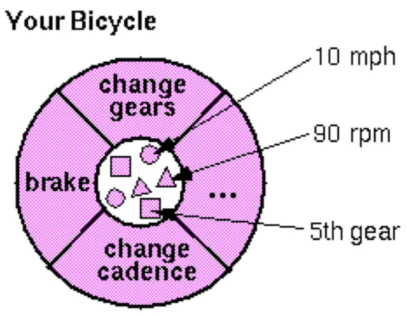
12

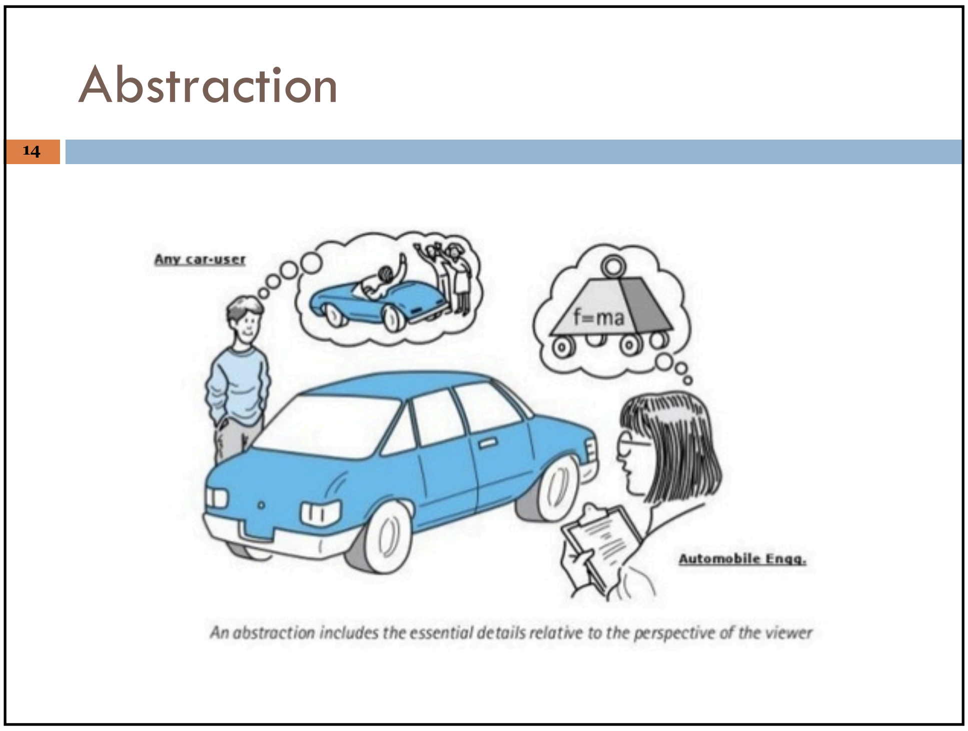




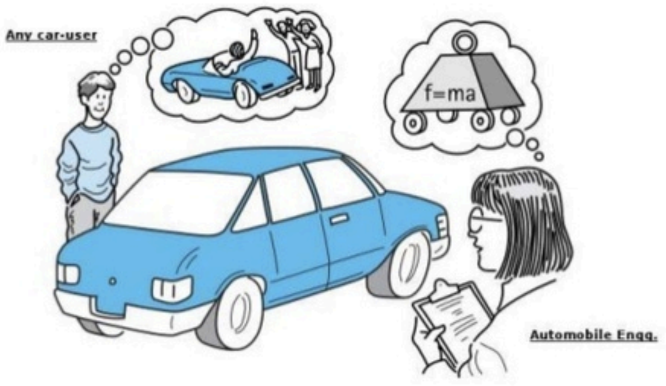
Encapsulation







Abstraction

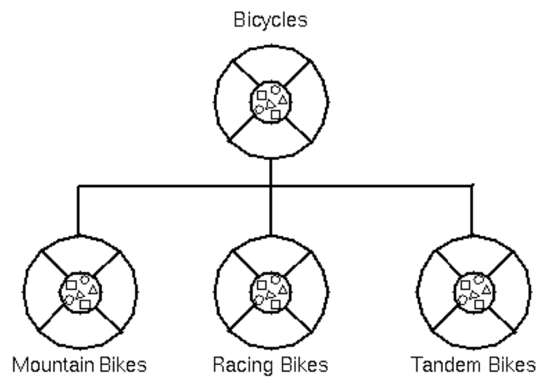


An abstraction includes the essential details relative to the perspective of the viewer

Inheritance

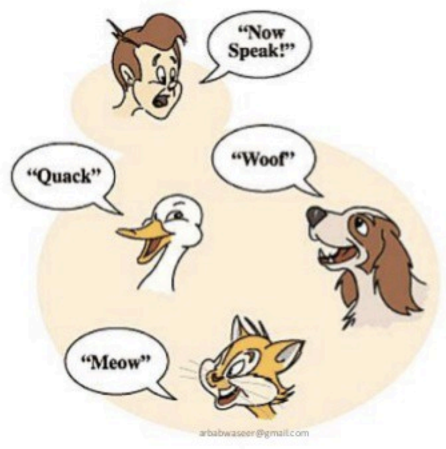
15

Hierarchy of Classes



Polymorphism

16



Example: Four Pillars

17

- When we think of a mobile phone as an object, its basic functionality for which it was invented were Calling & Receiving a call & Messaging.
- But nowadays thousands of new features and models were added and the features and number of models are still growing.

Object: Example

18

- a mobile manufacturing company can be an object. Each object can be different based on their characteristics. For example, here are two objects.
- Mobile mbl1 = **new** Mobile ();
- Mobile mbl2 = **new** Mobile ();

Class: Example

- Mainly a class would consist of a name, attributes, and operations.
- Considering the example, Mobile can be a class, which has some attributes like Profile Type, IMEI Number, Processor, and some more. It can have operations like Dial, Receive and SendMessage.

Abstraction: Example

- It is the most important pillar in OOPS. In our example of Mobile class and objects like Nokia, Samsung, iPhone.
- Some features of mobiles,
  - Dialling a number call some method internally which concatenate the numbers and displays it on screen but what is it doing we don't know.
  - Clicking on green button actual send signals to calling a person's mobile but we are unaware of how it is doing.

Encapsulation: Example

- Talking about Bluetooth which we usually have in our mobile. When we switch on a Bluetooth, I am able to connect to another mobile or Bluetooth enabled devices but I'm not able to access the other mobile features like dialling a number, accessing inbox etc. This is because the Bluetooth feature is given some level of abstraction.
- Another point is when mobile A is connected with mobile B via Bluetooth whereas mobile B is already connected to mobile C then A is not allowed to connect C via B. This is because of accessibility restriction.

Polymorphism: Example

- Let's say Samsung mobile has a 5MP camera available i.e. — it is having a functionality of CameraClick(). Now the same mobile is having Panorama mode available in-camera, so functionality would be the same but with mode – Method Overloading
- Another point wherein SendMessage() was intended to send message to a single person at a time but suppose Nokia had given provision for sending a message to a group at once. i.e. — Overriding the functionality to send message to a group.

Inheritance: Example

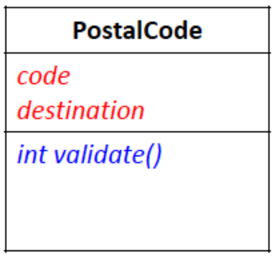
- Basic Mobile functionality is to send a message, dial and receive a call. So the brands of mobile are using this basic functionality by extending the mobile class functionality and adding their own new features to their respective brand.

Problem

- Develop a program which recognizes whether a given code belongs to any of the three country codes, namely, Canada, Britain and US, otherwise throws exceptions. It also tries to identify the location according to the given description of the postal codes.

Solution Approach 1: Brute Force Approach

- A PostalCode class with one method to determine
  - the validity
  - If found valid, then location.



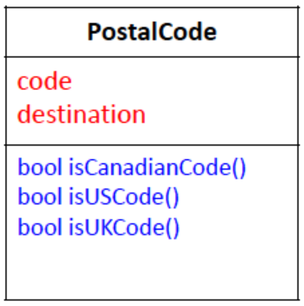
Solution Approach 1

- **Disadvantages**
- The method validate( ) becomes very complex
  (Cyclomatic complexity can be high)*
- Comprehension is not easy
- Debugging and modification of the code will be difficult
- Not modular and not scalable
- Functional-oriented
- Not a good OO Solution

Solution Approach 2

27

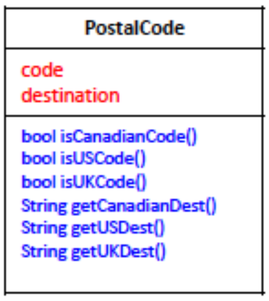
- Write separate postal code checking methods for the three countries
- If found valid, the method also compute the destination



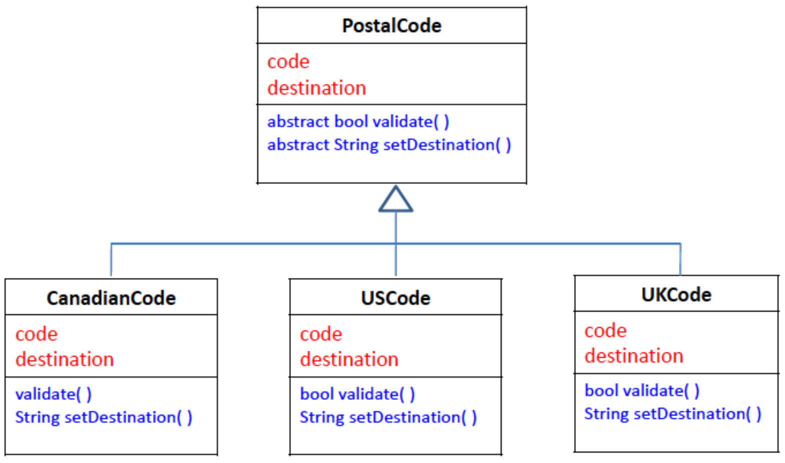
Solution Approach 2

28

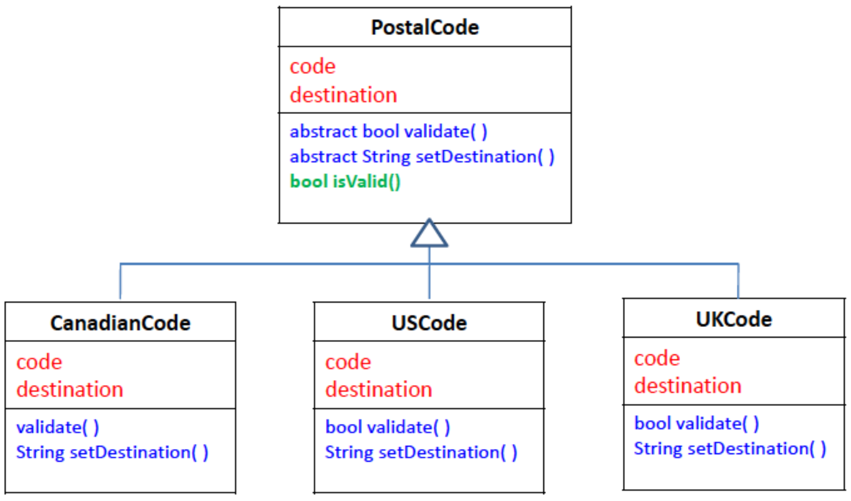
- Still
  - Functional-oriented
  - Scalability (Extendable)?
  - Is it Object-Oriented?



Solution Approach 3



Solution Approach 3 (cont..)



Identifying Objects and how they are related?

Overview

- Identifying Objects
  - – Identity
  - – Properties
  - – Behavior
- Identify Classes
  - – Name
  - – Class Attributes
  - – Class Methods
- Identify Relationships
- Develop a class diagrams

Identifying Objects

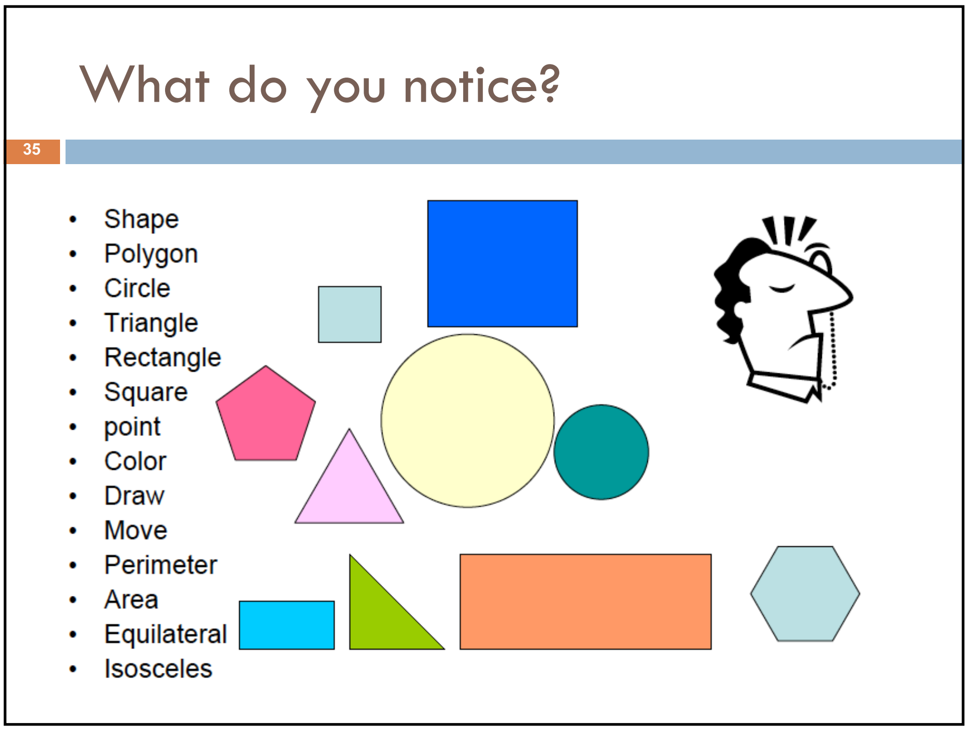
Problem #1: 2D Geometric Objects

Regular 2D shapes can be polygons or circles. A polygon may be a triangle, a rectangle, square or a circle. We can assign a color to a shape and draw it. The shape can be moved to a position. We should be able to determine perimeter and area of a given shape.

What do you notice?

Problem #1: 2D Geometric Objects

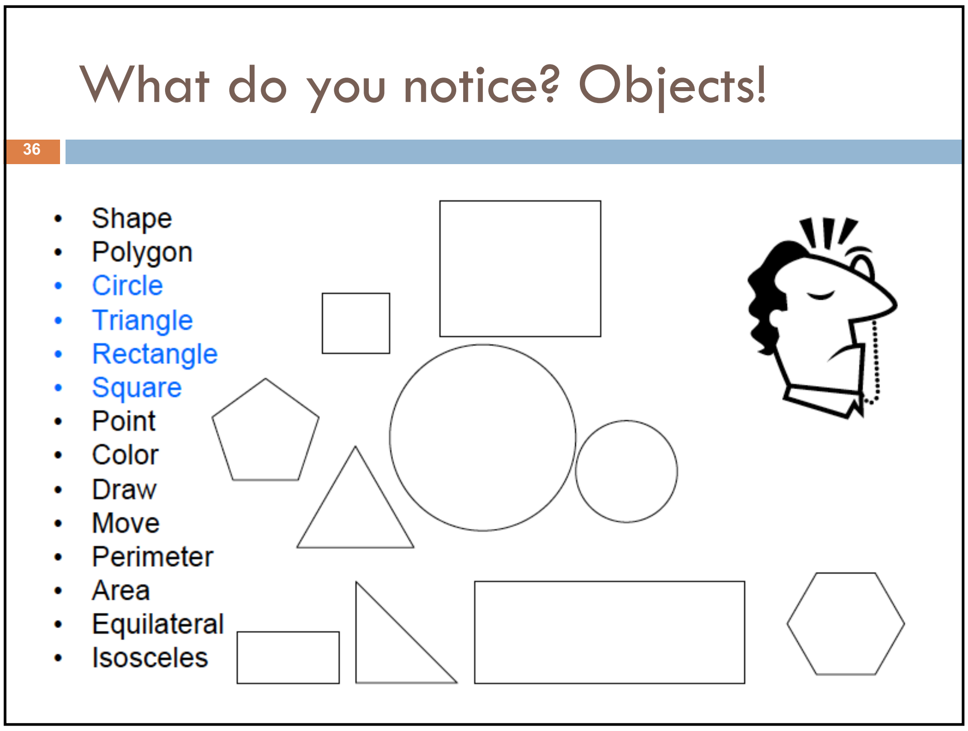
- Regular 2D **shapes** can be **polygons** or **circles**. A polygon consists of a number of **points** (>2). A polygon may be a **triangle**, a **rectangle**, **square** or a **circle**. We can assign a **color** to a shape and **draw** it. The shape can be **moved** to a **position**. We should be able to determine **perimeter** and **area** of a given shape. In case of a triangle, we determine whether, it is **equilateral** or **isosceles** triangle.



What do you notice?

35

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles



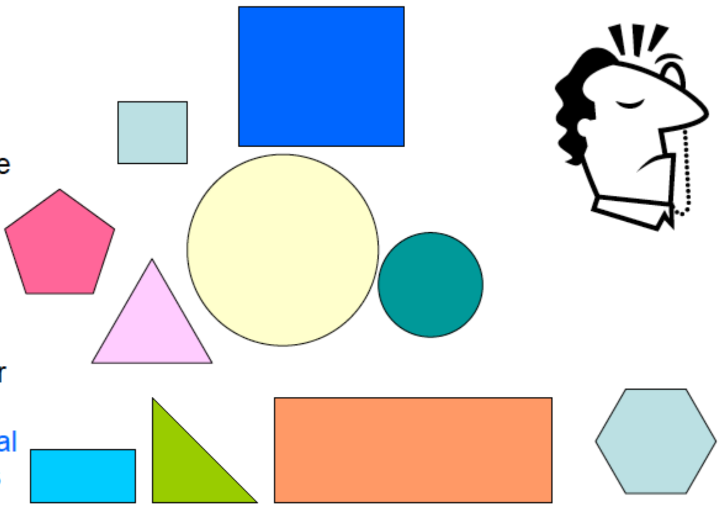
What do you notice? Objects!

36

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles

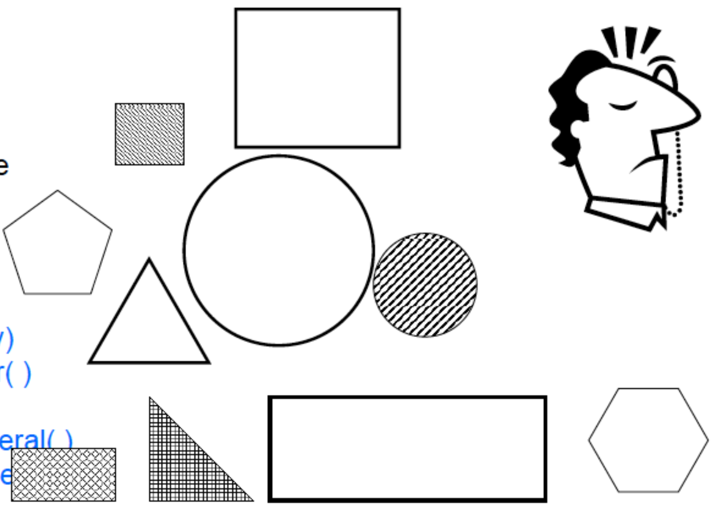
What do you notice? Properties!

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles



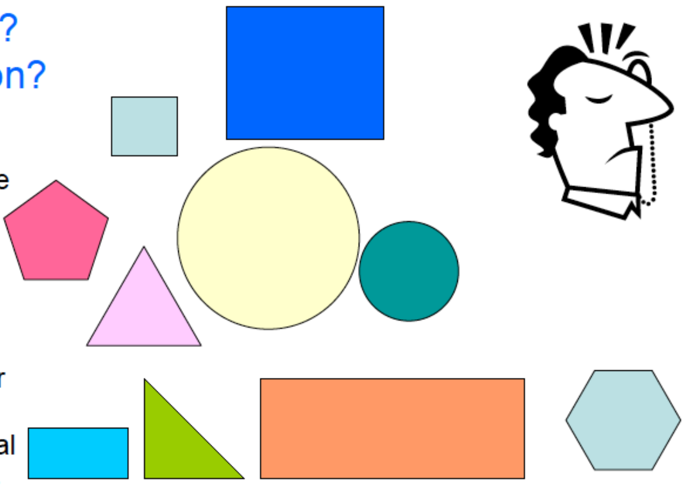
What do you notice? Behavior

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw( )
- Move(x, y)
- Perimeter( )
- Area( )
- Is Equilateral( )
- Is Isoscele



And What else?

- Shape?
- Polygon?
- Circle
- Triangle
- Rectangle
- Square
- Color
- Point
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles

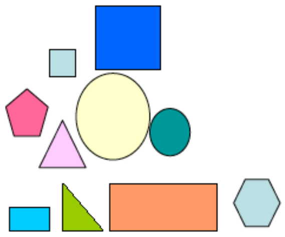


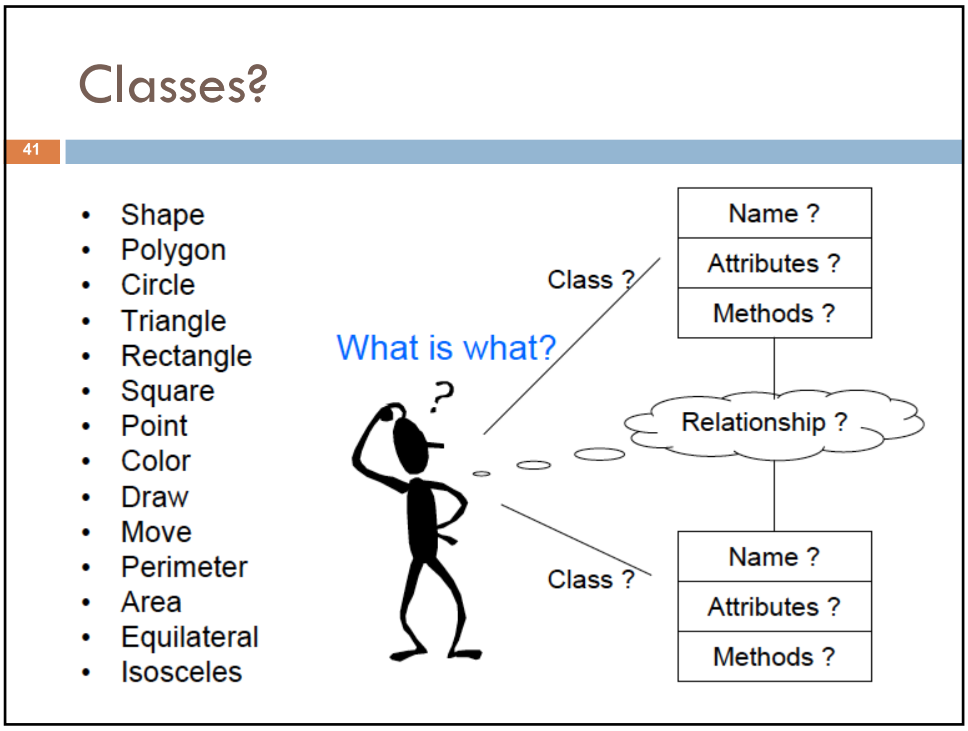
And What else?

- Shape?
- Polygon?
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles

- All are Shapes !
- Some of them are Polygons !
- A “kind-of” relationship
- Both are abstract !

- A Polygons consists of a number of points
- A “has-a” or “part-of” relationship!

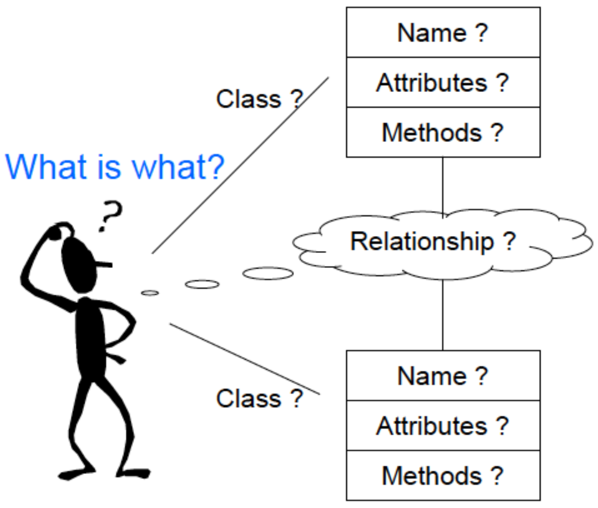


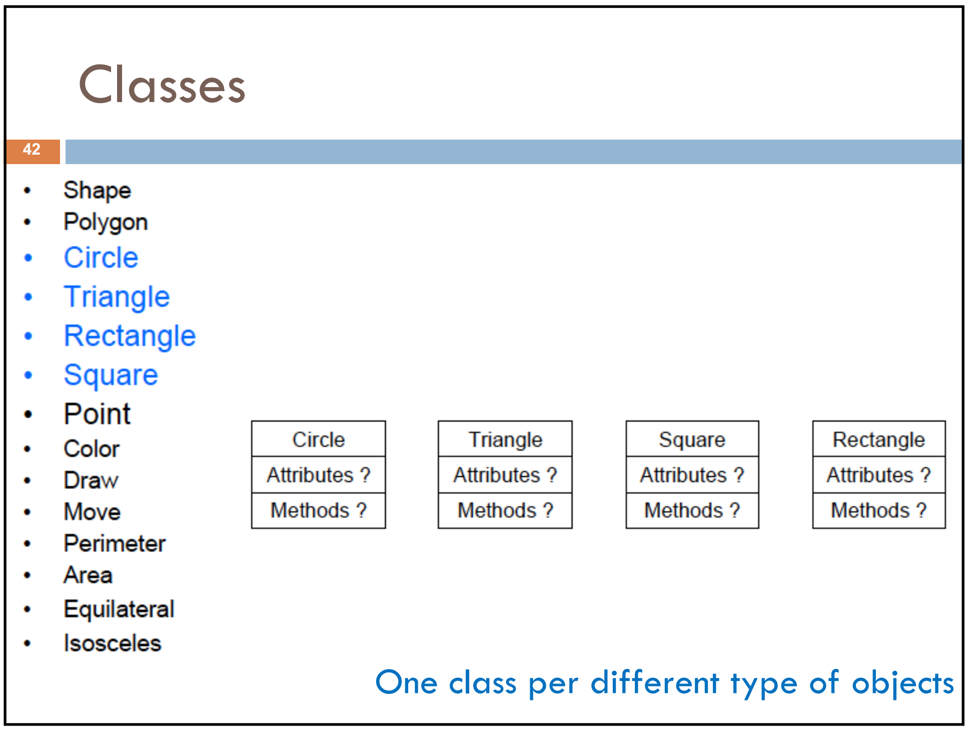


Classes?

41

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles





Classes

42

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles

Circle
Attributes ?
Methods ?

Triangle
Attributes ?
Methods ?

Square
Attributes ?
Methods ?

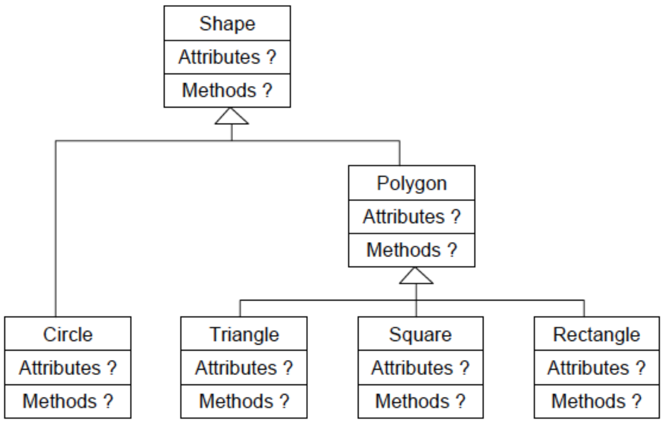
Rectangle
Attributes ?
Methods ?

One class per different type of objects

Classes

43

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles

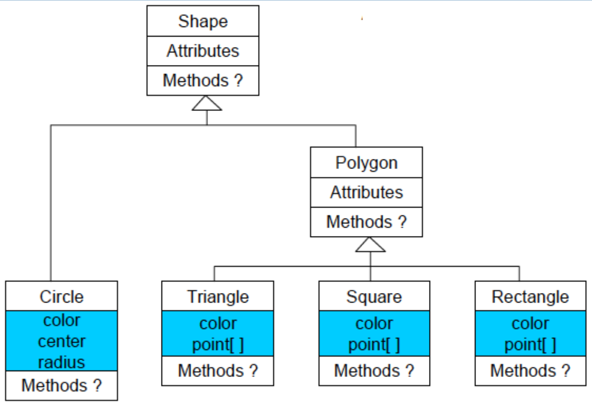


One class per different type of objects

Attributes

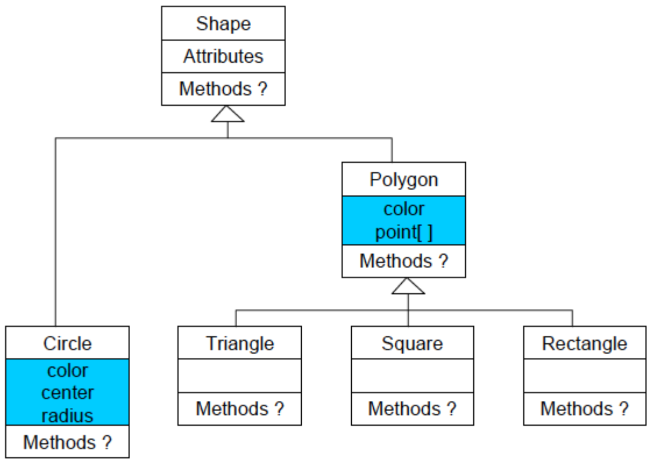
44

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles
- getColor
- setColor



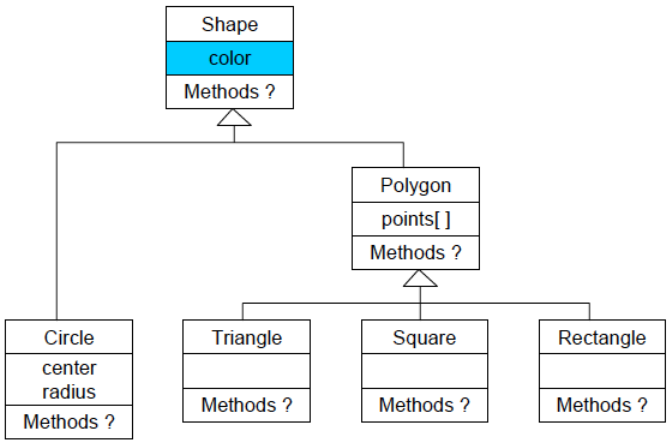
Attributes

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles
- getColor
- setColor



Attributes

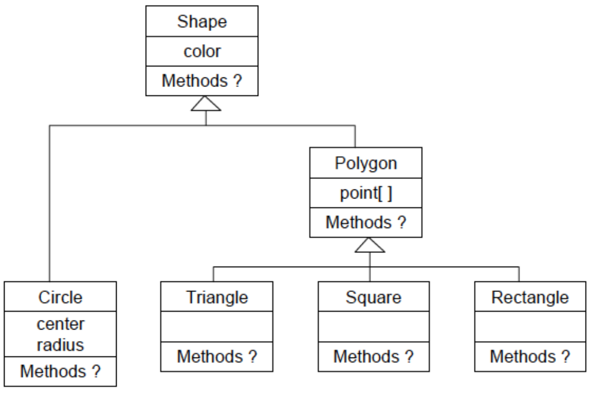
- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles
- getColor
- setColor



Methods

47

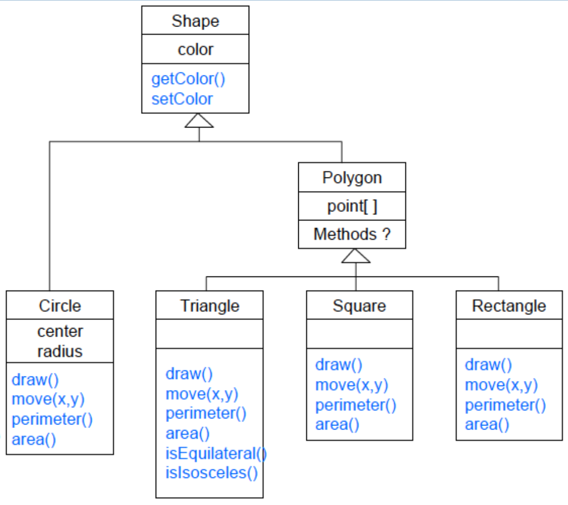
- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- Draw
- Move
- Perimeter
- Area
- Equilateral
- Isosceles
- getColor
- setColor

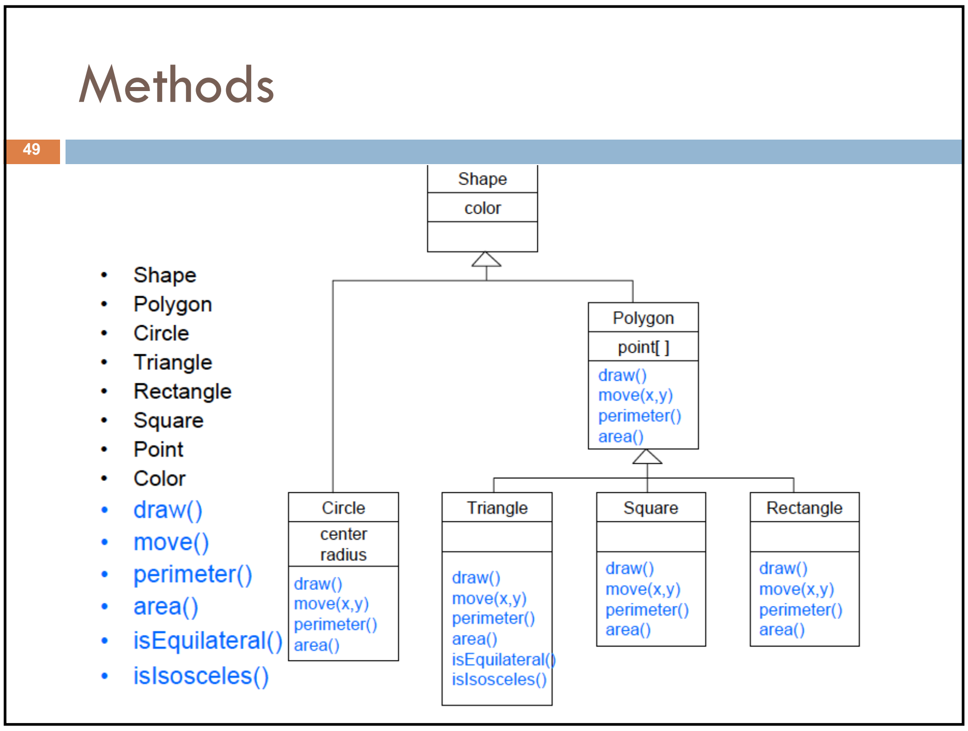


Methods

48

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- draw()
- move()
- perimeter()
- area()
- isEquilateral()
- isIsosceles()
- getColor()
- setColor()

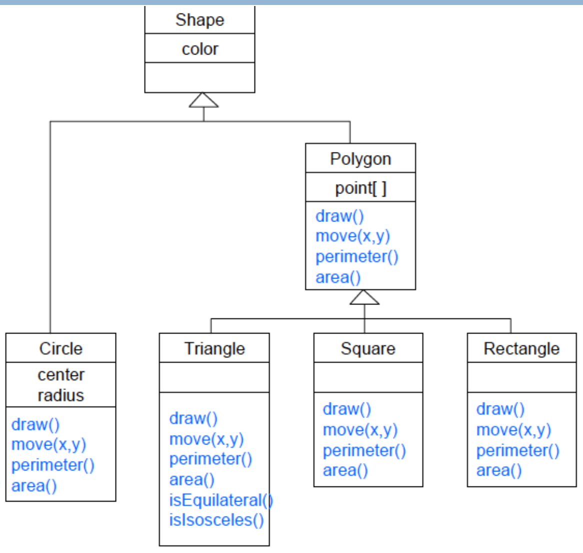


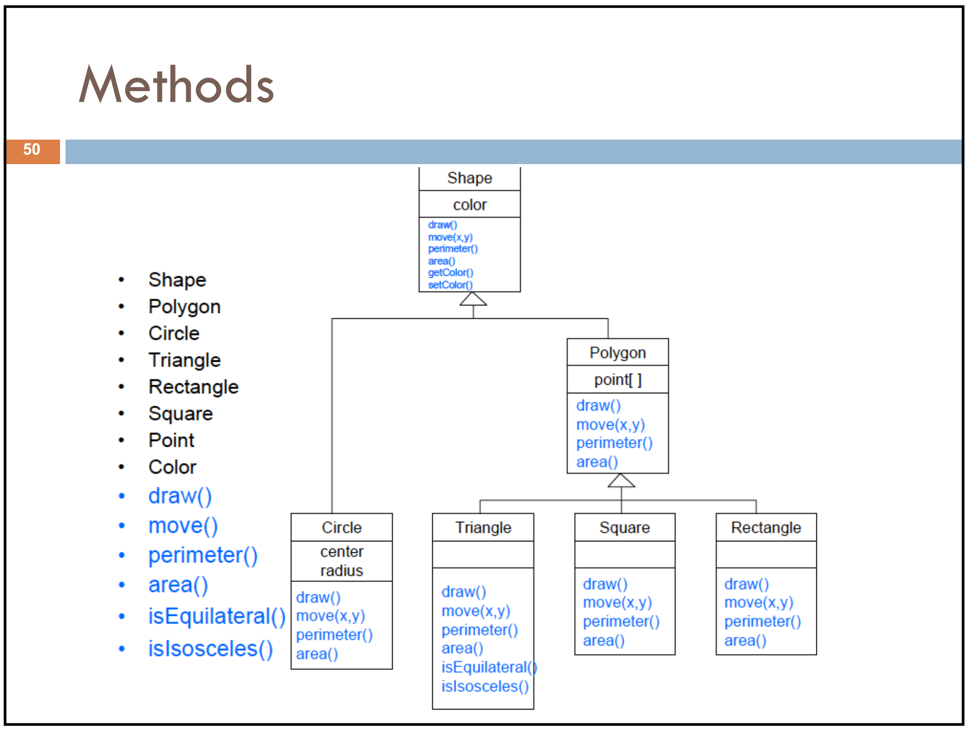


Methods

49

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- draw()
- move()
- perimeter()
- area()
- isEquilateral()
- isIsosceles()

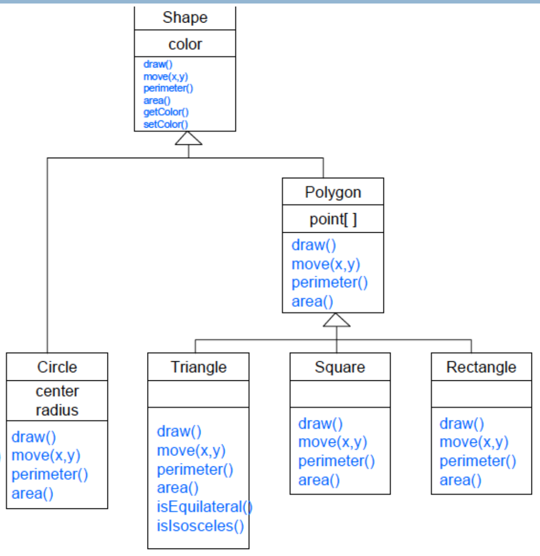




Methods

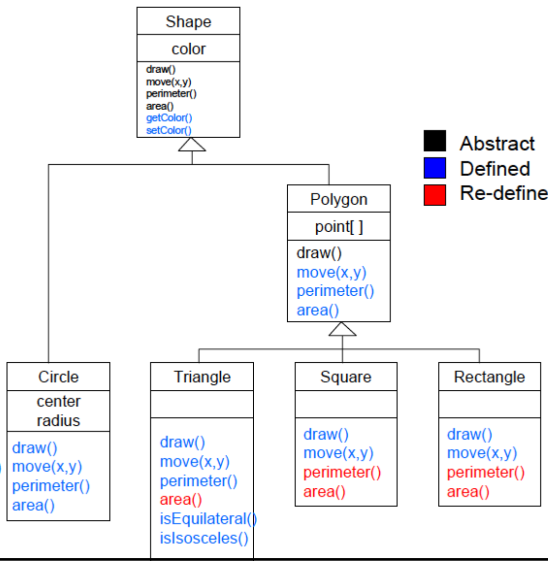
50

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- draw()
- move()
- perimeter()
- area()
- isEquilateral()
- isIsosceles()



Methods

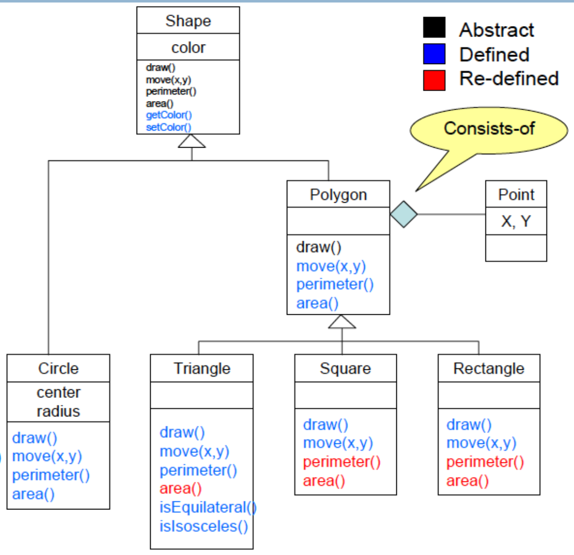
- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- draw()
- move()
- perimeter()
- area()
- isEquilateral()
- isIsosceles()



■ Abstract
🟦 Defined
🟥 Re-defined

Final Thoughts

- Shape
- Polygon
- Circle
- Triangle
- Rectangle
- Square
- Point
- Color
- draw()
- move()
- perimeter()
- area()
- isEquilateral()
- isIsosceles()



■ Abstract
🟦 Defined
🟥 Re-defined