Object-Orientation

(Why? Real-world examples; Industry practices)

Saurabh Tiwari, PhD
DAIICT Gandhinagar

Monday, 07 June 2021

Complexity?

2

A Simple Example on Complexity

3

Our capacity with Complexity

4

What we can infer?

5

- Our ability as human beings breaks at a point when complexity (chaos) increases.
- We have technique to tackle complexity (e.g., counting, arranging in order, diagrams etc.)
- Code is 1s and 0s at the bottom level (which is hard to think of!).
- High level Languages (e.g., Python, Java, C++, C#, Visual Basic, JavaScript) helps but not entirely!
- **We manages complexity in our daily life through OO (may be unknowingly)**

Is the world around us a script in execution?

6

Which way you think?

7

The Six Blind Men of Indostan

8

The Blind Men and the Elephant American poet John Godfrey Saxe (1816-1887) based the following poem on an Indian folktale.

The Six Blind Men of Indostan

9

It was six men of Indostan, To learning much inclined, Who went to see the elephant, (Though all of them were blind), That each by observation might satisfy his mind.

The first approached
the elephant,
And happening to fall
Against his broad and
sturdy side,
At once began to bawl:
"God bless me! But the
elephant
Is very like a wall!"

The second, feeling of the tusk,
Cried: "Ho! What have we here,
So very round and smooth and
sharp?
To me 'tis very clear,
This wonder of an elephant
Is very like a spear!"

The Six Blind Men of Indostan

10

The third approached the
animal,
And happening to take
The squirming trunk within his
hands,
Thus boldly up and spake:
"I see," quoth he, "the elephant
Is very like a snake!"

The fourth reached out an eager
hand,
And felt about the knee.
"What most this wondrous beast is
like is might plain," quoth he;
"Tis clear enough the elephant
Is very like a tree."

The fifth, who chanced to touch
the ear,
Said: "E'en the blindest man
Can tell what this resembles most:
Deny the fact who can,
This marvel of an elephant
Is very like a fan."

The sixth no sooner had begun
About the beast to grope,
Than seizing on the swinging
tail
That fell within his scope,
"I see," quoth he, "the elephant
Is very like a rope."

The Six Blind Men of Indostan

11

And so these men of Indostan Disputed loud and long, Each in his own opinion Exceeding stiff and strong.

Though each was partly right, All were in the wrong.

Handling Complexity

12

1. Abstraction
2. Decomposition
3. Modularity
4. Hierarchy

Abstraction

13

- Abstraction allows us to ignore unessential details

Technique to deal Complexity

14

- A technique used to master complexity ("divide and conquer")
- Two major types of decomposition
  - Functional decomposition
  - Object-oriented decomposition

- Functional decomposition
  - The system is decomposed into modules
  - Each module is a major function in the application domain
  - Modules can be decomposed into smaller modules.

Decomposition

15

- Object-oriented decomposition
  - The system is decomposed into classes ("objects")
  - Each class is a major entity in the application domain
  - Classes can be decomposed into smaller classes
- Object-oriented vs. functional decomposition

Which decomposition is the right one?

Functional Decomposition

16

Functional Decomposition:

Autoshape

17

Object-Oriented View

18

What is this?

19

An Eskimo!

20

A Face!

21

Depending on the purpose of the system different objects might be found: A nose is suddenly an elbow, hair is a cave, an ear turns out to be a glove.

Modularity

22

Modularization is the process of dividing a software system into multiple independent modules where each module works independently.

Advantages of Modularization:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

Highly cohesive structure

- Loosely coupled
- Interactions through clearly defined

Modularity

23

Coupling:

Coupling is the measure of the degree of interdependence between the modules.

A good software will have low coupling.

Cohesion:

Cohesion is a measure of the degree to which the elements of the module are functionally related.

OR

degree to which all elements directed towards performing a single task are contained in the component.

A good software design will have high cohesion.

Modularity

24

Hierarchies

25

We got abstractions and decomposition

- This leads us to chunks (classes, objects) which we view with object model
- This should be modular

- Another way to deal with complexity is to provide simple relationships between the chunks

- One of the most important relationships is hierarchy

Two important hierarchies
– "Part of" hierarchy
– "Is-kind-of" hierarchy

26

27

Object-Orientation?

Real World Objects

28

Real World Objects

29

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

Real World Objects

30

- Some Object contains other Objects

Software Objects

31

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

Software Object: Cycle

32

```
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

33

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

Class & Object

34

Class

Blueprint of a car

Object

Car

Class

35

Class and Objects

36

OO- Based on the Objects

37

Four Pillars of OO

38

Encapsulation

39

Abstraction

40

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

Inheritance

41

Hierarchy of Classes

Polymorphism

42

Example: Four Pillars

43

- 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

44

- 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

45

- 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

46

- 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

47

- 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

48

- 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

49

- 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

50

- 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

51

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

PostalCode
code *destination*
int validate()

Solution Approach 1

52

- **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

53

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

PostalCode
code destination
bool isCanadianCode() bool isUSCode() bool isUKCode()

Solution Approach 2

54

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

PostalCode
code destination
bool isCanadianCode() bool isUSCode() bool isUKCode() String getCanadianDest() String getUSDest() String getUKDest()

Solution Approach 3

55

Solution Approach 3 (cont..)

56

57

Identifying Objects and how they are related?

Overview

58

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

Identifying Objects

59

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?

60

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?

61

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

What do you notice? Objects!

62

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

What do you notice? Properties!

63

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

What do you notice? Behavior

64

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

And What else?

65

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

And What else?

66

- 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?

67

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

Classes

68

- 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

69

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

One class per different type of objects

Attributes

70

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

Attributes

71

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

Attributes

72

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

Methods

73

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

Methods

74

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

Methods

75

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

Methods

76

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

Methods

77

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

Final Thoughts

78

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

Real-World Example

79

C Code

Ladder

Read, Scale, Filter Inputs
&
Perform Input Alarming

Perform Control
&
Control Alarming

Perform Output Alarming
&
Filter, Scale, Write Outputs

Real-World Example

80

Real-World Example

81

Multiple Instances of the same object type

Real-World Example

82

Encapsulation

Think and Organize!

83

Email:
saurabh_t@daiict.ac.in

Thanks for attention ☺