

Intro to Computer Science II

Object-Oriented Design

Object-Oriented Design is the process of planning a system of interacting *objects* for the purpose of solving a software problem. It is one approach to software design.

1

Object-Oriented Design is the process of planning a system of interacting *objects* for the purpose of solving a software problem. It is one approach to software design.

- **Objects** – Any physical or logical elements.
- Objects are distinguished first by their classification (or just **class**)
 - Objects classified as **Dogs** are different than you and I, which are classified as **Human**
- Objects of a specific class are called **instances** of the class.
 - You and I are instances of **Human**

- Objects have a set of *characteristics* that make them unique
 - What are some of our characteristics that make each of us unique?
 - Eye color, Hair color, Sleeping, Hungry
 - In O-O terminology, these are called **attributes**, or **fields**, or **properties**
- Characteristics (**attributes**) have *values*
 - These values determine the **state** of an object at any time
 - Most values are temporal, changing over time (for example, hair!)
 - NOTE - If they are not temporal, then they may make good named constants in your code

Fruit

- Characteristics (attributes)
 - Name
 - Color
 - Weight
- Methods
 - be_eaten()

Examples of Classes and Objects

Apple(Fruit)

- Additional attributes
 - None
- Additional methods
 - throw()

Orange(Fruit)

- Additional features
 - None
- Additional methods
 - squeeze()

Person

- Attributes
 - Name
 - Age
 - Place of birth
- Methods
 - eat()
 - walk()
 - sleep()

Student(Person)

- Attributes
 - Major
 - Class year
 - GPA
- Methods
 - attend_class()
 - take_exam()
 - play_club_sports()

Employee(Person)

- Attributes
 - Department
 - Work schedule
- Methods
 - get_paid()
 - attend_meeting()

Your example(s)?

Encapsulation and O-O design

• Encapsulation

- The grouping of data and methods together into one package in such a way that the internal representation of the object is hidden
- All interaction with the object is performed only through the object's methods
- **Why is encapsulation an important part of the design process?**
 - An object should always manage its own internal state!
 - An object is responsible for itself and how it carries out its own actions

Encapsulation Example

• Our Array class example:

- How Array class is defined is hidden, whether an array of ctype objects, or a Python list
- To the outside world, all we need to know is how to use it

```
grades = Array2D(7, 3)
```

OO Design: Coupling vs. Cohesion

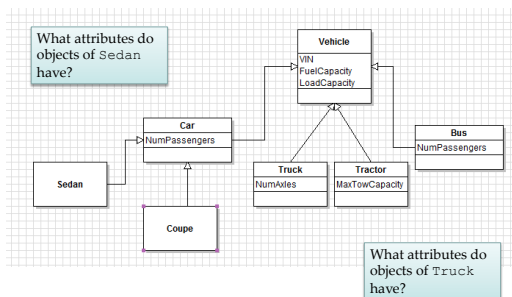
- **Coupling** – (aka dependency) – the degree to which each object relies on all of the other objects in the system
- **Cohesion** – the degree to which all of the functionality in an object are related
- What does a good OOD strive for?
 - **Low coupling**
 - High coupling means high interclass dependencies
 - Minimize coupling to avoid a "snowball effect" of change in one class
 - **High cohesion**
 - All public data and methods should all be related directly to the concept the class represents

Relationship: Inheritance

- The strongest class relationship
- Models the “is-a” relationship
- From an SE view, inheritance is POWERFUL, yet simple concept.
 - Idea – extend what you already have by adding only those capabilities / features you need
 - It can save an enormous amount of development time through code reuse!

16

Example: Vehicles



Code Example for Class Bird

```

class Bird:
    color = 'Yellow' # class field

    def __init__(self): #Constructor
        self.weight = 10 #instance field

    def fly(self):
        if self.weight > 15:
            self.lightenTheLoad() # calling a method
        else:
            print("FLYING!!!!")

    def lightenTheLoad(self):
        print('Splat!')

    def eat(self, food):
        self.weight = self.weight + food
  
```

```
big_bird = Bird()
```

```
big_bird.fly()
```

```
big_bird.eat(20)
```

```
big_bird.fly()
```

```

class Bird:
    color = 'Yellow' # class field
    def __init__(self): #Constructor
        self.weight = 10 #instance field
    def fly(self):
        if self.weight > 15:
            self.lightenTheLoad() # calling a method
        else:
            print("FLYING!!!!")
    def lightenTheLoad(self):
        print('Splat!')
    def eat(self, food):
        self.weight = self.weight + food

```

```

class Penguin(Bird): # Penguin inherits class Bird
    # Overriding the Bird constructor
    def __init__(self):
        Bird.__init__(self)
    # Overriding the Bird fly method
    def fly(self):
        print("What?? Penguins don't fly.")
    def swim(self):
        if self.weight > 15:
            self.lightenTheLoad()
            print('Splash! Splash')

```

```

wheezy = Penguin()
wheezy.fly()
wheezy.eat(10)
wheezy.swim()

```

Design Exercise

- Take out your computer
- Write the code for class Vehicle and its subclasses Car and Truck in a file named *vehicle.py*
- Write the code in a separate file named *vehicle_app.py* for testing the Vehicle class that creates a few Car and Truck objects and prints their information.