COEN 5830, Fall 2024
Introduction to Robotics
Lecture 6
Object-Oriented Programming 2
Encapsulation and Inheritance

Leopold Beuken (leopold.Beuken@Colorado.edu)

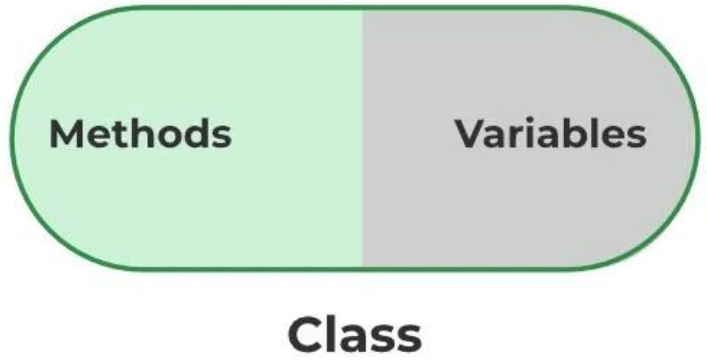
Thursday, 11/10/2024

Administrative

- Can you access the Robotics Seminar recordings?
- HW1 Posted on Canvas, due **9/17** at **11:59pm**
  - Please answer all questions in a **single** .py file

Encapsulation

- A **client** is a program which uses a class (or instances of classes)
- A class offers **services** to clients through which objects are accesses
  - The use of classes should be as simple as possible from the client's perspective
  - The integrity of any object is always preserved (state of object remains acceptable)
- The integrity of an object typically implies that **attributes** are of an **acceptable nature.**
- **Encapsulation** is one way of ensuring the **integrity** of objects by **hiding** attributes and/or methods from the client. Rather provide the client with a suitable **service** that for access.



Encapsulation - Getters and Setters

- **Getter -** A method dedicated to **returning the value** of an attribute
- **Setter –** A method dedicated to **setting the value** of an attribute

Encapsulation - Getters and Setters

- **Getter -** A method dedicated to **returning the value** of an attribute
- **Setter –** A method dedicated to **setting the value** of an attribute

```
class Wallet:
    def __init__(self):
        self.__money = 0

    # A getter method
    @property
    def money(self):
        return self.__money

    # A setter method
    @money.setter
    def money(self, money):
        if money >= 0:
            self.__money = money
```

```
wallet = Wallet()
print(wallet.money)

wallet.money = 50
print(wallet.money)

wallet.money = -30
print(wallet.money)
```

The client does not know that they are using getter and setter methods here. It just looks like they are accessing attributes directly.

Class Hierarchies (Inheritance)

- Sometimes you define a class, but then realize you need special traits for some, but not all instances of the same class.

```python
class Student:

    def __init__(self, name: str, id: str, email: str, credits: str):
        self.name = name
        self.id = id
        self.email = email
        self.credits = credits

class Teacher:

    def __init__(self, name: str, email: str, room: str, teaching_years: int):
        self.name = name
        self.email = email
        self.room = room
        self.teaching_years = teaching_years
```

Class Hierarchies (Inheritance)

- Sometimes you define a class, but then realize you need **special traits for some, but not all instances of the same class**.
  - We already know it's a bad idea to define class traits in our main program outside of the class definition
  - It also does not make sense to combine the two classes together, as some traits would not be used by some objects

```python
class Student:

    def __init__(self, name: str, id: str, email: str, credits: str):
        self.name = name
        self.id = id
        self.email = email
        self.credits = credits

class Teacher:

    def __init__(self, name: str, email: str, room: str, teaching_years: int):
        self.name = name
        self.email = email
        self.room = room
        self.teaching_years = teaching_years
```

Inheritance helps us solve this problem. It allows **child classes** to **inherit** traits **from parent classes**. In addition to inherited traits, the child class can also have its **own unique traits**.

Access Modifiers

- Traits defined as **private** in parent classes are **not directly accessible** in child classes.
- **Protected** traits are accessible to derived **child classes**, but still **not available** to **clients** that use the classes

Access modifier	Example	Visible to client	Visible to derived class
Public	`self.name`	yes	yes
Protected	`self._name`	no	yes
Private	`self.__name`	no	no

Protected traits are defined using a **single underscore**. Note that this is merely a **naming convention** and these traits are technically still available to clients. It is considered **bad practice** for clients to access these traits though.