327 Object-oriented Programming

Lecture 4

9/13/2021

Professor Barron

Today...

- getters/setters
- python “properties”
- notebook case-study
- end of Chapter 2 and starting Chapter 3
- using super()
- packing/unpacking args

Getters/Setters

- Also called accessors/mutators
- Expose private parts of an object to the public interface
- Better than using instance variables directly
  - why?
- Often a bad idea
  - Why does another object need access?
  - Could the task be achieved within the original object?

- https://www.infoworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html

Python “properties”

- Convenience of dot notation access
- Benefits of method access
- Still should only use getters/setters when needed

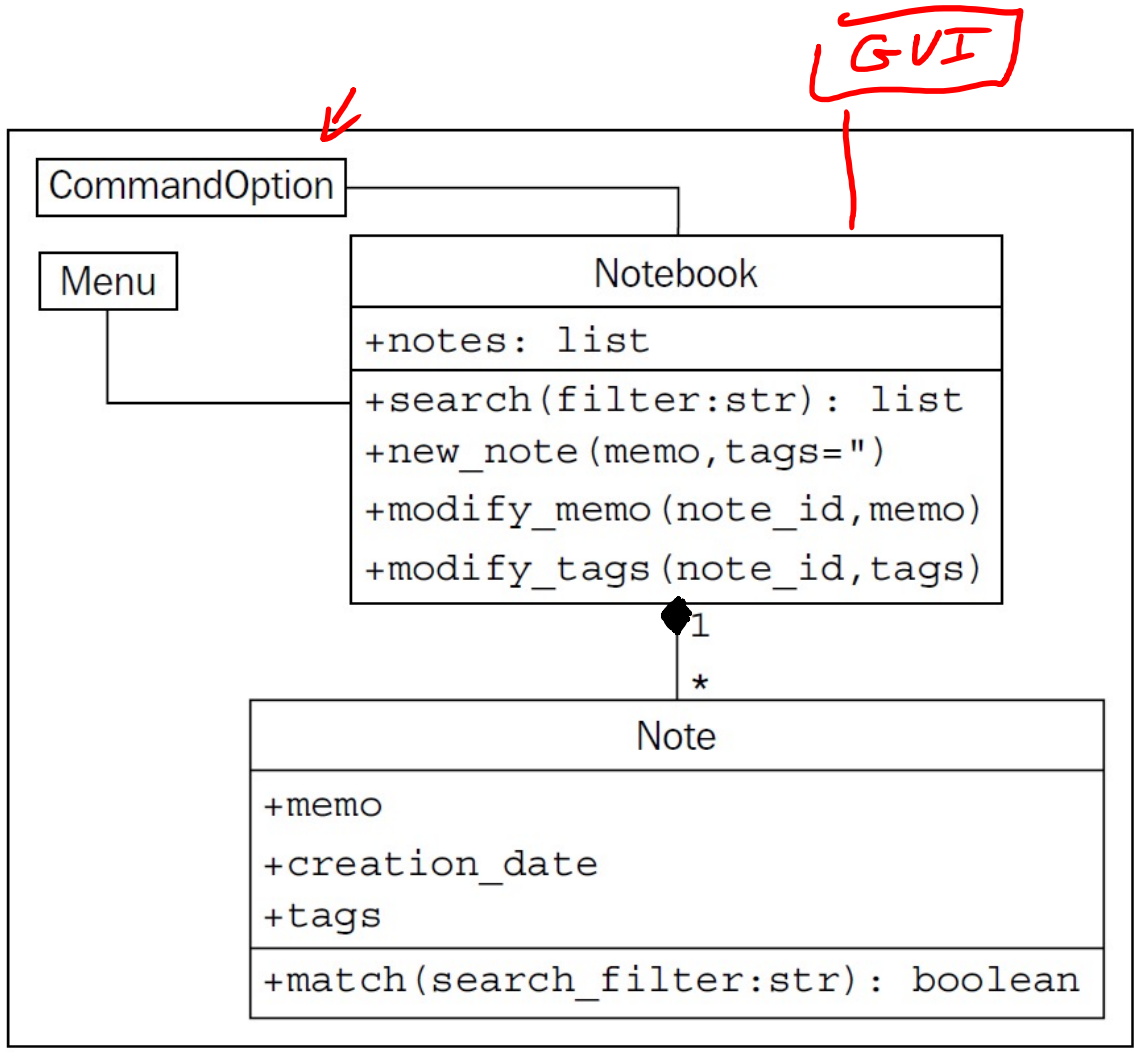
```
1   class Color:
2       def __init__(self, rgb_value, name):
3           self.rgb_value = rgb_value
4           self._name = name
5
6       def _set_name(self, name):
7           if not name:
8               raise Exception("Invalid Name")
9           self._name = name
10
11      def _get_name(self):
12          return self._name
13
14      name = property(_get_name, _set_name)
```

Python “properties”

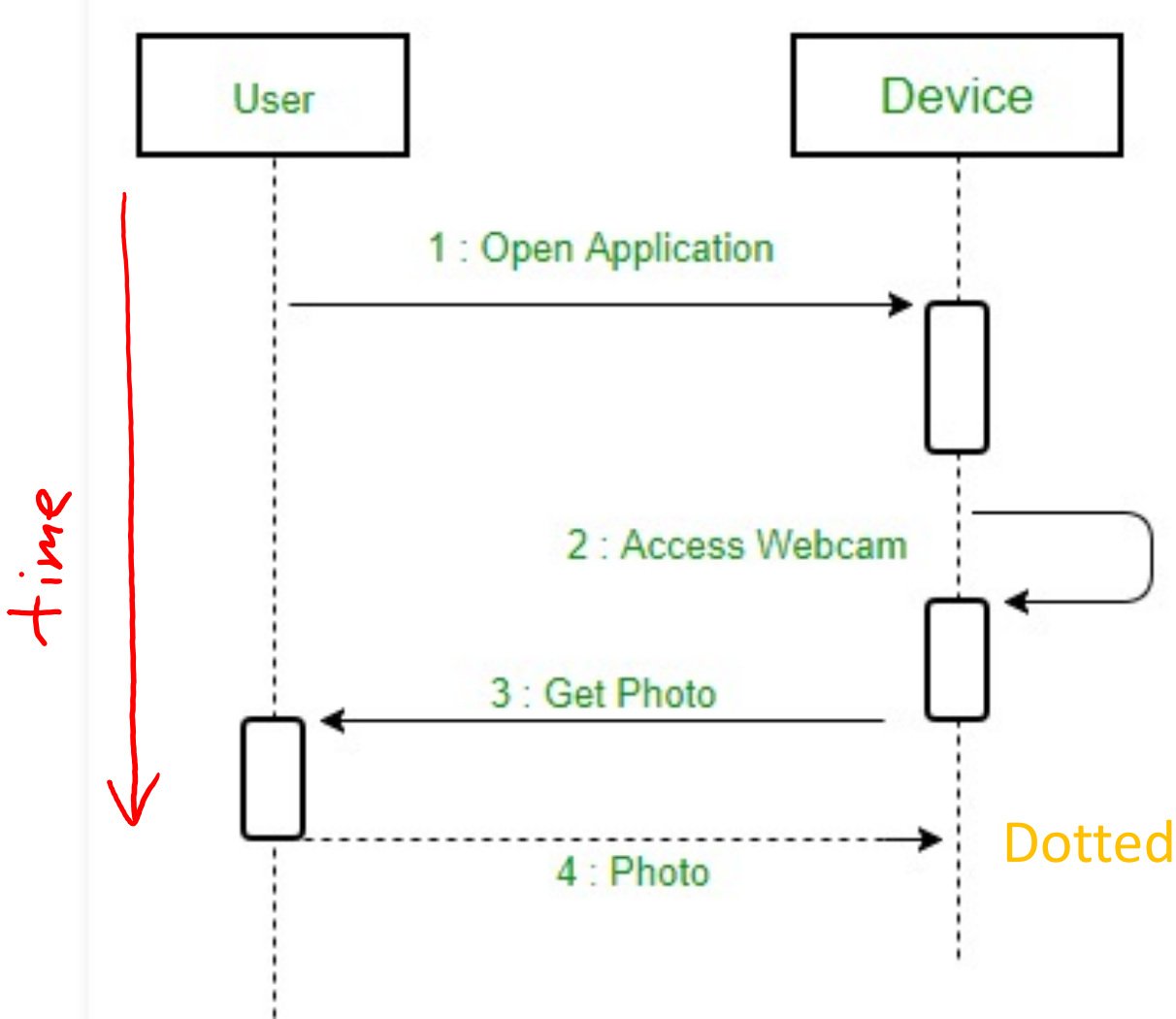
`property(`*`getter`*`, `*`setter`*`, `*`deleter`*`, `*`docstring`*`)`

```
1   class Color:
2       def __init__(self, rgb_value, name):
3           self.rgb_value = rgb_value
4           self._name = name
5
6       def _set_name(self, name):
7           if not name:
8               raise Exception("Invalid Name")
9           self._name = name
10
11      def _get_name(self):
12          return self._name
13
14      name = property(_get_name, _set_name)
```

Notebook case study



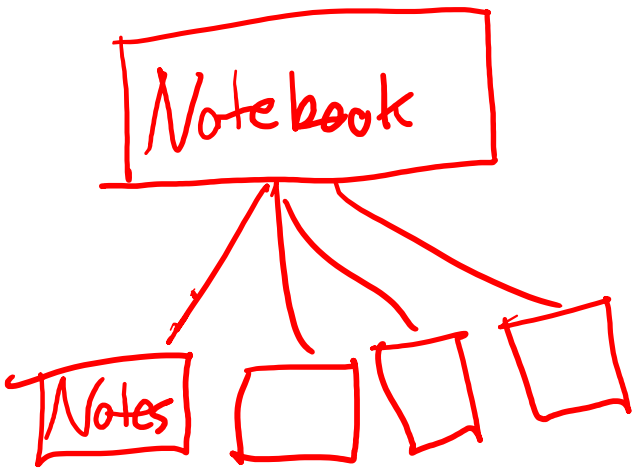
UML sequence diagram

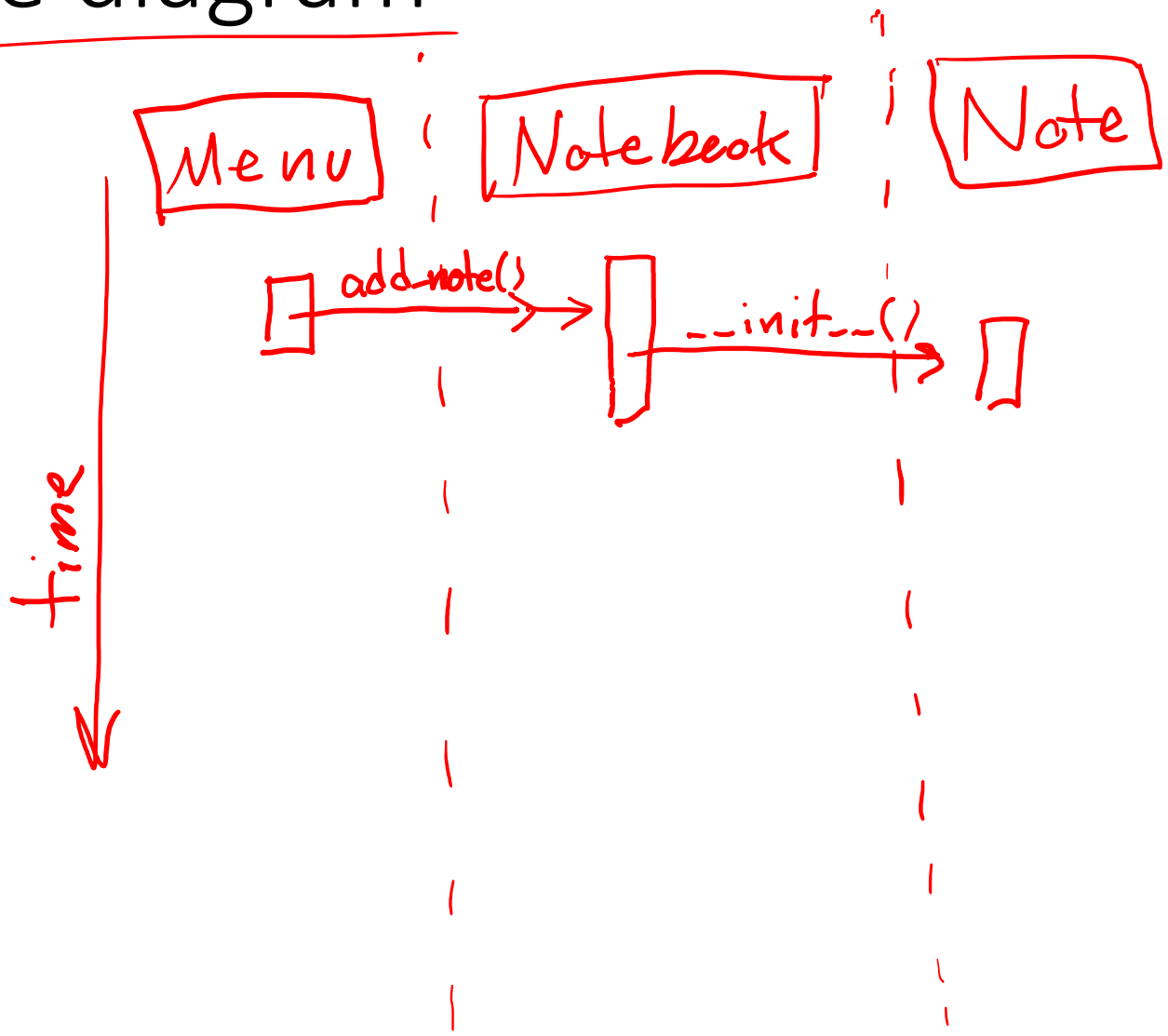


Dotted arrow usually indicates a reply message

Notebook sequence diagram

Object diagram





Using/overriding parent class

- super()
  - Same as super(CurrentClass, self)
- Often needed in __init__, but may also be used elsewhere
- May appear anywhere in a method

- If you are adding parameters then you might include those first and do this…

```
def __init__(self, new_parameter, *args, **kwargs)
    self.foo = new_parameter
    super().__init__(*args, **kwargs)
```

*args and **kwargs

- Packing arguments into tuple or dict
  - def foo(*args, **kwargs):
- Unpacking a list

  a = [1, 2, 3]

  foo(*a)
- Unpacking a dict

  a = {“a”:1, “b”:2}

  foo(**a)

```
for x in *args:
    # do something
      w/ x
```

```
def foo(a=None, b=None):
```