

Adding Classes to a Package

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



Adam Spannbauer

Machine Learning Engineer at Eastman

Object oriented programming



Anatomy of a class

working in work_dir/my_package/my_class.py

```
# Define a minimal class with an attribute
class MyClass:
    """A minimal example class

    :param value: value to set as the ``attribute`` attribute
    :ivar attribute: contains the contents of ``value`` passed in init
    """

# Method to create a new instance of MyClass
def __init__(self, value):
    # Define attribute with the contents of the value param
    self.attribute = value
```

Using a class in a package

working in work_dir/my_package/__init__.py

```
from .my_class import MyClass
```

working in work_dir/my_script.py

```
import my_package

# Create instance of MyClass
my_instance = my_package.MyClass(value='class attribute value')

# Print out class attribute value
print(my_instance.attribute)
```

```
'class attribute value'
```

The self convention

working in work_dir/my_package/my_class.py

```
# Define a minimal class with an attribute
class MyClass:
    """A minimal example class

    :param value: value to set as the ``attribute`` attribute
    :ivar attribute: contains the contents of ``value`` passed in init
    """

    # Method to create a new instance of MyClass
    def __init__(self, value):
        # Define attribute with the contents of the value param
        self.attribute = value
```

```
my_instance = my_package.MyClass(value='class attribute value')
```

Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

Leveraging Classes

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

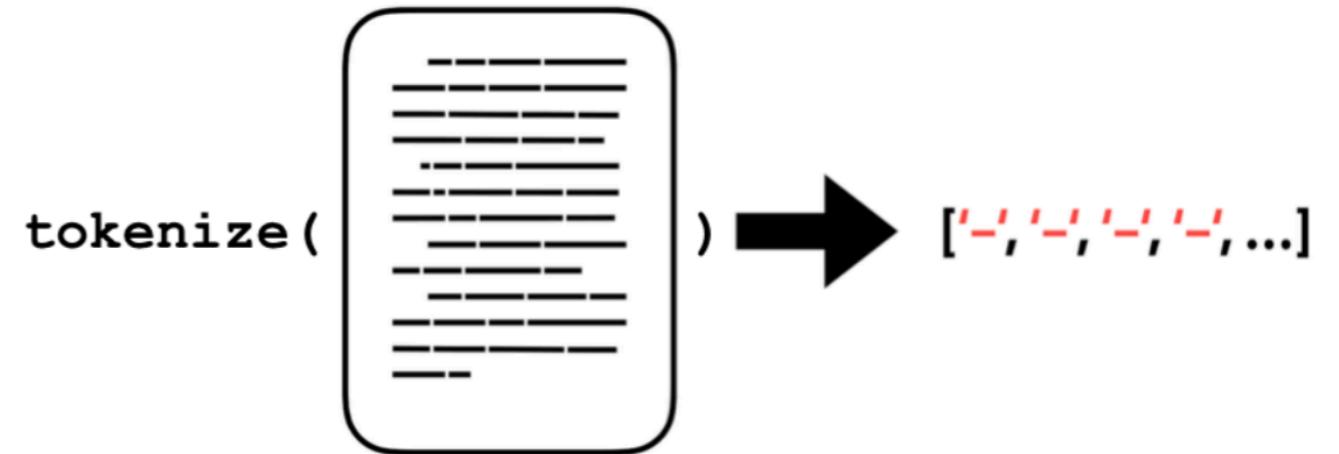


Adam Spannbauer

Machine Learning Engineer at Eastman

Extending Document class

```
class Document:  
    def __init__(self, text):  
        self.text = text
```



Current document class

```
class Document:  
    def __init__(self, text):  
        self.text = text
```

```
def __init__():  
    ...
```

```
    tokenize()
```



```
[..., ..., ..., ...]
```

Revising `__init__`

```
class Document:  
    def __init__(self, text):  
        self.text = text  
        self.tokens = self._tokenize()  
  
doc = Document('test doc')  
print(doc.tokens)
```

```
['test', 'doc']
```

Adding `_tokenize()` method

```
# Import function to perform tokenization
from .token_utils import tokenize

class Document:

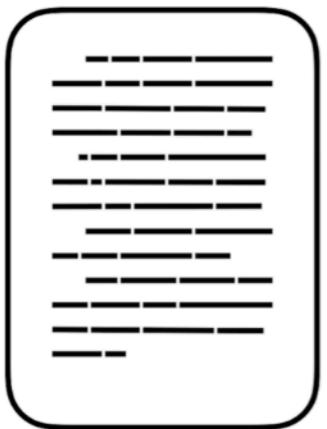
    def __init__(self, text, token_regex=r'[a-zA-Z]+'):
        self.text = text
        self.tokens = self._tokenize()

    def _tokenize(self):
        return tokenize(self.text)
```

Non-public methods

```
def __init__():
```

...



[..., ..., ..., ..., ...]



The risks of non-public methods

- Lack of documentation
- Unpredictability



Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

Classes and the DRY principle

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



Adam Spannbauer

Machine Learning Engineer at Eastman

Creating a SocialMedia Class



The DRY principle



The DRY principle



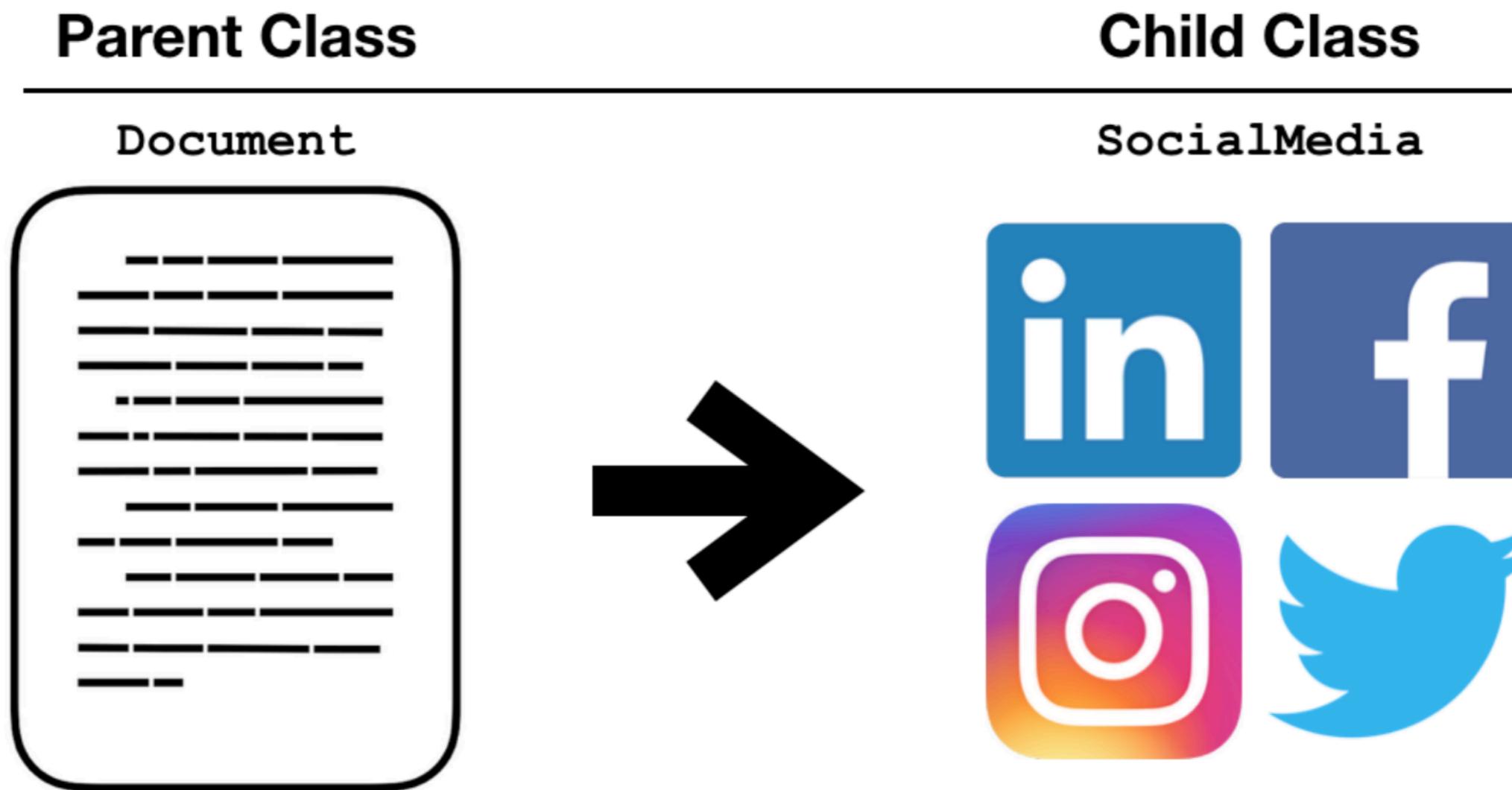
The DRY principle



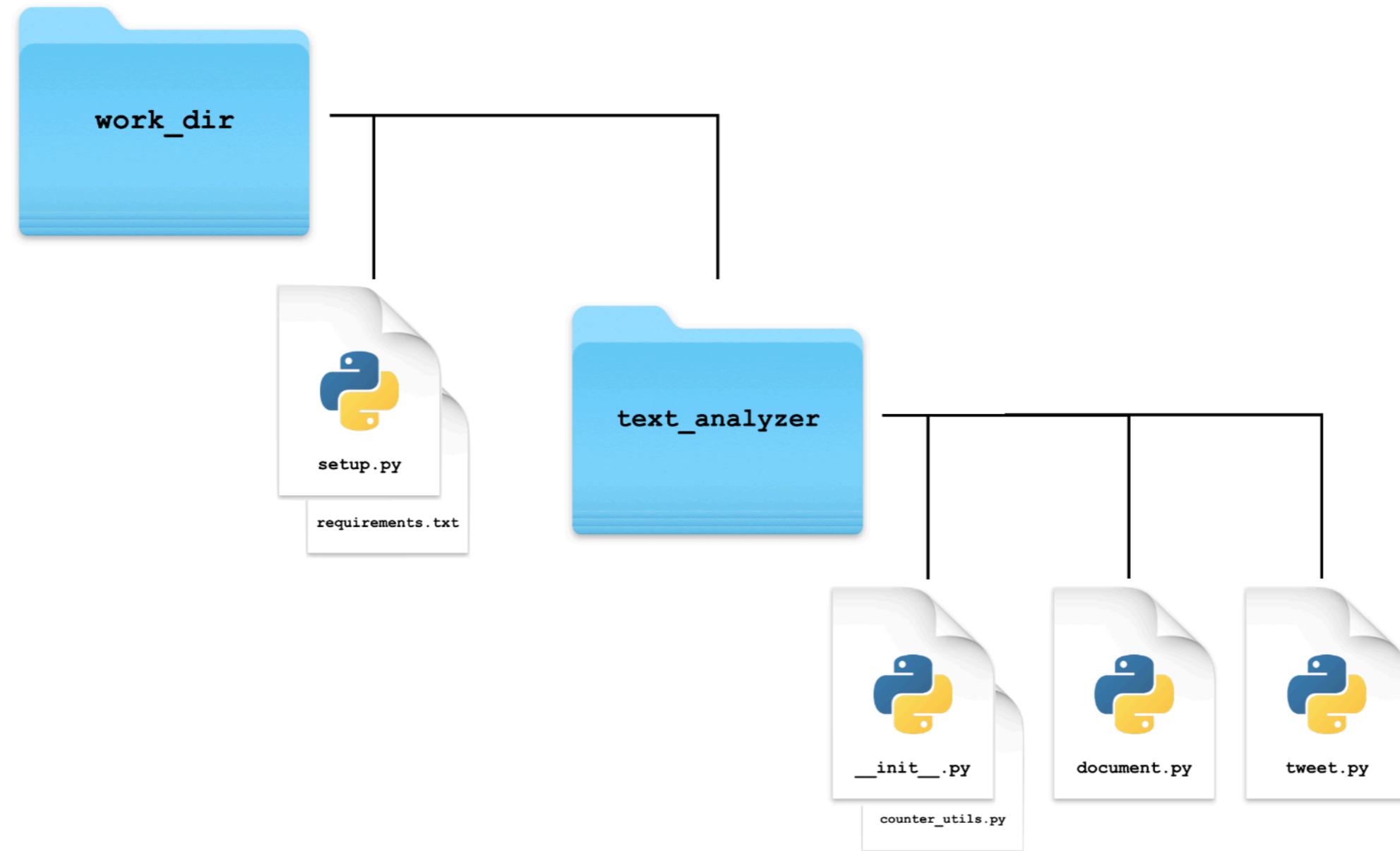
The DRY principle



Intro to inheritance



Inheritance in Python



Inheritance in Python

```
# Import ParentClass object
from .parent_class import ParentClass

# Create a child class with inheritance
class ChildClass(ParentClass):
    def __init__(self):
        # Call parent's __init__ method
        ParentClass.__init__(self)
        # Add attribute unique to child class
        self.child_attribute = "I'm a child class attribute!"

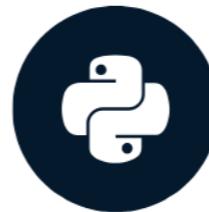
# Create a ChildClass instance
child_class = ChildClass()
print(child_class.child_attribute)
print(child_class.parent_attribute)
```

Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

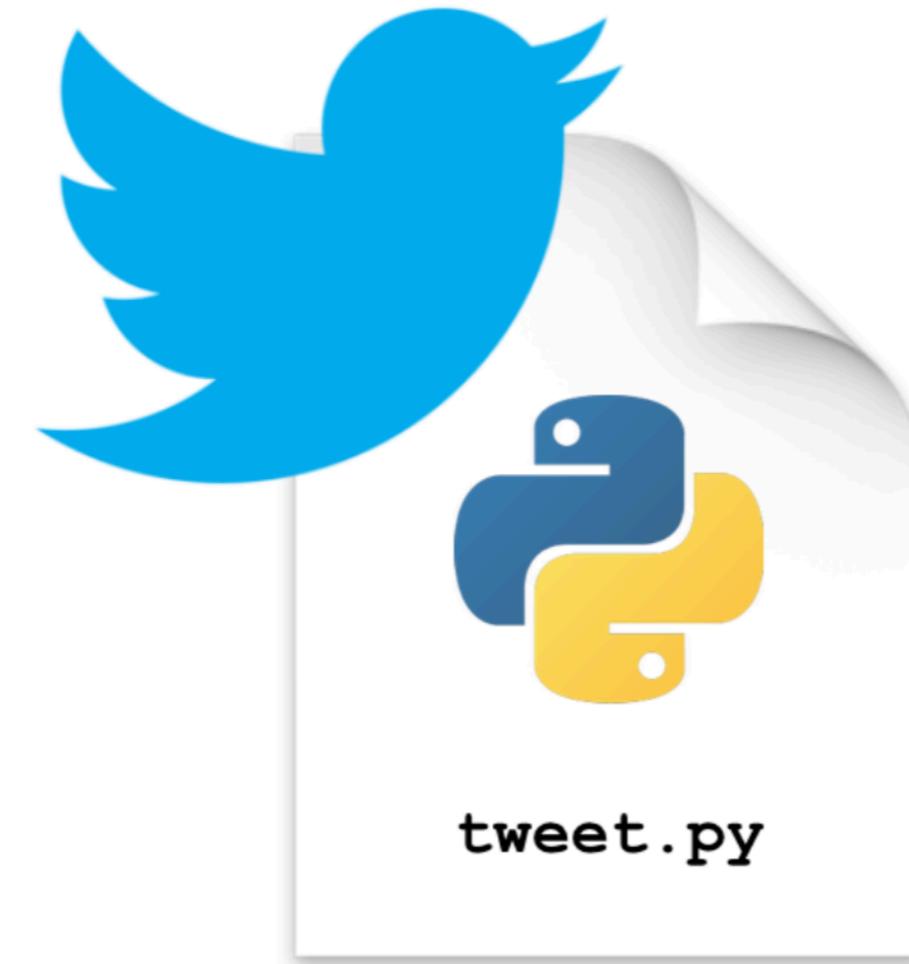
Multilevel inheritance

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

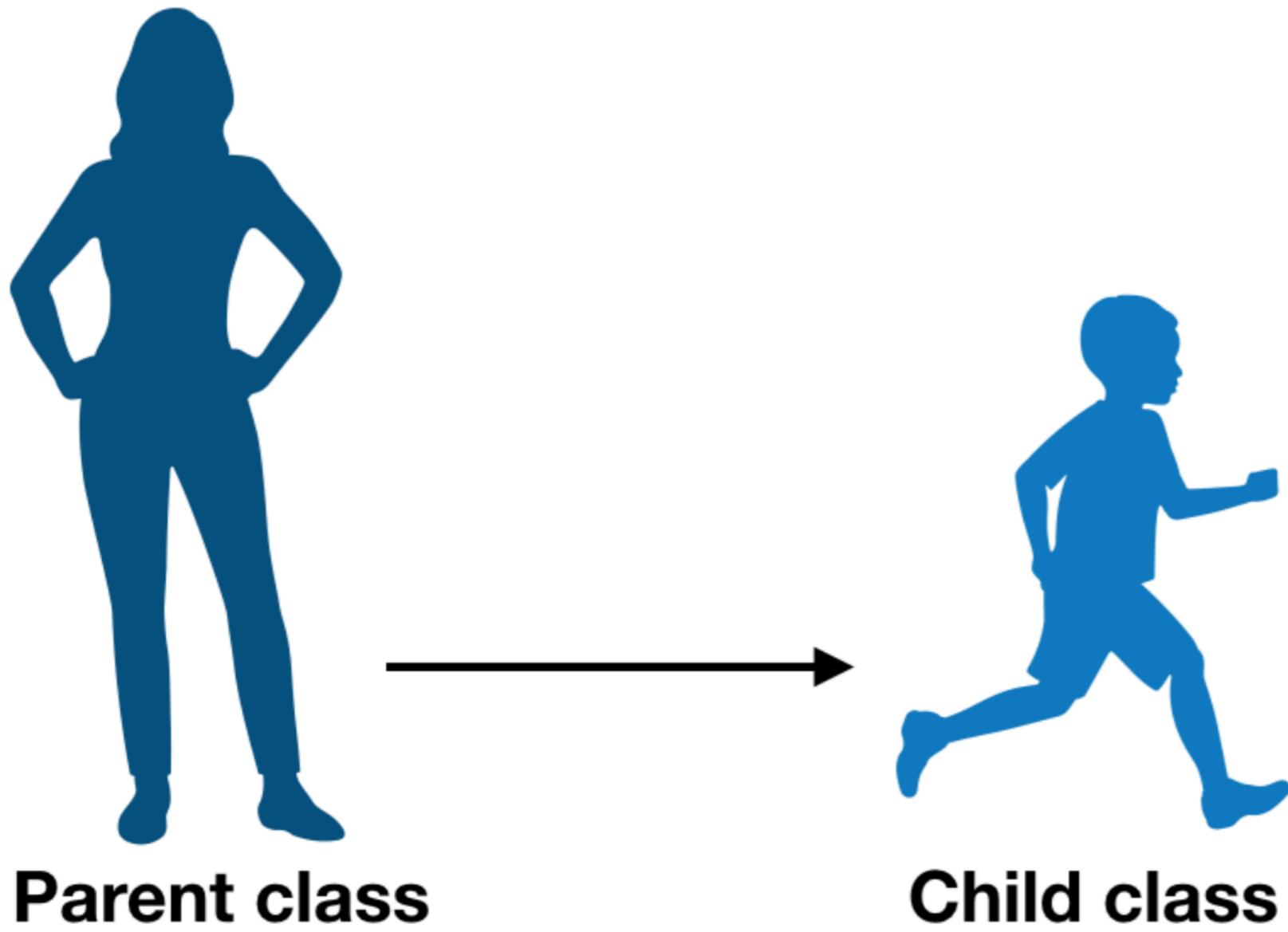


Adam Spannbauer
Machine Learning Engineer at Eastman

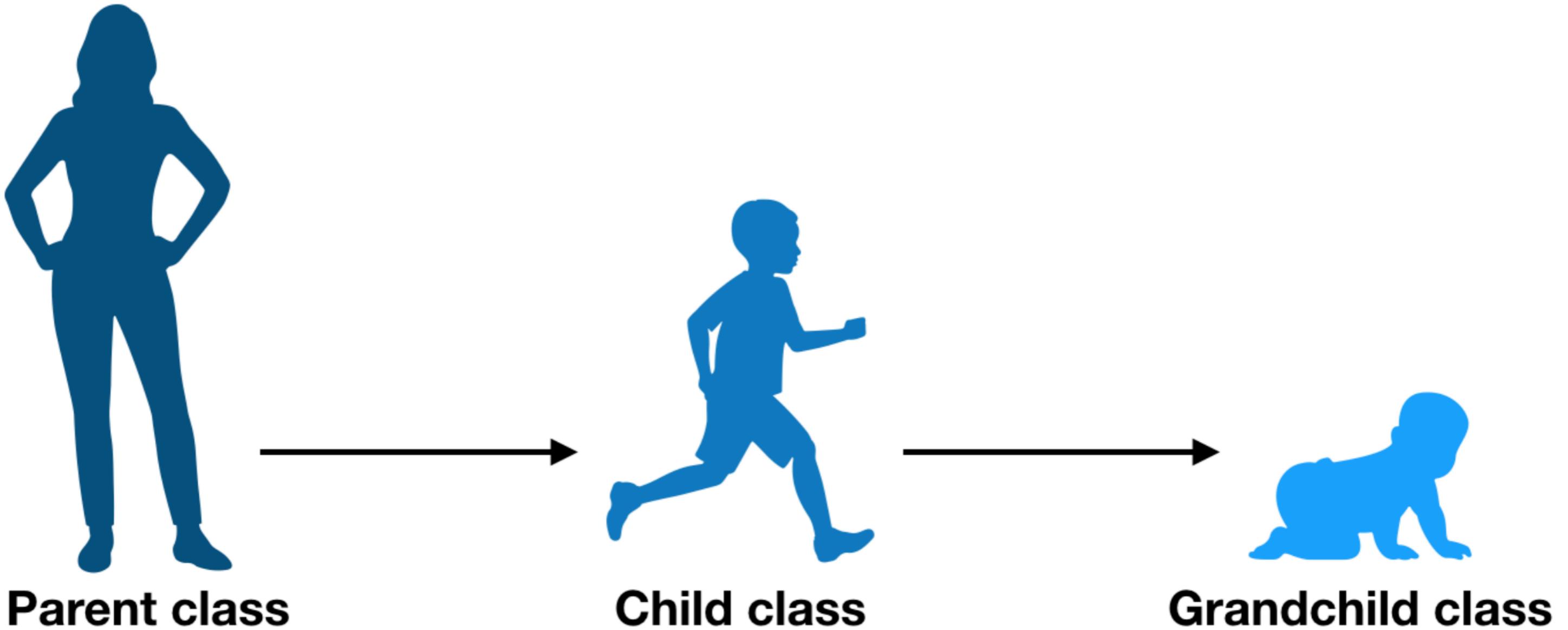
Creating a Tweet class



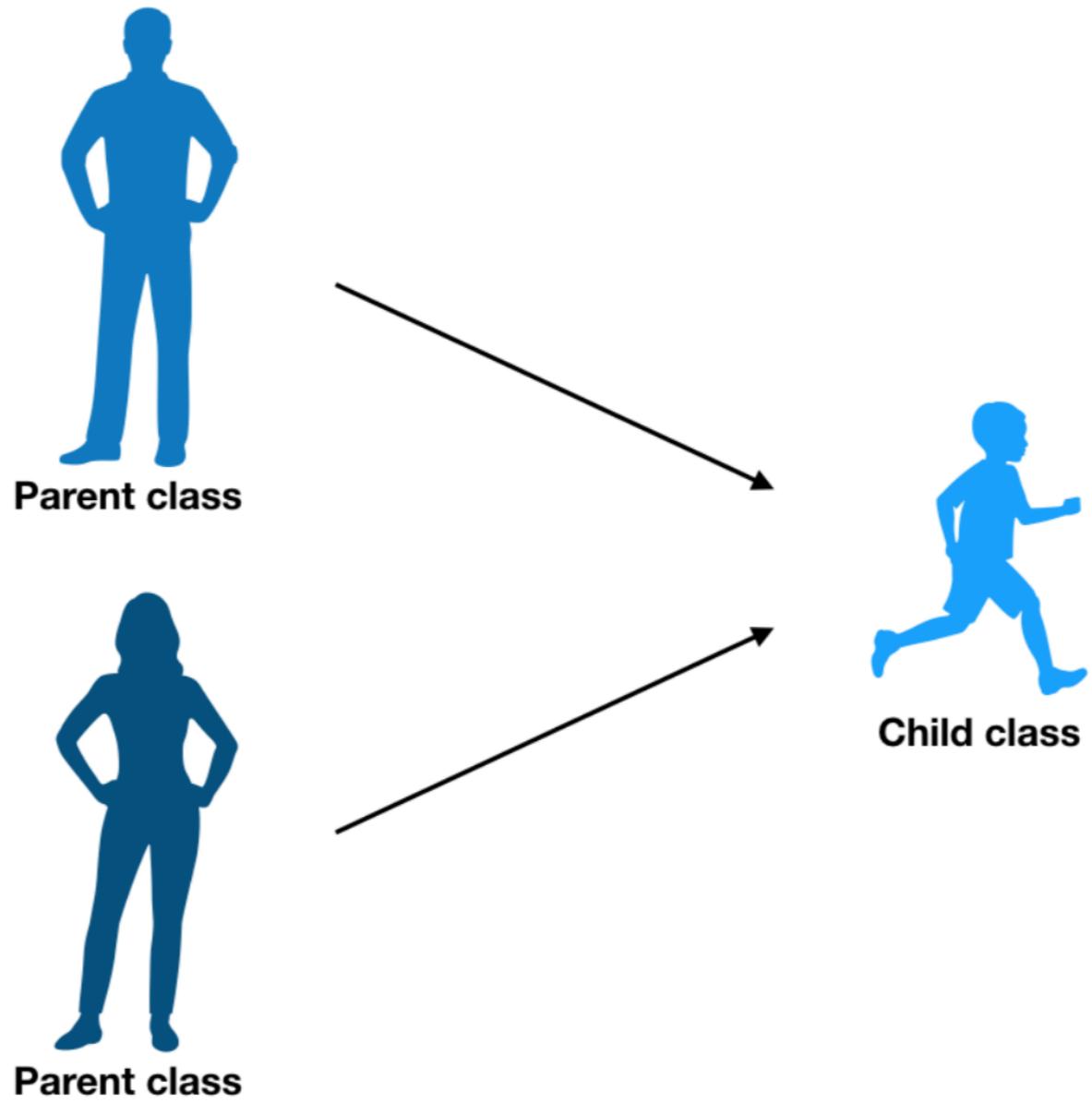
Multilevel inheritance



Multilevel inheritance



Multiple inheritance



Multilevel inheritance and super

```
class Parent:  
    def __init__(self):  
        print("I'm a parent!")  
  
class Child(Parent):  
    def __init__(self):  
        Parent.__init__()  
        print("I'm a child!")  
  
class SuperChild(Child):  
    def __init__(self):  
        super().__init__()  
        print("I'm a super child!")
```

Learn more about multiple inheritance & `super()`.

Multilevel inheritance and super

```
class Parent:  
    def __init__(self):  
        print("I'm a parent!")  
  
class SuperChild(Parent):  
    def __init__(self):  
        super().__init__()  
        print("I'm a super child!")  
  
class Grandchild(SuperChild):  
    def __init__(self):  
        super().__init__()  
        print("I'm a grandchild!")  
  
grandchild = Grandchild()
```

```
I'm a parent!  
I'm a super child!  
I'm a grandchild!
```

Keeping track of inherited attributes

```
# Create an instance of SocialMedia  
sm = SocialMedia('@DataCamp #DataScience #Python #sklearn')  
# What methods does sm have? -\_( )_/-  
dir(sm)
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',  
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',  
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',  
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',  
'__str__', '__subclasshook__', '__weakref__', '_count_hashtags',  
'_count_mentions', '_count_words', '_tokenize', 'hashtag_counts',  
'mention_counts', 'text', 'tokens', 'word_counts']
```

Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON