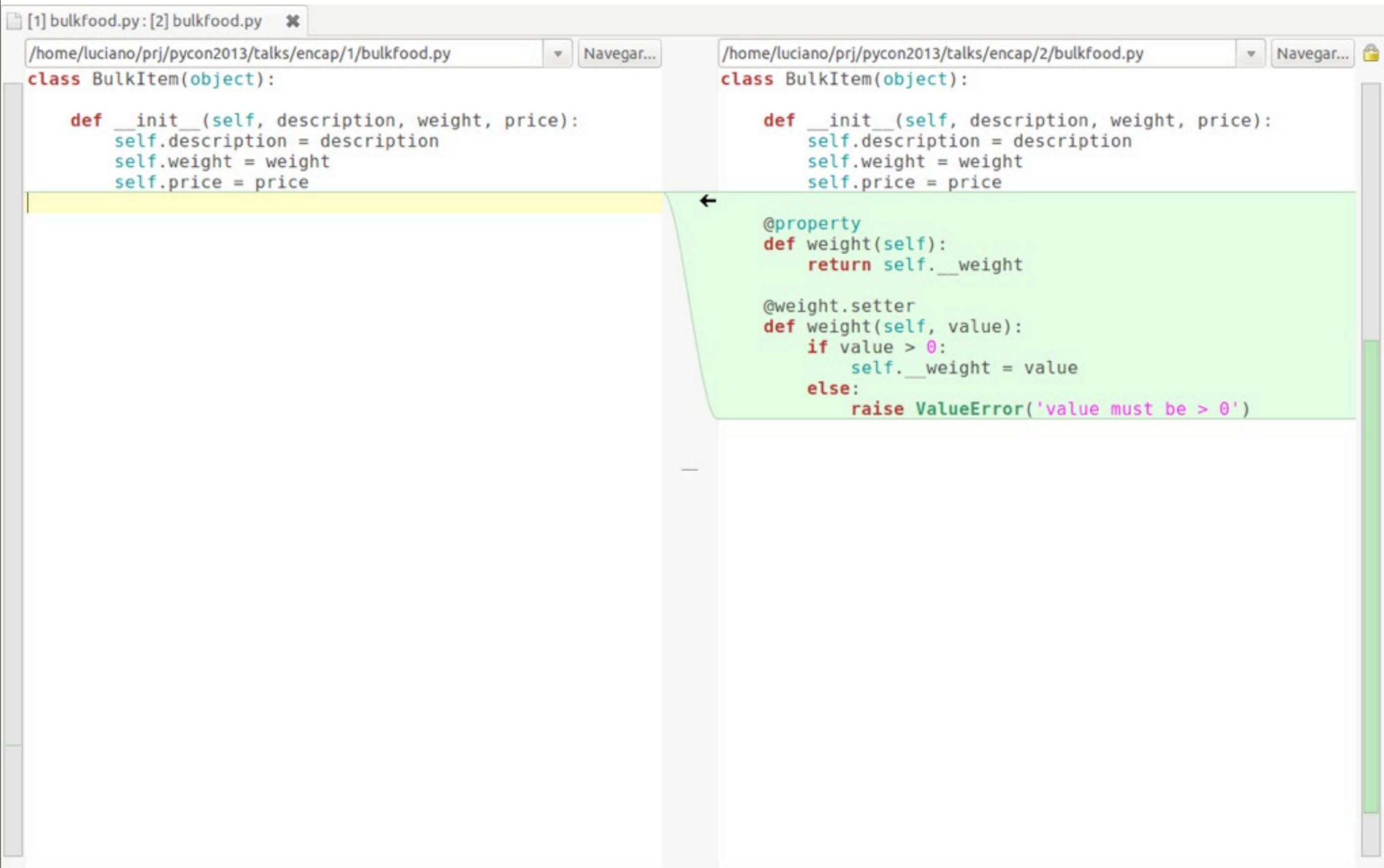


1 the simplest thing

BulkItem
description
weight
price
__init__

```
class BulkItem(object):  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price
```

② validation via property



The image displays a side-by-side comparison of two Python code files, illustrating the evolution of a class to include validation via properties.

Left Pane (bulkfood.py):

```
class BulkItem(object):  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price
```

Right Pane (bulkfood.py):

```
class BulkItem(object):  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price  
  
    @property  
    def weight(self):  
        return self.__weight  
  
    @weight.setter  
    def weight(self, value):  
        if value > 0:  
            self.__weight = value  
        else:  
            raise ValueError('value must be > 0')
```

A yellow highlight is present under the `self.weight = weight` line in the left pane. A green highlight covers the property definition in the right pane. A curved arrow points from the yellow highlight to the green highlight, signifying the replacement of direct assignment with a validated property.

② validation via property

BulkItem
description
__weight
price
__init__
weight {prop. get}
weight {prop. set}

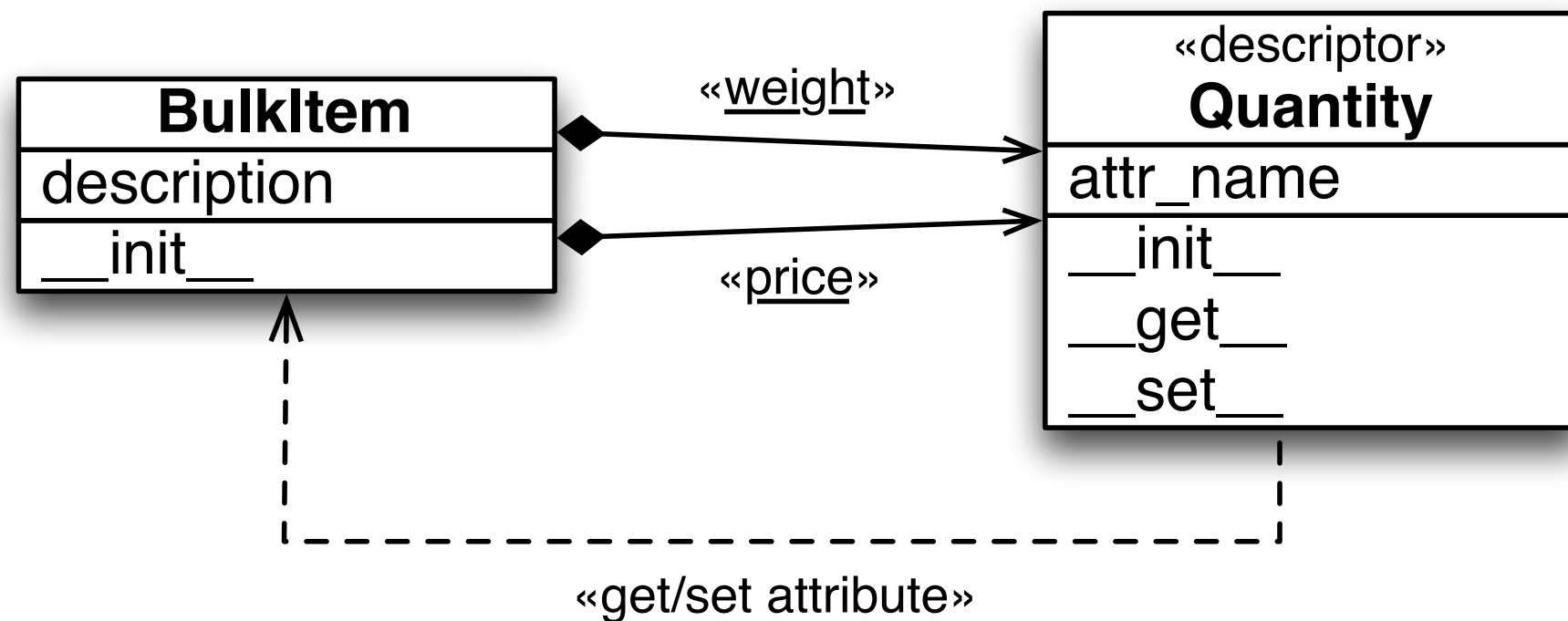
```
class BulkItem(object):  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price  
  
    @property  
    def weight(self):  
        return self.__weight  
  
    @weight.setter  
    def weight(self, value):  
        if value > 0:  
            self.__weight = value  
        else:  
            raise ValueError('value must be > 0')
```

This works, but what if **price** needs a similar treatment?

③ validation via descriptors

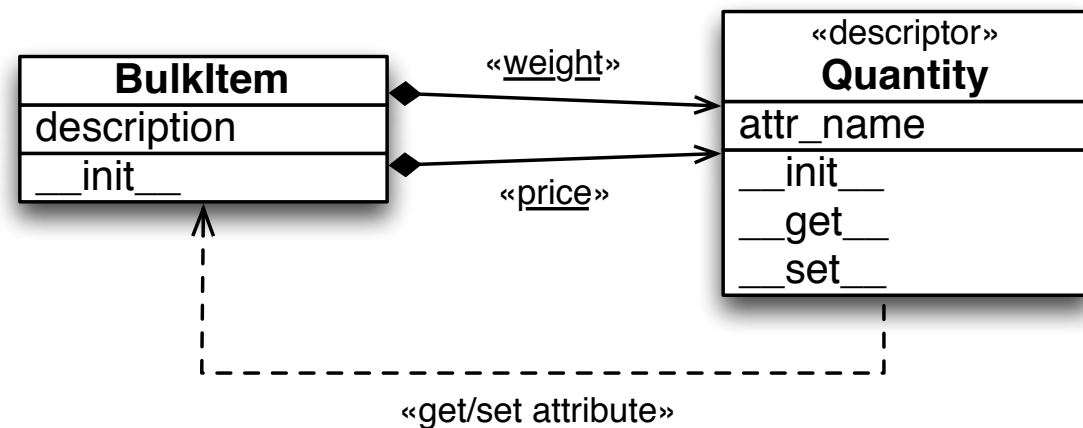
```
[2] bulkfood.py: [3] bulkfood.py ✕  
/home/luciano/prj/pycon2013/talks/encap/2/bulkfood.py  Navegar...  /home/luciano/prj/pycon2013/talks/encap/3/bulkfood.py  Navegar...  
  
class BulkItem(object):  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price  
  
    @property  
    def weight(self):  
        return self.__weight  
  
    @weight.setter  
    def weight(self, value):  
        if value > 0:  
            self.__weight = value  
        else:  
            raise ValueError('value must be > 0')  
  
→ ← class Quantity(object):  
    def __init__(self):  
        prefix = self.__class__.__name__  
        key = id(self)  
        self.attr_name = '%s_%s' % (prefix, key)  
  
    def __get__(self, instance, owner):  
        return getattr(instance, self.attr_name)  
  
    def __set__(self, instance, value):  
        if value > 0:  
            setattr(instance, self.attr_name, value)  
        else:  
            raise ValueError('value must be > 0')  
  
class BulkItem(object):  
    weight = Quantity()  
    price = Quantity()  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price
```


③ validation via descriptors



Descriptors enable reuse of validation logic through **composition**

③ validation via descriptors



```
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

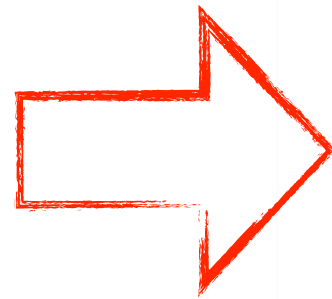
    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BulkItem(object):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

③ validation via descriptors



Field data is stored in
BulkItem instance
attributes with generic
names like

Quantity_14199423

```
class Quantity(object):

    def __init__(self):
        prefix = self.__class__.__name__
        key = id(self)
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BulkItem(object):
    weight = Quantity()
    price = Quantity()

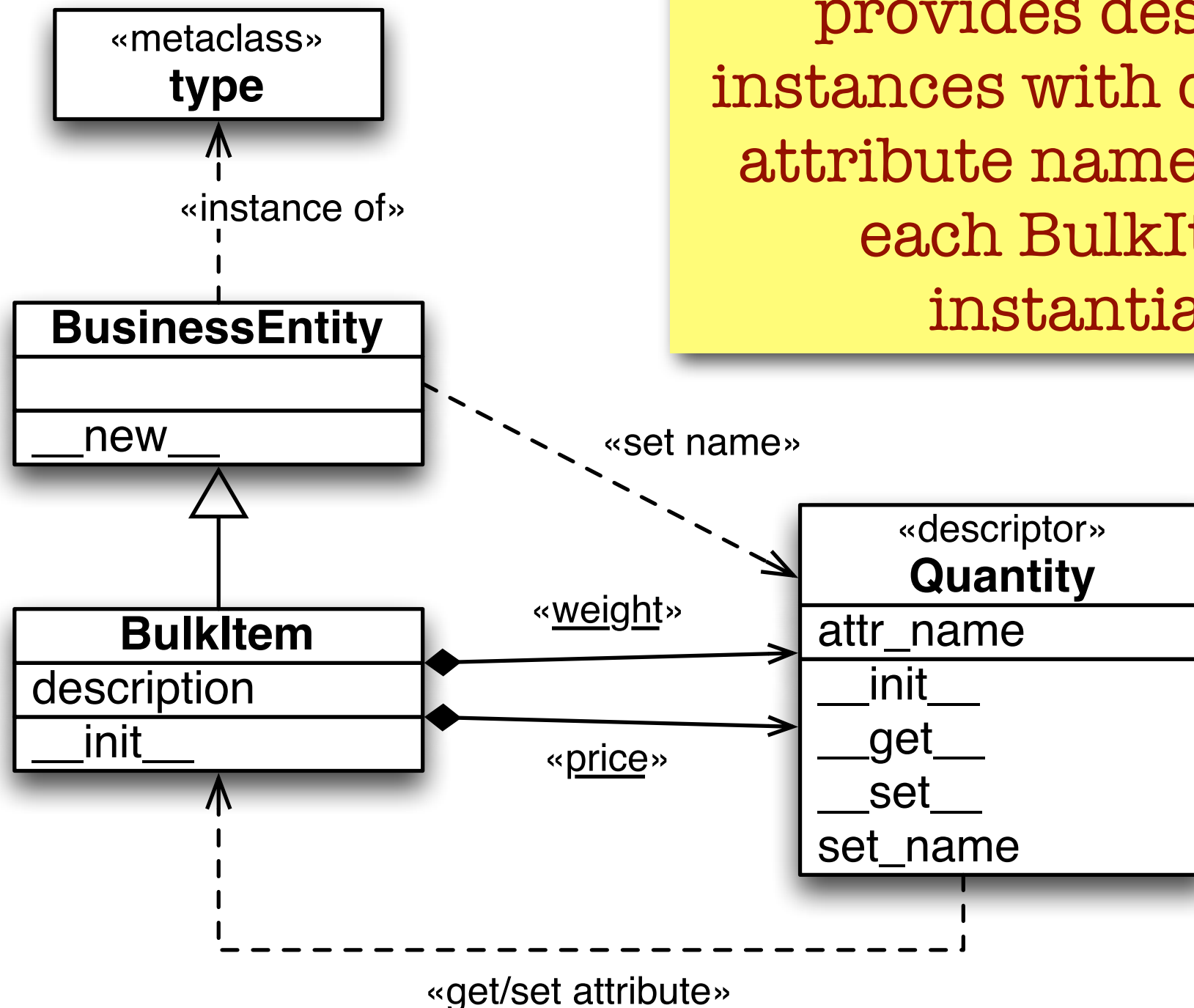
    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```


④ proper names for attrs

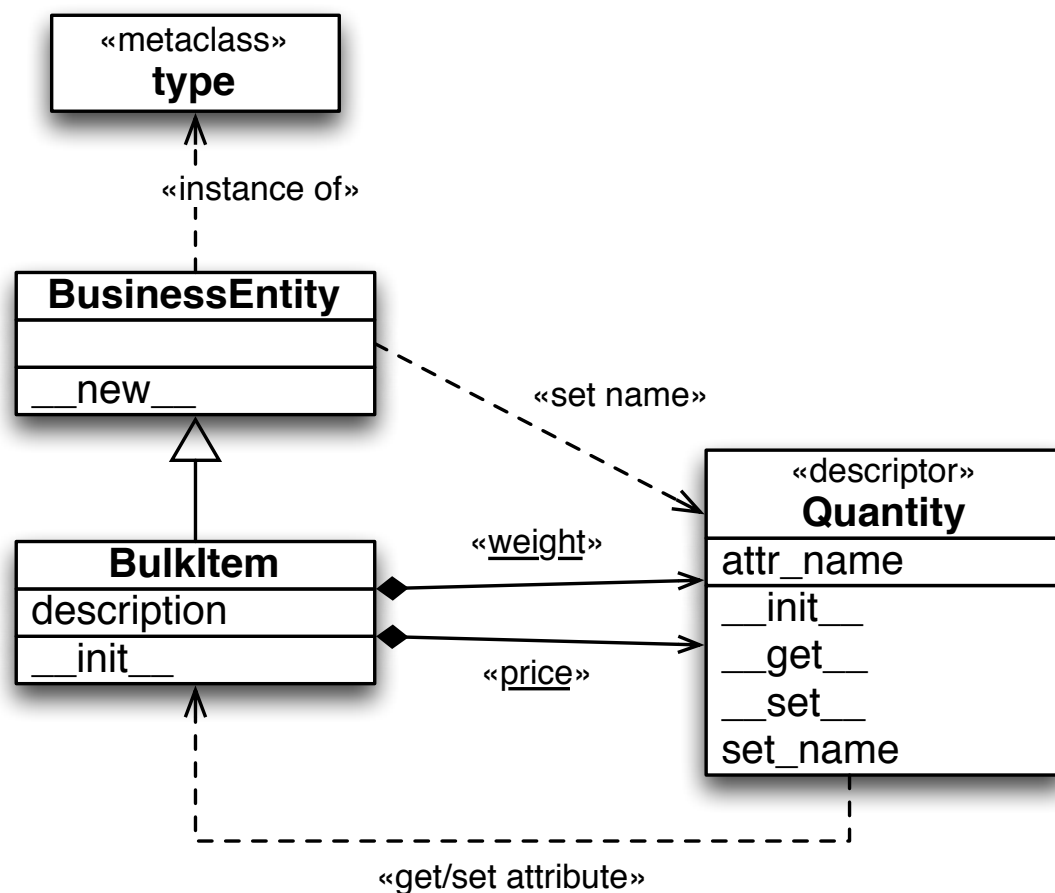
```
[3] bulkfood.py: [4] bulkfood.py ✕  
/home/luciano/prj/pycon2013/talks/encap/3/bulkfood.py Navegar...  
class Quantity(object):  
    def __init__(self):  
        prefix = self.__class__.__name__  
        key = id(self)  
        self.attr_name = '%s_%s' % (prefix, key)  
  
    def __get__(self, instance, owner):  
        return getattr(instance, self.attr_name)  
  
    def __set__(self, instance, value):  
        if value > 0:  
            setattr(instance, self.attr_name, value)  
        else:  
            raise ValueError('value must be > 0')  
  
class BulkItem(object):  
    weight = Quantity()  
    price = Quantity()  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price  
  
/home/luciano/prj/pycon2013/talks/encap/4/bulkfood.py Navegar...  
class Quantity(object):  
    def __init__(self):  
        self.set_name(self.__class__.__name__, id(self))  
  
    def set_name(self, prefix, key):  
        self.attr_name = '%s_%s' % (prefix, key)  
  
    def __get__(self, instance, owner):  
        return getattr(instance, self.attr_name)  
  
    def __set__(self, instance, value):  
        if value > 0:  
            setattr(instance, self.attr_name, value)  
        else:  
            raise ValueError('value must be > 0')  
  
class BusinessEntity(object):  
    def __new__(cls, *args, **kwargs):  
        for key, attr in cls.__dict__.items():  
            if isinstance(attr, Quantity):  
                attr.set_name('__' + cls.__name__, key)  
        return super(BusinessEntity, cls).__new__(cls, *args)  
  
class BulkItem(BusinessEntity):  
    weight = Quantity()  
    price = Quantity()  
  
    def __init__(self, description, weight, price):  
        self.description = description  
        self.weight = weight  
        self.price = price
```


④ proper names for attrs

BusinessEntity.__new__
provides descriptor
instances with descriptive
attribute names at when
each BulkItem is
instantiated



④ proper names for attrs



```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BusinessEntity(object):
    def __new__(cls, *args, **kwargs):
        for key, attr in cls.__dict__.items():
            if isinstance(attr, Quantity):
                attr.set_name('__' + cls.__name__, key)
        return super(BusinessEntity, cls).__new__(cls, *args, **kwargs)

class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```


5 avoiding redundant work

[4] bulkfood.py: [5] bulkfood.py ✕

/home/luciano/prj/pycon2013/talks/encap/4/bulkfood.py

```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BusinessEntity(object):
    def __new__(cls, *args, **kwargs):
        for key, attr in cls.__dict__.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+cls.__name__, key)
        return super(BusinessEntity, cls).__new__(cls, *args, **kwargs)

class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

/home/luciano/prj/pycon2013/talks/encap/5/bulkfood.py

```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, bases, dict_)
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)

class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta

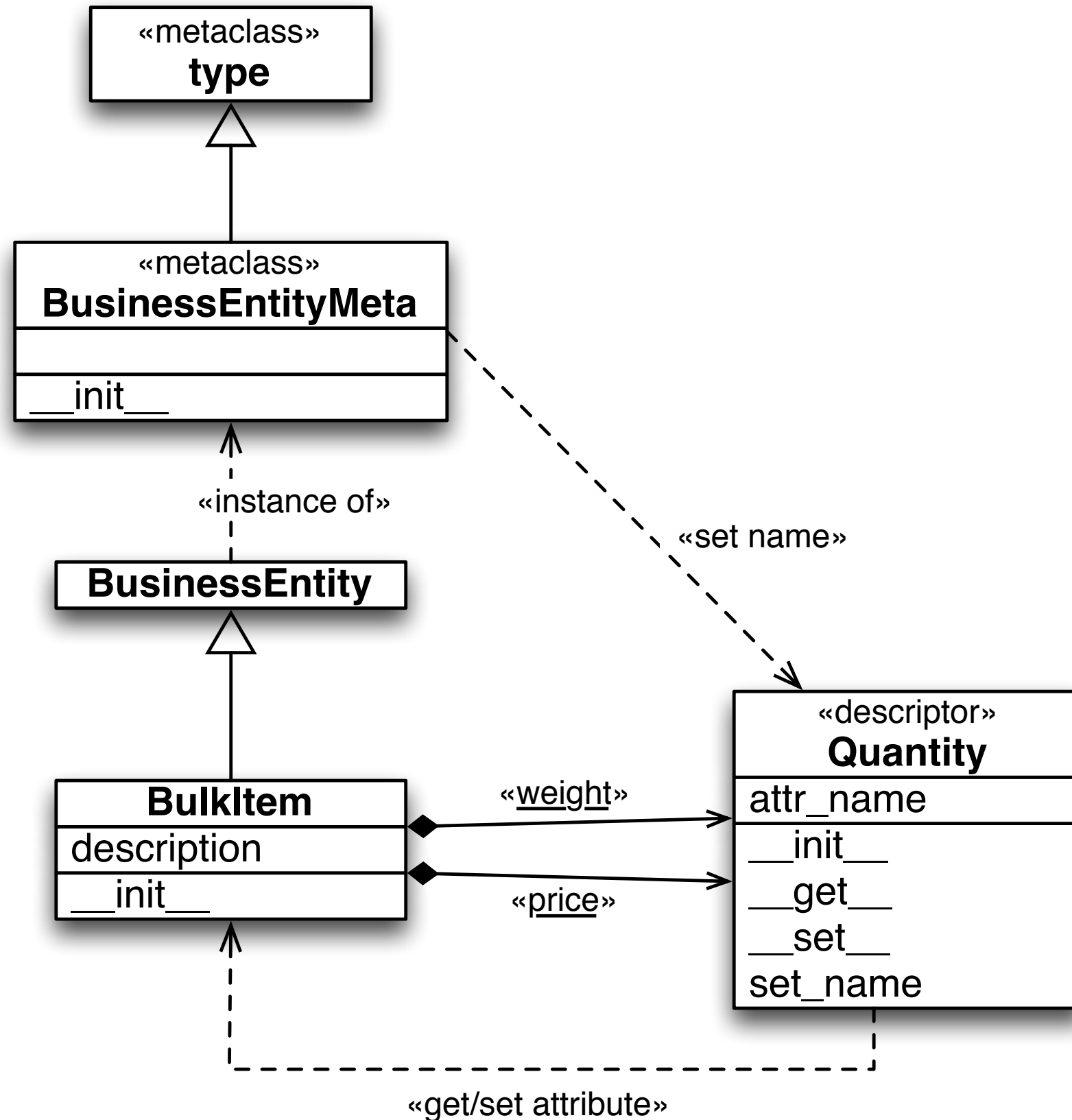
class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```

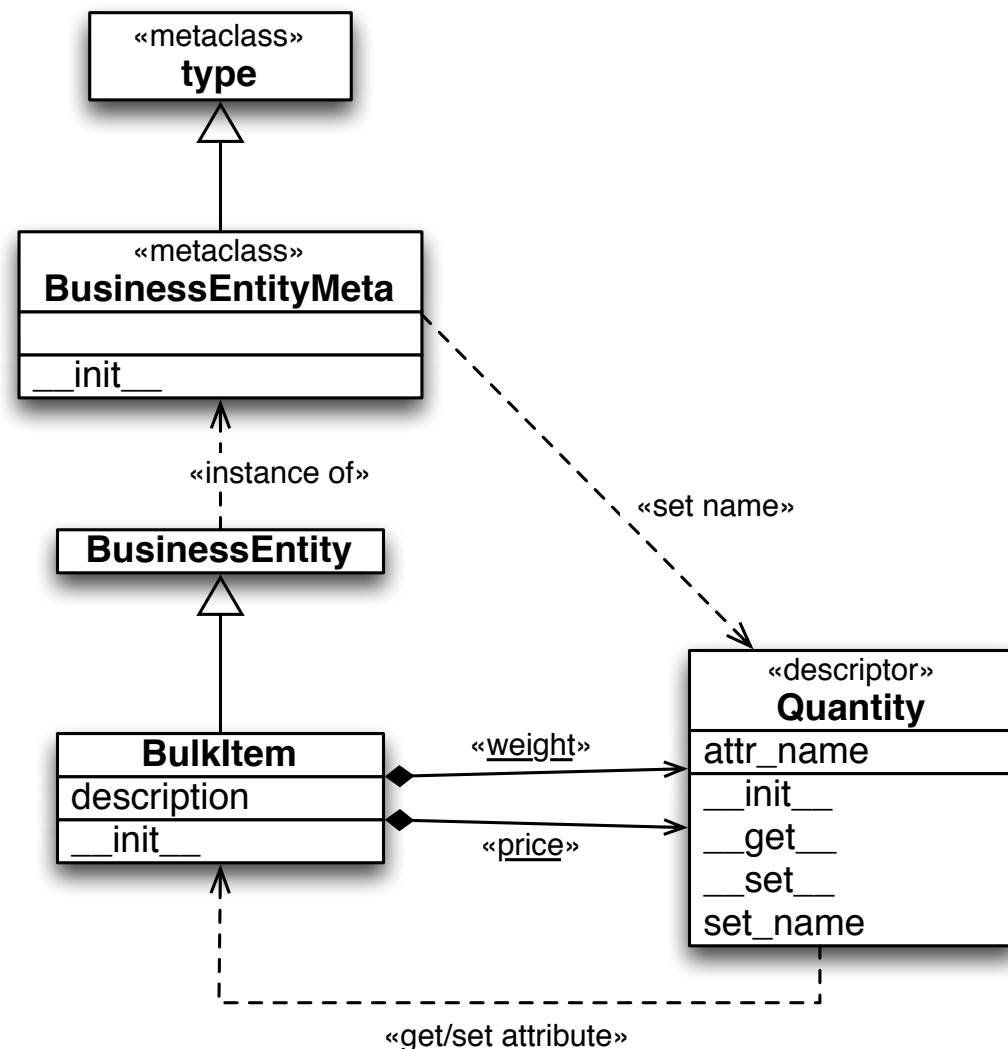
→ ←

→ ←

⑤ avoiding redundant work



5 avoiding redundant work



```

class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, base
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)

class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta

class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
  
```

5 avoiding redundant work

The **BusinessEntityMeta** metaclass provides descriptor instances with descriptive attribute names at import time

```
class Quantity(object):

    def __init__(self):
        self.set_name(self.__class__.__name__, id(self))

    def set_name(self, prefix, key):
        self.attr_name = '%s_%s' % (prefix, key)

    def __get__(self, instance, owner):
        return getattr(instance, self.attr_name)

    def __set__(self, instance, value):
        if value > 0:
            setattr(instance, self.attr_name, value)
        else:
            raise ValueError('value must be > 0')

class BusinessEntityMeta(type):
    def __init__(mcs, name, bases, dict_):
        super(BusinessEntityMeta, mcs).__init__(name, bases, dict_)
        for key, attr in dict_.items():
            if isinstance(attr, Quantity):
                attr.set_name('__'+name, key)

class BusinessEntity(object):
    __metaclass__ = BusinessEntityMeta

class BulkItem(BusinessEntity):
    weight = Quantity()
    price = Quantity()

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price
```