Value Object Documentation

Release 1.0.0

Paweł Zadrożny

Mar 19, 2021

Contents:

Value Object is an attempt to make DDD implementation of Value. It's not the same as Data Class proposed in **PEP 557** https://www.python.org/dev/peps/pep-0557/ but shares some similarities like frozen attributes.

CHAPTER 1

Value Object

Info DDD Value Object implementation.

Author Paweł Zadrożny @pawelzny <pawel.zny@gmail.com>

1.1 Features

- Value Objects are immutable.
- Two objects with the same values are considered equal
- Access to values with dot notation: `value.my_attr`
- Access to values by key: `value['my_attr']`

1.2 Installation

```
pipenv install vo  # or pip install vo
```

Package: https://pypi.org/project/vo/

1.3 Documentation

- Full documentation: http://vo.readthedocs.io
- Public API: http://vo.readthedocs.io/en/latest/api.html
- Examples and usage ideas: http://vo.readthedocs.io/en/latest/examples.html

1.4 Quick Example

Value accept any `key=value` pairs. These pairs will be attached to object as attributes. Once created values are immutable. Attributes can't be changed or deleted.

```
>>> from vo import Value
>>> book = Value(title='Learning Python',
...              authors=['Mark Lutz', 'David Ascher'],
...              publisher="O'REILLY")
>>> book
Value(authors=['Mark Lutz', 'David Ascher'], publisher="O'REILLY", title='Learning␣
↪Python')

>>> str(book)
'{"authors": ["Mark Lutz", "David Ascher"], "publisher": "O\'REILLY", "title":
↪"Learning Python"}'
```

Warning: Any attempt of value modification or delete will raise `ImmutableInstanceError`

```
>>> from vo import Value
>>> book = Value(title='Learning Python',
...              authors=['Mark Lutz', 'David Ascher'],
...              publisher="O'REILLY")
>>> book.title = 'Spam'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    raise ImmutableInstanceError()
  vo.value.ImmutableInstanceError: Modification of Value frozen instance is forbidden.
```

1.4.1 Values access

Values can be accessed like object attributes or like dict keys.

```
>>> from vo import Value
>>> book = Value(title='Learning Python',
...              authors=['Mark Lutz', 'David Ascher'],
...              publisher="O'REILLY")
>>> book.title == book['title']
True

>>> book.authors == book['authors']
True
```

1.4.2 Objects comparison

Let's take the same book example.

```
>>> from vo import Value
>>> book1 = Value(title='Learning Python',
...               authors=['Mark Lutz', 'David Ascher'],
...               publisher="O'REILLY")
>>> book2 = Value(title='Learning Python',
...               authors=['Mark Lutz', 'David Ascher'],
...               publisher="O'REILLY")
>>> book1 == book2
True

>>> book1 is book2
False
```

1.4.3 Value lookup

Check if value exists.

```
>>> from vo import Value
>>> book = Value(title='Learning Python',
...              authors=['Mark Lutz', 'David Ascher'],
...              publisher="O'REILLY")
>>> 'title' in book
True

>>> 'price' in book
False

>>> book.title
'Learning Python'

>>> book.price
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: 'Value' object has no attribute 'price'
```

CHAPTER 2

Examples

Yes, I know it's dangerous to follow code examples. Usually examples aren't in sync with real source code.

But I found a solution ... I hope!

Note:

All examples are derived from real code hooked to Pytest.
Every change in source code enforce change in examples.
Outdated examples == failed build.

You can check at https://github.com/pawelzny/vo/blob/master/tests/test_examples.py

See also:

Look at *Public API* for more details.

2.1 Basics

Value objects can be used as is straight from library. You still can extend them but for simple usage its not necessary.

2.1.1 Create Value Object

Value takes all kwargs (key=value) and add them as object attribute. Assigning values are made only once on `__init__` and after that no values can be changed.

```
from vo import Value

book_ddd = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
book_tdd = Value(title='TDD', author='Life', price=99.98, currency='USD')
```

2.1.2 Access to attributes

Properties can be accessed with dot or key notation.

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')

assert book.title == 'DDD'
assert book.author == book['author']
assert book['price'] == 120.44
```

2.1.3 Value Objects comparison

Note: Two objects with the same values are considered **equal**, but **not the same**.

Compare different values

```
from vo import Value

book_ddd = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
book_tdd = Value(title='TDD', author='Life', price=99.98, currency='USD')

assert book_ddd != book_tdd
```

Compare similar values

```
from vo import Value

book_ddd = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
book_clone = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')

assert book_ddd == book_clone
assert book_ddd is not book_clone
```

2.2 Advanced

More real life example of Value Object usage.

2.2.1 Basic inheritance

Using Value Object directly is easy, fast, and just works. However due to dynamic attribute assignment on `__init__` your favourite IDE / Editor can't generate hints.

This is when inheritance come handy.

```
from vo import Value

class Book(Value):
    title = None
    author = None
    price = None
    currency = None

book = Book(title='DDD', author='Pythonista', price=120.44, currency='USD')
```

2.2.2 Wonky behaviour

Weird behaviour but completely correct.

Warning: Value Object does not validate given attributes. Validation is up to you.

```
from vo import Value

class Book(Value):
    title = None
    author = None
    price = None
    currency = None

book = Book(spam='Foo')

# whaaaat!?

assert 'spam' in book
assert book.title is None
assert book.price is None
assert book.title is None
assert book.currency is None
```

2.2.3 Attribute validation

Most of the time you will want to make inheritance like below, but remember to not assign attribute by your own. Always delegate to `super().__init__()`

```
from vo import Value

class Book(Value):
    title = None
    author = None
    price = None
    currency = None

    def __init__(self, title, author, price, currency):
        # make validation if needed

        # always delegate assignment to Parent!
        super().__init__(title=title, author=author, price=price, currency=currency)
```

(continues on next page)

(continued from previous page)

```
book = Book(title='DDD', author='Pythonista', price=120.44, currency='USD')
```

2.2.4 Usage Ideas

Value Object is helpful always when source data must not be modified.

Frozen response

Note: **Requests** package has been faked for purpose of this example to avoid unnecessary and unrelated dependency.

Before executing this example make sure you have **requests** installed in your environment with `pip install requests`

```
from vo import Value

class Quote(Value):
    _id = None
    title = None
    content = None
    link = None

    def __init__(self, _id, title, content, link):
        # validation if needed
        super().__init__(_id=_id, title=title, contet=content, link=link)

response = requests.get('https://quotesondesign.com/wp-json/posts'
                        '?filter[orderby]=rand'
                        '&filter[posts_per_page]=2')

quotes = [Quote(x['ID'], x['title'], x['content'], x['link']) for x in response.
↪json()]
```

2D Coordinates

```
from vo import Value

class Point2D(Value):
    x = 0
    y = 0

    def __init__(self, x, y):
        # validation if needed
        super().__init__(x=x, y=y)

    def __add__(self, other):
        return Point2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Point2D(self.x - other.x, self.y - other.y)
```

(continues on next page)

(continued from previous page)

```
p1 = Point2D(2, 5)
p2 = Point2D(2, 5)
p3 = Point2D(-3, 10)

assert p1 == p2
assert p1 != p3
assert p1 + p2 == Point2D(4, 10)
assert p3 - p1 == Point2D(-5, 5)
```

Money object

Danger:

This example is not meant to run on production !!
It doesn't implement validation and many more comparison methods.

But its nice to present general idea of Money Object.

```
import decimal
from vo import Value

class Money(Value):
    amount = None
    currency = None

    def __init__(self, amount, currency):
        # plenty of validation
        super().__init__(amount=decimal.Decimal(amount), currency=currency)

    def __lt__(self, other):
        return self.amount < other.amount

    def __gt__(self, other):
        return self.amount > other.amount

    def __add__(self, other):
        return Money(amount=self.amount + other.amount, currency='USD')

    def __sub__(self, other):
        return Money(amount=self.amount - other.amount, currency='USD')

assert Money(200, 'USD') > Money(120, 'USD')
assert Money(100, 'USD') < Money(120, 'USD')
assert Money(100, 'USD') + Money(200, 'USD') == Money(300, 'USD')
assert Money(100, 'USD') - Money(50, 'USD') == Money(50, 'USD')
```

2.3 Forbidden actions

> **Warning:** All attempt to value modification ends up with `ImmutableInstanceError` exception.

2.3.1 Modification

Modification of existing attribute is forbidden. Let's create a book, and then try to change its title.

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
all_good = True

try:
    book.title = 'BDD > DDD'  # or book['title'] = 'SPAM'
except ImmutableInstanceError:
    all_good = False

assert all_good is False
assert book.title == 'DDD'
```

Adding new attributes also raises exception. Let's add publisher property to the book.

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
all_good = True

try:
    book.publisher = 'SPAM'  # or book['publisher'] = 'SPAM'
except ImmutableInstanceError:
    all_good = False

assert all_good is False
assert 'publisher' not in book
```

2.3.2 Deletion

Properties of value object can't be deleted no matter what!

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
all_good = True

try:
    del book.title  # or del book['title']
except ImmutableInstanceError:
    all_good = False

assert all_good is False
assert 'title' in book
```

2.4 Data dumps

Convert value object to different data types.

2.4.1 To dict

Note: Actually `.to_dict()` method returns `collections.OrderedDict`.

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
dump = book.to_dict()

assert isinstance(dump, OrderedDict)
assert dump == OrderedDict([('author', 'Pythonista'), ('currency', 'USD'),
                            ('price', 120.44), ('title', 'DDD')])
```

2.4.2 To bytes

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
dump = book.to_bytes()

assert isinstance(dump, bytes)
assert dump == b'\'{"author": "Pythonista", "currency": "USD", ' \
               b'"price": 120.44, "title": "DDD"}\''
```

2.4.3 To JSON

```
from vo import Value

book = Value(title='DDD', author='Pythonista', price=120.44, currency='USD')
dump = book.to_json()

assert isinstance(dump, str)
assert dump == '{"author": "Pythonista", "currency": "USD", ' \
               '"price": 120.44, "title": "DDD"}'
```

CHAPTER 3

Public API

See also:

Check out *Examples* derived from real and fully tested source code.

class vo.value.**Value**(***kwargs*)

Basic implementation of DDD immutable Value Object.

Implementation provides:

- Immutability of once created object
- Comparison of two objects
- Conversion to other data types

Param Any `key=value` pairs.

to_dict() → collections.OrderedDict

Dump all values to OrderedDict.

All keys are sorted in ascending direction. Dump does not include hash and checksum.

Returns dict with values

Return type collections.OrderedDict

to_json() → str

Dump values to JSON.

Feed for JSON comes from `.to_dict()` method.

Returns JSON string.

Return type str

to_bytes() → bytes

Dump values to bytes.

Feed for byte string comes from `.to_json()` method.

Returns byte string

Return type bytes

exception vo.value.**ImmutableInstanceError**(*message: str = None*)

Raised on any attempt to modify values in Value object.

Parameters **message** (*str*) – Optional message.

message = 'Modification of Value instance is forbidden.'

CHAPTER 4

Credits

4.1 Development

- Paweł Zadrożny @pawelzny <pawel.zny@gmail.com>

4.2 Contributors

None yet. Why not be the first?

Read more how to contribute on *Contributing*.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/pawelzny/vo/issues

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

5.1.4 Write Documentation

authentication could always use more documentation, whether as part of the official authentication docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pawelzny/vo/issues

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *vo* for local development.

1. Fork the *vo* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/vo.git
```

3. Install your local copy into a virtualenv. Assuming you have PipEnv installed, this is how you set up your fork for local development:

```
$ cd vo/
$ make install-dev
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make test-all
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4, 3.5, 3.6, and for PyPy3.5. Check https://circleci.com/gh/pawelzny/vo and make sure that the tests pass for all supported Python versions.

CHAPTER 6

LICENSE

ISC License

Python Module Index

v

Index