Refactoring for Performance

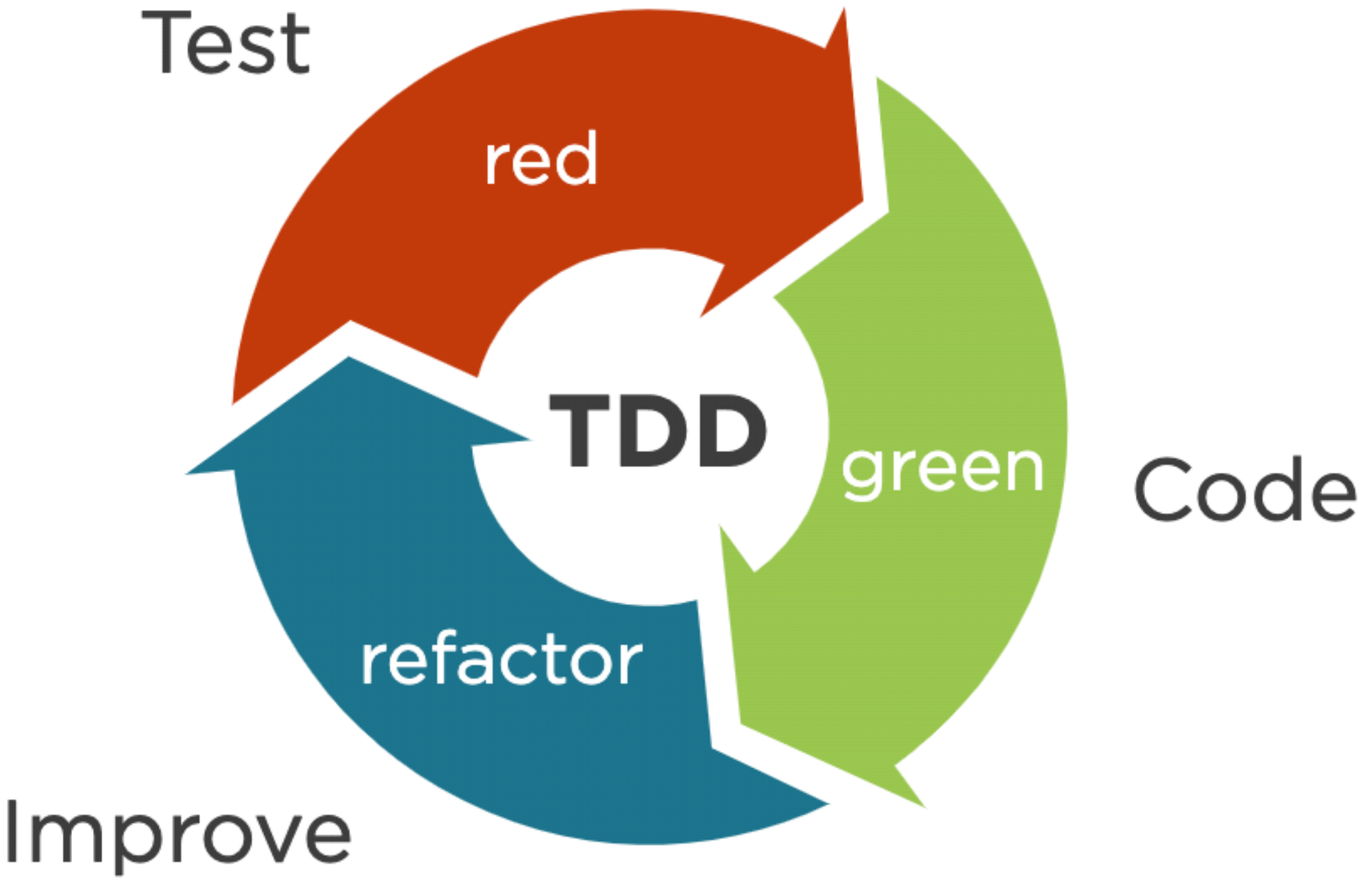
Robert Smallshire
COFOUNDER - SIXTY NORTH

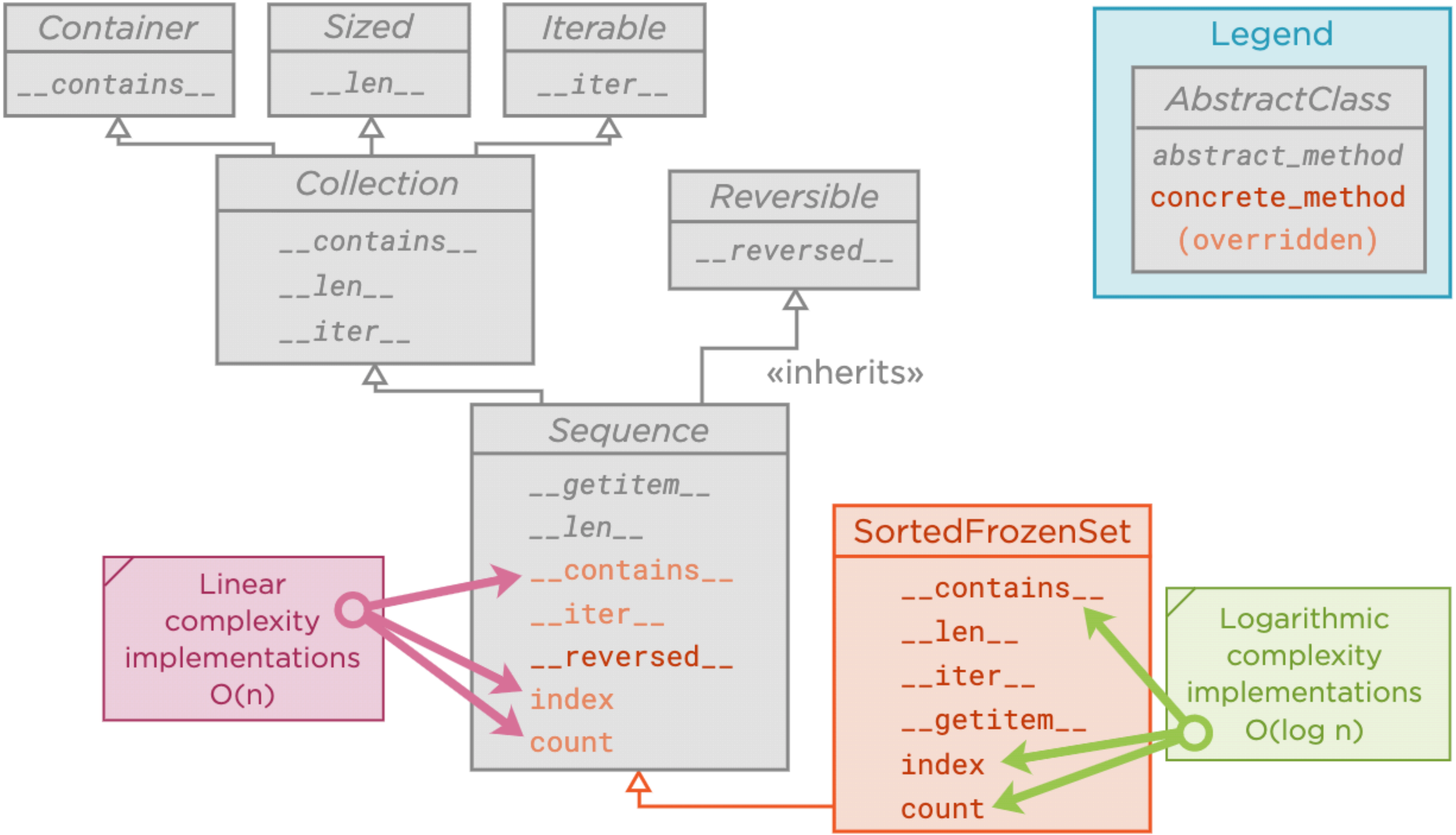
@robsmallshire

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Test Driven Development




Container
__contains__
Sized
__len__
Iterable
__iter__
Collection
__contains__
__len__
__iter__
Reversible
__reversed__
«inherits»
Sequence
__getitem__
__len__
__contains__
__iter__
__reversed__
index
count
SortedFrozenSet
__contains__
__len__
__iter__
__getitem__
index
count
Linear complexity implementations O(n)
Logarithmic complexity implementations O(log n)
Legend
AbstractClass
abstract_method
concrete_method
(overridden)

Recamán's Sequence

```python
from itertools import count

def recaman():
    """Generate Recaman's sequence.

    https://en.wikipedia.org/wiki/Recamán%27s_sequence
    https://oeis.org/A005132
    """
    seen = set()
    a = 0
    for n in count(1):
        yield a
        seen.add(a)
        c = a - n
        if c < 0 or c in seen:
            c = a + n
        a = c
```

Recamán Realized

```
>>> from itertools import islice
>>> from recaman import recaman
>>> list(islice(recaman(), 50))
[0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 2
2, 10, 23, 9, 24, 8, 25, 43, 62, 42, 63,
 41, 18, 42, 17, 43, 16, 44, 15, 45, 14,
 46, 79, 113, 78, 114, 77, 39, 78, 38, 7
9, 37, 80, 36, 81, 35, 82, 34, 83]
>>>
```

sorted-set / sorted_frozen_set.py

Unittests in test_sorted_frozen_set.py

test_sorted_frozen_set.py | sorted_frozen_set.py

```
14    def __contains__(self, item):
15        try:
16            self.index(item)
17            return True
18        except ValueError:
19            return False
20
21    def __len__(self):
22        return len(self._items)
23
24    def __iter__(self):
25        return iter(self._items)
```

SortedFrozenSet

Run: Unittests in test_sorted_frozen_set.py

Tests passed: 54 of 54 tests – 4 ms

Test Results 4 ms

```
Testing started at 12:02 ...
/Users/sixty-north/.virtualenvs/sorted-set/bin/python /Applications/PyCharm
  .app/Contents/helpers/pycharm/_jb_unittest_runner.py --path
  test_sorted_frozen_set.py
Launching unittests with arguments python -m unittest test_sorted_frozen_set.py
  in /var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpbfjhoxg9/build/sorted-set
```

Tests passed: 54 (moments ago)

20:1 LF UTF-8 4 spaces Python 3.8 (sorted-set)

bisect — Array bisection algorithm

Source code: Lib/bisect.py

This module provides support for maintaining a list in sorted order without having to sort the list after each insertion. For long lists of items with expensive comparison operations, this can be an improvement over the more common approach. The module is called `bisect` because it uses a basic bisection algorithm to do its work. The source code may be most useful as a working example of the algorithm (the boundary conditions are already right!).

The following functions are provided:

bisect.**bisect_left**(*a*, *x*, *lo=0*, *hi=len(a)*)

Locate the insertion point for *x* in *a* to maintain sorted order. The parameters *lo* and *hi* may be used to specify a subset of the list which should be considered; by default the entire list is used. If *x* is already present in *a*, the insertion point will be before (to the left of) any existing entries. The return value is suitable for use as the first parameter to `list.insert()` assuming that *a* is already sorted.

The returned insertion point *i* partitions the array *a* into two halves so that `all(val < x for val in a[lo:i])` for the left side and `all(val >= x for val in a[i:hi])` for the right side.

bisect.**bisect_right**(*a*, *x*, *lo=0*, *hi=len(a)*)
bisect.**bisect**(*a*, *x*, *lo=0*, *hi=len(a)*)

Similar to `bisect_left()`, but returns an insertion point which comes after (to the right of) any existing entries of *x* in *a*.

The returned insertion point *i* partitions the array *a* into two halves so that `all(val <= x for val`