* collections / containers

Implement ADT (abstract data type) Bag. (+ Iterator)

ADT Iterator

- Has access to the interior structure of the Bag and it has a current element from the Bag.

Domain: I = {i | i is an iterator over b ∈ $\mathcal{B}$}

Interface:

init(i, b)

pre: b ∈ $\mathcal{B}$

post: i ∈ I, i is an iterator over b. i refers to the first element of b, or it is invalid if b is empty

'i' valid → get current → move to the next

valid(i)

pre: i ∈ I

post: $valid \leftarrow \begin{cases} true, if\ the\ current\ element\ from\ i\ is\ a\ valid\ one \\ false, otherwise \end{cases}$

first(i)

pre: i ∈ I

post: i' ∈ I, the current element from i' refers to the first element from the bag or i' is invalid if the bag is empty

next(i)

pre: i ∈ I, valid(i)

post: i' ∈ I, the current element from i' refers to the next element from the bag

throws: exception, if it is not valid

(the container has been modified)

getCurrent(i)

pre: i ∈ I, valid(i)

post: getCurrent ∈ TElem, getCurrent is the current element from i

throws: exception if i is not valid

```
def printBag(bag):

    it = bag.iterator()
    while it.valid():
        print(it.getCurrent())
        it.next()
    print("Over. Let's start again")
    it.first()
    while it.valid():
        print(it.getCurrent())
        it.next()
```

Bag vs List

- no order of elements

  ex: bag [1,2,3,1,5,2,6,2]) same bag, different list
  [1,1,2,2,2,3,5,6]

  same elements, but in different order

- no position (in the interface)

Bag vs Set

- a set has unique elements, while a bag can be seen as a multiset

* interface is the term used in c++ for what we have when defining a class

---

using Python
using 2 representations by using list (dynamical array)

① R1 - have all elements

1	2	3	1	5	2	6	2

② R2 - keep each element once and their frequency

(1,2)	(2,3)	(5,1)	(6,1)

1	2	5	6

2	3	1	1

① class Bag:

```
    def __init__(self):
        self.__elems = []

    def __add__(self, new_elem):
        self.__elems.append(new_elem)

    def __remove__(self, e):
        if e in self.__elems:
            self.__elems.remove(e)
            return True
        else:
            return False
    ...
    def nrOccurences(self, e):
        cnt = 0
        for el in self.__elems:
            if e == el:
```

* we can work considering that the precondition is met

```
            cut += 1
        return cut

    def iterator(self):
        return BagIterator(self)

class BagIterator:
    def __init__(self, b):
        self.__bag = b
        self.__current = 0

    def __next__(self):
        if self.__current >= self.__bag.size():
            raise ValueError("No next element")
        else:
            self.__current += 1

    def getCurrent(self):
        if self.valid():
            return self.__bag.__Bag__elems[self.__current]
        else:
            raise ValueError()
```

the container

this is how you access private fields

(2)

```
class BagF:
    def __init__(self):
        self.__elems = []
        self.__freq = []

    def __add__(self, e):
        if e in self.__elems:
            self.__freq[self.__elems.index(e)]
        else:
            self.__elems.append(e)
            self.__freq.append(1)

    def __remove__(self, e):
        if e in self.__elements:
            if self.__freq[self.__elems.index(e)] == 1:
                self.__freq.pop(self.__elems.index(e))
                self.__elems.remove(e)
                return True
            else:
                self.__freq[i] -= 1
                return True
        else:
            return False
```

```
def noOccurances(self, e):
    if e in self.__elems:
        return self.__freq[self.__elements.index(e)]
    else:
        return 0
```

TElem = (equality test)
← (assignment)

TComp = (equality test)
← (assignment)
>, < (less than / greater than)

```
class BagIterator:
    def __init__(self, b):
        self._bag = b
        self.__currentE = 0
        self.__currentF = 1

    def next(self):
        if self.valid():
            if self._currentF >= self._bag._Bag__freq[self.__currentE]:
                self._currentE += 1
                self._currentF = 1
            else:
                self.__currentF += 1
        else:
            raise ValueError()

    def valid(self):
        if self.__currentE < len(self.__bag._Bag__elems):
            return True
        return False
```