Nota seminar = 50% lucrare scrisă (seminar 5 – 50 de minute)
25% activitate seminar
15% prezenţa (este permisă o absenţă nemotivată)
10% stadiul dezvoltării proiectului

Nota finală = 60% examen scris
20% nota proiect
20% nota seminar

TAD Colecţie

Problema

Să se construiască o colecţie de numere întregi şi să se afişeze folosind un iterator. Aceeaşi cerinţă pentru o colecţie de caractere.

- container, asemenea unei mulţimi, dar în care se pot repeta elementele
- ordinea elementelor nu contează
- exemplu: **c = {**1, 2, 3, 4, 3, 2, 1, 1, 2, 4**}**

Domeniu

$\mathcal{C} = \{c \mid c$ – colecţie cu elemente generice (de tip **TElement**)$\}$

Operaţii

creează(c)

pre: -

post: c – colecţie vidă, $c \in \mathcal{C}$

adaugă(c, e)

pre: $c \in \mathcal{C}$, $e \in$ **TElement**

post: $c' \in \mathcal{C}, c' \leftarrow c \cup \{e\}$

{s-a adăugat elementul în colecție}

șterge(c, e)

pre: $c \in \mathcal{C}$, $e \in$ **TElement**

post: $c' \in \mathcal{C}, c' \leftarrow c \setminus \{e\}$

{s-a șters o apariție a elementului din colecție}

caută(c, e)

pre: $c \in \mathcal{C}$, $e \in$ **TElement**

post: ***caută*** = adevărat, dacă $e \in c$

caută = fals, altfel

dim(c)

pre: $c \in \mathcal{C}$

post: ***dim*** = numărul de elemente ale lui ***c***; $dim \in \mathbb{N}^*$

vidă(c)

pre: $c \in \mathcal{C}$

post: ***vidă*** = adevărat, dacă ***c*** – colecție vidă

vidă = fals, altfel

iterator(c, i)

pre: $c \in C$

post: $i \in I$, i – iterator pe colecția c

TAD IteratorCol

- entitate care permite iterarea (parcurgerea) unei colecții și preluarea elementelor acesteia
- colecția nu oferă acces la elementele sale, ea trebuie să furnizeze un mecanism de accesare a elementelor folosind un iterator
- procesul de iterare este separat de reprezentarea folosită pentru container
- iteratorul trebuie sa conțină:
    - o referință spre colecția pe care o iterează
    - o referință spre elementul curent din iterație – *curent*

Domeniu

$I = \{i \mid i$ – iterator pe $c \in C\}$

Operaţii

creează(i, c)

pre: $c \in C$

post: $i \in I$, s-a creat iteratorul i pe colecția c (elementul *curent* din iterator referă primul element din colecție)

următor(i)

pre: ***i*** ∈ I

post: *curent'* referă următorul element din colecție, față de cel referit de *curent*

element(i,e)

pre: ***i*** ∈ I, *curent* referă o poziție validă din colecție

post: ***e*** ∈ **Telement, *e*** – elementul din colecție referit de *curent*

valid(i)

pre: ***i*** ∈ I

post: ***valid*** **=** adevărat, dacă *curent* – referă o poziție validă din colecție
valid **=** fals, altfel

- 2 moduri de reprezentare pentru o colecție:
  1. Înșiruire de elemente care se pot repeta
  2. Perechi element + frecvență: $e_1, e_2, ..., e_n$
     $f_1, f_2, ..., f_n$

Implementare colectie + iterator – reprezentarea 1 (înșiruire de elemente care se pot repeta):

```
class Colectie:
    def __init__(self):
        self.__c = []

    def adauga(self, e):
        self.__c.append(e)

    def sterge(self, e):
        try:
            self.__c.remove(e)
        except ValueError:
            pass
```

```
    def cauta(self, e):
        if e in self.__c:
            return True
        return False

    def dim(self):
        return len(self.__c)

    def vida(self):
        if self.dim() == 0:
            return True
        return False

    def getCol(self):
        return self.__c

    def iterator(self):
        return Iterator(self)


class Iterator:
    def __init__(self, c):
        self.__col = c
        self.__curent = 0

    def urmator(self):
        self.__curent += 1

    def valid(self):
        return self.__curent < self.__col.dim()

    def element(self):
      return self.__col.getCol()[self.__curent]
```

Implementare colectie + iterator – reprezentarea 2 (perechi element + frecvență):

```
class ColectieF:
    def __init__(self):
        self.__elem = []
        self.__frecv = []

    def adauga(self, e):
        if e not in self.__elem:
            self.__elem.append(e)
            self.__frecv.append(1)
        else:
            idx = self.__elem.index(e)
            self.__frecv[idx] += 1

    def sterge(self, e):
        if e not in self.__elem:
            pass
        else:
```

```
            idx = self.__elem.index(e)
            if self.__frecv[idx] == 1:
                self.__elem.pop(idx)
                self.__frecv.pop(idx)
            else:
                self.__frecv[idx] -= 1

    def vida(self):
        return len(self.__elem) == 0

    def dim(self):
        nr = 0
        for f in self.__frecv:
            nr += f
        return nr

    def getElem(self):
        return self.__elem

    def getFrecv(self):
        return self.__frecv

    def iterator(self):
        return IteratorF(self)


class IteratorF:
    def __init__(self, col):
        self.__c = col
        self.__curent = 0
        self.__k = 0

    def urmator(self):
        if self.__k < self.__c.getFrecv()[self.__curent] - 1:
            self.__k += 1
        else:
            self.__curent += 1
            self.__k = 0

    def element(self):
        return self.__c.getElem()[self.__curent]

    def valid(self):
        return self.__curent < len(self.__c.getElem())
```

Implementare main:

```
from Colectie import *
from ColectieF import *

def creeazaIntregi(c):
    c.adauga(1)
    c.adauga(2)
    c.adauga(3)
```

```
        c.adauga(2)
        c.adauga(1)

    def tipar(c):
        i = c.iterator()
        while (i.valid()):
            print(i.element())
            i.urmator()

    def main():
        # c = Colectie()
        c = ColectieF()
        creeazaIntregi(c)
        tipar(c)

main()
```