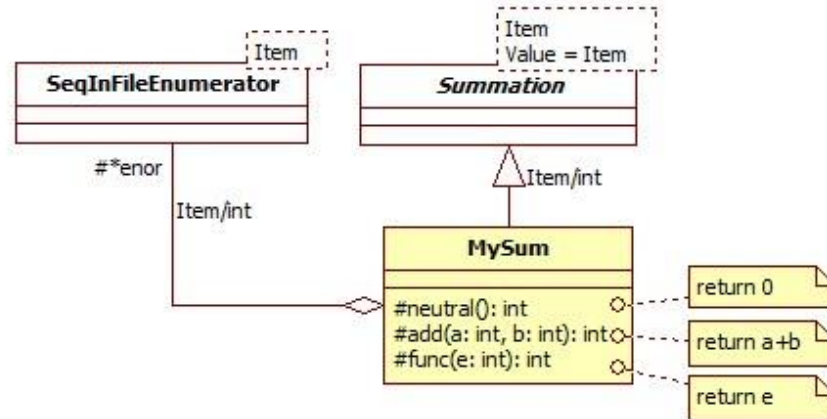# *Class template library*

1. Sum up all the (integer) numbers stored in a text file.

   Post: $s = \Sigma_{e \in f'}\, e$ $f{:}\mathrm{infile}(\mathbb{Z})$

   Analogy:

enor(Item)	~	infile($\mathbb{Z}$)
func(e)	~	e
Value,+,Ø	~	$\mathbb{Z}$,+,0

   ```
                                        ┌─────────────────┐
                          ┌ ─ ─ ─ ─ ┐   ¦ Item            ¦
                          ¦ Item    ¦   ¦ Value = Item    ¦
                          └ ─ ─ ─ ─ ┘   └ ─ ─ ─ ─ ─ ─ ─ ─ ┘
              SeqInFileEnumerator          Summation

                  #*enor                      △ Item/int
                                              │
                      Item/int               MySum
                                      ┌──────────────────────────┐   ┌──────────┐
                                      │ #neutral(): int        ○─┼───│ return 0 │
                                      │ #add(a: int, b: int): int○┼───│return a+b│
                                      │ #func(e: int): int     ○─┼───│ return e │
                                      └──────────────────────────┘   └──────────┘
   ```
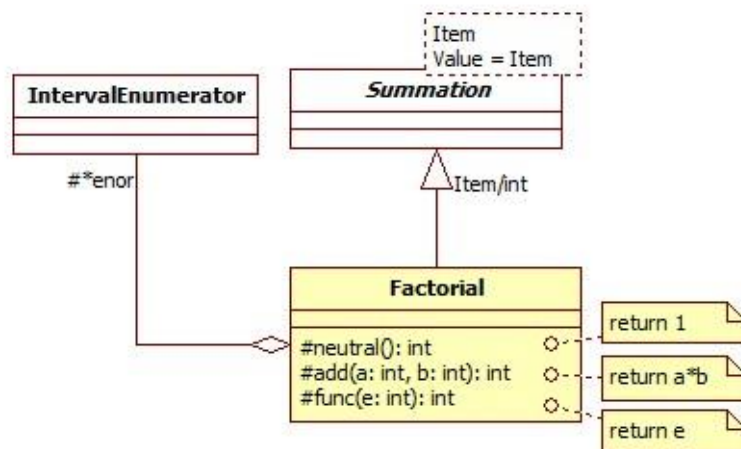
2. Calculate the factorial of a natural number.

   Post: $fact = \Pi_{i=2..n}\, i$ $n{:}\mathbb{N}$

   analógia táblázat:

enor(Item)	~	i=2 .. n
f(e)	~	i
Value, +, Ø	~	$\mathbb{N}$, *, 1

   ```
                                        ┌─────────────────┐
                                        ¦ Item            ¦
                                        ¦ Value = Item    ¦
                                        └ ─ ─ ─ ─ ─ ─ ─ ─ ┘
              IntervalEnumerator           Summation

                  #*enor                      △ Item/int
                                              │
                                             Factorial
                                      ┌──────────────────────────┐   ┌──────────┐
                                      │ #neutral(): int        ○─┼───│ return 1 │
                                      │ #add(a: int, b: int): int○┼───│return a*b│
                                      │ #func(e: int): int     ○─┼───│ return e │
                                      └──────────────────────────┘   └──────────┘
   ```
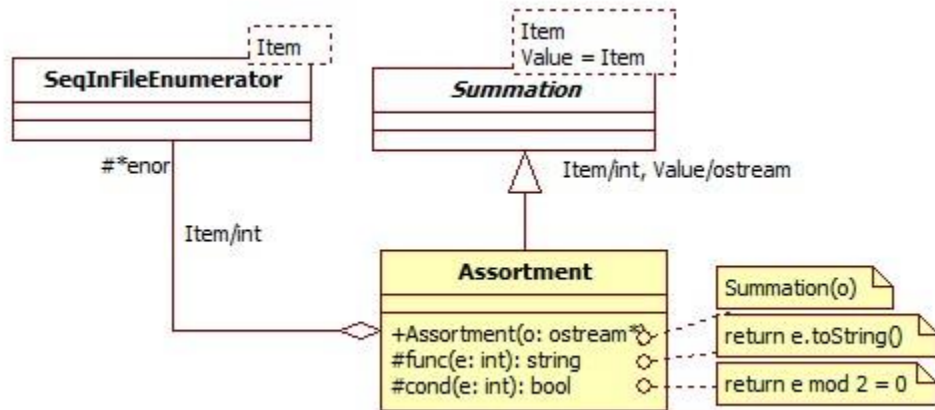
3. Assort the even numbers of a text file into another file

       Post: $y = \bigoplus_{\substack{e \in f' \\ 2|e}} <e>$ $f:infile(\mathbb{Z})$

      Analogy:

          enor(Item)   ~   infile($\mathbb{Z}$)

          f(e)   ~ $<e>$ if $2|e$, $<>$ else

          Value, +, $\emptyset$ ~ $\mathbb{Z}^*, \oplus, <>$
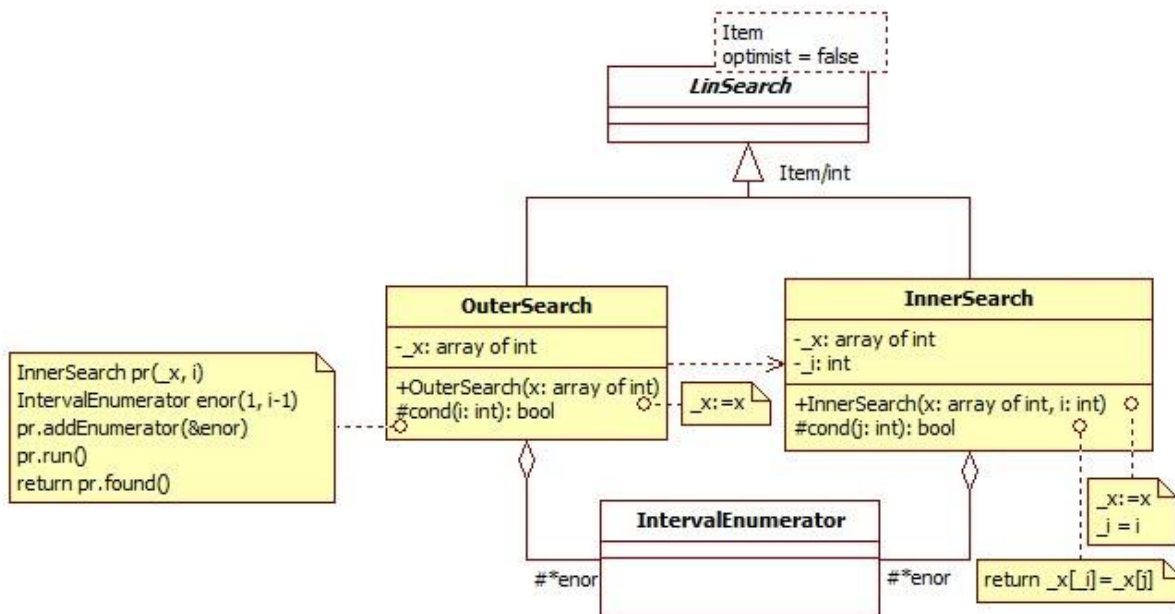

4. Which is the first repetitive item of an array?

      Post: $l, ind = search_{i=2..n} (search_{j=1..i-1} (x[i]=x[j]))$ $(x: \mathbb{Z}^n)$

      Analogy:

    enor(Item) ~ $i=2..n$ enor(Item) ~ $j=1..i-1$

    cond(e)   ~ $search_{j=1..i-1} (x[i]=x[j])$ cond(e)   ~ $x[i]=x[j]$
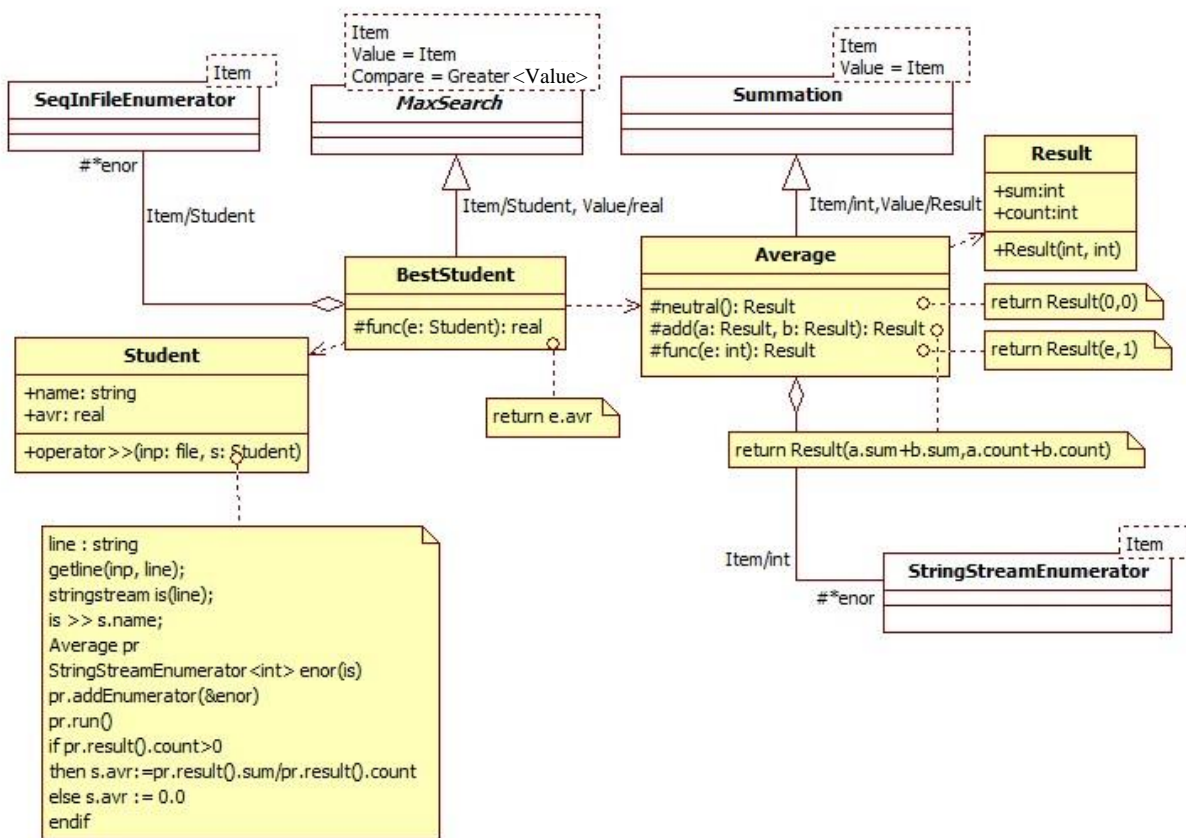
5. Who is the best student of a course (student with the best average)? Name and marks of one student are stored in one line of the input text file.

Post:
$$Student = rec(name : String, avr : \mathbb{R})$$
$$max, elem = MAX_{st \in f} (st.avr) \quad (f:infile(Student))$$
$$st.avr = (\Sigma_{e \in marks}\, e\, /\, \Sigma_{e \in marks}\, 1) \;\; if\; \Sigma_{e \in marks}\, 1 > 0,\;\; 0\;\; else$$

Analogy:

enor(Item)	~	infile(Student)	enor(Item) ~ stringstream<int>
f(e)	~	average(st)	f(e) ~ e
Value, <	~	$\mathbb{R}, <$	Value,+,$\emptyset$ ~ $(\mathbb{N}, \mathbb{N})$, $(+,+)$, $(0,0)$

6. Who is the best student of a course (student with the best average)? Name and one mark of a student are stored in one line of the input text file. One student may have several marks.
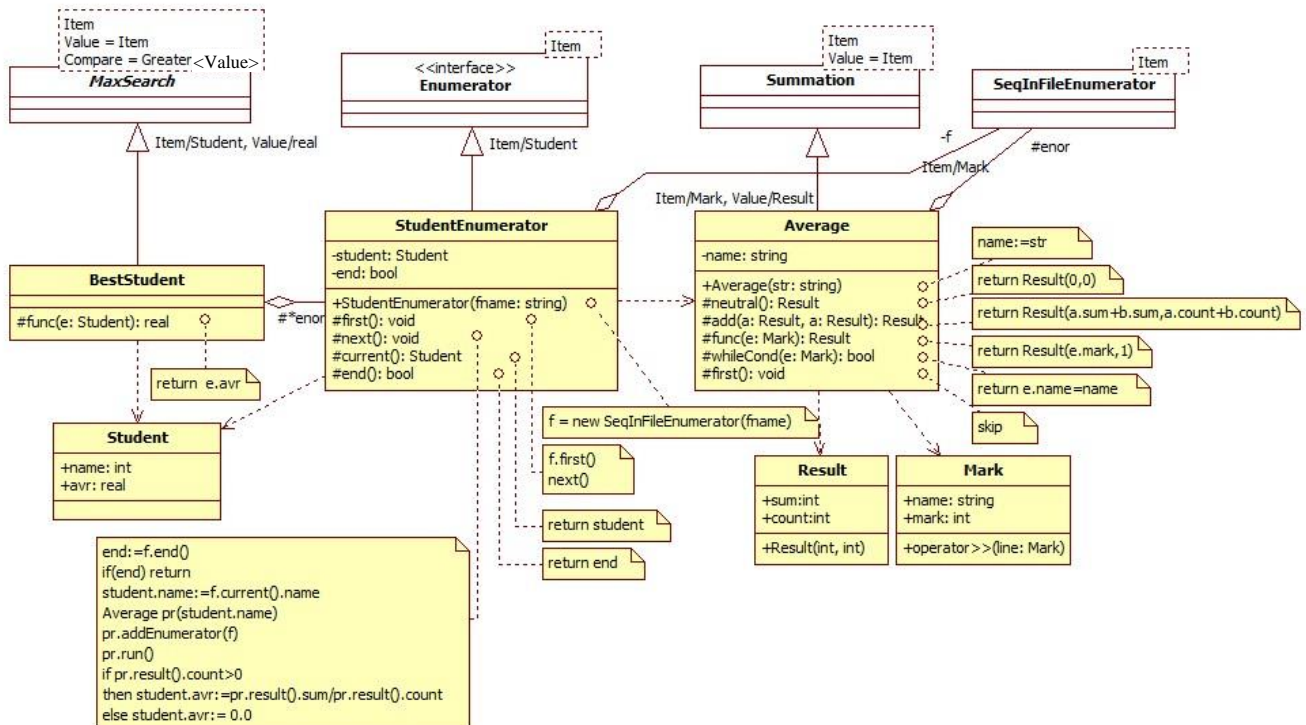
Post:
$$Student = rec(name : String, avr : \mathbb{R})$$
$$max, elem = MAX_{e \in f}(e.avr) \qquad (f:enor(Student))$$

Analogy:

enor(Item)	~	enor(Student)
f(e)	~	e.avr
H, <	~	$\mathbb{R}, <$

Abstract enumerator: enumerates the students' name and average, though, in the original file, name-mark pairs are stored.

Item
Value = Item
Compare = Greater<Value>
MaxSearch

<<interface>>
Enumerator
Item

Item
Value = Item
Summation

Item
SeqInFileEnumerator

Item/Student, Value/real

Item/Student

Item/Mark, Value/Result

-f
#enor
Item/Mark

StudentEnumerator
-student: Student
-end: bool
+StudentEnumerator(fname: string)
#first(): void
#next(): void
#current(): Student
#end(): bool

Average
-name: string
+Average(str: string)
#neutral(): Result
#add(a: Result, a: Result): Result
#func(e: Mark): Result
#whileCond(e: Mark): bool
#first(): void

name:=str
return Result(0,0)
return Result(a.sum+b.sum,a.count+b.count)
return Result(e.mark,1)
return e.name=name
skip

BestStudent
#func(e: Student): real

#**enor

return e.avr

Student
+name: int
+avr: real

f = new SeqInFileEnumerator(fname)

f.first()
next()

return student

return end

end:=f.end()
if(end) return
student.name:=f.current().name
Average pr(student.name)
pr.addEnumerator(f)
pr.run()
if pr.result().count>0
then student.avr:=pr.result().sum/pr.result().count
else student.avr:= 0.0

Result
+sum:int
+count:int
+Result(int, int)

Mark
+name: string
+mark: int
+operator>>(line: Mark)

7. Find the length of the longest island if measured elevations can be found in a text file.

Post:

max, elem = $\text{MAX}_{e \in f}$ (e.length)   (f:enor(Island) Island=rec(length : $\mathbb{N}$, no : $\mathbb{N}$))

Analogy:

t   ~   f

f(e)   ~   e.length

H, <   ~ $\mathbb{Z}$, <

abstract enumerator: enumerates the index and the length of the islands.

SeqInFileEnumerator *Item*

MaxIsland
#func(e: Island): real

Item/Island, Value/int

Item
Value = Item
Compare = Greater<Value>
MaxSearch

return e.length

Island
+length: int
+no: int

Item/int

#*enor

IslandEnumerator
- _island: Island
- _end: bool
+IslandEnumerator(fname: string)
#first(): void
#next(): void
#current(): Island
#end(): bool

-*f

Item/Island

Item
<<interface>>
Enumerator

_f = new SeqInFileEnumerator(fname)

f->first()
_island.no:=0
next()

return _island

return _end

CrossingSee pr1
pr1.addEnumerator(_f)
pr1.run()
_end:=_f->end()
if(_end) return
_island.no:=_island.no+1
IslandLength pr2
pr2.addEnumerator(_f)
pr2.run()
_island.length:=pr2.result()

#*enor

CrossingSee
#cond(e: int): bool
#first()

Item
Selection

Item/int

skip

return e<>0

#*enor

IslandLength
+IslandLength()
#whileCond(e: int): bool
#first(): void

return e>0

skip

Item/int

Item
Counting

Item/int

Other tasks:

1. Sum up the the odd numbers found in a text file, that can be found after the first even file. (Selection vs. Lin. search)
2. Sum up the the odd numbers between the first two even numbers in a text file.
3. Write a program that decides if there is an even number in a text file and gives back the sum of the odd numbers. The file can be opened only once and its content cannot be stored.
4. In a text file, there are results of different courses belonging to several students. One line consists of the name of the student (one word), the name of the course (one word), then comment-mark pairs that the student has received during that course. One student may have results in several lines of the file, but his/her marks are stored consecutively in the file (it is ordered by student name). Which student has passed the least courses? A student passes a course if every mark it at least 2.