Tervminták II. (Híd, Bejáró, Gyártófüggvény)

Gregorics Tibor

gt@inf.elte.hu

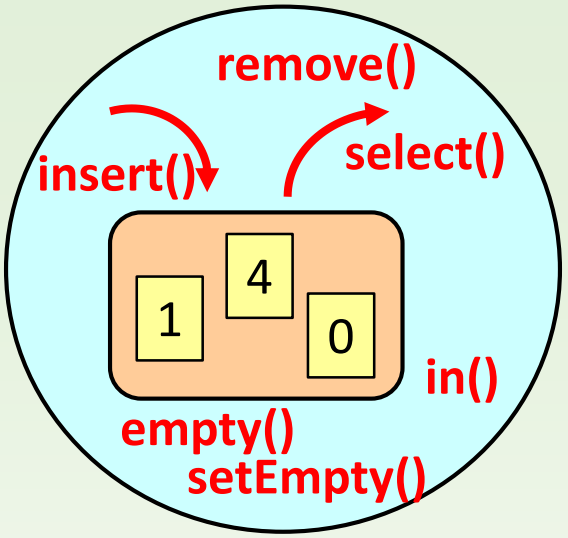
http://people.inf.elte.hu/gt/oep

1.Feladat

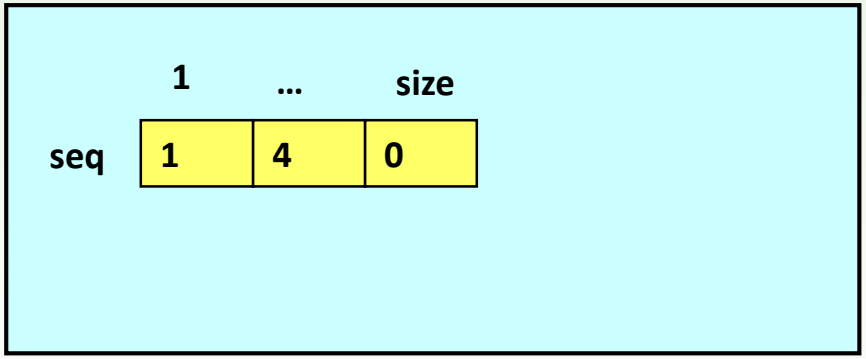
Készítsünk olyan kódot, amely segítségével természetes számokat tároló halmazok hozhatók létre.

- Egy halmaz-objektum reprezentációja attól függjön, hogy tudunk-e felső korlátot (max) adni a halmazban tárolandó természetes számokra.
  - Általában, a halmaz elemeit egy sorozat fogja tárolni.
  - Speciálisan, a halmazt egy max+1 méretű logikai értékű tömb reprezentálja majd úgy, hogy a halmaz elemei a tömb azon indexei lesznek, ahol a tömb igaz értéket tárol.
- Szeretnénk a reprezentációt elrejteni a halmazt használók elől: egy halmaz létrehozásánál csak azt kérdezzük meg, hogy tud-e a felhasználó felső korlátot mondani a halmazba kerülő természetes számokra, de ne kelljen tudnia arról, hogy ehhez milyen reprezentációt használunk.

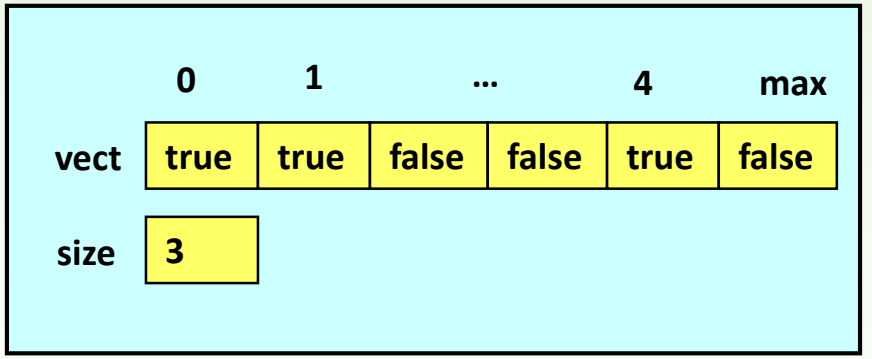
Kétféle reprezentáció



Sorozat



Tömb



1. Dinamikusan változó hosszúságú sorozat.

2. Műveletek számítási bonyolultsága többnyire lineáris, de az empty és a select konstans idejű.

1. Rögzített méretű tömb és külön a halmazbeli elemek száma.

2. Műveletek számítási bonyolultsága többnyire konstans, de a select és a setEmpty lineáris.

Halmaz típus sorozattal

$set(\mathbb{N})$

Típus-specifikáció

típusértékek

Olyan véges elemű halmazok, amelynek elemei természetes számok.

műveletek

üresé teszi a halmazt (setEmpty)
$h:=\emptyset$ $h:set(\mathbb{N})$
betesz egy elemet a halmazba (insert)
$h:=h\cup\{e\}$ $h:set(\mathbb{N}), e:\mathbb{N}$
kivesz egy elemet a halmazból (remove)
$h:=h-\{e\}$ $h:set(\mathbb{N}), e:\mathbb{N}$
kiválasztja a halmaz egy elemét (select)
$e:=mem(h)$ $h:set(\mathbb{N}), e:\mathbb{N}$
üres-e a halmaz (empty)
$l:= h=\emptyset$ $h:set(\mathbb{N}), l:\mathbb{L}$
benne van-e egy elem a halmazban (in)
$l:= e\in h$ $h:set(\mathbb{N}), e:\mathbb{N}, l:\mathbb{L}$

reprezentáció

$seq : \mathbb{N}^*$

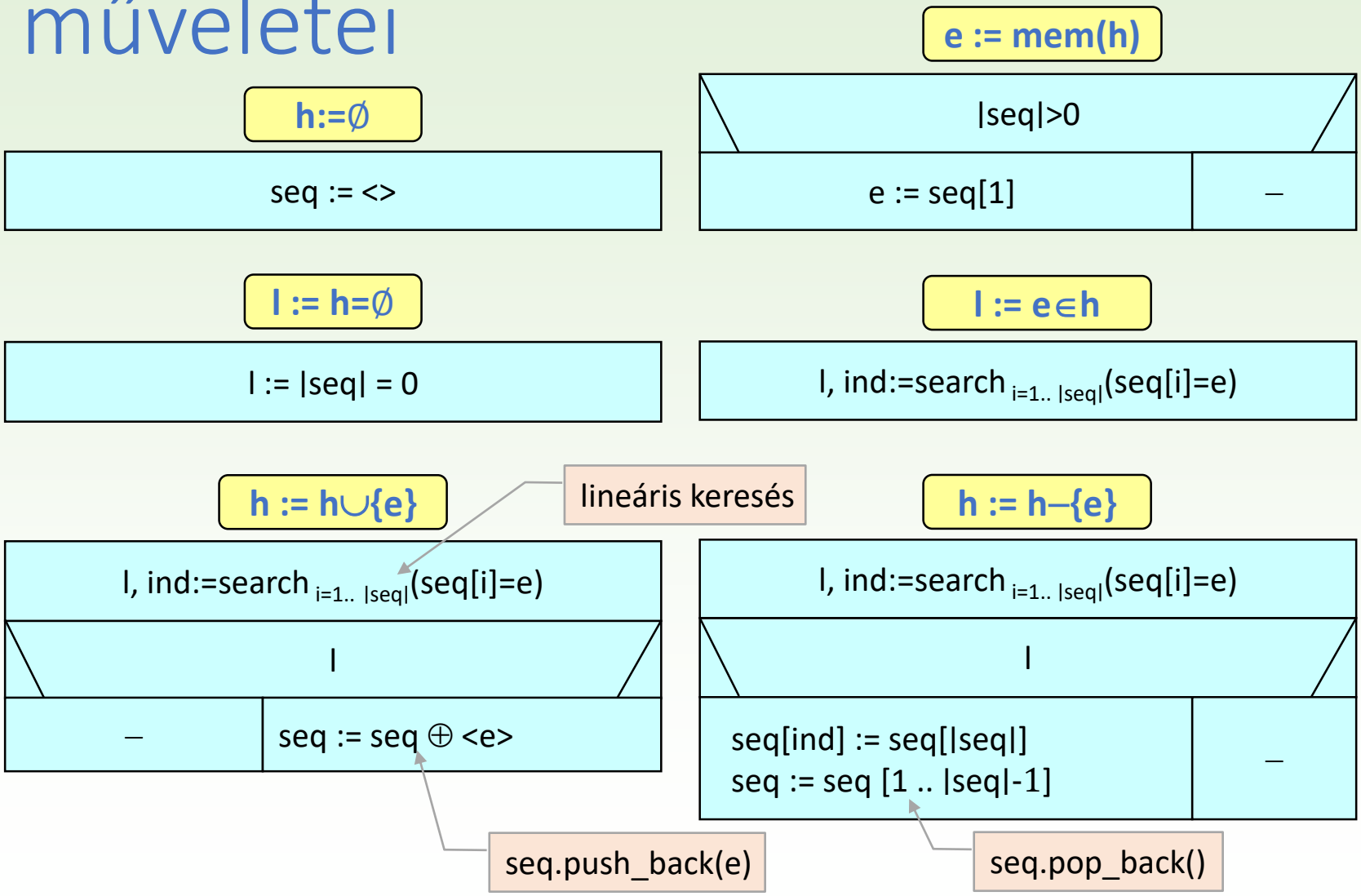
C++-ban: vector<int>

implementáció

műveletek programjai

Típus-megvalósítás

Sorozattal reprezentált halmaz műveletei



Halmaz típus tömbbel

set([0..max])

Típus-specifikáció

típusértékek

Olyan halmazok, amelynek elemei
0 és max közé eső természetes számok.

műveletek

üresé teszi a halmazt (setEmpty)
h:=∅ h:set([0..max])
betesz egy elemet a halmazba (insert)
h:=h∪{e} h:set([0..max]), e:ℕ
kivesz egy elemet a halmazból (remove)
h:=h–{e} h:set([0..max]), e:ℕ
kiválasztja a halmaz egy elemét (select)
e:=mem(h) h:set([0..max]), e:ℕ
üres-e a halmaz (empty)
l:= h=∅ h:set([0..max]), l:𝕃
benne van-e egy elem a halmazban (in)
l:= e∈h h:set([0..max]), e:ℕ, l:𝕃

reprezentáció

vect : $\mathbb{L}^{0..max}$
size : ℕ

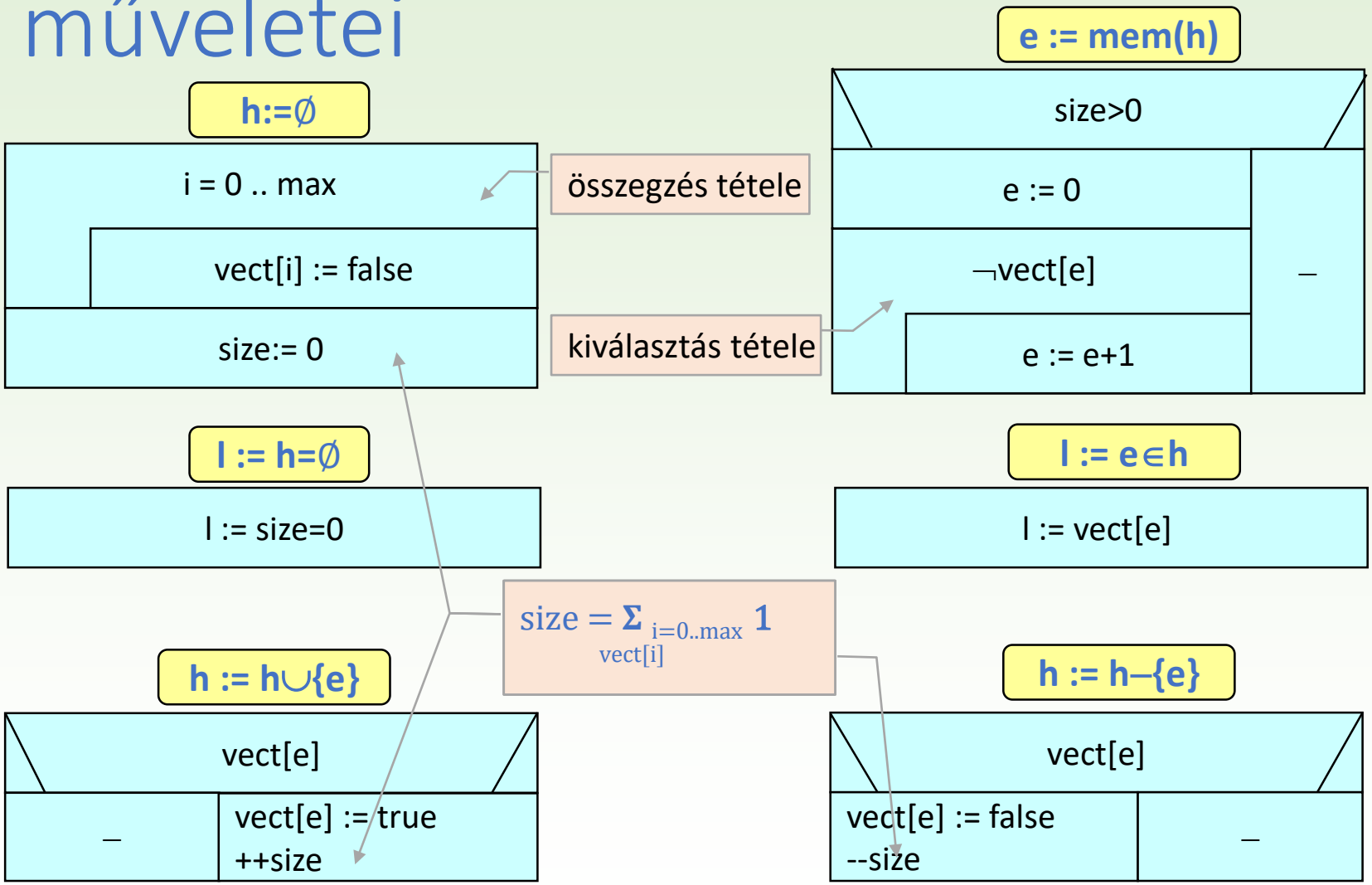
invariáns: $size = \sum_{i=0..max,\ vect[i]} 1$

implementáció

műveletek programjai

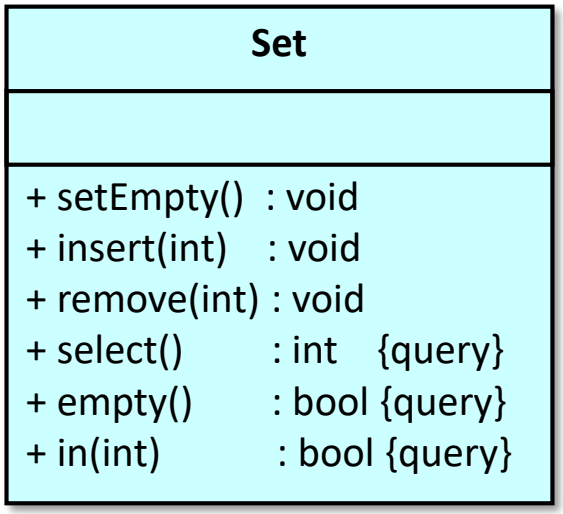
Típus-megvalósítás

Tömbbel reprezentált halmaz műveletei



Halmaz osztály publikus része

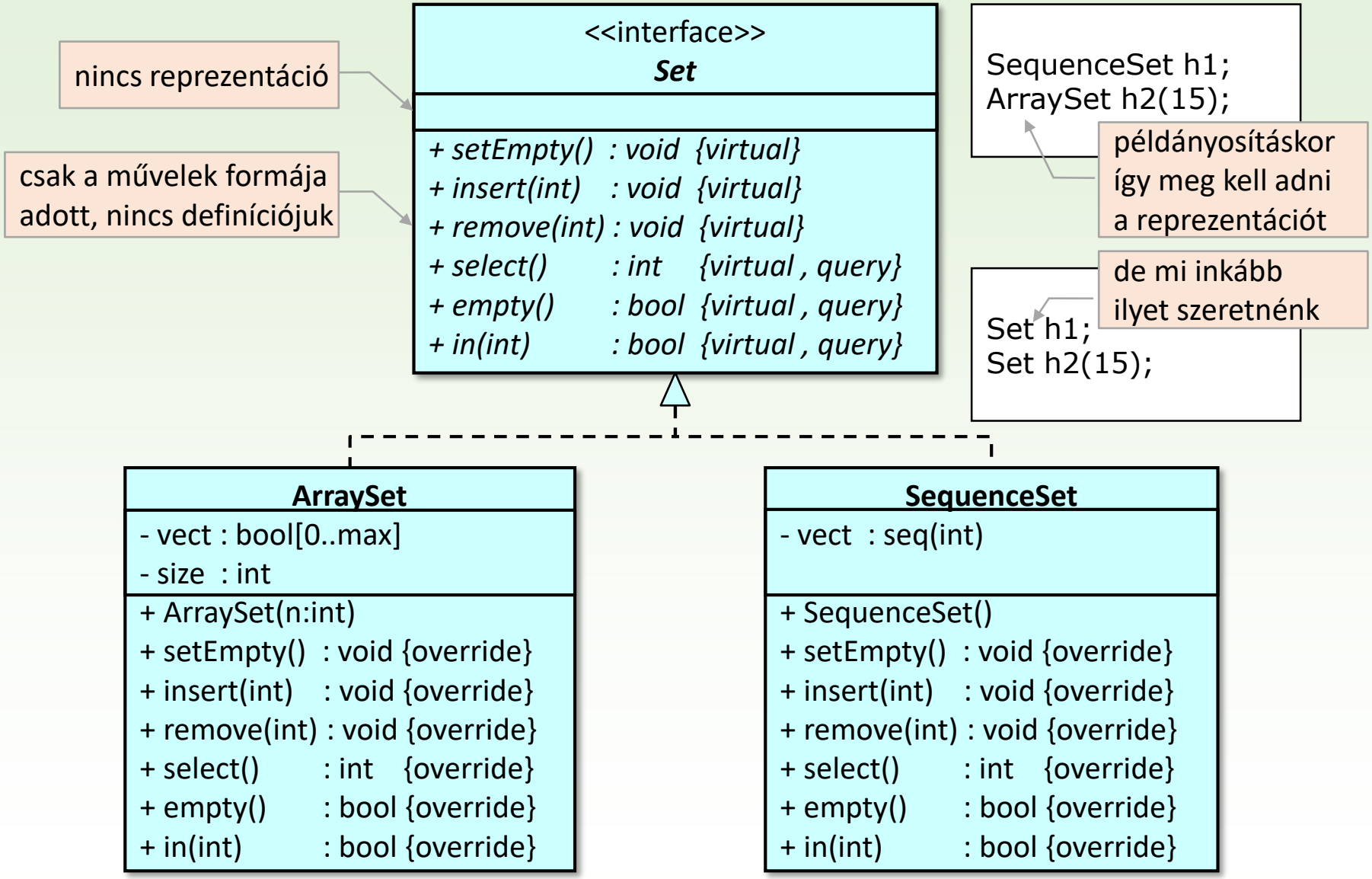
```
class Set
{
      public:

            void setEmpty();
            void insert(const int &e);
            void remove(const int &e);
            int    select()  const;
            bool empty()  const;
            bool in(int e) const;

      private:
            …
};
```



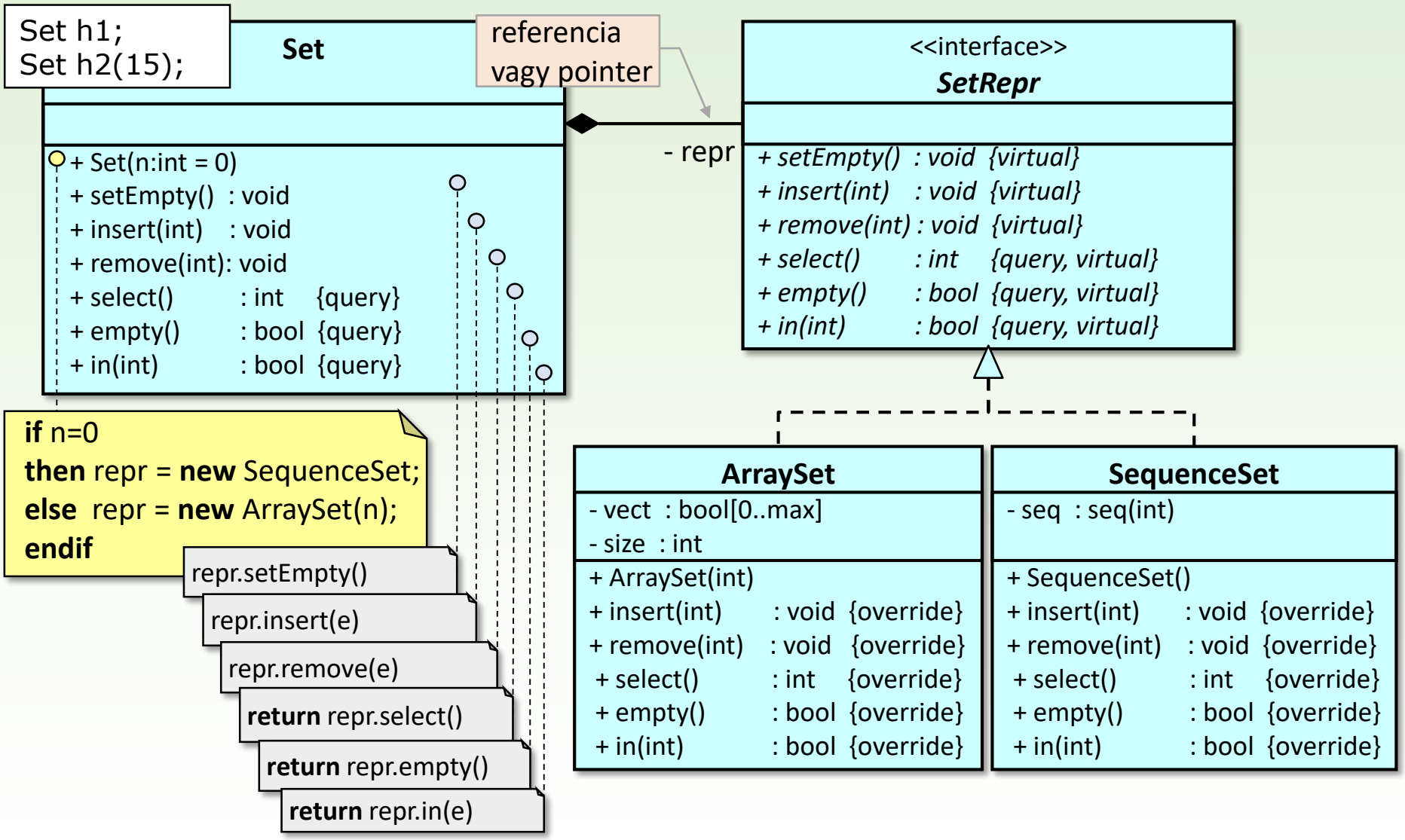
Hogyan írható le egyszerre mindkét reprezentáció?

set.h

Osztálydiagram 1. változat

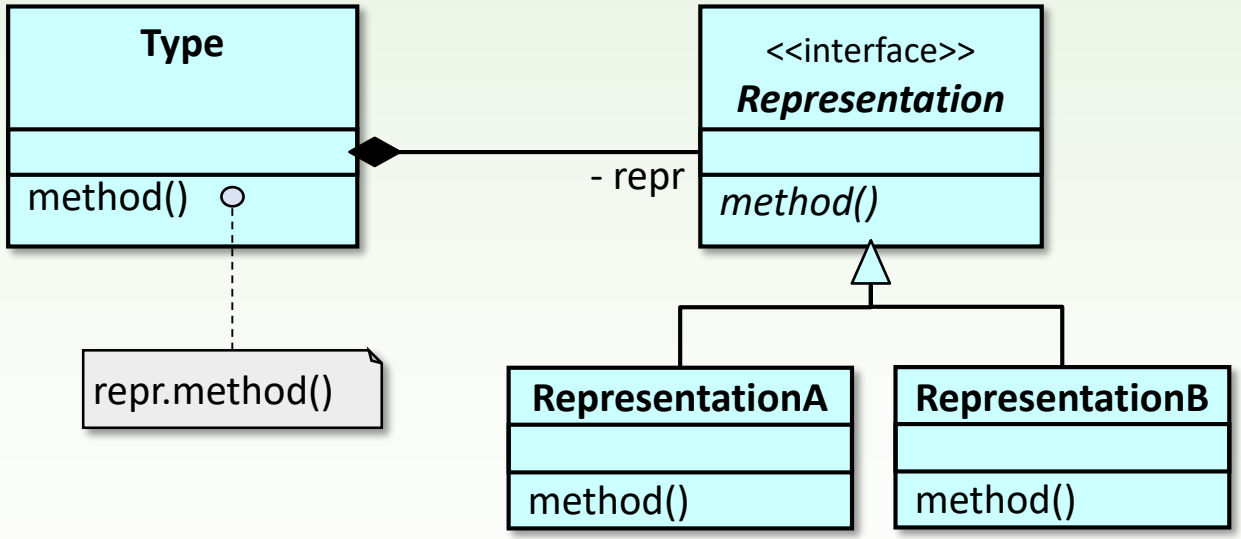


Osztálydiagram 2. változat



Híd (bridge) tervezési minta

- Egy osztály reprezentációját leválasztjuk az osztályról azért, hogy az rugalmasan kicserélhető legyen.



A tervezési minták az objektum-alapú modellezést támogató osztálydiagram minták, amelyek az újrafelhasználhatóság, rugalmas módosíthatóság, hatékonyság biztosításában játszanak szerepet.

Halmaz osztály inline módon

```
#include "setrepr.h"
#include "array_set.h"
#include "sequence_set.h"

class Set {
public:
      Set(int n = 0) {
            if (0 == n) _repr = new SequenceSet;
            else        _repr = new ArraySet(n);
      }
      ~Set() { delete _repr; }
      void setEmpty()          { _repr->setEmpty(); }
      void insert(int e)       { _repr->insert(e); }
      void remove(int e)       { _repr->remove(e); }
      int    select()  const   { return _repr->select(); }
      bool empty()  const      { return _repr->empty(); }
      bool in(int e) const     { return _repr->in(e); }
private:
      SetRepr *_repr;

      Set(const Set& h);
      Set& operator=(const Set& h);
};
```

Set
+ Set(n:int = 0) + setEmpty() : void + insert(int) : void + remove(int) : void + select() : int {query} + empty() : bool {query} + in(int) : bool {query}

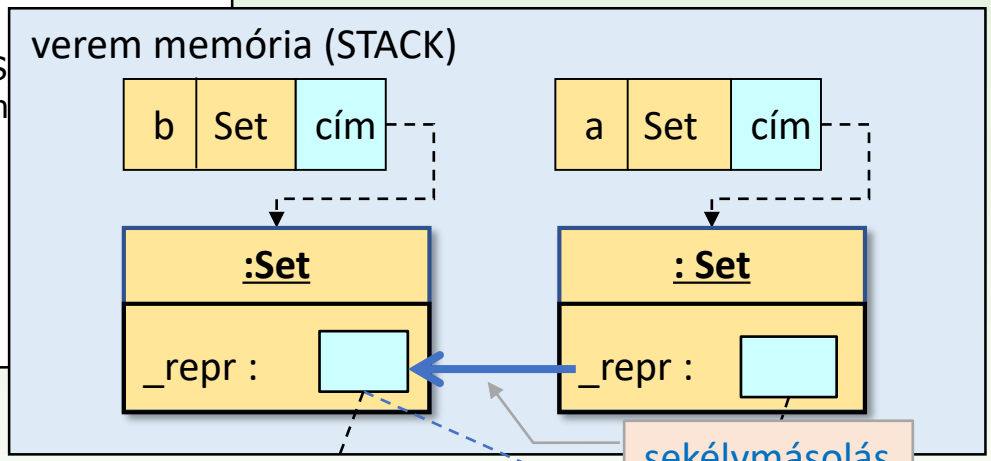
pointer

Itt az alapértelmezett másoló konstruktor és értékadás operátor rosszul működne. Legyenek privátok, így nem használhatók. Később felülírhatjuk és publikussá tehetjük.

set.h

Kitérő: amikor az alapértelmezett másolás és értékadás rossz

```
class Set {
public:
  Set(int n = 0) {
    if (0 == n) _repr = new SequenceS
    else        _repr = new ArraySet(n
  }
  ~Set() { delete _repr; }
      ...
private:
  SetRepr *_repr;
};
```




```
Set a, b;
b = a;
```

értékadás operátor

memória szivárgást okoz

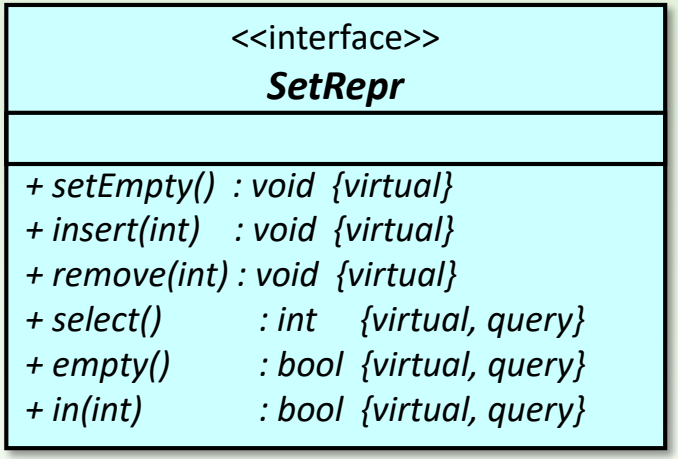
látszólag független halmazok azonosak

mélymásolás kellene

```
Set b = a;
```

másoló konstruktor

Reprezentáció interfésze



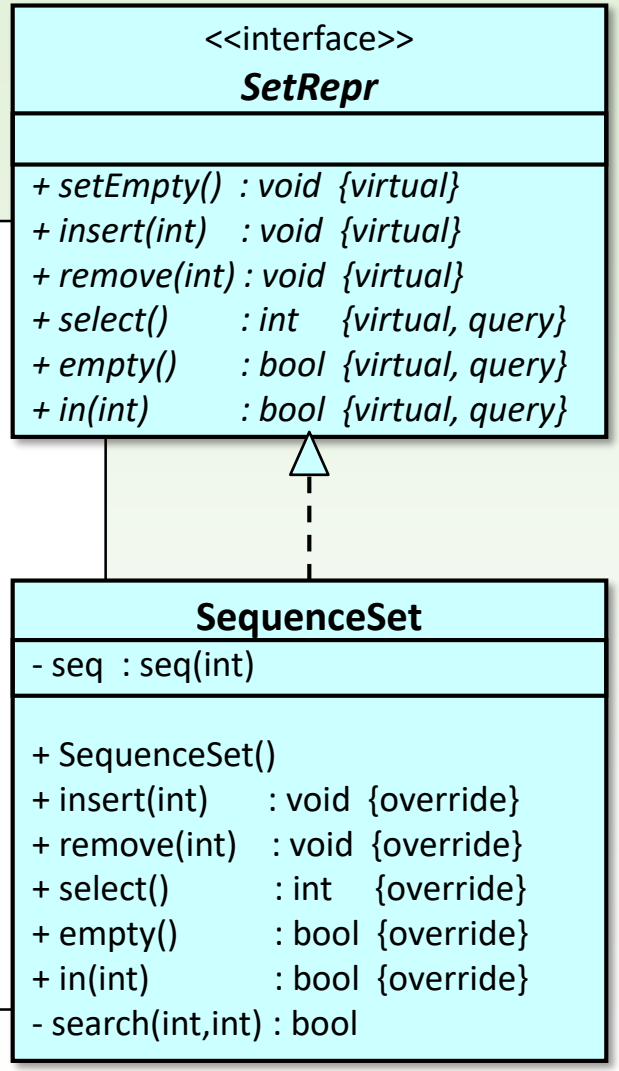

```
class SetRepr
{
public:
        virtual void setEmpty()         = 0;
        virtual void insert(int e)      = 0;
        virtual void remove(int e)      = 0;
        virtual int  select() const     = 0;
        virtual bool empty() const      = 0;
        virtual bool in(int e) const    = 0;
        virtual ~SetRepr(){}
};
```

setrepr.h

Sorozat-reprezentáció

```
#include "setrepr.h"
#include <vector>

class SequenceSet : public SetRepr{
public:
      SequenceSet ():SetRepr() { setEmpty(); }
      void setEmpty()            override;
      void insert(int e)         override;
      void remove(int e)         override;
      int  select()    const     override;
      bool empty() const         override;
      bool in(int e) const       override;
private:
      std::vector<int> _seq;
      bool search(int e, unsigned int &ind) const;
};
```

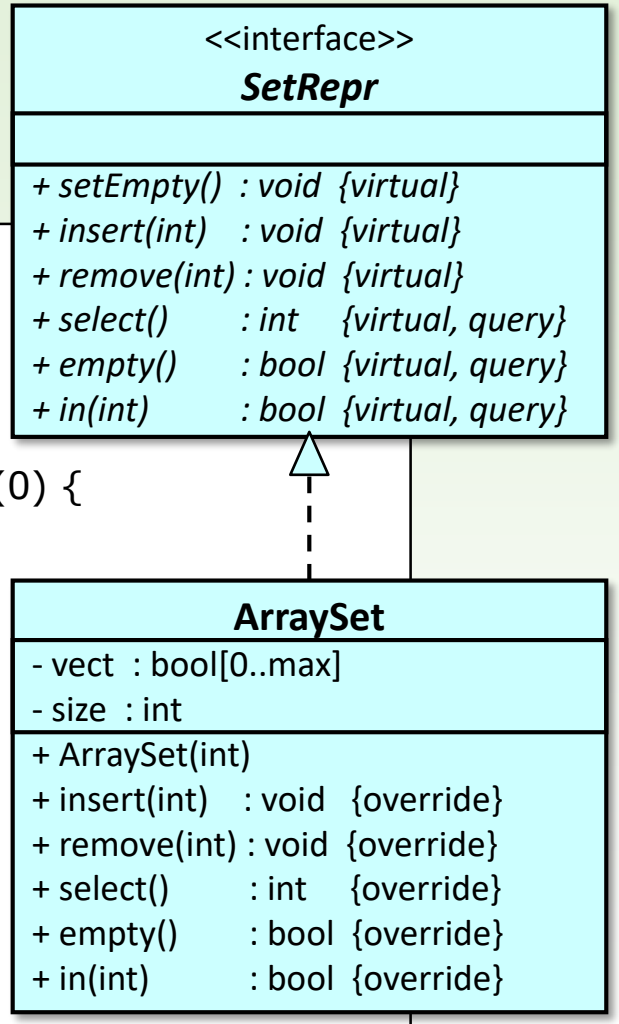
sequence_set.h



Tömb-reprezentáció

```
#include "setrepr.h"
#include <vector>

class ArraySet : public SetRepr{
public:
        ArraySet (int n) : SetRepr(), _vect(n+1), _size(0) {
                setEmpty();
        }
        void setEmpty()          override;
        void insert(int e)       override;
        void remove(int e)       override;
        int    select() const    override;
        bool empty() const       override;
        bool in(int e) const     override;
private:
        std::vector<bool> _vect;
        int _size;
};
```

array_set.h



Kivételek osztályai

```
#include <exception>
#include <sstream>

class EmptySetException : public std::exception {
public:
	const char* what() const noexcept override {
		return "Empty set";
	}
};

class IllegalElementException : public std::exception {
private:
	int _e;
public:
	IllegalElementException(int e): _e(e) {}
	const char* what() const noexcept override {
		std::ostringstream os;
		os <<  "Illegal element: " << _e;
		std::string str = os.str();
		char* msg = new char[str.size() + 1];
		std::copy(str.begin(), str.end(), msg);
		msg[str.size()] = '\0';
		return msg;
	}
};
```

szabványos kivétel osztályok

setrepr.h

Kivételek dobása

```
Set h(100);
try {
      h.insert(101);
      int e = h.select()
} catch(std::exception &ex){
      cout << ex.what() << endl;
}
```

```
int Set::select() const
{
      if (empty()) throw EmptySetException();
      return repr->select();
}
```

kivétel példányosítás és dobás

```
bool ArraySet::in(int e) const
{
      if (e<0 || e>int(vect.size())-1) throw IllegalElementException(e);
      return _vect[e];
}

bool ArraySet::insert(int e) const
{
      if (e<0 || e>int(vect.size())-1) throw IllegalElementException(e);
      return _vect[e];
}
```

2.Feladat

Keressünk egy természetes számokat tartalmazó halmazban olyan számot, amely nagyobb a halmaz legalább három másik eleménél!

(Ez a keresés biztos sikertelen lesz, ha nincs a halmazban legalább négy szám, és biztosan sikeres, ha van.)

- A feladat megoldható a halmaz elemei közti lineáris kereséssel, amely során minden elemnél egy számlálással határozzuk meg azt, hogy hány nálánál kisebb érték van a halmazban.
- Mindkét programozási tételhez a halmaz elemeit kell felsorolni.

Specifikáció

A : h:set(ℕ), l:𝕃, n:ℕ

Ef: $h = h_0$

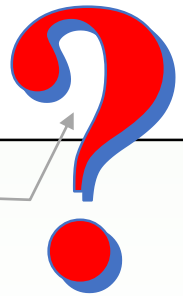
Uf: $l, n = SEARCH_{e \in h_0}(kisebbekszáma(h_0, e) \geq 3)$

$$kisebbekszáma(h_0, e) = \sum_{\substack{u \in h_0 \\ e > u}} 1$$

a halmaz egy lehetséges felsorolása:

first()	~	-
current()	~	select()
next()	~	remove(select())
end()	~	empty()

```
bool l = false;
int n;
for ( ; !l && !h.empty(); h.remove(h.select())){
      n = h.select();
      int c = 0;
      for ( ; !h.empty(); h.remove(h.select())){
            if (n > h.select()) ++c;
      }
      l = c>=3;
}
```

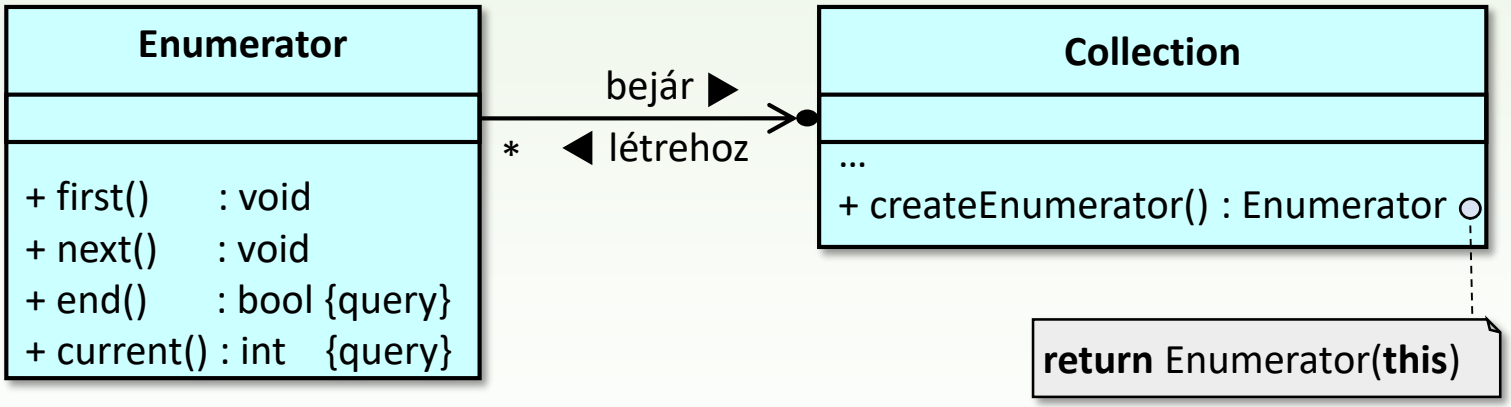


Ez a megoldás rossz, mert

- a két felsorolás nem független: mindkettő ugyanazt a felsorolást használja (a belső ciklus nem újrakezdi, hanem folytatja a külső ciklusban elkezdett felsorolást, sőt be is fejezi azt)
- a felsorolás módosítja a halmazt: a belső ciklus már első alkalommal törli a halmaz összes elemét.

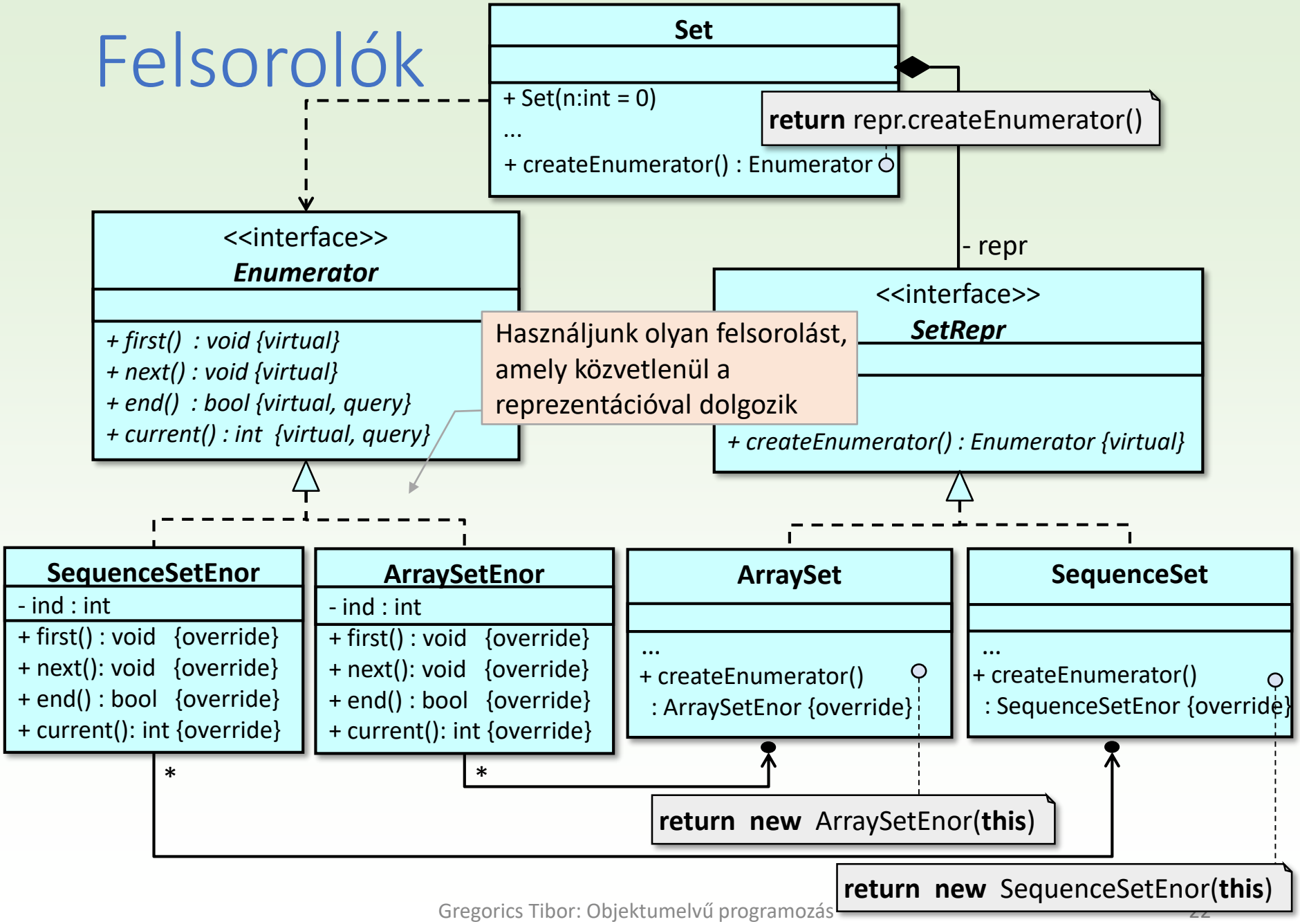
Bejáró (iterátor) tervezési minta

- Egy gyűjtemény elemeinek felsorolását (bejárását) egy attól független objektum (felsoroló) végzi, amely eléri a felsorolandó gyűjteményt (hivatkozik rá vagy annak konstans másolatára). A felsoroló objektumot a gyűjtemény hozza létre.



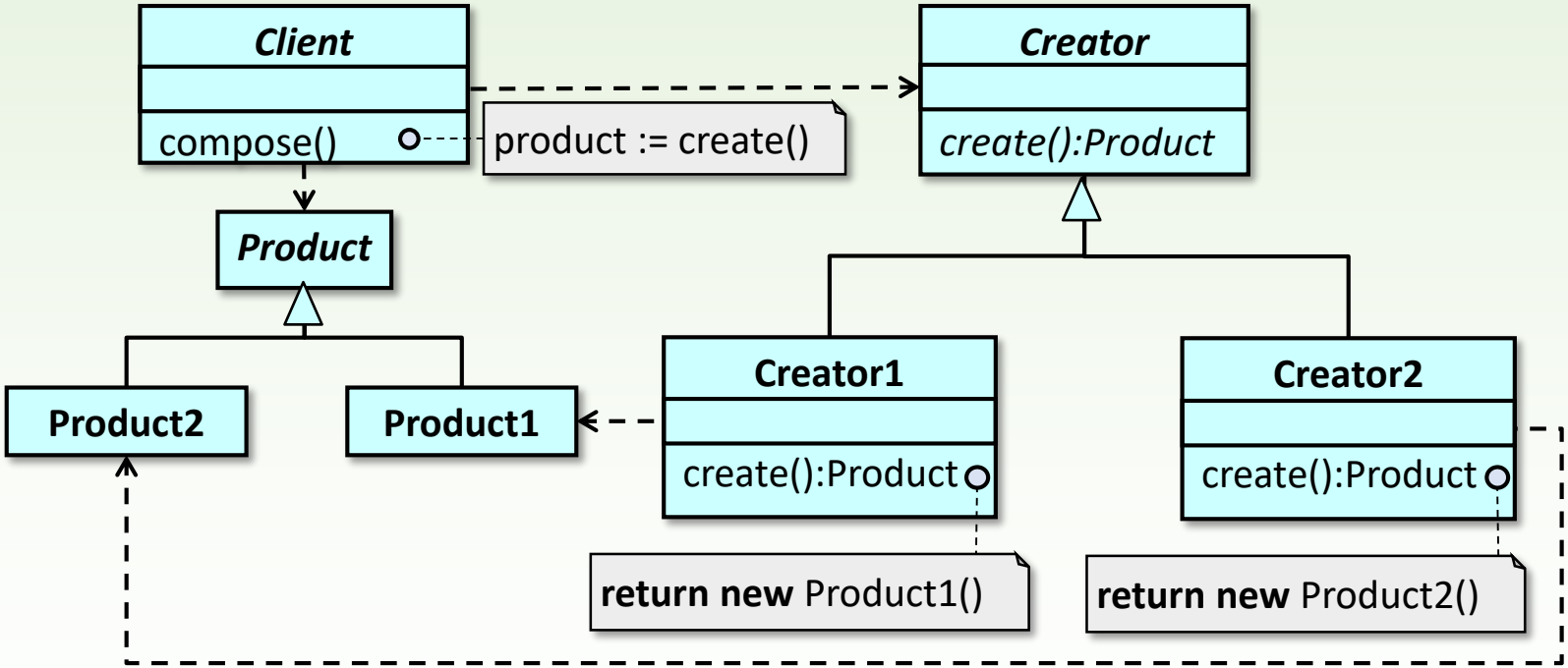
A tervezési minták az objektum-alapú modellezést támogató osztálydiagram minták, amelyek az újrafelhasználhatóság, rugalmas módosíthatóság, hatékonyság biztosításában játszanak szerepet.

Felsorolók


Set
+ Set(n:int = 0)
...
+ createEnumerator() : Enumerator
return repr.createEnumerator()
- repr
<<interface>>
Enumerator
+ first() : void {virtual}
+ next() : void {virtual}
+ end() : bool {virtual, query}
+ current() : int {virtual, query}
Használjunk olyan felsorolást, amely közvetlenül a reprezentációval dolgozik
<<interface>>
SetRepr
+ createEnumerator() : Enumerator {virtual}
SequenceSetEnor
- ind : int
+ first() : void {override}
+ next(): void {override}
+ end() : bool {override}
+ current(): int {override}
ArraySetEnor
- ind : int
+ first() : void {override}
+ next(): void {override}
+ end() : bool {override}
+ current(): int {override}
ArraySet
...
+ createEnumerator()
: ArraySetEnor {override}
SequenceSet
...
+ createEnumerator()
: SequenceSetEnor {override}
*
*
return new ArraySetEnor(this)
return new SequenceSetEnor(this)

Gyártófüggvény tervezési minta (factory method)

- A kliens nem tudja, milyen típusú termék-objektumot kell létrehoznia, és ezt a felelősséget átruházza a segítő alosztályok egyikére.



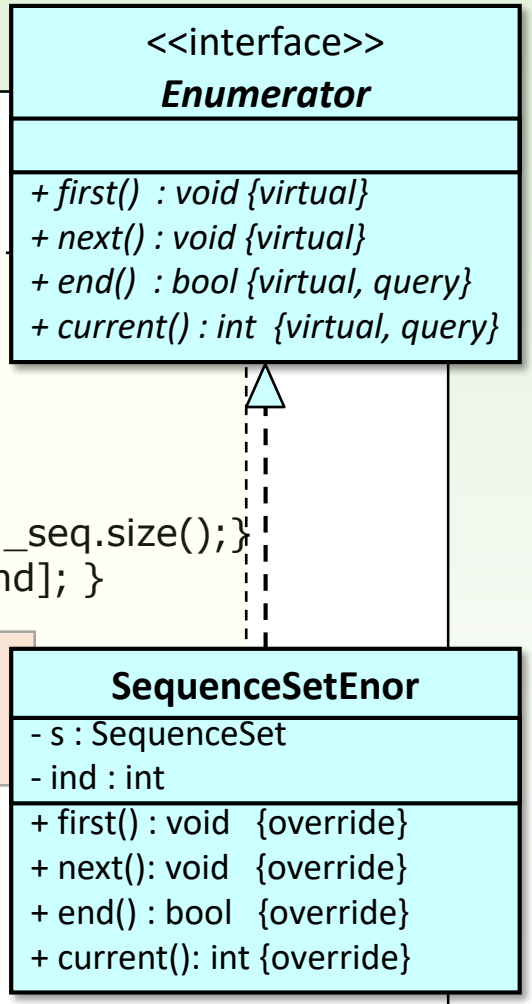
A tervezési minták az objektum-alapú modellezést támogató osztálydiagram minták, amelyek az újrafelhasználhatóság, rugalmas módosíthatóság, hatékonyság biztosításában játszanak szerepet.

SequenceSet felsorolója

```
class SequenceSet{
public:
    ...
    class SequenceSetEnor : public Enumerator{
    public:
        SequenceSetEnor(SequenceSet *h): _s(h) {}
        void first()            override { _ind = 0; }
        void next()             override { ++_ind;}
        bool end()    const override { return _ind ==_s->_seq.size();}
        int current() const override { return _s->_seq[_ind]; }
    private:
        SequenceSet  *_s;
        unsigned int  _ind
    };

    Enumerator* createEnumerator() override{
        return new SequenceSetEnor(this);
    }
private:
    std::vector<int> _seq;
};
```

A beágyazott olyan mintha friend lenne: látja a beágyazó osztály rejtett tagjait is.

A halmaz elemeinek felsorolását az azt reprezentáló sorozat elemeinek felsorolása valósítja meg.

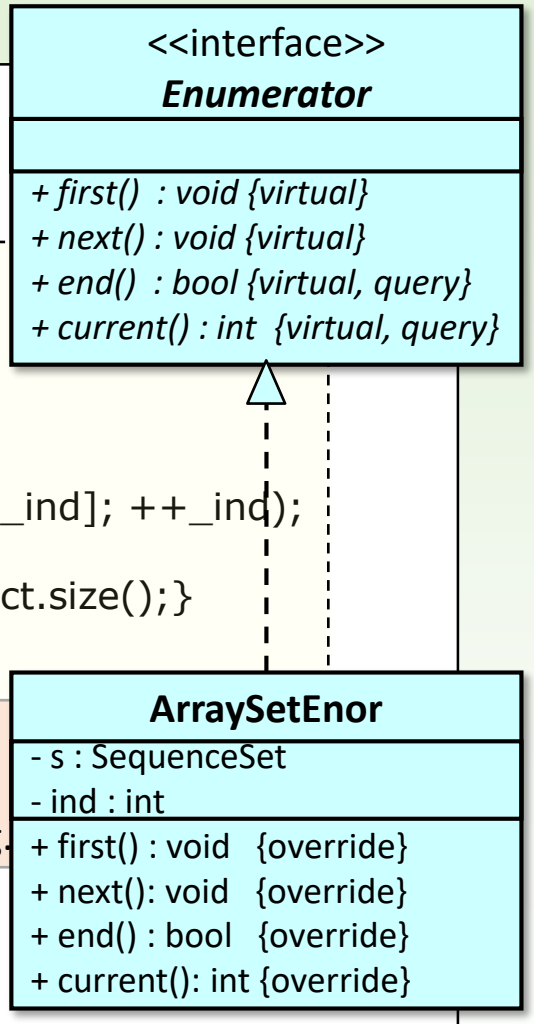


sequence_set.h

ArraySet felsorolója

```
class ArraySet{
public:
    …
    class ArraySetEnor : public Enumerator{
    public:
        ArraySetEnor(ArraySet *h): _s(h) {}
        void first()     override { _ind = -1; next(); }
        void next()      override {
            for (++_ind;_ind<_s->_vect.size() && !_s->_vect[_ind]; ++_ind);
        }
        bool end()       const override {return _ind==_s->_vect.size();}
        int  current()   const override {return _ind; }
    private:
        ArraySet  *_s;
        unsigned int _ind
    };

    Enumerator* createEnumerator() override{
        return new ArraySetEnor(this);
    }
private:
    std::vector<bool> _vect;
    int _size;
};
```

A halmaz elemeinek felsorolását az azt reprezentáló tömb true értékeihez tartozó indexek felsorolása valósítja meg.



array_set.h

Főprogram

```
Set h;
// Beolvasás
…

Enumerator* enor1 = h.createEnumerator();
bool l = false;
for (enor1->first(); !l && !enor1->end(); enor1->next()){
        int n = enor1->current();
        int c = 0;
        Enumerator* enor2 = h.createEnumerator();
        for (enor2->first(); !enor2->end(); enor2->next()){
                if (n > enor2->current()) ++c;
        }
        l = c>=3;
}

if (l) cout << "A keresett szám: " << n << endl;
else   cout << "Nincs keresett szám.\n";
```

háttérben:
a repr–>createEnumerator() hatására
példányosodik egy SequenceSetEnor(this)

main.cpp

3.Feladat

Tegyük biztonságossá a felsorolást!

- Probléma: ha a felsorlás közben megváltozik a halmaz (pl. a setEmpty(), insert(), remove(), értékadás operátor, destruktor hatására), akkor a felsorolás elromolhat.
- Megoldás: Zárjuk ki az ilyen műveleteket végrehajtását felsorolás közben.

```
Set h;
 ...
Enumerator * enor = h.createEnumerator();
for (enor->first(); !enor->end(); enor->next()){
      h.remove(enor->current());
}
```

Hibát okozhat a felsorolás során végrehajtott törlés, hiszen elveszítjük a felsorolás aktuális elemét.

Kizárás megvalósítása

```
Set h;
try {
        …
        h.remove(enor->current());
        …
} catch(std::exception &ex){
        cout << ex.what() << endl;
}
```

```
class Set {
public:
        …
        void Set::remove(int e)
        {
                if (_repr->getEnumCount()!=0)
                        throw UnderTraversalException();
                _ref->remove(e);
        }
        …
};
```

set.h

A kritikus művelet dobjon kivételt, ha az adott halmazon éppen felsoroló dolgozik. Ehhez ismerni kell a halmazon dolgozó felsorolók számát.

```
class UnderTraversalException : public std::exception {
public:
        const char* what() const noexcept override {
                return "Under traversal";
        }
};
```

setrepr.h

már nem interfész, de még absztrakt osztály

```
class SetRepr {
public:
        SetRepr(): _enumeratorCount(0){}
        …
        int getEnumCount() const { return _enumeratorCount; }
protected:
        int _enumeratorCount;
};
```

aktív felsorolók száma

setrepr.h

Felsoroló-számlálás a SequenceSet-ben

```
class SequenceSet : public SetRepr{
public:
    SequenceSet() : SetRepr() { setEmpty(); }
    …
    class SequenceSetEnor : public Enumerator{
    public:
        SequenceSetEnor(SequenceSet *h): _s(h)
        { ++(_s->_enumeratorCount); }
        ~ SequenceSetEnor() { --(_s->_enumeratorCount); }
        …
    };

    Enumerator* createEnumerator() override {
        return new SequenceSetEnor(this);
    }
};
```

lenullázza a felsorolók számlálóját

Amikor egy új felsorolót példányosítunk egy SequenceSet objektumhoz, akkor annak felsoroló-számlálóját növeljük.

A felsoroló megszűnésekor a felsoroló-számlálót csökkentjük.

sequence_set.h

Felsoroló-számlálás az ArraySet-ben

```
class ArraySet : public SetRepr {
public:
    ArraySet(int n) : SetRepr(), _vect(n+1), _size(0) {
        setEmpty();
    }
    ...
    class ArraySetEnor : public Enumerator {
    public:
        ArraySetEnor(ArraySet *h): _s(h)
        { ++(_s->_enumeratorCount); }
        ~ArraySetEnor() { --(_s->_enumeratorCount); }
        ...
    };

    Enumerator * createEnumerator() override {
        return new ArraySetEnor(this);
    }
};
```

lenullázza a felsorolók számlálóját

Amikor egy új felsorolót példányosítunk egy ArraySet objektumhoz, akkor annak felsoroló-számlálóját megnöveljük.

a felsoroló megszűnésekor a felsoroló-számlálót csökkentjük

array_set.h

4.Feladat

- Azért, hogy ne csak integereket lehessen a halmazainkban tárolni, az osztályok helyett használjunk osztálysablonokat, amelyeknél partaméterként adhassuk meg a halmazban tárolt elemek típusát.
- Vegyük figyelembe, hogy a tömbös reprezentációval kizárólag a felső korláttal rendelkező természetes számokat tároló halmazokat tudunk ábrázolni, azaz ebben az esetben a sablonparaméter csak **int** lehet.

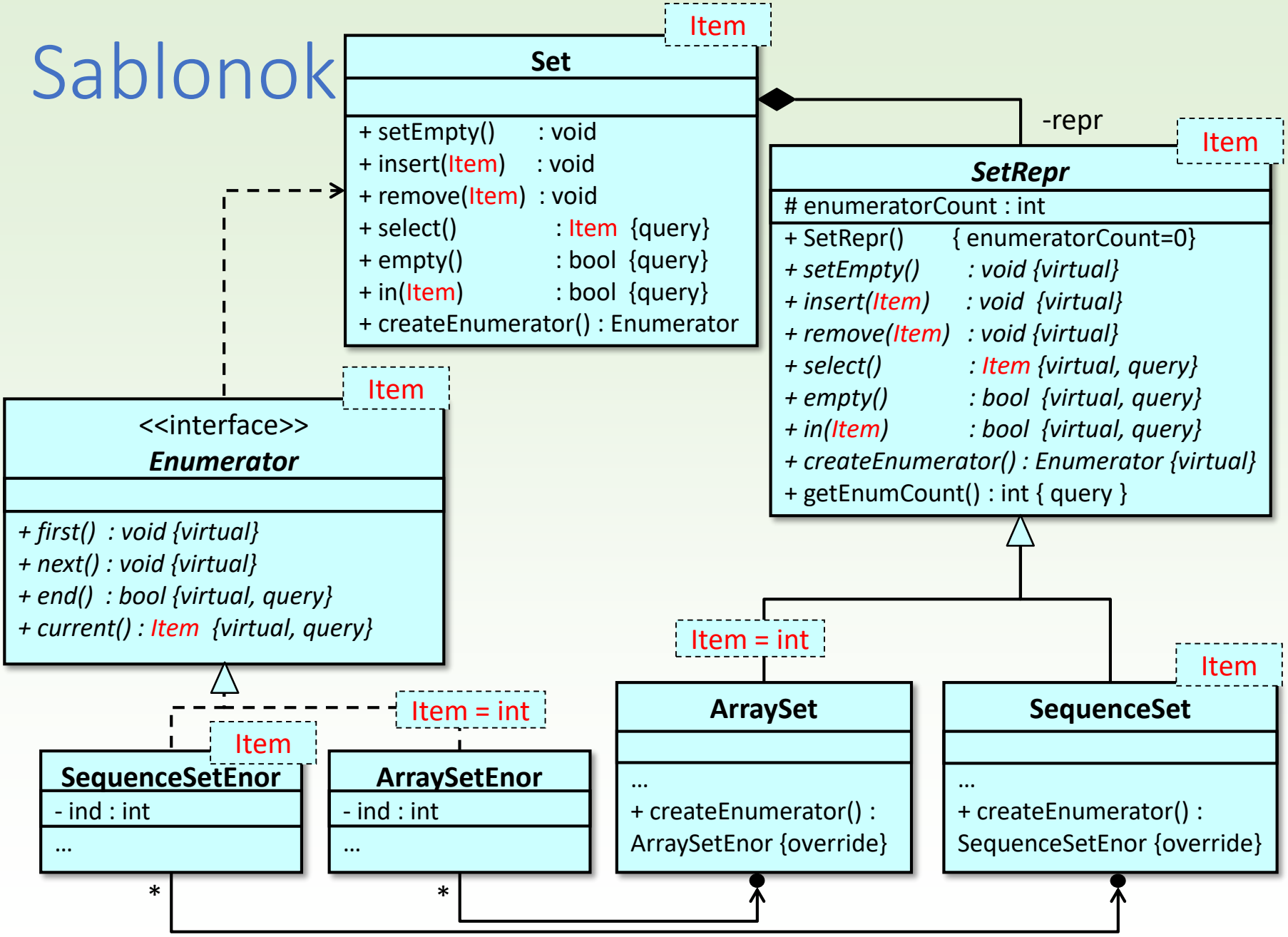
fordítási időben osztályként példányosodik az osztálysablon
futási időben objektumként példányosodik az osztály

```
Set<int>    h1(100);
Set<int>    h2;
Set<string> h3;

h1.insert(42);
h2.insert(1456);
h3.insert("alma");

Enumerator<int>    *enor1 = h1.createEnumerator();
Enumerator<string> *enor2 = h3.createEnumerator();
```

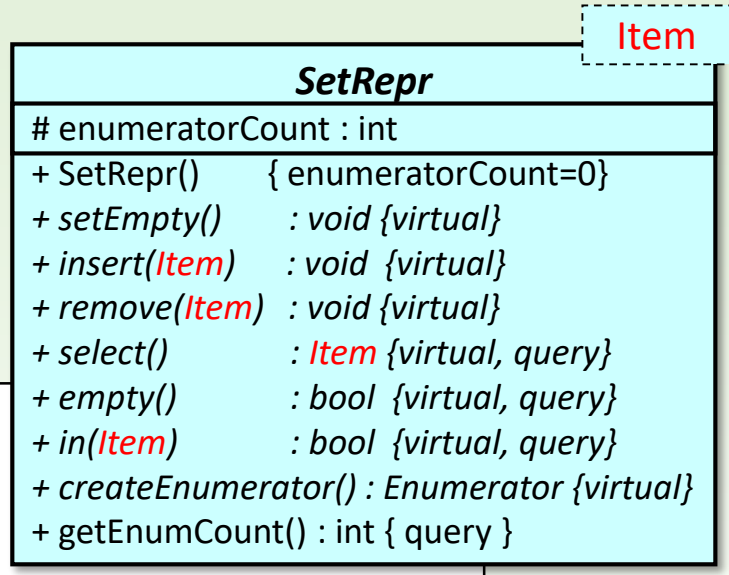
A felsoroló osztályok is osztálysablonok példányai lesznek, amelyek típusparaméterét egyeztetni kell a halmazéval.

main.cpp

Sablonok


Item
Set
+ setEmpty() : void
+ insert(Item) : void
+ remove(Item) : void
+ select() : Item {query}
+ empty() : bool {query}
+ in(Item) : bool {query}
+ createEnumerator() : Enumerator
-repr
Item
SetRepr
enumeratorCount : int
+ SetRepr() { enumeratorCount=0}
+ setEmpty() : void {virtual}
+ insert(Item) : void {virtual}
+ remove(Item) : void {virtual}
+ select() : Item {virtual, query}
+ empty() : bool {virtual, query}
+ in(Item) : bool {virtual, query}
+ createEnumerator() : Enumerator {virtual}
+ getEnumCount() : int { query }
Item
<<interface>>
Enumerator
+ first() : void {virtual}
+ next() : void {virtual}
+ end() : bool {virtual, query}
+ current() : Item {virtual, query}
Item = int
ArraySet
...
+ createEnumerator() : ArraySetEnor {override}
Item
SequenceSet
...
+ createEnumerator() : SequenceSetEnor {override}
Item = int
Item
SequenceSetEnor
- ind : int
...
ArraySetEnor
- ind : int
...
*
*

SetRepr osztály-sablonja



jelöli a sablont, megadja a sablon paramétereit

```
template <typename Item>
class SetRepr {
public:
      SetRepr() : _enumeratorCount(0) {}
      virtual ~SetRepr(){};
      virtual void setEmpty()        = 0;
      virtual void insert(Item e)    = 0;
      virtual void remove(Item e)    = 0;
      virtual Item select()    const = 0;
      virtual bool empty()     const = 0;
      virtual bool in(Item e)const = 0;
      virtual Enumerator<Item>* createEnumerator() = 0;
      int getEnumCount() const { return _enumeratorCount; }
protected:
      int _enumeratorCount;
};
```

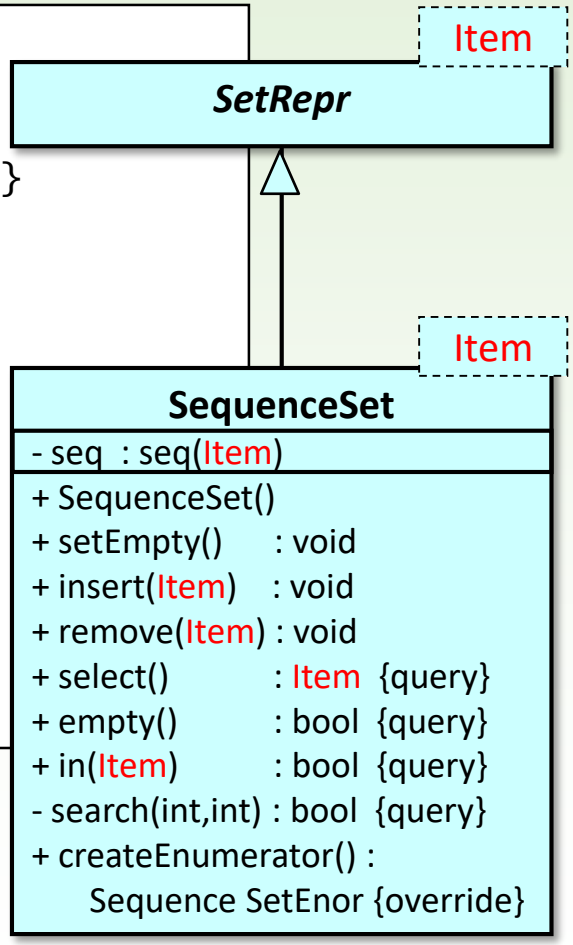
setrepr.hpp

SequenceSet osztály-sablonja

```
template <typename Item>
class SequenceSet : public SetRepr<Item>{
public:
      SequenceSet () : SetRepr<Item> { setEmpty(); }
      void setEmpty()       override;
      void insert(Item e)   override;
      void remove(Item e)   override;
      Item select()   const override;
      bool empty()    const override;
      bool in(Item e)const override;
      …
private:
      std::vector<Item> _seq;
      bool search(Item e, unsigned int &ind) const;
};
…
```

sequence_set.hpp

Ugyanabba az állományba kerül az osztálysablon definíciója (.h) és a sablon-metódusainak definíciója (.cpp).



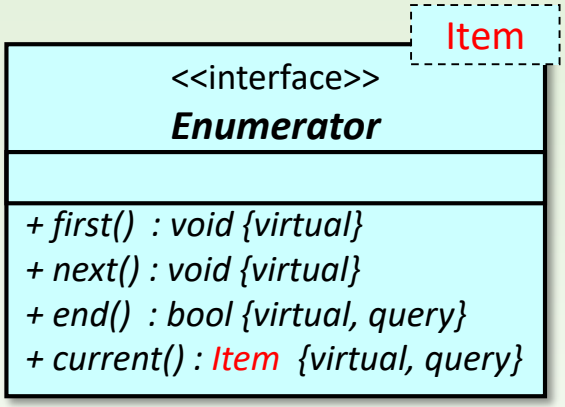
SequenceSet metódus-sablonjai

Nemcsak egy osztály, hanem egy függvény is lehet sablon, mint ahogy egy sablonosztály metódusai is sablonok.

```
template <typename Item>
void SequenceSet<Item>::insert(int e)
{
      unsigned int ind;
      if (!search(e,ind)) _seq.push_back(e);
}
template <typename Item>
void SequenceSet<Item>::remove(int e)
{
      unsigned int ind;
      if (search(e,ind)){
            _seq[ind] = _seq[_seq.size()-1];
            _seq.pop_back();
      }
}
template <typename Item>
int SequenceSet<Item>::select() const
{
      return _seq[0];
}

...
```

sequence_set.hpp

Enumerator interfész-sablon



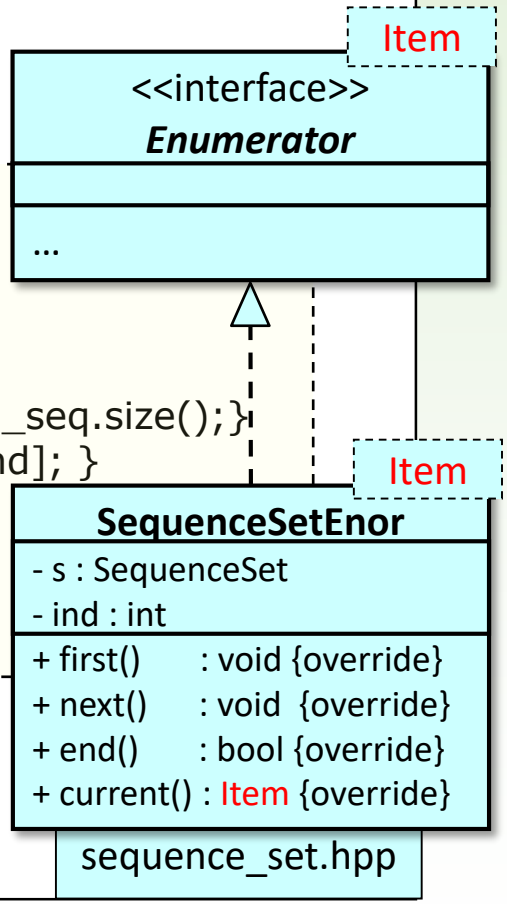

```
template <typename Item>
class Enumerator {
public:
        virtual void first()              = 0;
        virtual void next()               = 0;
        virtual bool end()      const     = 0;
        virtual Item current()  const     = 0;
        virtual ~Enumerator(){};
};
```

enumerator.hpp

SequenceSetEnor osztály-sablonja

a beágyazás miatt ez is egy Item típus-paraméterű sablon, de nem kell kiírni újra, hogy ez egy template

```
template <typename Item>
class SequenceSet : public SetRepr<Item>{
public:
      SequenceSet () : SetRepr<Item> { setEmpty(); }
      ...
      class SequenceSetEnor : public Enumerator<Item>{
      public:
            SequenceSetEnor(SequenceSet<Item> *h): _s(h) {}
            void first()              override { _ind = 0; }
            void next()               override { ++_ind;}
            bool end()      const override { return _ind ==_s->_seq.size();}
            Item current() const override { return _s->_seq[_ind]; }
      private:
            SequenceSet<Item>  *_s;
            unsigned int  _ind;
      };

      Enumerator<Item>* createEnumerator() override{
            return new SequenceSetEnor<Item> (this);
      }
};
```



ArraySet és ArraySetEnor osztály

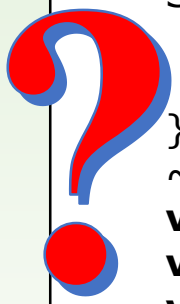
Sem az ArraySet, sem az ArraySetEnor nem lesz sablon, de a SetRepr helyett a SetRepr<int>-ből, illetve az Enumerator helyett az Enumerator<int>ből származnak.

```
class ArraySet : public SetRepr<int>{
public:
 ArraySet (int n): SetRepr<int>(), _vect(n+1), _size(0) { setEmpty(); }
    ...
    class ArraySetEnor : public Enumerator<int>{
          ...
    };

    Enumerator<int>* createEnumerator() override{
          return new ArraySetEnor(this);
    }
};
```

array_set.hpp

Set osztály-sablon

Fordítási hiba: Ha Item nem az int (pl. Set<string> esetén), akkor ez az értékadás hibás (típus konfliktus), mert az ArraySet a SetRepr<int> leszármazottja, a _repr típusa pedig a SetRepr<Item>.



```
template <typename Item>
class Set {
public:
    Set(int n = 0) {
        if (0 == n) _repr = new SequenceSet<Item>;
        else        _repr = new ArraySet(n);
    }
    ~Set() { delete _repr; }
    void setEmpty()      { _repr->setEmpty(); }
    void insert(Item e)  { _repr->insert(e); }
    void remove(Item e) { _repr->remove(e); }
    Item select()  const  { return _repr->select(); }
    bool empty()   const { return _repr->empty(); }
    bool in(Item e)const { return _repr->in(e); }

    Enumerator<Item>* createEnumerator() { _repr->createEnumerator(); }
private:
    SetRepr<Item> *_repr;

    Set(const Set& h) ;
    Set& operator=(const Set& h);
};
```

Item

Set

+ Set(n:int = 0)
+ setEmpty() : void
+ insert(Item) : void
+ remove(Item) : void
+ select() : Item {query}
+ empty() : bool {query}
+ in(Item) : bool {query}

Viszont erre az ágra csak Item=int esetén lenne szükség. Csak ekkor kellene lefordítani.

set.hpp

Sablon specializáció

- Lehetőség van egy osztály-sablon, vagy akár csak egy függvény-sablon különféle definícióira a sablonparaméter értékétől függően.

```
template <typename Item>
class Set {
public:
        Set(int n = 0) { _repr = createSetRepr<Item>(n); }
        …
private:
        SetRepr<Item>* _repr;

        static SetRepr<Item>* createSetRepr(int n) {
                return new SequenceSet<Item>;
        }
};
```

Ez egy olyan gyártófüggvény-sablon, amelynek két definíciója is van: az általános mellett arra a speciális esetre, amikor Item=int.

általános osztályszintű gyártófüggvény-sablon

set.hpp

```
template<>
SetRepr<int>* Set<int>::createSetRep(int n) {
        if (0 == n) return new SequenceSet<int>;
        else        return new ArraySet(n);
}
```

speciális osztályszintű gyártófüggvény-sablon

set.hpp