

Tervminták II. (Híd, Bejáró, Gyártófüggvény)

Halmaz és bejárása

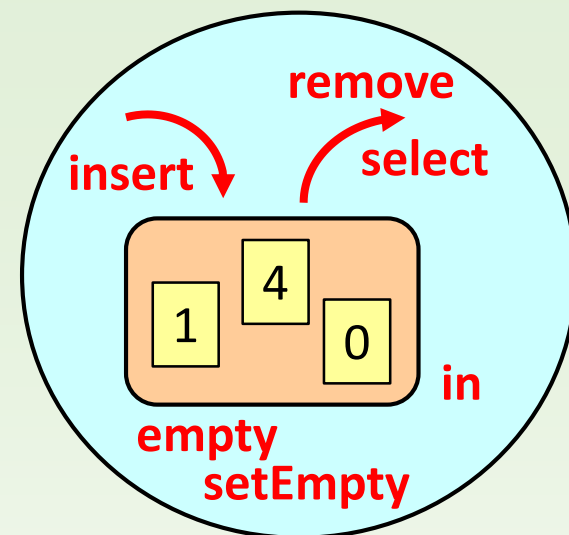
Osztály-sablonok

1.Feladat

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

- Egy halmaz-objektum reprezentációja attól függ, hogy tetszőleges egész számokat akarunk-e abban tárolni, vagy speciálisan olyan természetes számokat, amelyekre adott egy felső korlát (**max**).
 - Általában **egy sorozatban** fogjuk tárolni a halmaz elemeit.
 - Speciálisan **egy logikai értékű tömb** reprezentál majd egy halmazt úgy, hogy a halmaz elemei a tömb azon indexei, ahol a tömb igaz értéket tárol.
- Szeretnénk a reprezentációt elrejtetni 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 nem kell tudnia arról, hogy ehhez milyen reprezentációt használunk.

Kétféle reprezentáció



Tömb

	0	1	...	4	max	
vect	true	true	false	false	true	false
size	3					

Sorozat

	1	...	size
seq	1	4	0

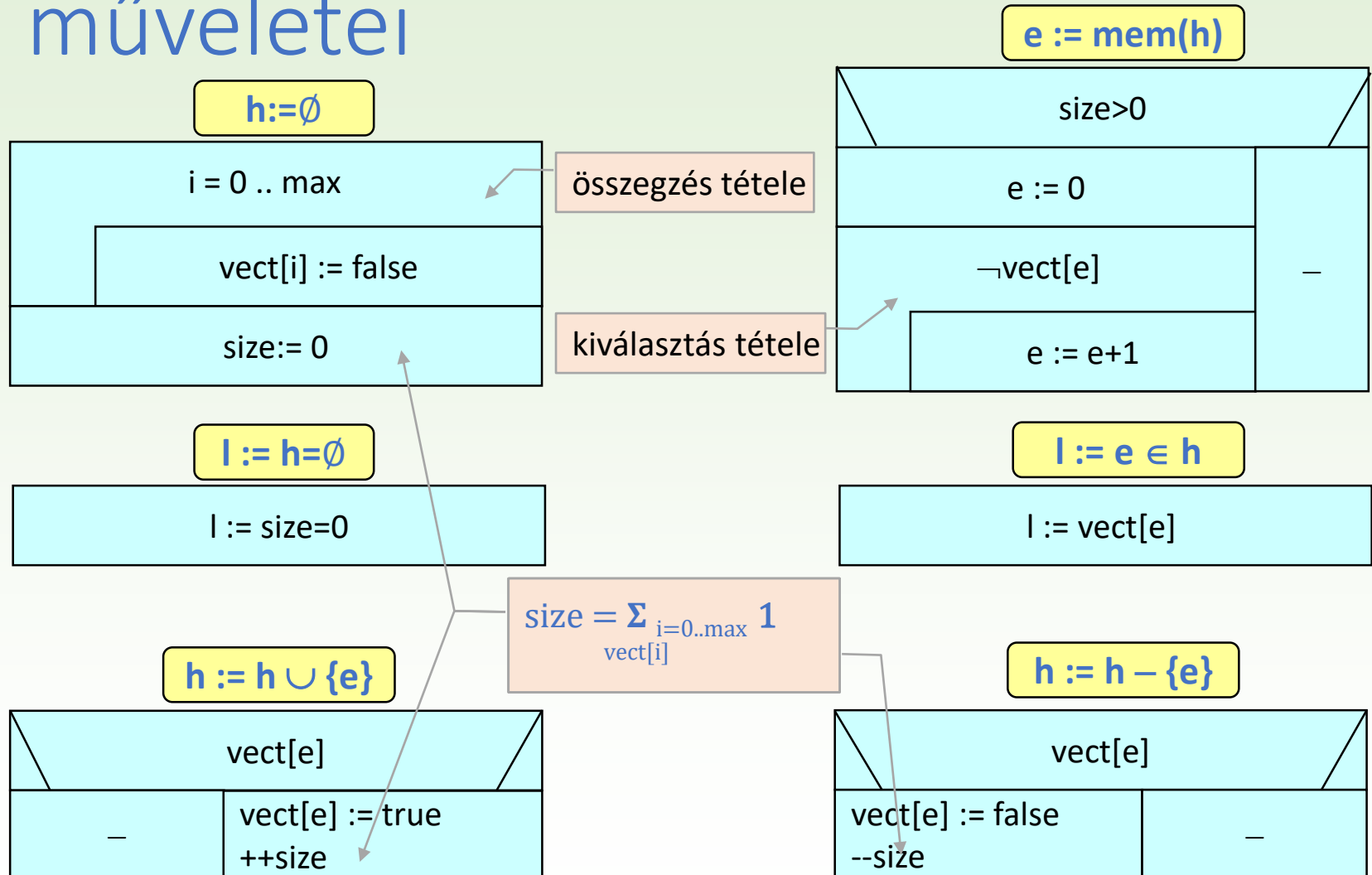
1. Rögzített méretű tömb és külön a halmazbeli elemek száma.
2. Műveletek számítási bonyolultsága többnyire konstans, de a select és a setEmpty lineáris.

1. Dinamikusán változó hosszúságú sorozat.
2. Műveletek számítási bonyolultsága többnyire lineáris, de az empty és a select konstans idejű.

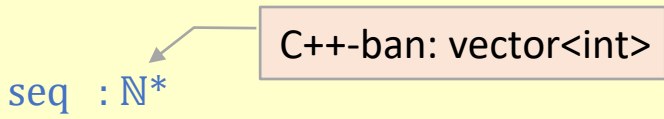
Halmaz típus tömbbel

set([0..max])		Típus-specifikáció	
típusértékek	<p>Olyan halmazok, amelyek elemei 0 és max közé eső természetes számok.</p> <p>(Formálisan: a $2^{\{0..max\}}$ hatványhalmaz.)</p>	<p>üresé teszi a halmazt (setEmpty)</p> <p>$h := \emptyset$ $h := \text{set}([0..max])$</p> <p>betesz egy elemet a halmazba (insert)</p> <p>$h := h \cup \{e\}$ $h := \text{set}([0..max]), e: \mathbb{N}$</p> <p>kivesz egy elemet a halmazból (remove)</p> <p>$h := h - \{e\}$ $h := \text{set}([0..max]), e: \mathbb{N}$</p> <p>kiválasztja a halmaz egy elemét (select)</p> <p>$e := \text{mem}(h)$ $h := \text{set}([0..max]), e: \mathbb{N}$</p> <p>üres-e a halmaz (empty)</p> <p>$l := h = \emptyset$ $h := \text{set}([0..max]), l: \mathbb{L}$</p> <p>benne van-e egy elem a halmazban (in)</p> <p>$l := e \in h$ $h := \text{set}([0..max]), e: \mathbb{N}, l: \mathbb{L}$</p>	műveletek
reprzetentáció	<p>$\text{vect} : \mathbb{L}^{0..max}$</p> <p>$\text{size} : \mathbb{N}$</p> <p>invariáns: $\text{size} = \sum_{i=0..max} 1_{\text{vect}[i]}$</p>	műveletek programjai	implementáció
Típus-megvalósítás			

Tömbbel reprezentált halmaz műveletei



Halmaz típus sorozattal

set(N)		Típus-specifikáció	
típusértékek	<p>Olyan véges elemű halmazok, amelynek elemei természetes számok.</p> <p>(Formálisan: a $\{ h \in 2^{\mathbb{N}} \mid h < \infty \}$ halmaz, azaz a $2^{\mathbb{N}}$ hatványhalmaz véges elemű halmazai.)</p>	<p>üresé teszi a halmazt (setEmpty)</p> <p>$h := \emptyset \quad h: \text{set}(\mathbb{N})$</p> <p>betesz egy elemet a halmazba (insert)</p> <p>$h := h \cup \{e\} \quad h: \text{set}(\mathbb{N}), e: \mathbb{N}$</p> <p>kivesz egy elemet a halmazból (remove)</p> <p>$h := h - \{e\} \quad h: \text{set}(\mathbb{N}), e: \mathbb{N}$</p> <p>kiválasztja a halmaz egy elemét (select)</p> <p>$e := \text{mem}(h) \quad h: \text{set}(\mathbb{N}), e: \mathbb{N}$</p> <p>üres-e a halmaz (empty)</p> <p>$l := h = \emptyset \quad h: \text{set}(\mathbb{N}), l: \mathbb{L}$</p> <p>benne van-e egy elem a halmazban (in)</p> <p>$l := e \in h \quad h: \text{set}(\mathbb{N}), e: \mathbb{N}, l: \mathbb{L}$</p>	műveletek
reprzetentáció	<p>$\text{seq} : \mathbb{N}^*$</p> <p></p>	műveletek programjai	implementáció
		Típus-megvalósítás	

Sorozattal reprezentált halmaz műveletei

$h := \emptyset$

$seq := \langle \rangle$

$l := h = \emptyset$

$l := |seq| = 0$

$e := mem(h)$

$|seq| > 0$

$e := seq[1]$

$l := e \in h$

$l, ind := search_{i=1..|seq|}(seq[i]=e)$

$h := h \cup \{e\}$

lineáris keresés

$l, ind := search_{i=1..|seq|}(seq[i]=e)$

$seq := seq \oplus \langle e \rangle$

$seq.push_back(e)$

$h := h - \{e\}$

$l, ind := search_{i=1..|seq|}(seq[i]=e)$

$seq[ind] := seq[|seq|]$
 $seq := seq[1 .. |seq|-1]$

$seq.pop_back()$

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;

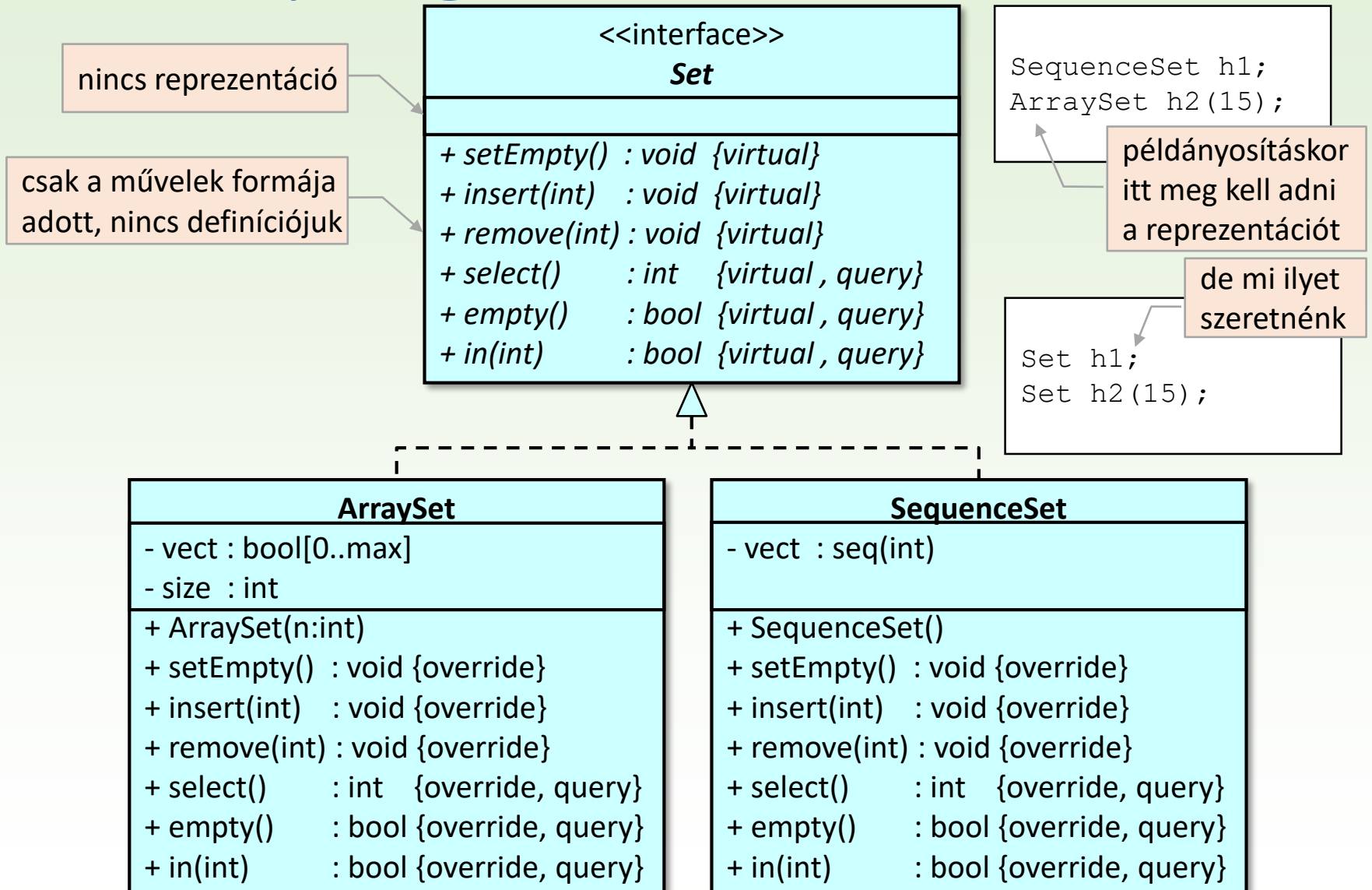
    private:
        ...
};
```

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

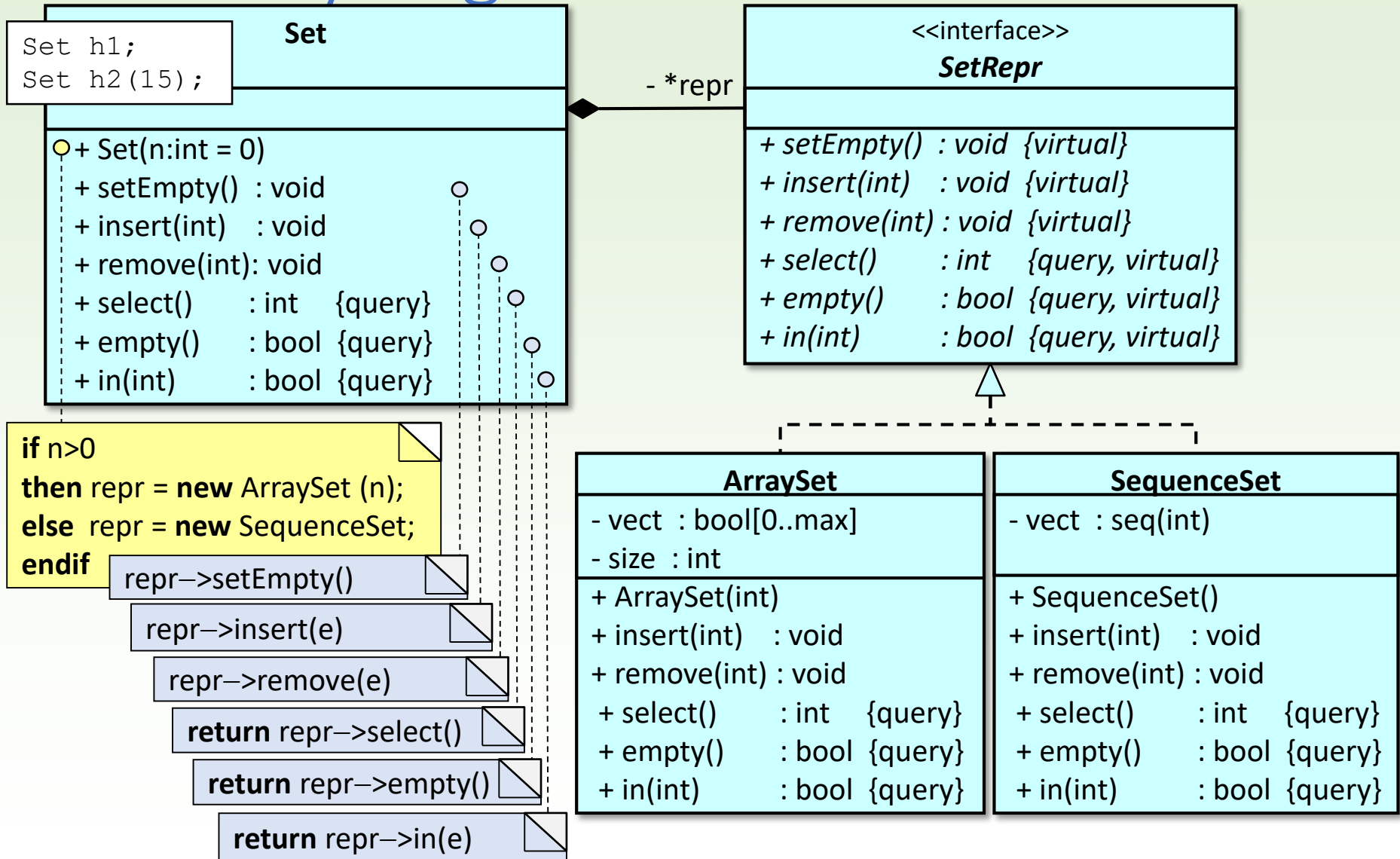
set.h

Hogyan írható le egyszerre
mindkét reprezentáció?

Osztálydiagram 1. változat

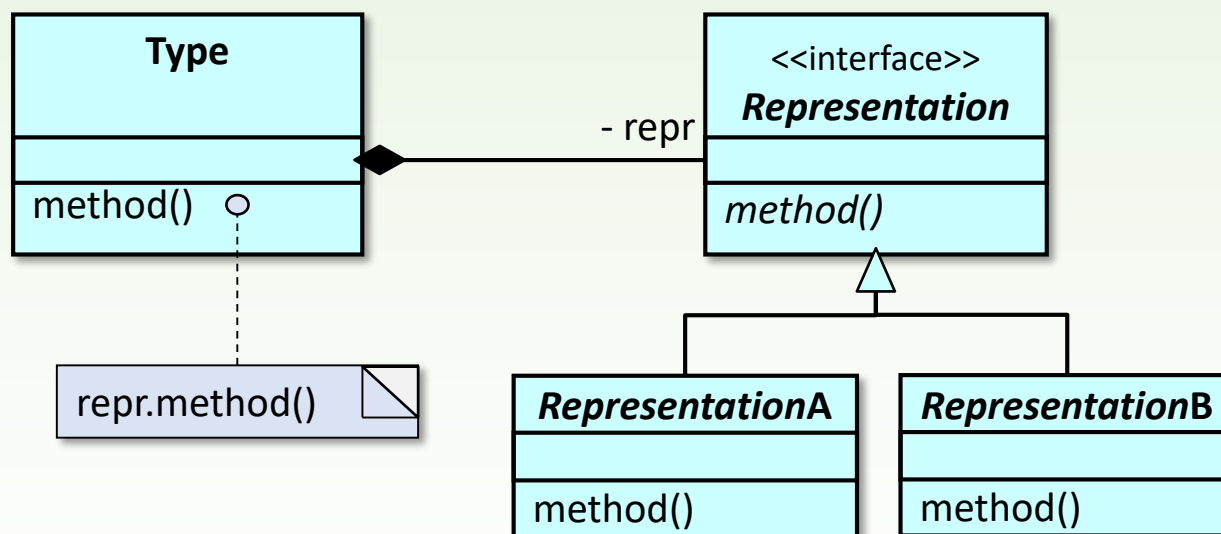


Osztálydiagram 2. változat



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, 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 (n>0) _repr = new ArraySet(n);
        else     _repr = new SequenceSet;
    }
    ~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);
};
```

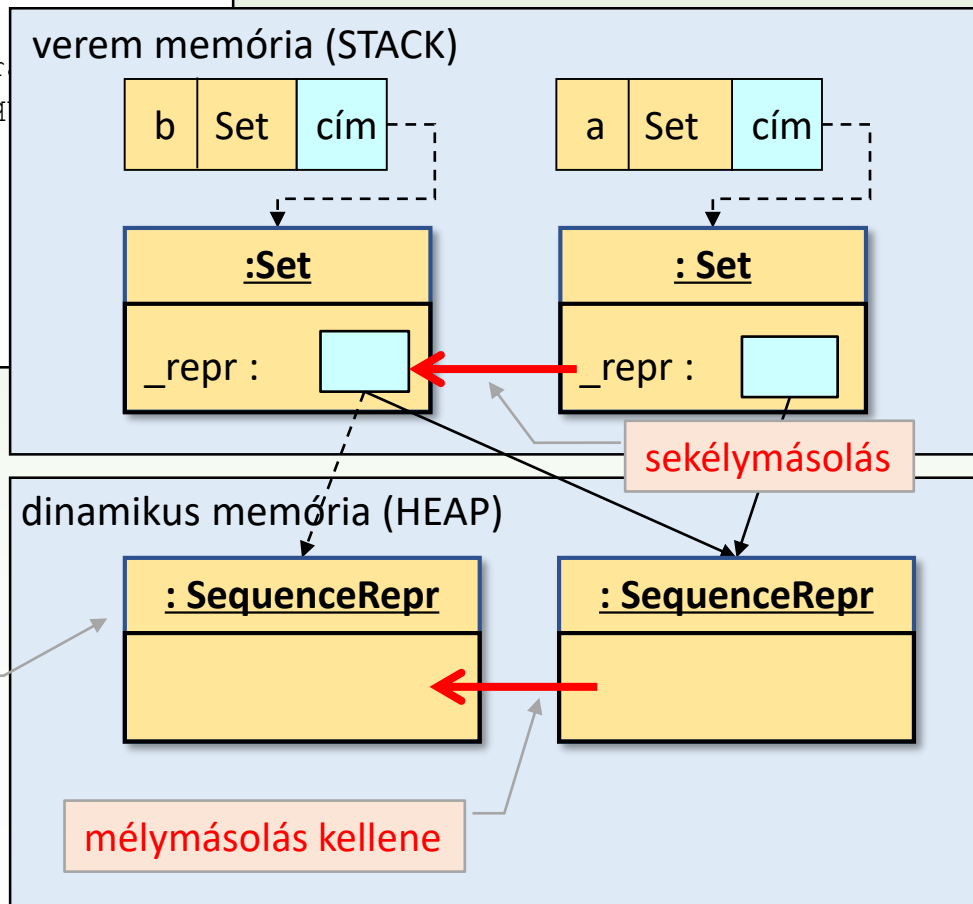
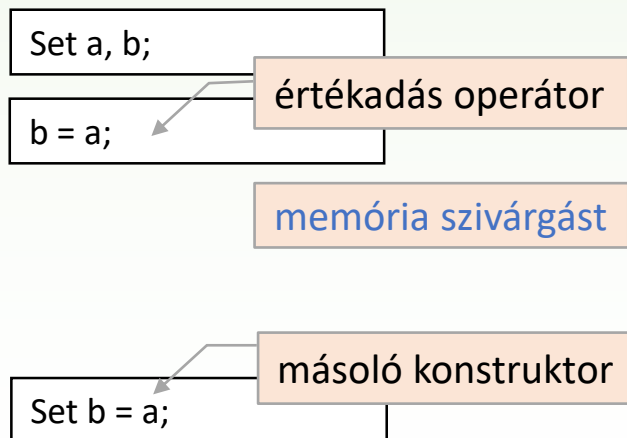
Az alapértelmezett másoló konstruktor és értékadás operátor rosszul működne, de privátként nem használhatók. Később definiálhatjuk és publikussá tehetjük őket.

Set
<ul style="list-style-type: none">+ Set(n:int = 0)+ setEmpty() : void+ insert(int) : void+ remove(int) : void+ select() : int {query}+ empty() : bool {query}+ in(int) : bool {query}

set.h

Kitérő: amikor az alapértelmezett másolás és értékadás rossz

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



Reprezentáció interfésze

<<interface>> SetRepr	
+ setEmpty()	: void {virtual}
+ insert(int)	: void {virtual}
+ remove(int)	: void {virtual}
+ select()	: int {virtual, query}
+ empty()	: bool {virtual, query}
+ in(int)	: bool {virtual, query}

```
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() {}
};
```

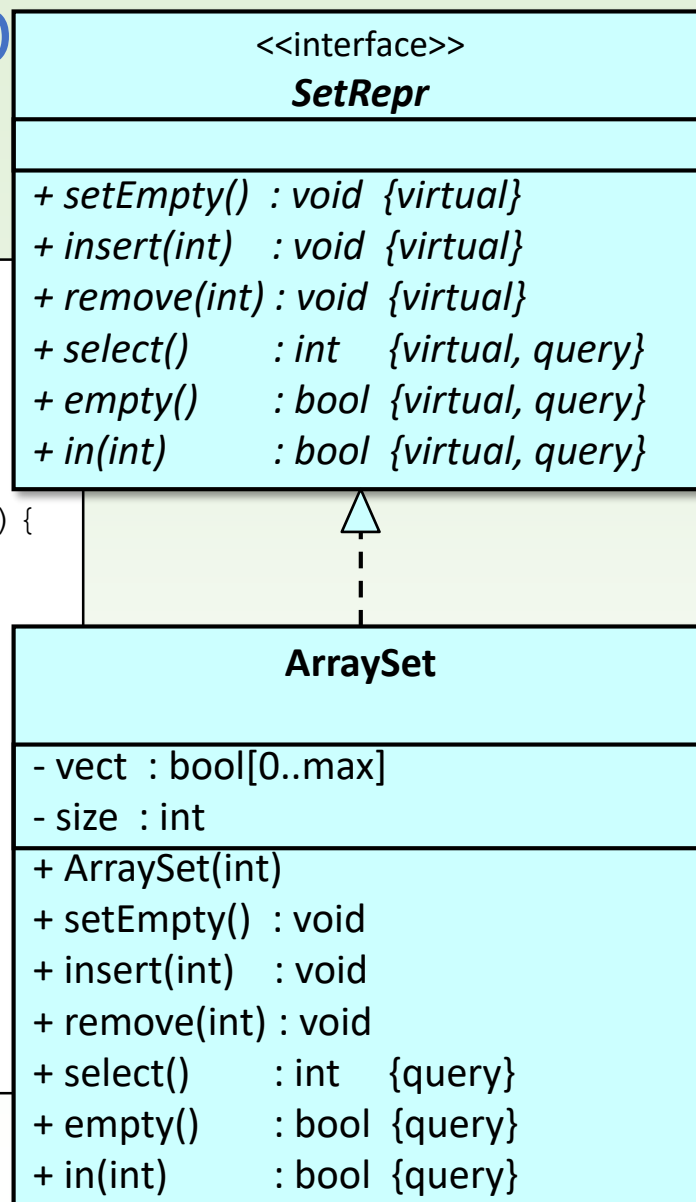
setrepr.h

Tömb-reprezentáció

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

class ArraySet : public SetRepr{
public:
    ArraySet (int n): _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>
```

szabványos kivétel osztályok

```
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;
    }
};
```

setrepr.h

Kivételek dobása

```
Set h(100);  
try {  
    h.insert(101);  
} catch(exception &ex){  
    cout << ex.what() << endl;  
}
```

```
int ArraySet::select() const  
{  
    if(_size==0) throw SetEmptyException();  
    int e;  
    for(e=0; !_vect[e]; ++e);  
    return e;  
}
```

kivétel példányosítás
és dobás

```
bool ArraySet::empty() const  
{  
    return _size==0;  
}
```

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

kivétel példányosítás
és dobás

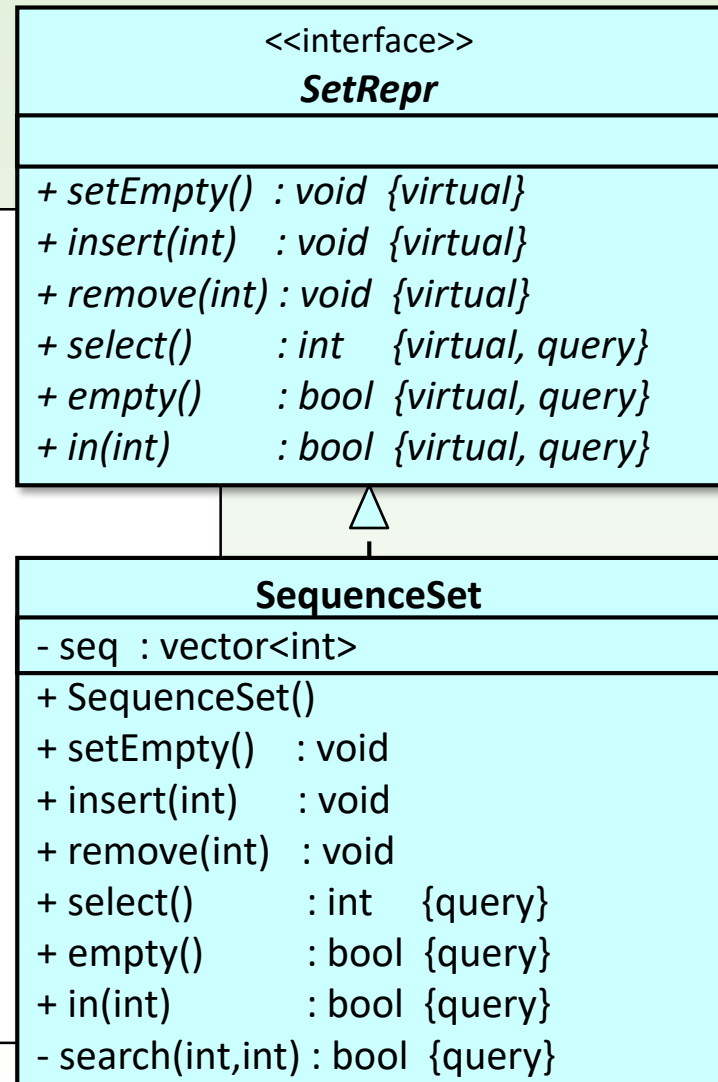
array_set.cpp

Sorozat-reprezentáció

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

class SequenceSet : public SetRepr{
public:
    SequenceSet () { _seq.clear(); }
    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



2.Feladat

Keressünk egy természetes számokat tartalmazó halmazban olyan számot, amely nagyobb a halmaz legalább három másik eleménél! (Ez a keresés eleve sikertelen, ha nincs a halmazban legalább négy szám.)

- 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 : $h:\text{set}(\mathbb{N}), l:\mathbb{L}, n:\mathbb{N}$

Ef: $h = h_0$

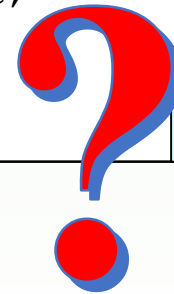
Uf: $l, n = \text{search}_{e \in h_0} (\text{kisebbdarab}(h_0, e) \geq 3)$

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

a halmaz standard felsorolása:

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

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



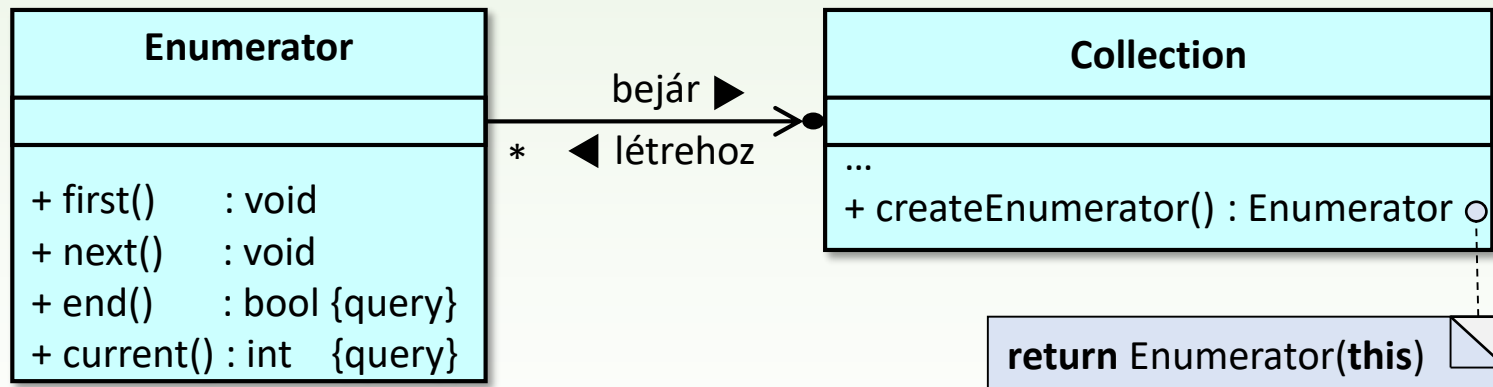
main.cpp

Ez a megoldás rossz:

- a standard felsorolás módosítja a halmazt, így a belső ciklus első alkalommal kitörli a halmaz összes elemét
- a két felsorolás nem független, a külső és belső felsorolás összefonódik

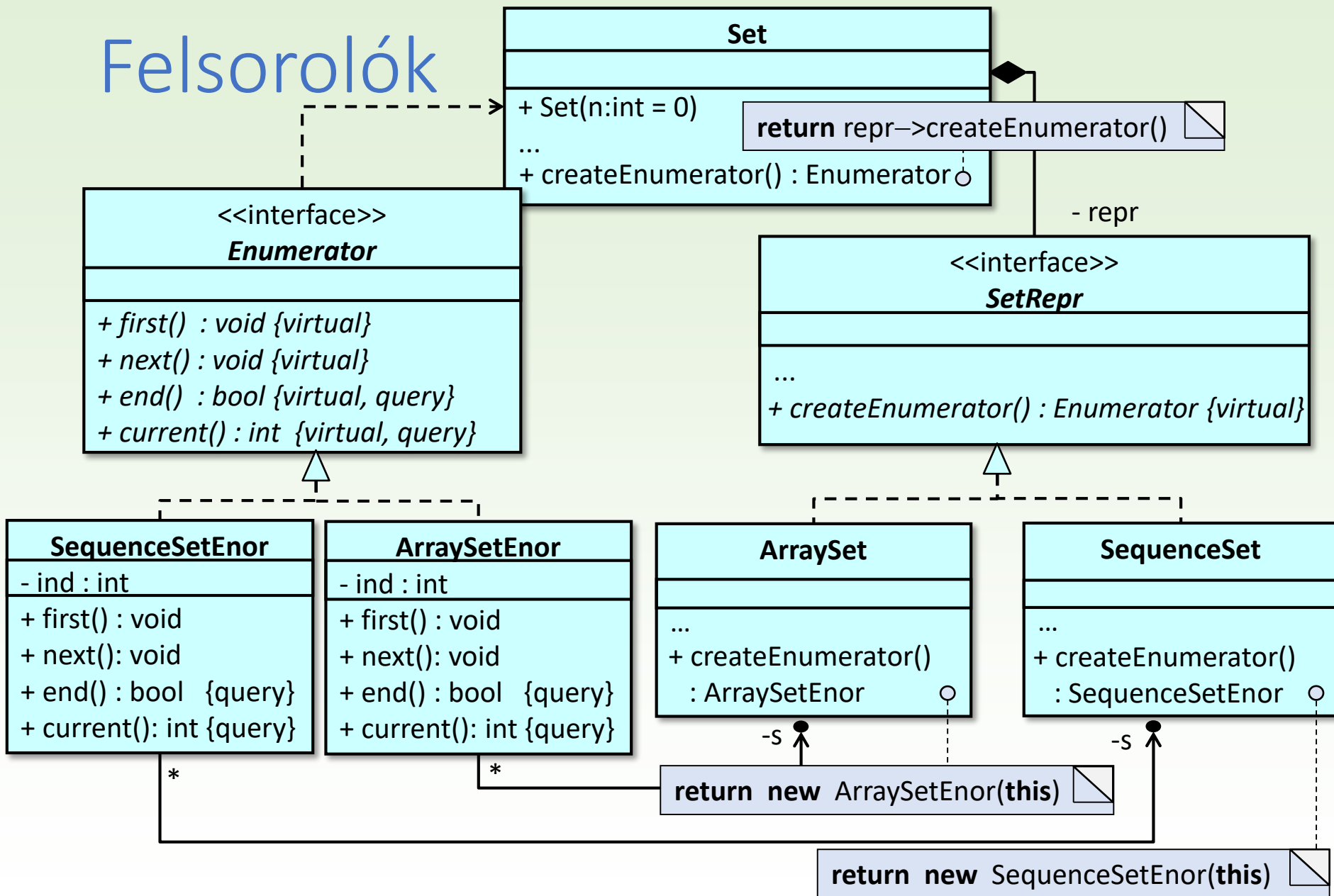
Bejáró (iterátor) tervezési minta

- 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.



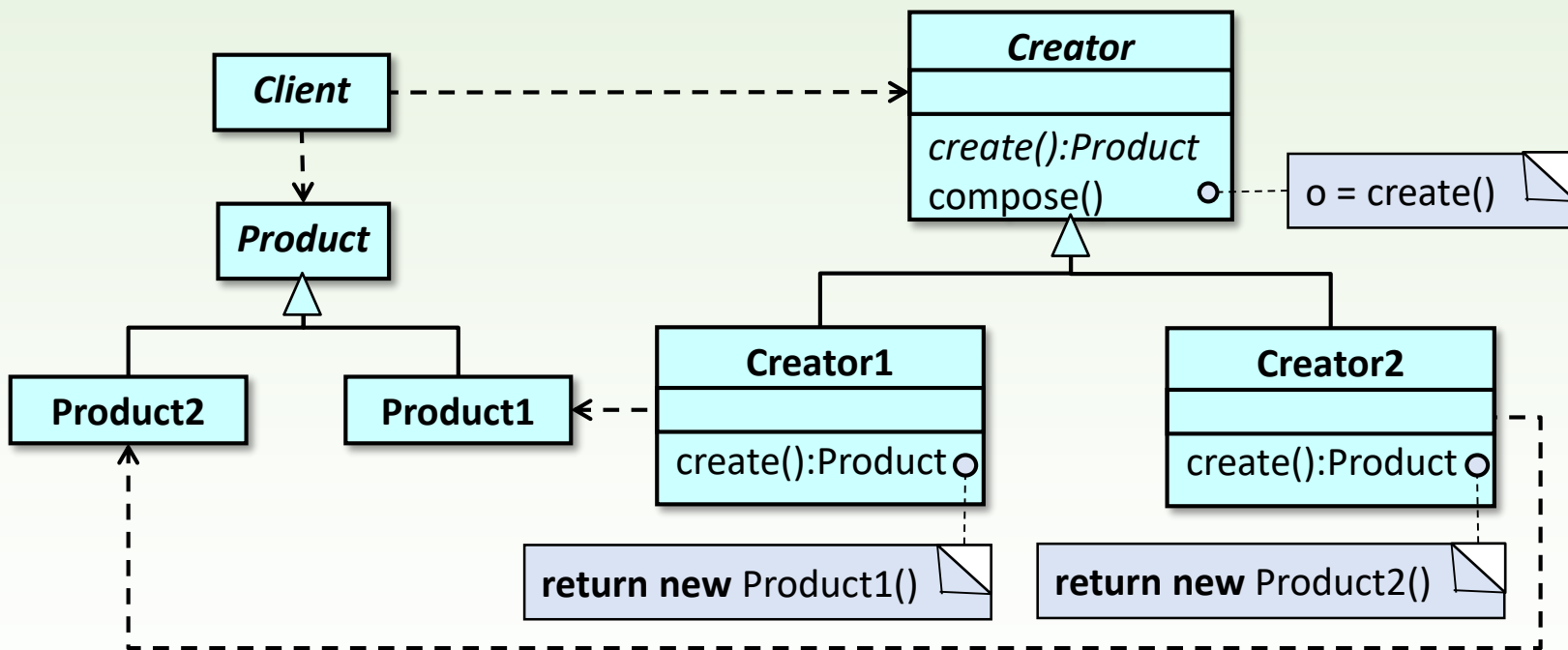
A **tervezési minták** az objektum-alapú modellezést támogató osztálydiagram minták, amelyek az újrafelhasználhatóság, módosíthatóság, hatékonyság biztosításában játszanak szerepet.

Felsorolók



Gyártófüggvény (factory method) tervezési minta

- 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, módosíthatóság, hatékonyság biztosításában játszanak szerepet.

Főprogram

```
Set h;
// Beolvasás
...

bool l = false;
int n;
Enumerator* enor1 = h.createEnumerator();
for(enor1->first(); !l && !enor1->end(); enor1->next()){
    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:
repr->createEnumerator()
return new SequenceSetEnor(this)

main.cpp

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);  
}  
};
```

A halmaz elemeinek felsorolását
a halmazt reprezentáló sorozat
elemeinek felsorolása valósítja meg.

<<interface>>

Enumerator

+ first() : void {virtual}
+ next() : void {virtual}
+ end() : bool {virtual,query}
+ current() : int {virtual,query}



SequenceSetEnor

- s : SequenceSet

- ind : int

+ first() : void
+ next(): void
+ end() : bool {query}
+ current(): int {query}

sequence_set.h

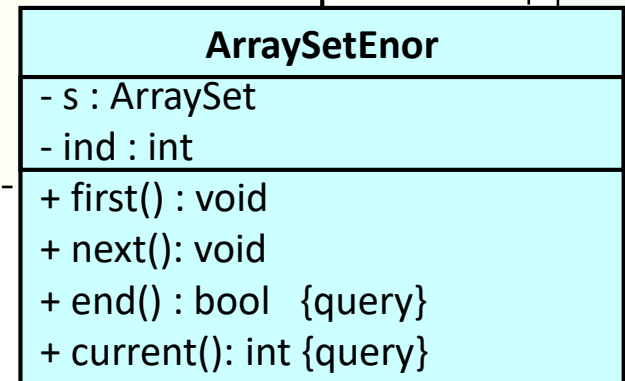
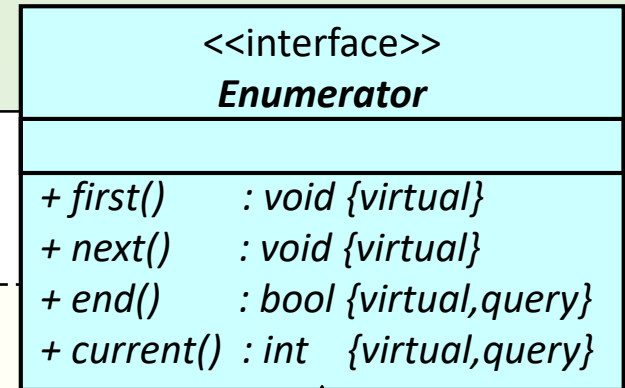
ArraySet felsorolója

A halmaz elemeinek felsorolását a halmazt reprezentáló tömb true értékeihez tartozó indexek felsorolása valósítja meg.

```
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);
    }
};
```



array_set.h

3.Feladat

Tegyük biztonságossá a felsorolást!

- Probléma: egy halmaz elemeinek felsorolása hibás lehet, ha a felsorolás közben olyan műveletet hajtunk végre, amelyik megváltoztatja a halmazt.
- Kritikus műveletek: `setEmpty()`, `insert()`, `remove()`, értékadás operátor, destruktork.
- Megoldás: Ne engedjünk kritikus műveletet végrehajtani felsorolás közben.

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

Hibát okoz a felsorolás által érintett aktuális elem törlése, ha a sorozat-reprezentációt használjuk.



main.cpp

Kizárás megvalósítása

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

setrepr.h

```
class Set {  
public:
```

```
...  
void Set::remove(int e)  
{
```

```
    if(_repr->getEnumCount() != 0) throw UnderTraversalException();  
    _ref->remove(e);  
}
```

```
~Set() {
```

```
    if(_repr->getEnumCount() != 0) throw UnderTraversalException();  
    delete repr;  
}
```

```
}
```

```
...
```

```
};
```

A kritikus művelet kivételt dob,
ha a halmazon felsoroló dolgozik.

set.h

```
class SetRepr {  
public:
```

```
    SetRepr(): _enumeratorCount(0) {}  
...
```

```
    int getEnumCount() const { return _enumeratorCount; }
```

```
protected:
```

```
    int _enumeratorCount;
```

```
};
```

ez már nem interfész, de még absztrakt osztály

aktív felsorolók száma

setrepr.h

ArraySet

```
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 növeljük.

a felsoroló megszűnésekor a felsoroló-számlálót csökkentjük

array_set.h

SequenceSet

```
class SequenceSet : public SetRepr{  
public:
```

```
    SequenceSet(): SetRepr() {}  
    ...
```

```
    class SequenceSetEnor : public Enumerator{  
    public:
```

```
        SequenceSetEnor(SequenceSet *h): _s(h)
```

```
        { ++(_s->_enumeratorCount); }
```

```
        ~ SequenceSetEnor() { --(_s->_enumeratorCount); }
```

```
        ...
```

```
    };
```

```
    Enumerator* createEnumerator() override {
```

```
        return new SequenceSetEnor(this);
```

```
    }
```

```
};
```

lenullázza a felsoroló számlálót

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

A felsoroló megszűnésekor a felsoroló-számlálót csökkentjük.

sequence_set.h

4.Feladat

- ❑ Az eddigi osztályok helyett használjunk **osztálysablonokat** azért, hogy partaméterként adhassuk meg a halmazban tárolt elemek típusát.
- ❑ Vegyük azonban figyelembe, hogy a tömbös reprezentációval kizárólag a felső korláttal rendelkező természetes számokat tároló halmazt 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;
```

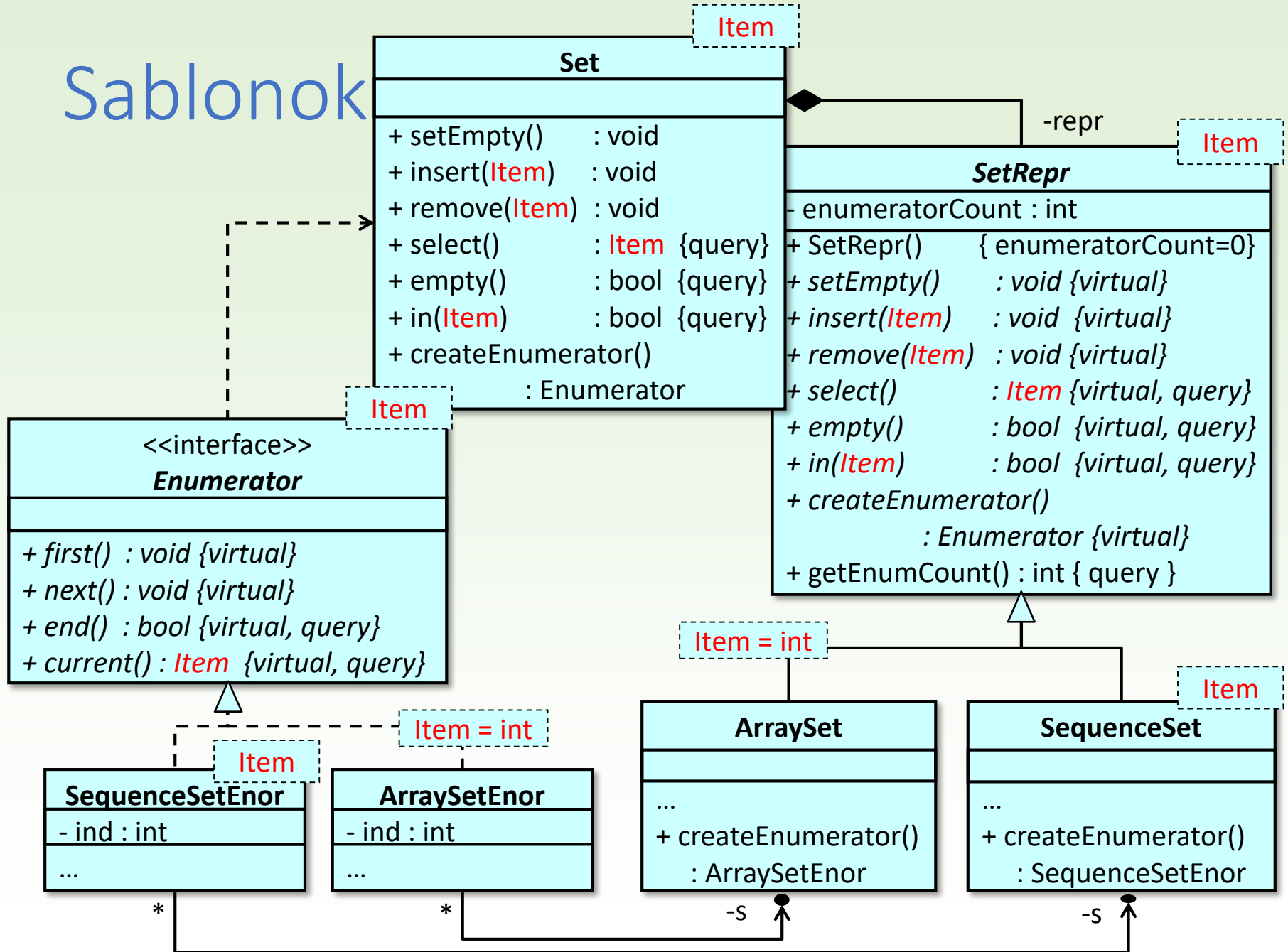
```
h1.insert(12);  
h2.insert(666);  
h3.insert("alma");
```

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

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

main.cpp

Sablonok



Reprezentáció interfész-sablónja

jelöli a sablon, megadja
a sablon paramétereit

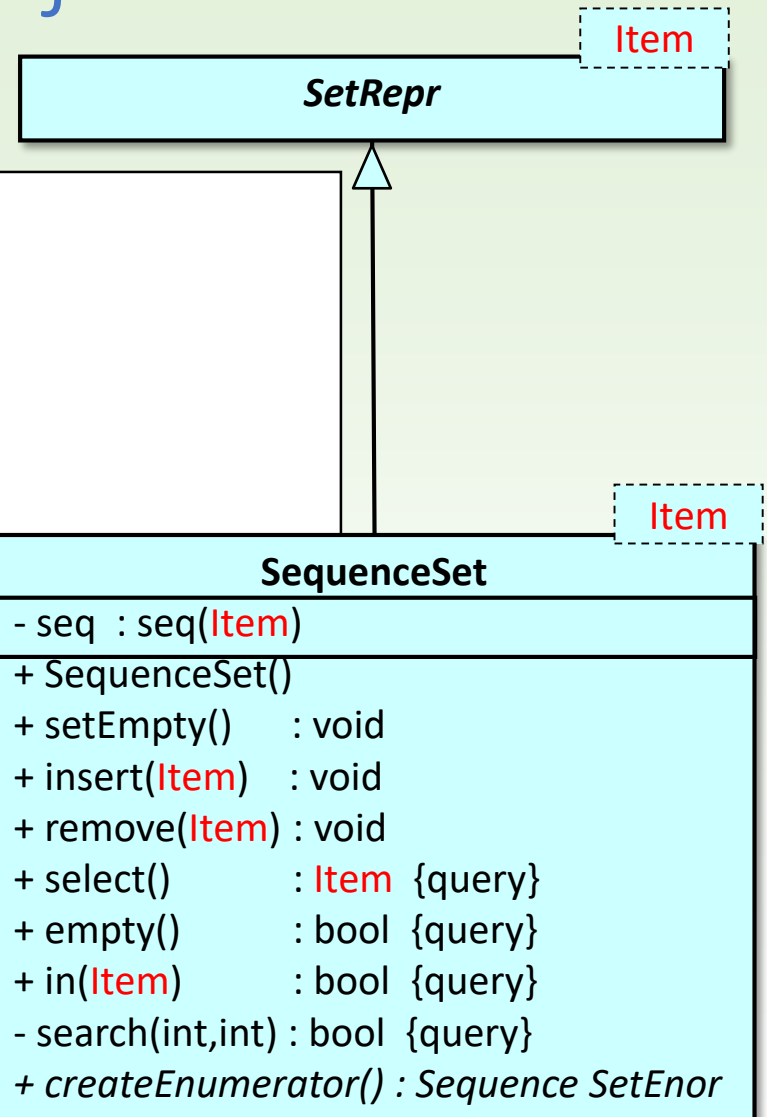
```
template <typename Item>
class SetRepr {
public:
    SetRepr(): _enumeratorCount(0) {}
    virtual ~SetRepr() {};
    virtual void setEmpty() = 0;
    virtual void insert(Item e) = 0;
    virtual void remove(Item e) = 0;
    virtual Item select() const = 0;
    virtual bool empty() const = 0;
  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

SetRepr	
# enumeratorCount : int	
+ SetRepr()	{ enumeratorCount=0}
+ setEmpty()	: void {virtual}
+ insert(<i>Item</i>)	: void {virtual}
+ remove(<i>Item</i>)	: void {virtual}
+ select()	: <i>Item</i> {virtual, query}
+ empty()	: bool {virtual, query}
+ in(<i>Item</i>)	: bool {virtual, query}
+ createEnumerator()	: Enumerator {virtual}
+ getEnumCount()	: int { query }

Item

setrepr.h

SequenceSet sablonja



```
template <typename Item>
class SequenceSet : public SetRepr<Item>{
public:
    SequenceSet () { _seq.clear(); }
    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
};
...
sequence_set.hpp
```

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

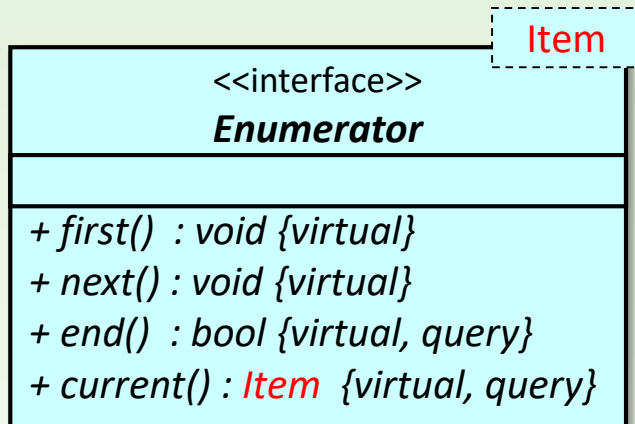
SequenceSet sablon metódusai

```
...  
template <typename Item>  
void SequenceSet<Item>::setEmpty() { _seq.clear(); }  
  
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  
{  
    if( _seq.size()==0) throw EmptySetException();  
    return _seq[0];  
}
```

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

sequence_set.hpp

Felsoroló interfész-sablona



```
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.h

SequenceSetEnor sablonja

```
template <typename Item>
class SequenceSet : public SetRepr<Item>{
public:
```

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

```
...
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);
}
};
```

<<interface>>
Enumerator

...



Item

Item

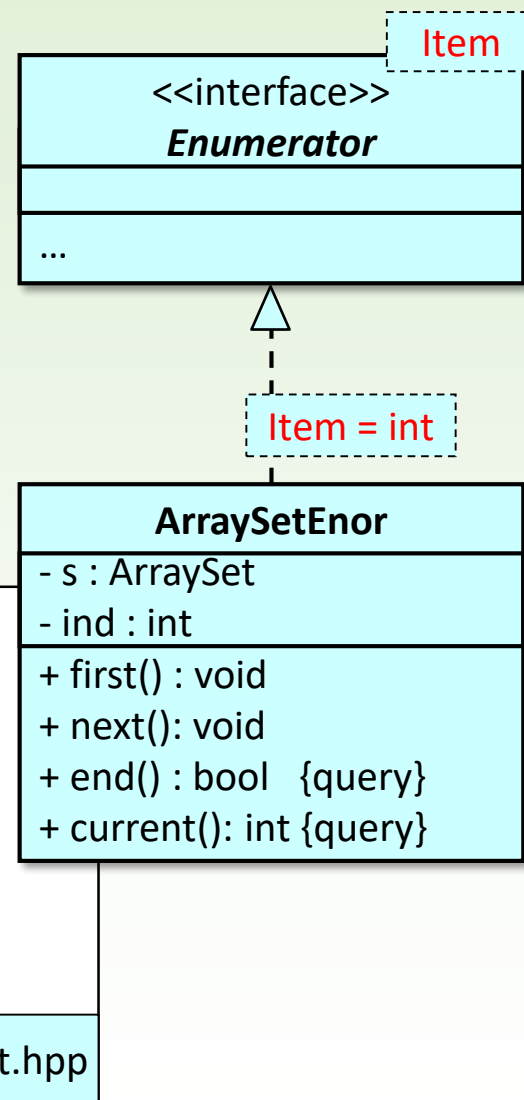
SequenceSetEnor

- ind : int
- s : SequenceSet

+ first() : void {virtual}
+ next() : void {virtual}
+ end() : bool {virtual, query}
+ current() : Item {virtual, query}

sequence_set.hpp

ArraySet felsorolója



az ArraySet nem változik, kivéve hogy a származtatásnál az őosztály int-re vett változatára kell hivatkozni

```
class ArraySet : public SetRepr<int>{
public:
    class ArraySetEnor : public Enumerator<int>{
        ...
    };

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

array_set.hpp

Halmaz osztálysablon

```
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);
```

```
};
```

Fordítási hiba:

Ez az értékadás hibás lesz, ha Item nem az int típus, mert az ArraySet nem leszármazottja ilyenkor a SetRepr<Item>-nek.

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

Erre az ágra viszont csak Item=int esetén van szükség.

set.h

Elágazás szerinti példányosítás helyett paraméter függő példányosítás

```
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>;
    }
};
```

egy gyártófüggvény-sablon
példányosítja a reprezentációt

Set<Item> osztály általános
gyártófüggvény-sablonja

set.h

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

speciálisan a Set<int> osztály
gyártófüggvény-sablonja

set.h