Halmaz típus
1.rész
Halmaz reprezentálása

Gregorics Tibor

gt@inf.elte.hu

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

Cél

- Természetes számok halmazát eltároló objektumokra van szükség.
- Az ezek reprezentációját attól tegyük függővé, hogy ismerjük-e előre a felső korlátját az eltárolandó természetes számoknak.
  - Ha nem ismert a felső korlát, akkor a halmazt az a sorozat reprezentálja, amelyben a halmazbeli elemeket elhelyezhetjük.
  - Ha ismert a felső korlát (max), akkor a halmazt egy olyan 0-tól indexelt max+1 hosszú logikai értékeket tartalmazó tömb reprezentálja, amely csak azon indexeknél tárol igaz értéket, amely index a halmaz eleme.
- Szeretnénk azonban a reprezentációt elrejteni a halmaz típust használók elől: a halmaz létrehozásakor csak azt kérdezzük meg, hogy van-e felső korlátja a halmazba kerülő természetes számoknak.

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)
$e:\in h$ $h:set(\mathbb{N}), e:\mathbb{N}$

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

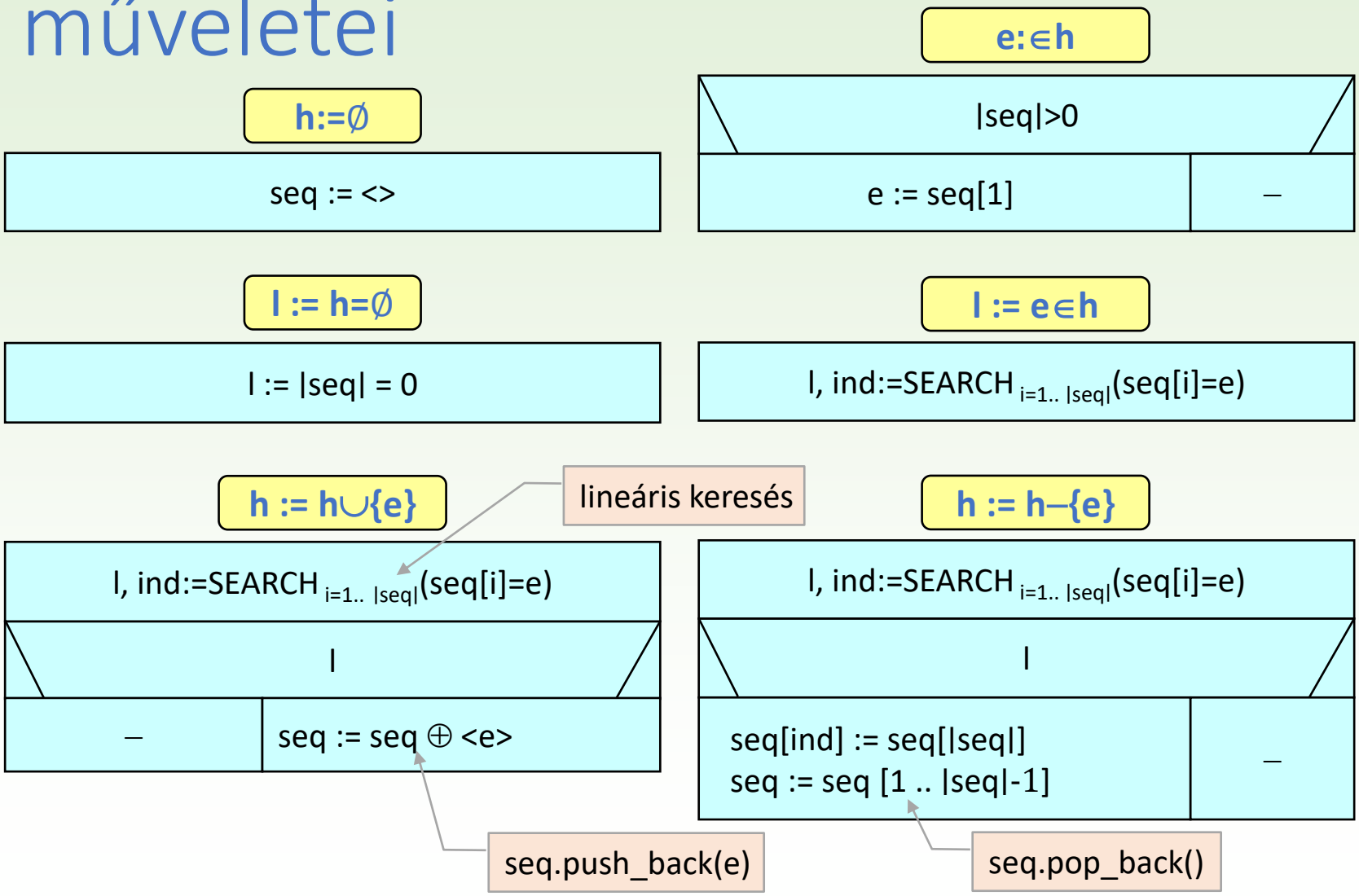
$seq : \mathbb{N}^*$

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)
$h:=\emptyset$ $h:set(\mathbb{N})$
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)
$e:\in h$ $h:set(\mathbb{N}), e:\mathbb{N}$
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

$vect : \mathbb{L}^{0..max}$
$size : \mathbb{N}$

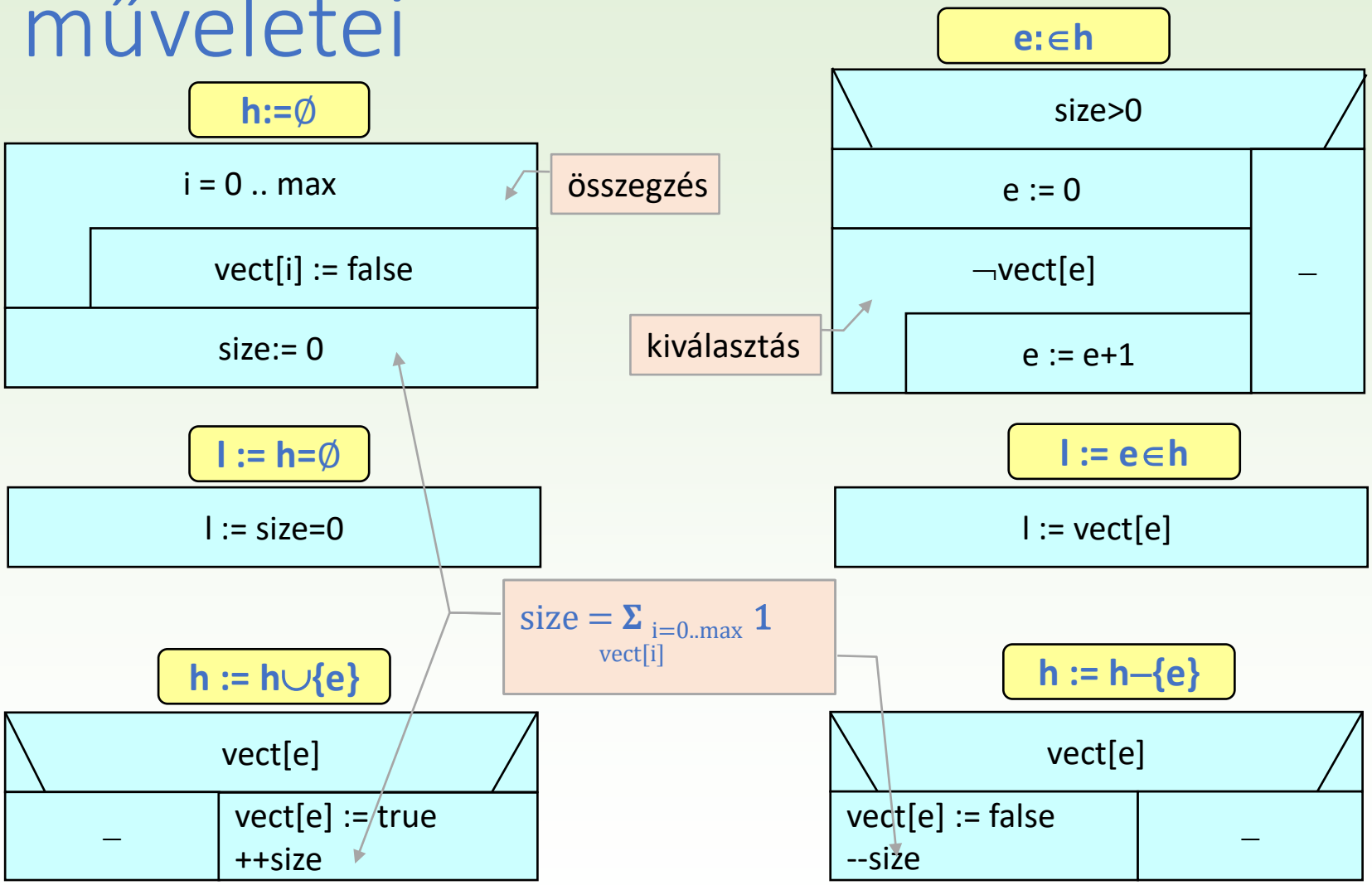
invariáns: $size = \sum_{i=0..max,\ vect[i]} 1$

implementáció

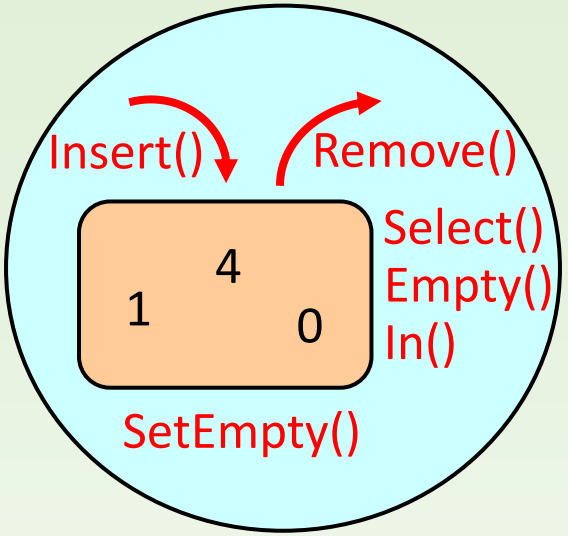
műveletek programjai

Típus-megvalósítás

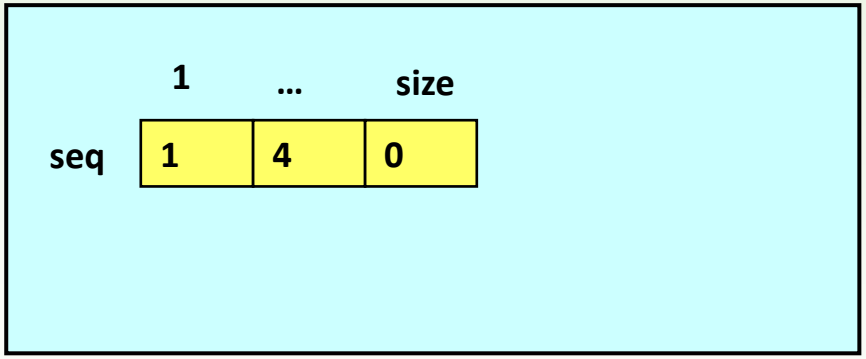
Tömbbel reprezentált halmaz műveletei



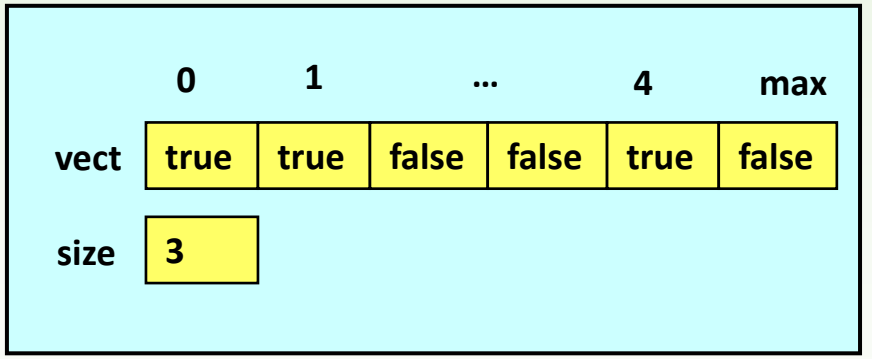
Reprezentációk összehasonlítása



Sorozat



Tömb



Dinamikusan változó hosszúságú sorozat.

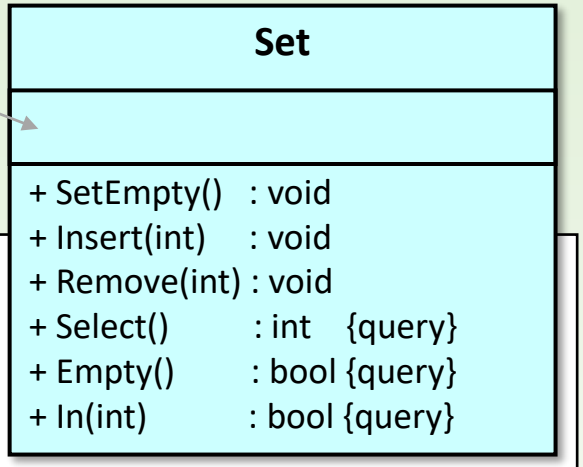
Rögzített méretű tömb és külön a halmazbeli elemek száma.

Az Insert(), Remove(), In() számítási bonyolultsága lineáris, az Empty(), Select(), SetEmpty() konstans idejű.

Az Insert(), Remove(), In(), Empty() számítási bonyolultsága konstans, a Select(), SetEmpty() lineáris.

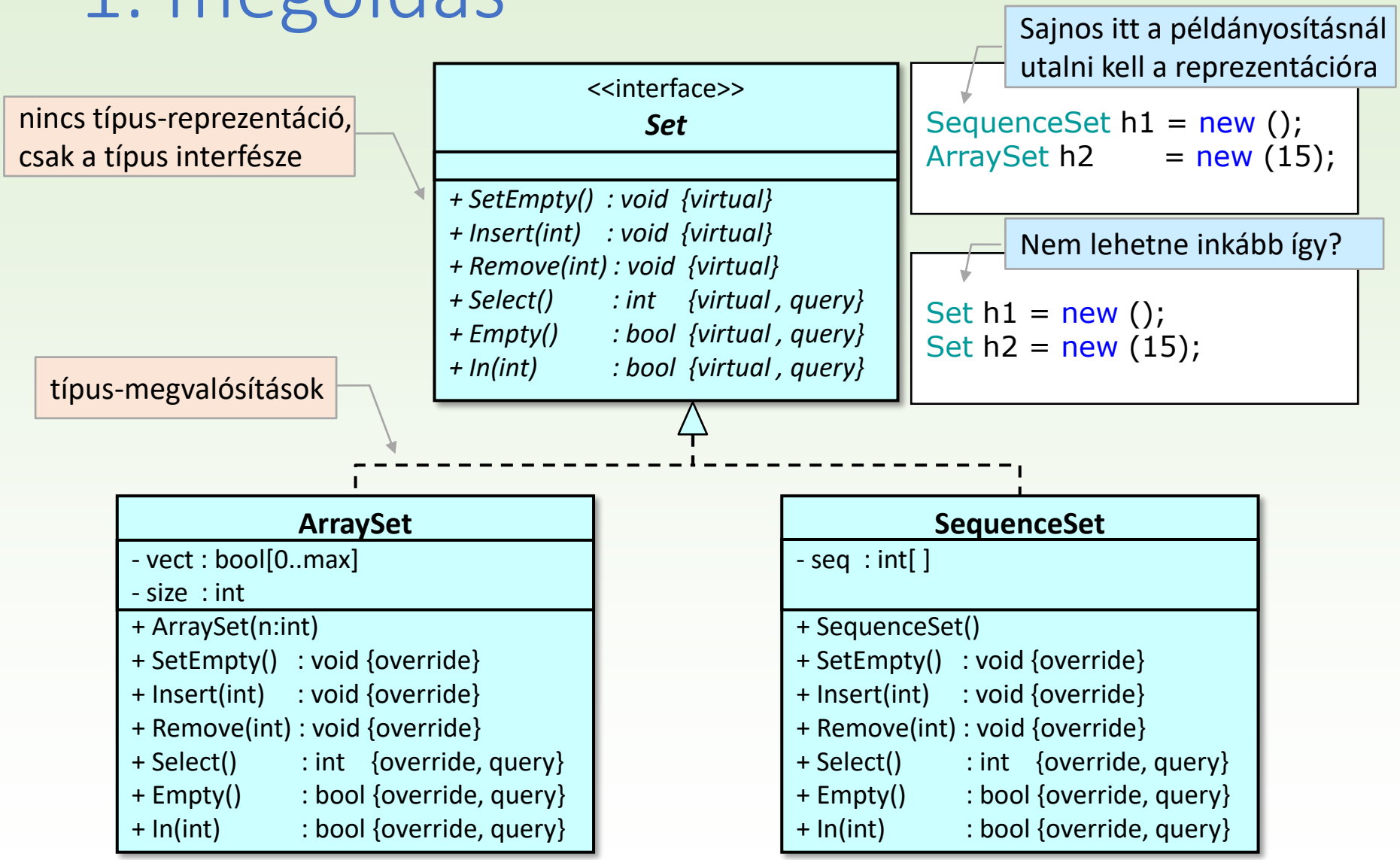
Halmaz osztály szolgáltatásai

Hogyan írható le egyszerre mindkét reprezentáció?



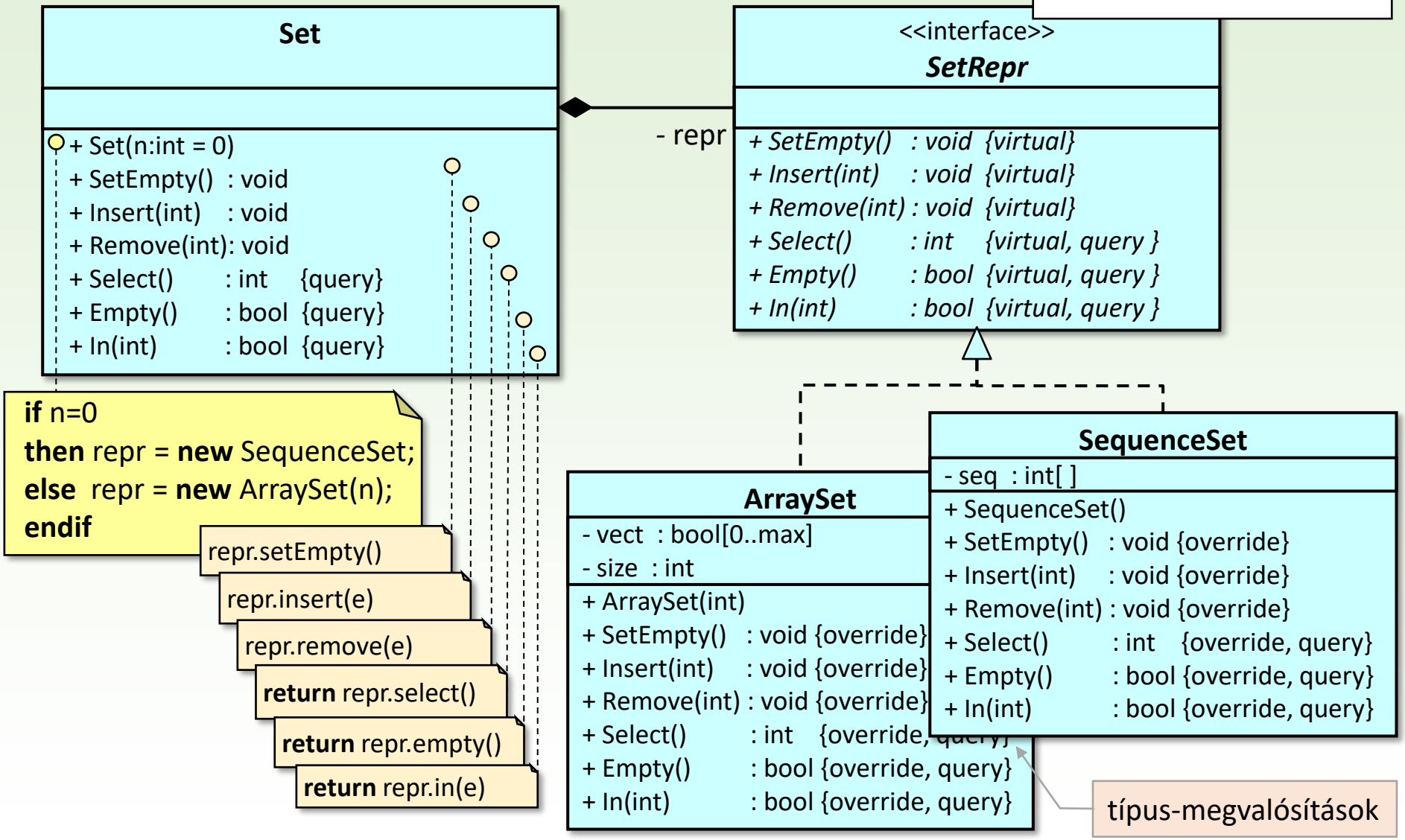

```
class Set
{
  public class EmptySetException : Exception { }
  public class IllegalElementException : Exception
  {
    public int e;
    public IllegalElementException(int n) { e = n; }
  }


  public void SetEmpty()    { ... }
  public void Insert(int e) { ... }
  public void Remove(int e) { ... }
  public int  Select()      { ... }
  public bool Empty()       { ... }
  public bool In(int e)     { ... }
}
```

1. megoldás


Sajnos itt a példányosításnál utalni kell a reprezentációra
nincs típus-reprezentáció, csak a típus interfésze
<<interface>>
Set
+ SetEmpty() : void {virtual}
+ Insert(int) : void {virtual}
+ Remove(int) : void {virtual}
+ Select() : int {virtual , query}
+ Empty() : bool {virtual , query}
+ In(int) : bool {virtual , query}
SequenceSet h1 = new ();
ArraySet h2 = new (15);
Nem lehetne inkább így?
Set h1 = new ();
Set h2 = new (15);
típus-megvalósítások
ArraySet
- vect : bool[0..max]
- size : int
+ ArraySet(n:int)
+ SetEmpty() : void {override}
+ Insert(int) : void {override}
+ Remove(int) : void {override}
+ Select() : int {override, query}
+ Empty() : bool {override, query}
+ In(int) : bool {override, query}
SequenceSet
- seq : int[ ]
+ SequenceSet()
+ SetEmpty() : void {override}
+ Insert(int) : void {override}
+ Remove(int) : void {override}
+ Select() : int {override, query}
+ Empty() : bool {override, query}
+ In(int) : bool {override, query}

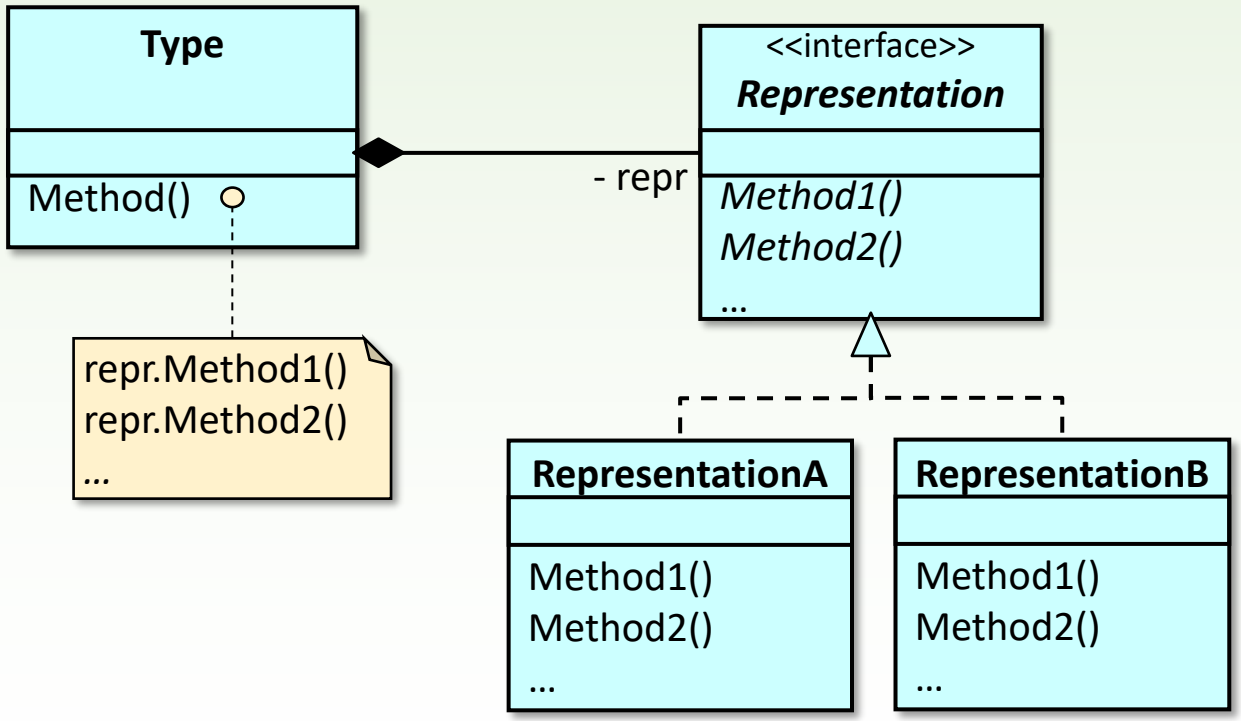
2. megoldás

```
Set h1 = new ();
Set h2 = new (15);
```



Híd (bridge) tervezési minta

- Egy típus reprezentációját leválasztjuk a típust biztosító osztályról azért, hogy azt rugalmasan, futási időben választhassuk ki.



Halmaz osztály (Set.cs)

Set
- repr : SetRepr
+ Set(n:int = 0) + SetEmpty() : void + Insert(int) : void + Remove(int): void + Select() : int {query} + Empty() : bool {query} + In(int) : bool {query}

```
class Set
{
   private ISetRepr repr;

   public Set(int n = 0)
   {
      if (0 == n) repr = new SequenceSet();
      else repr = new ArraySet(n);
   }

   public void SetEmpty()    { repr.SetEmpty(); }
   public void Insert(int e) { repr.Insert(e); }
   public void Remove(int e) { repr.Remove(e); }
   public int Select()
   {
      if (Empty()) throw new EmptySetException();
      return repr.Select();
   }
   public bool Empty() { return repr.Empty(); }

   public bool In(int e) { return repr.In(e); }
}
```

```
interface ISetRepr
{
   void SetEmpty();
   void Insert(int e);
   void Remove(int e);
   int  Select();
   bool Empty();
   bool In(int e);
}
```

Sorozat-reprezentáció

SequenceSet : SetRepr
- seq : int[]
+ SetEmpty() : void {override} + Insert(int) : void {override} + Remove(int) : void {override} + Select() : int {override, query} + Empty() : bool {override, query} + In(int) : bool {override, query}

```
class SequenceSet : ISetRepr
{
   private List<int> seq = new ();

   public SequenceSet()      { seq.Clear(); }
   public void SetEmpty()    { seq.Clear(); }
   public void Insert(int e) { if (!seq.Contains(e)) seq.Add(e); }
   public void Remove(int e) { seq.Remove(e); }
   public bool Empty()       { return seq.Count==0;}
   public int  Select()      { return seq[0]; }
   public bool In(int e)     { return seq.Contains(e); }
}
```

List műveletei

Tömb-reprezentáció

ArraySet : SetRepr
- vect : bool[0..max] - size : int
+ ArraySet(int) + SetEmpty() : void {override} + Insert(int) : void {override} + Remove(int) : void {override} + Select() : int {override, query} + Empty() : bool {override, query} + In(int) : bool {override, query}

```
class ArraySet : ISetRepr
{
   private bool[] vect;
   private int size;

   public ArraySet(int n)
   {
       vect = new bool[n+1];
       for (int i = 0; i < n; ++i) vect[i] = false;
       size = 0;
   }
   public void SetEmpty()
   {
      for (int i = 0; i < vect.Length; ++i) vect[i] = false;
      size = 0;
   }
   public void Insert(int e)
   {
      if (e < 0 || e >= vect.Length) throw new IllegalElementException(e);
      if (!vect[e]) { vect[e] = true; ++size; }
   }

   ...
}
```

Tömb-reprezentáció folytatás

```
class ArraySet : ISetRepr
{
   ...
   public void Remove(int e)
   {
      if (e < 0 || e >= vect.Length) throw new IllegalElementException(e);
      if (vect[e]) { vect[e] = false; --size;}
   }
   public int Select()
   {
      int e;
      for (e = 0; !vect[e]; ++e);
      return e;
   }
   public bool Empty() { return size == 0; }
   public bool In(int e)
   {
      if (e < 0 || e >= vect.Length) throw new IllegalElementException(e);
      return vect[e];
   }
}
```

Halmaz típus
2.rész
Felelősség átruházás, sekély és mélymásolás

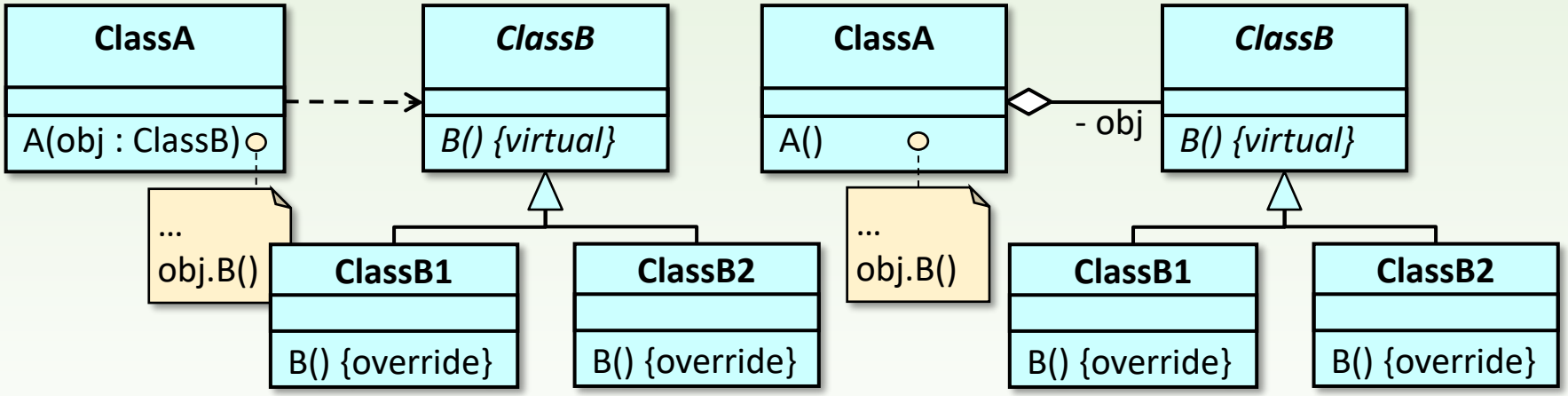
Gregorics Tibor

gt@inf.elte.hu

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

Objektum befecskendezés

- Ez az objektumelvű technika, amely a rugalmasság érdekében egy objektum tevékenységnek egy részét egy másik objektum metódusában írja le, két objektum összkapcsolásával biztosítja a tevékenység végrehajtását, amitől a tevékenység rugalmasan módosíthatóvá válik.



Stratégia: a *ClassA*-nak az A() metódusa felhasználja egy *ClassB* interfészű objektum B() metódusát.

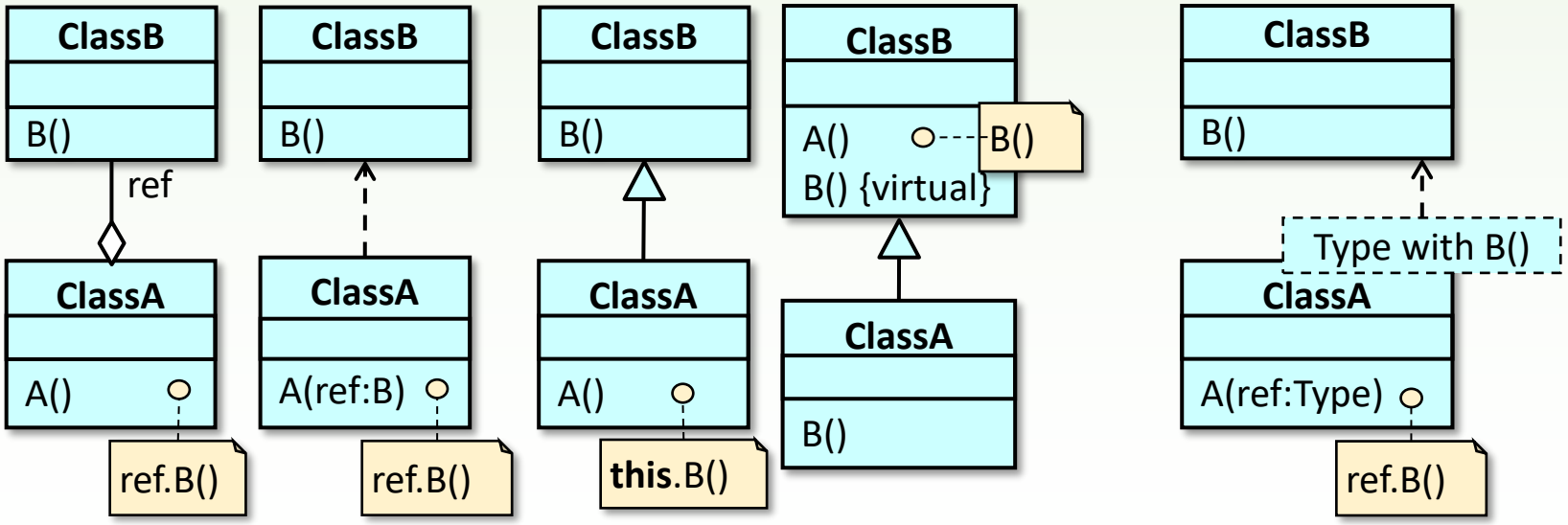
Látogató: a *ClassA* osztály alosztályaiba származtatott *A*() metódus egy *ClassB* ősű objektumtól (is) függ.

Híd: a *ClassA* osztállyal leírt típus megvalósítását egy abba komponált objektum szolgáltatja.

Felelősség átruházás technikái

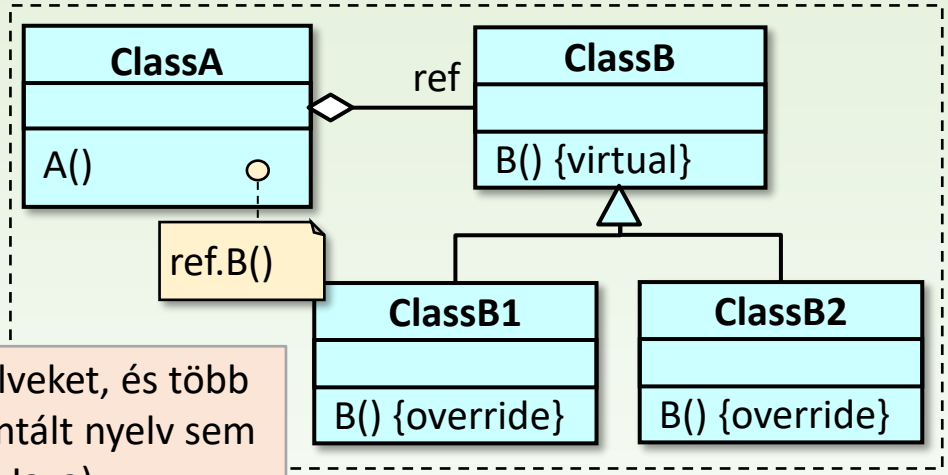
- A felelősség átruházás (*dependency injection*) egy objektum viselkedését (metódusainak működését) másik osztály kódjától teszi függővé.

- Objektum befecskendezéssel: az objektum metódusa egy másik objektum metódusát hívja.
- Származtatással: az objektum metódusa az ősosztályának nem felülírt metódusát hívja.
- Osztálysablonnal: az objektum metódusa a sablonparaméterében adott osztály metódusát hívja.

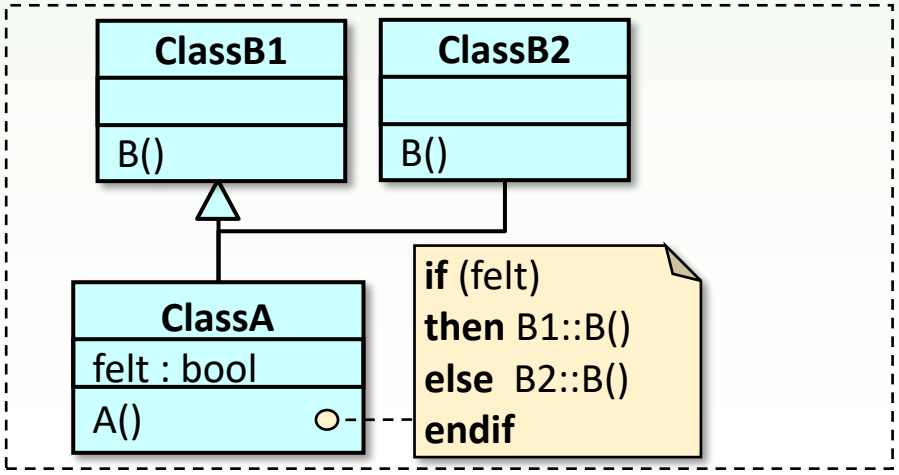


Felelősség átruházások összevetése

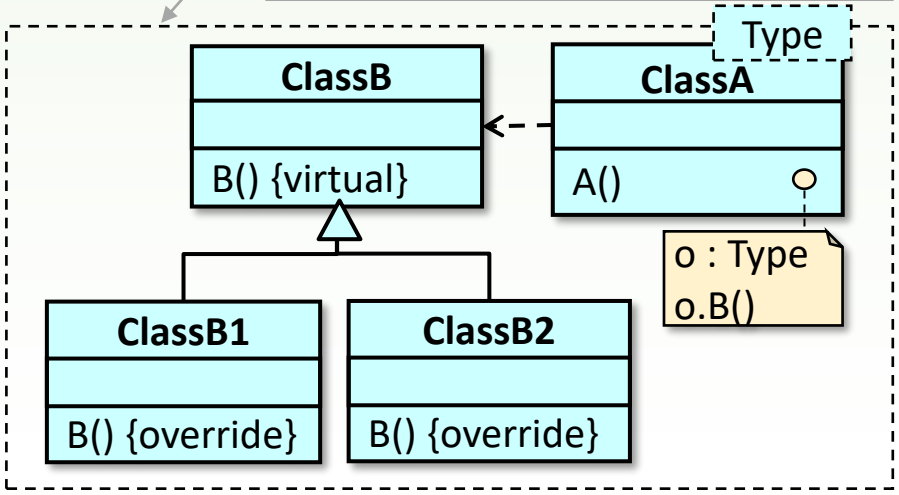
a ClassA bármelyik objektumára futási időben állítható be, hogy a ClassB1-nek vagy az ClassB2-nek a B() metódusát használja



sérti a SOLID elveket, és több objektum-orientált nyelv sem támogatja (C#, Java)



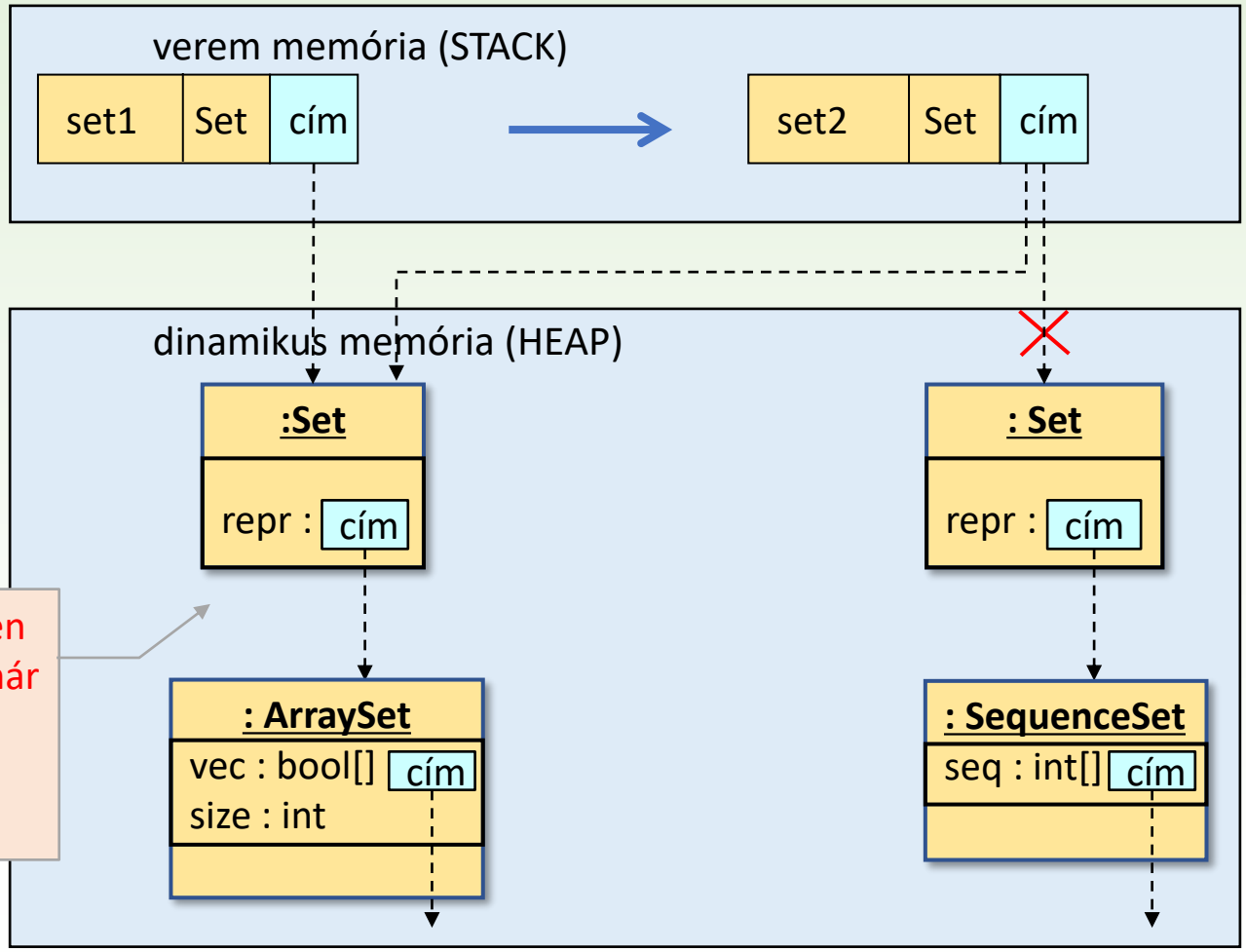
fordítási időben kell a Type helyére a ClassB1 vagy a ClassB2 típust írni, és az így kapott ClassA osztály minden objektumára ugyanaz a beállítás lesz érvényes.



Sekély másolás

```
Set set1 = new (10);
Set set2 = new ();
```

```
set2 = set1;
```



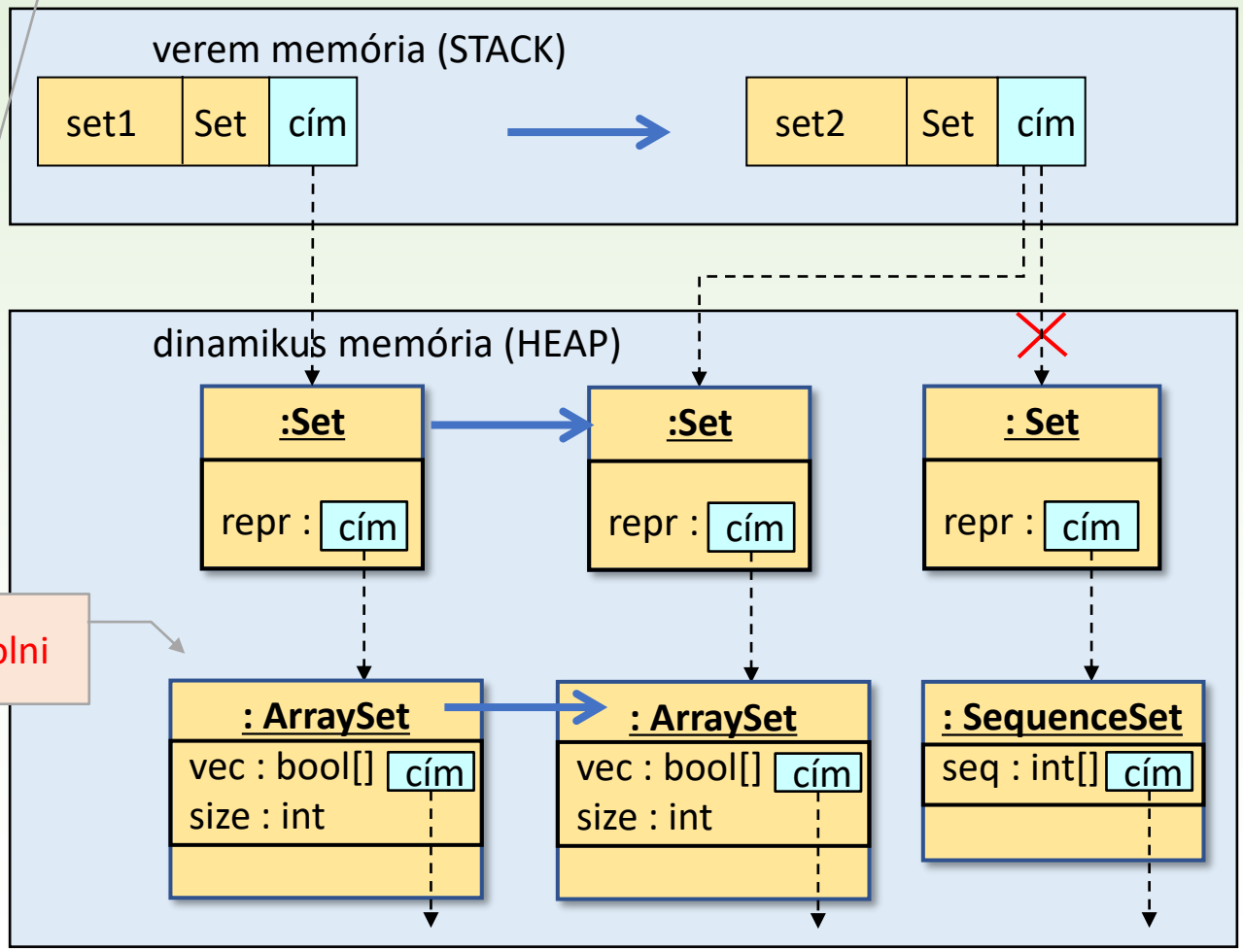
A látszólag egymástól független halmazok az értékadás után már azonosak lesznek:
- ha az egyik változik, vele együtt változik a másik is

Mély másolás

Szabványos megoldás: a lemásolandó objektum osztályában megvalósítjuk az `ICloneable` interfész `Clone()` metódusát, amelynek másolatot kell készíteni az objektumról.
Hátránya: a `Clone()` metódus `object`-tel tér vissza, amelyet kasztolni kell.

```
Set set1 = new (10);
Set set2 = new ();
```

```
set2 = (Set)set1.Clone();
```



Mindent le kell másolni

Klónozás

```
Set set1 = new (10);
Set set2 = new ();
set2 = (Set)set1.Clone();
```

```
class Set : ICloneable
{
  private ISetRepr repr;
  public object Clone()
  {
     return new Set() { repr = (ISetRepr)repr.Clone() };
  }
   ...
}
```

a this.repr klónozása

inicializáló blokk: példányosítás és adattag inicializálás egyben

```
class ArraySet : ISetRepr, ICloneable
{
  private bool[] vect;
  private int size;
  public object Clone()
  {
    ArraySet copy = new (size)
    { vect = (bool[])vect.Clone() };
    copy.size = size;
    return copy;
  }
  ...
}
```

a bool[] tömb Clone() metódusát használjuk

```
class SequenceSet : ISetRepr, ICloneable
{
  private List<int> seq = new ();
  public object Clone()
  {
     return new SequenceSet()
      { seq = new List<int>(seq) };
  }
  ...
}
```

A List<int>-nek nincs Clone() metódusa, de hívhatjuk az ún. másoló konstruktorát, amely mélymásolást végez.

Másoló konstruktor

```
Set set1 = new (10);
Set set2 = new (set1);
```

```
class Set
{
   private ISetRepr repr;
   public Set(Set other)
   {
      if      (other.repr is SequenceSet seqrepr) repr = new SequenceSet(seqrepr);
      else if (other.repr is ArraySet arrayrepr)  repr = new ArraySet(arrayrepr);
   }
   ...
}
```

típus vizsgálat és értékadás

a reprezentációs osztályok másoló konstruktoraira támaszkodunk

```
class SequenceSet : ISetRepr
{
   private List<int> seq = new ();
   public SequenceSet(SequenceSet source) { seq = new List<int>(source.seq);
}
   ...
}
```

ez a `List<int>` másoló konstruktora

```
class ArraySet : ISetRepr
{
  private bool[] vect;
  int size;
  public ArraySet(ArraySet source)
  {
    vect = new bool[source.Length];
    source.vect.CopyTo(vect, 0);
    size = source.size;
  }
  ...
}
```

```
for (int i = 0; i < vect.Length; ++i)
{
  vect[i] = source.vect[i];
}
```

Halmaz típus
3.rész
Halmaz elemeinek felsorolása

Gregorics Tibor

gt@inf.elte.hu

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

Egy feladat

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

(A keresés biztos sikertelen, ha nincs a halmazban legalább négy szám.)

- A feladat megoldható egy lineáris keresésbe ágyazott számlálással: az első olyan elemét keressük a halmaznak, amelyre teljesül, hogy a nálánál kisebb halmazbeli elemek száma nagyobb vagy egyenlő, mint 3. Mindkét algoritmus mintához a halmaz elemeit kell felsorolni.

A = (h:set($\mathbb{N}$), l:$\mathbb{L}$, n:$\mathbb{N}$)

Ef = (h = h_0)

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

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

Naiv megoldás

```
Set h = new (25);
...
bool l = false;
int elem;
for (; !l && !h.Empty(); h.Remove(h.Select()) )
{
   int e = h.Select();
   int c = 0;
   for (; !h.Empty(); h.Remove(h.Select()) )
   {
      int f = h.Select();
      if (e > f) ++c;
   }
   if (l = (c >= 3)) elem = e;
}
```

```
for(h.First(); !h.End(); h.Next())
{
    int e = h.Current();
}
```

A halmaz műveletei alkalmasak felsorolásra:

First()	~	–
Current()	~	Select()
Next()	~	Remove(Select())
End()	~	Empty()

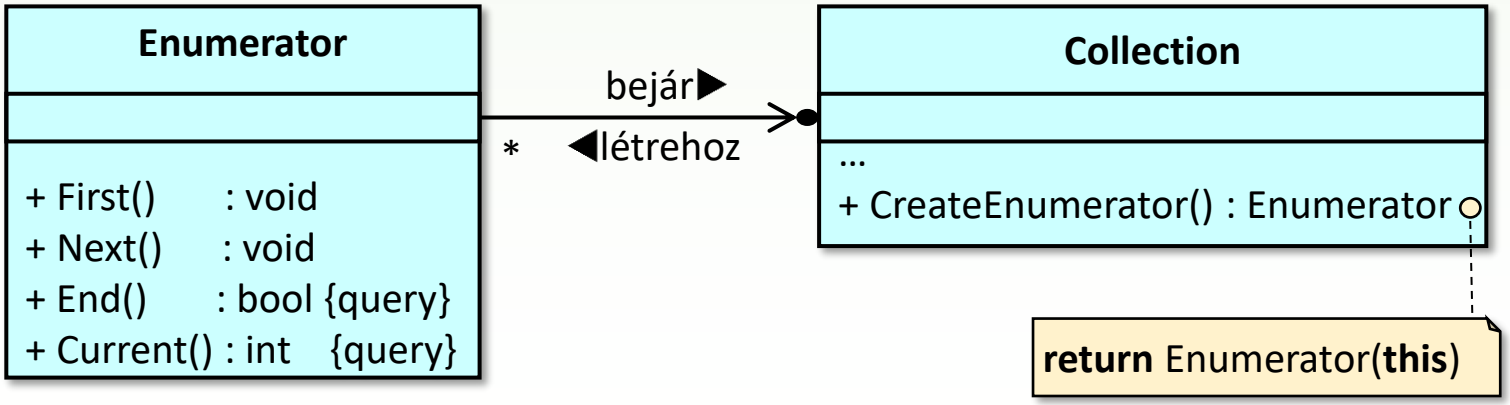
Select() legyen determinisztikus

Ez egy rossz megoldás, mert az egymásba ágyazott felsorolások nem függetlenek: a belső ciklus nem egy új felsorolást kezd az eredeti halmazon, hanem folytatja a külső ciklus felsorolását, sőt be is fejezi azt;
módosítják a gyűjteményt: a belső ciklus első lefutása kiüríti a halmazt, így a külső ciklus „Next()”-je ezután kivételt dob.

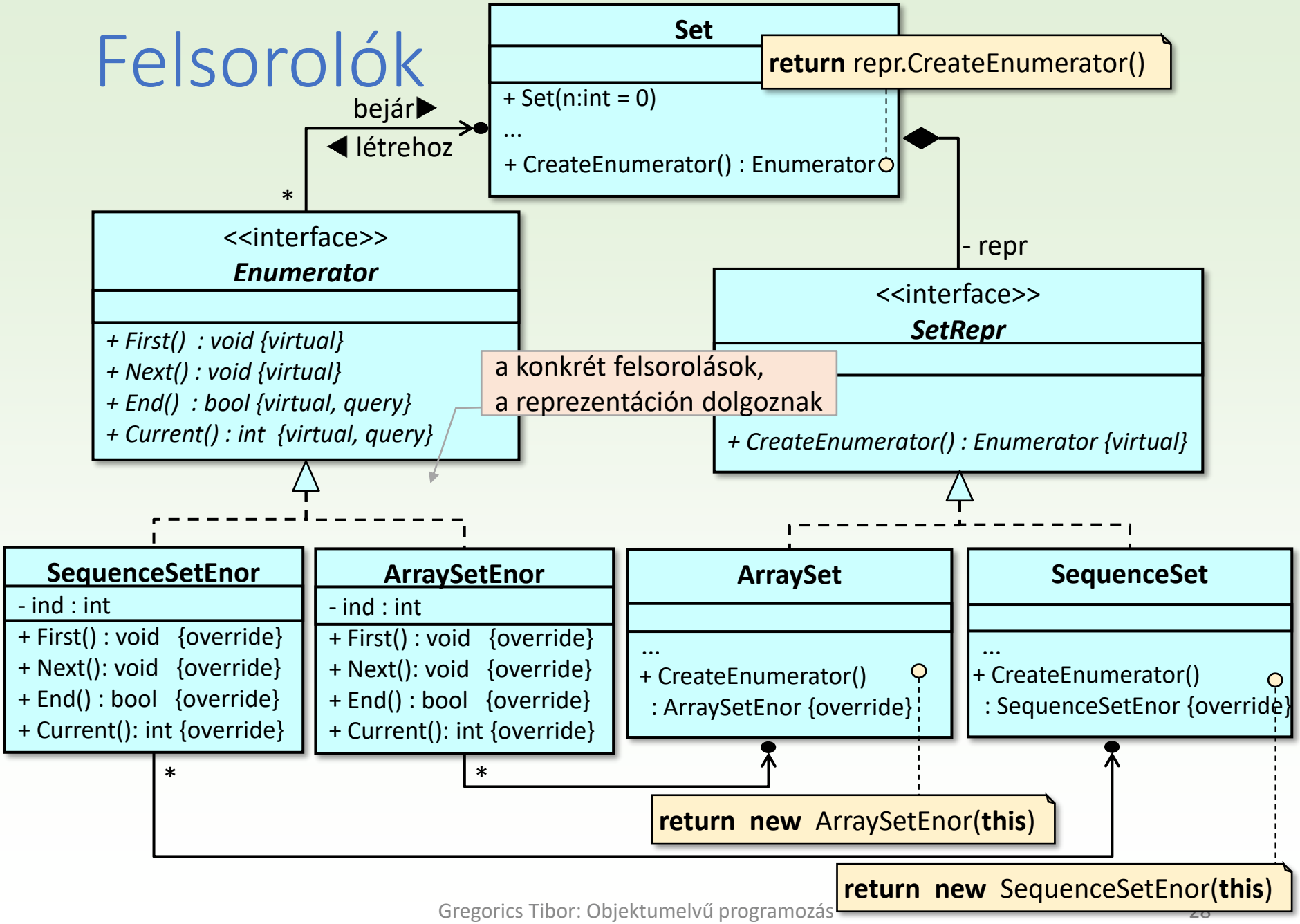
Felsoroló (iterátor) tervezési minta

- Egy gyűjtemény elemeinek felsorolását (bejárását) a gyűjteménytől elkülönülő, önálló objektummal (felsorolóval) érdemes végezni.
- A felsorolónak el kell tudni érni a felsorolandó gyűjteményt: hivatkozni kell tudnia rá, de nem módosítja azt.
- A felsoroló objektumot a felsorolni kívánt gyűjtemény hozza létre, akár többet is, és nyilvántarthatja az őt bejáró felsorolókat.

Célszerű, ha a felsorolás műveletei csak a felsorolandó gyűjteménynek a lekérdező (getter) metódusaira támaszkodnak azért, hogy egyszerre több felsoroló is dolgozhasson a gyűjteményen anélkül egymást zavarnák.

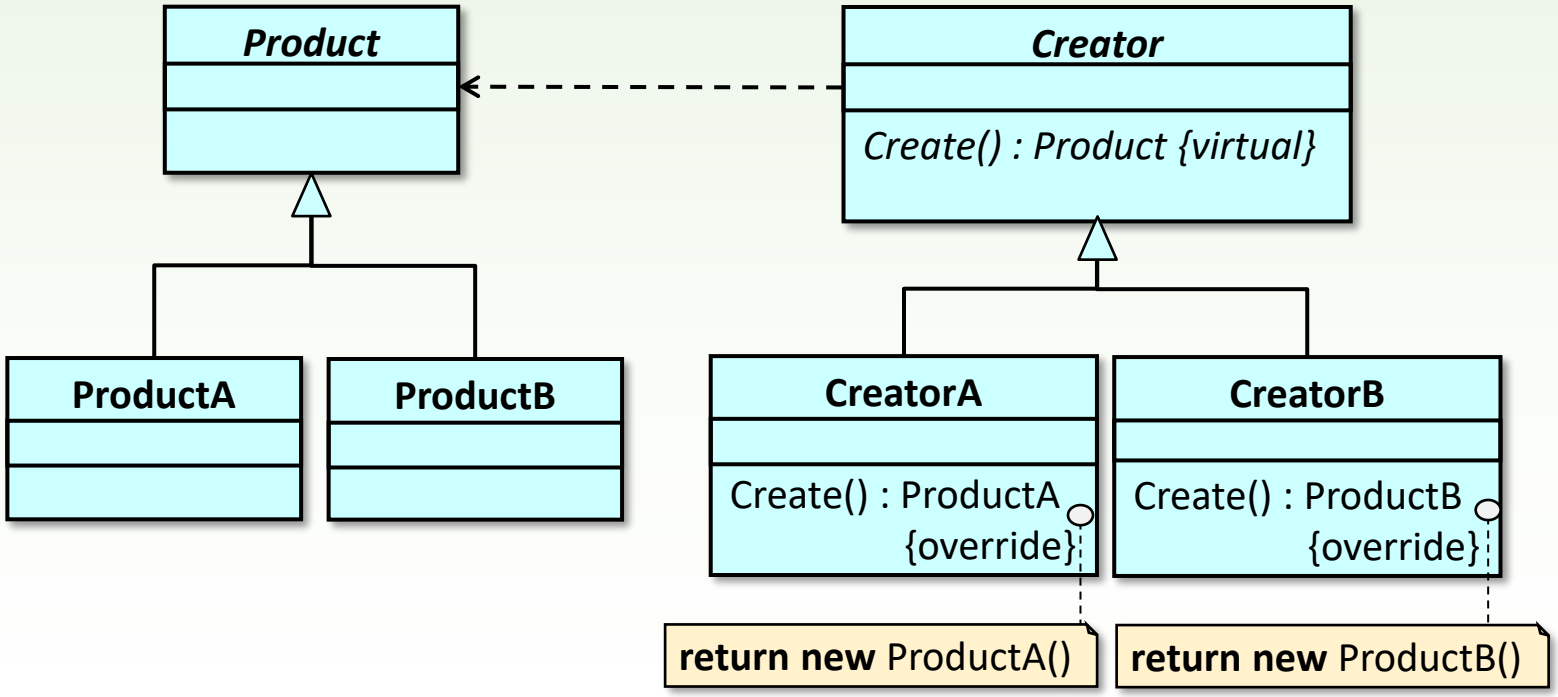


Felsorolók


Set
return repr.CreateEnumerator()
+ Set(n:int = 0)
...
+ CreateEnumerator() : Enumerator
bejár▶
◀ létrehoz
*
- repr
<<interface>>
Enumerator
+ First() : void {virtual}
+ Next() : void {virtual}
+ End() : bool {virtual, query}
+ Current() : int {virtual, query}
<<interface>>
SetRepr
+ CreateEnumerator() : Enumerator {virtual}
a konkrét felsorolások,
a reprezentáción dolgoznak
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 (factory method) tervezési minta

- Egy ősosztály alosztályainak objektumait egy segédosztály metódusával (Create) hozzuk létre, amely a példányosítást átruházza a segédosztály megfelelő alosztályára.



Felsorolás felsoroló objektumokkal

Külön felsoroló objektumok sorolják fel egymástól függetlenül a halmaz elemeit, miközben a halmaz nem változik.

```
Set h  = new (15);
…
bool l = false;
int elem = 0;
IEnumerator enor1 = h.CreateEnumerator();
for( enor1.First(); !enor1.End(); enor1.Next())
{
   int c = 0;
   IEnumerator enor2 = h.CreateEnumerator();
   for (enor2.First(); !enor2.End(); enor2.Next())
      if ( enor1.Current() > enor2.Current() ) ++c;
   if ( (l = (c >= 3)) ) { elem = enor1.Current() ; break; }
}
```

```
interface IEnumerator
{
    void First();
    void Next();
    bool End();
    int  Current();
}
```

```
class Set : ICloneable
{
  ...
  private ISetRepr repr;
  ...
  public IEnumerator CreateEnumerator() { return repr.CreateEnumerator(); }
}
```

SequenceSet felsorolója

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

SequenceSetEnor

- s : SequenceSet
- ind : int

+ First() : void {override}
+ Next(): void {override}
+ End() : bool {override}
+ Current(): int {override}

```
class SequenceSet : ISetRepr
{
   private List<int> seq = new ();
   ...
   public class SequenceSetEnor : IEnumerator
   {
      private readonly SequenceSet s;
      private int ind;

      public SequenceSetEnor(SequenceSet h) { s = h; }

      void First()  {ind = 0;}
      void Next()   {++ind;}
      bool End()    {return ind >= s.seq.Count;}
      int  Current(){return s.seq[ind];}
   }

   public override MyEnumerator CreateEnumerator() { return new SequenceSetEnor(this);}
}
```

beágyazott osztály

egy interfész metódusainak implementációi publikusak

ArraySet felsorolója

```
class ArraySet : ISetRepr
{
   private bool[] vect;
   ...
   public class ArraySetEnor : IEnumerator
   {
      private readonly ArraySet s;
      private int ind;

      public ArraySetEnor(ArraySet h) { s = h; }

      private void FindNext()
      {
         for (++ind; ind < s.vect.Length && !s.vect[ind]; ++ind);
      }

      void First()   {ind = -1; FindNext();}
      void Next()    {FindNext(); }
      bool End()     {return ind>=s.vect.Length;}
      int  Current() {return ind;}
   }

   public override MyEnumerator CreateEnumerator() { return new ArraySetEnor(this); }
}
```

A halmaz elemei a halmazt reprezentáló tömb azon indexei, ahol a tömb true értéket tárol. Ezen indexeket soroljuk fel.

beágyazott osztály

soron következő true érték keresése a vect tömbben (kiválasztás algoritmusa)

egy interfész metódusainak implementációi publikusak

ArraySetEnor

- s : ArraySet
- ind : int

+ First() : void {override}
+ Next(): void {override}
+ End() : bool {override}
+ Current(): int {override}
- FindNext():void

Felsoroló objektumok létrehozása

```
interface ISetRepr : IClonable
{
   ...
   public abstract IEnumerator CreateEnumerator();
}
```

```
class SequenceSet : ISetRepr
{
   private List<int> seq;
   ...
   public override IEnumerator CreateEnumerator()
   {
      return new SequenceSetEnor(this);
   }
}
```

```
class ArraySet : ISetRepr
{
   private bool[] vect;
   ...
   public override IEnumerator CreateEnumerator()
   {
      return new ArraySetEnor(this);
   }
}
```

```
public class SequenceSetEnor : IEnumerator
{
   void First()   { … }
   void Next()    { … }
   bool End()     { … }
   int  Current() { … }
}
```

```
public class ArraySetEnor : IEnumerator
{
   void First()   { … }
   void Next()    { … }
   bool End()     { … }
   int  Current() { … }
}
```

Javítás

Tegyük biztonságosabbá a halmaz felsorolását!

- Probléma: Elvben hibát okozhat, ha felsorolás közben valamilyen változtatást végzünk a felsorolandó gyűjteményen.
- Megoldás: Akadályozzuk meg egy gyűjtemény módosító műveleteinek végrehajtását, ha felsorolás folyik éppen a gyűjteményen.

```
Set h;
...
MyEnumerator enor = h.CreateEnumerator();
foreach (int e in enor )
{
   h.Remove(e);
}
```

egy halmaz Remove() művelete dobjon kivételt, ha a halmaz felsorolás alatt áll.

Kizárás megvalósítása

```
class Set
{
    public class UnderTraversalException : Exception { }
    ...
    public void Remove(int e)
    {
        if (repr.EnumeratorCount!=0) throw new UnderTraversalException();
        repr.Remove(e);
    }
    ...
}
```

ismerni kell a halmazon dolgozó felsorolók számát

`ISetRepr` interfészből `SetRepr` absztrakt osztály

```
abstract class SetRepr
{
    public int EnumeratorCount { get; protected set;} = 0;

    ...
}
```

Felsorolók számának karbantartása

```
class SequenceSet : SetRepr
{
   public SequenceSet() : base() { ... }
   ...
   public class SequenceSetEnor : MyEnumerator
   {
      private readonly SequenceSet s;
      private int ind;

      public SequenceSetEnor(SequenceSet h)
      {
         s = h;
         if(s.seq.Count>0) ++s.EnumeratorCount;
      }
      public override void First() { ind = 0; }
      public override void Next()  { ++ind;}
      public override bool End()   { return ind>=s.seq.Count; }
      public override int Current(){ return s.seq[ind]; }
      ~SequenceSetEnor() { --s.EnumeratorCount; }
   }

   public override MyEnumerator CreateEnumerator() {return new SequenceSetEnor(this);}
}
```

A SetRepr ősosztály konstruktora nullázza le a felsoroló-számlálót

felsoroló létrehozásakor növeljük, a felsoroló-számlálót

a felsoroló megszűnésekor csökkentjük a felsoroló-számlálót

Felsorolók számának karbantartása

```
class ArraySet : SetRepr
{
   public ArraySet() : base() { ... }
   ...
   public class ArraySetEnor : MyEnumerator
   {
      private readonly ArraySet s;
      private int ind;

      public ArraySetEnor(ArraySet h)
      {
         s = h;
         if(s.seq.Count>0) ++s.EnumeratorCount;
      }
      private void FindNext()
      {
         for (++ind; ind < s.vect.Length && !s.vect[ind]; ++ind) ;
      }
      public override void First()  { ind = -1; FindNext(); }
      public override void Next()   { FindNext(); }
      public override bool End()    { return ind == s.vect.Length; }
      public override int Current() { return ind; }
      ~ArraySetEnor() { --s.EnumeratorCount; }
   }

   public override MyEnumerator CreateEnumerator() { return new ArraySetEnor(this);}
}
```

A SetRepr ősosztály konstruktora nullázza le a felsoroló-számlálót

felsoroló létrehozásakor növeljük, a felsoroló-számlálót

a felsoroló megszűnésekor csökkentjük a felsoroló-számlálót

Felsorolás felsoroló objektumokkal

```
Set h  = new (15);
…
try {
   bool l = false;
   int elem = 0;
   IEnumerator enor1 = h.CreateEnumerator();
   for( enor1.First(); !enor1.End(); enor1.Next())
   {
      int c = 0;
      IEnumerator enor2 = h.CreateEnumerator();
      for (enor2.First(); !enor2.End(); enor2.Next())
         if ( enor1.Current() > enor2.Current() ) ++c;
      if ( (l = (c >= 3)) ) { elem = enor1.Current() ; break; }
   }
} catch() {Set.UnderTraversalException ex}
```

Egyszerűbben: foreach ciklussal

```
Set h  = new (15);
…
bool l = false;
int elem = 0;
foreach ( int e in h )
{
   int c = 0;
   foreach ( int f in h )
   {
      if (e > f) ++c;
   }
   if ( (l = (c >= 3)) )
   {
      elem = e; break;
   }
}
```

```
class Set : IEnumerable
{
   private ISetRepr repr;
   …
   IEnumerator IEnumerable.GetEnumerator()
   {
      foreach (var e in repr) yield return e;
   }
}
```

```
interface ISetRepr : IEnumerable { … }
```

```
class SequenceSet : ISetRepr
{
   private List<int> seq = new ();
   …
   IEnumerator IEnumerable.GetEnumerator()
   {
      foreach (var e in seq) yield return e;
   }
}
```

```
class ArraySet : ISetRepr
{
   private bool[] vect;
   …
   IEnumerator IEnumerable.GetEnumerator()
   {
      foreach(int i in Enumerable.Range(0,vect.Length-1))
         if (vect[i]) yield return i;
   }
}
```

Halmaz típus
4.rész
Halmaz osztály-sablon

Gregorics Tibor

gt@inf.elte.hu

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

Általánosítás

- Olyan halmaz példányosítását is szeretnénk támogatni, amely elemeinek típusát szabadon megválaszthatjuk.
- Ehhez a halmaz osztályát (és reprezentációját is) generikussá alakítjuk: egy típus-paraméterrel jelezzük az elemek típusát. Szükséges lesz arra is, hogy a felsoroló osztályokat is generikussá tegyük.
- A tömbös reprezentációval azonban továbbra is csak felső korláttal rendelkező természetes számokat tartalmazó halmazokat ábrázolhatunk.

fordítási időben osztályként példányosodik az osztálysablon

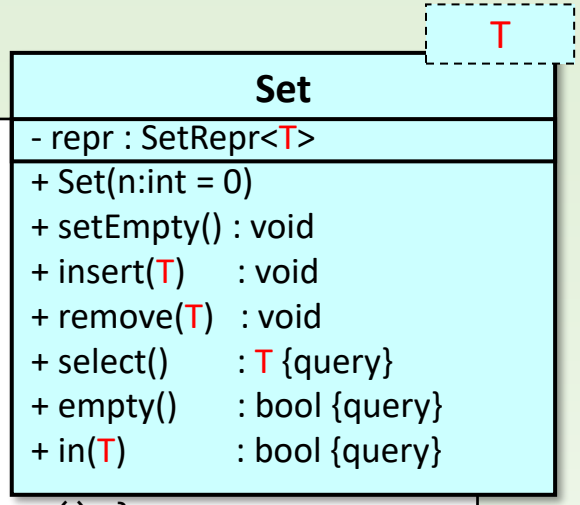
futási időben objektumként példányosodik az osztály

tömbös reprezentáció esetén az elemi típus csak int lehet

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

h1.Insert(-490);
h2.Insert("alma");
h3.Insert(42);
```

Set osztály

```
class Set<T> : IEnumerable, ICloneable
{
   private SetRepr<T> repr;

   public Set(int n = 0)   { … }
   public Set(Set<T> other){ … }
   public object Clone()
   {
      return new Set<T>() { repr = (SetRepr<T>)repr.Clone() };
   }
   public void SetEmpty()  { repr.SetEmpty(); }
   public void Insert(T e) { repr.Insert(e); }
   public void Remove(T e) { repr.Remove(e); }
   public T    Select()    { repr.SetEmpty(); }
   public bool Empty()
   {
      if (Empty()) throw new EmptySetException();
      return repr.Select();
   }
   public bool In(T e)     {return repr.Empty(); }
   public IEnumerator IEnumerable.GetEnumerator()
   {
      foreach (var e in repr) yield return e;
   }
}
```

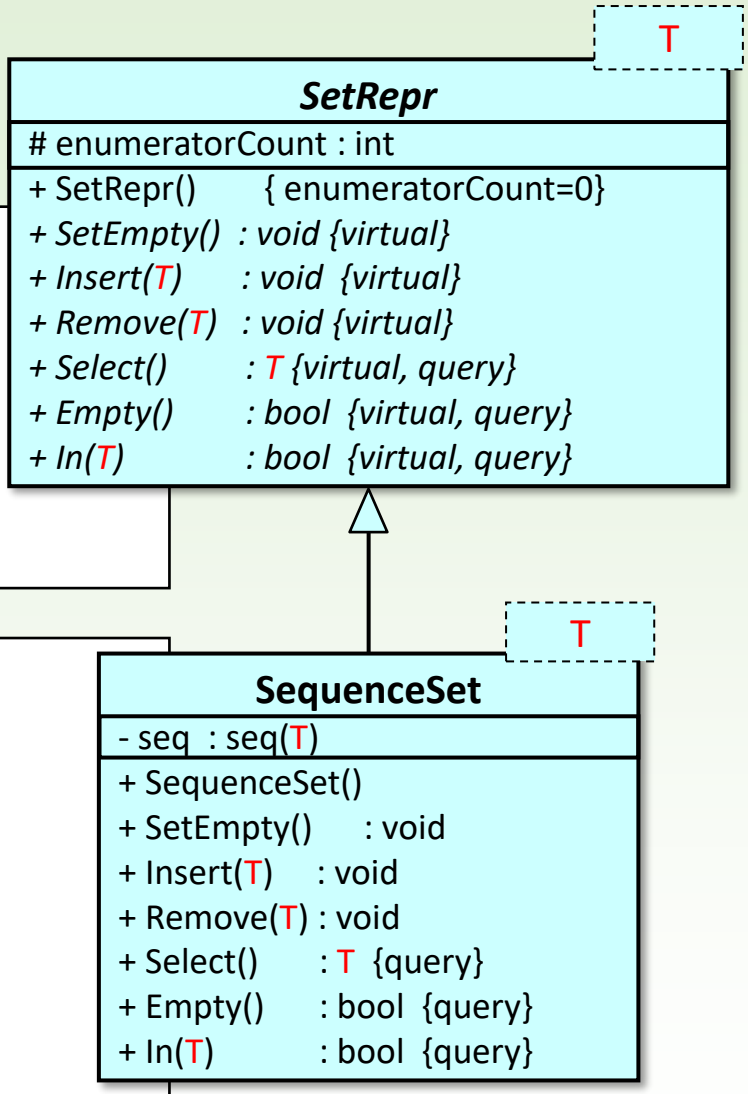


SequenceSet osztály

```
interface SetRepr<T> : IEnumerable, ICloneable
{
    object Clone();
    void Insert(T e);
    void Remove(T e);
    T Select();
    bool In(T e);
}
```

típus-paraméter

```
class SequenceSet<T> : SetRepr<T>
{
    private readonly List<T> seq = new ();
    public SequenceSet<T>() { ... }
    public SequenceSet<T>(SequenceSet<T>) { ... }
    public object Clone()   { ... }
    public void Insert(T e) { ... }
    public void Remove(T e) { ... }
    public T    Select()    { ... }
    public bool In(T e)     { ... }
    ...
}
```

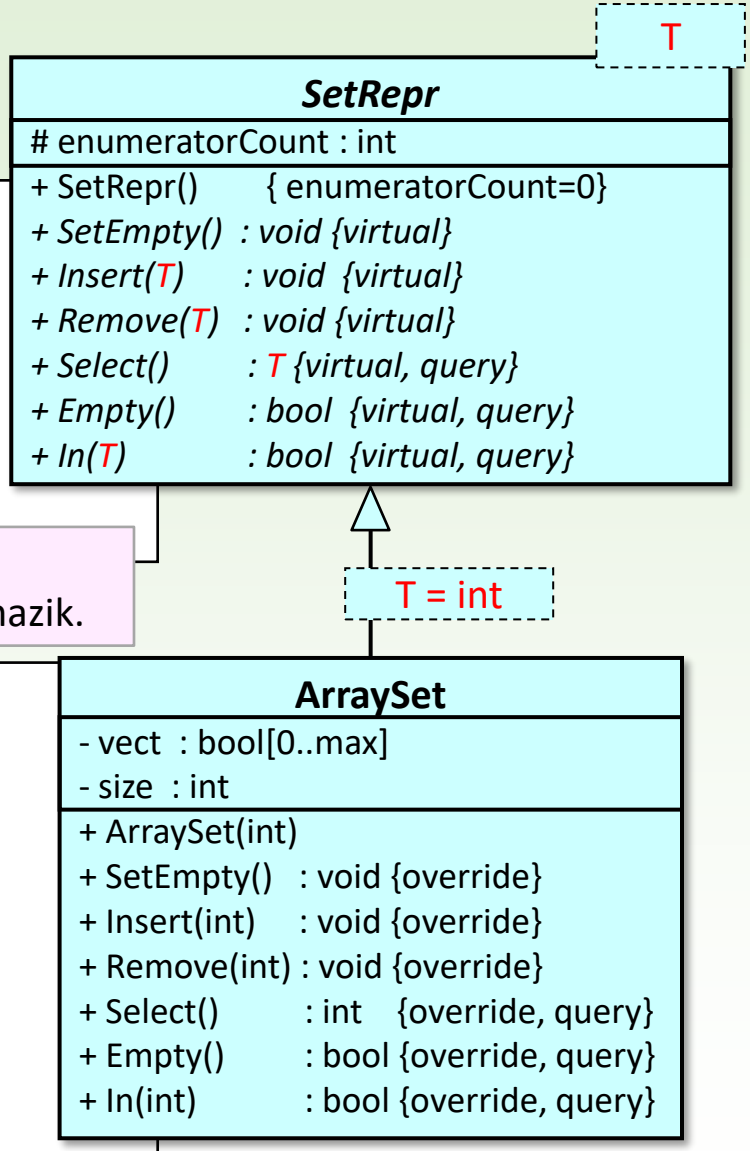


ArraySet osztály

```
interface SetRepr<T> : IEnumerable, ICloneable
{
    object Clone();
    void Insert(T e);
    void Remove(T e);
    T Select();
    bool In(T e);
}
```

Az ArraySet már nem generikus:
a SetRepr helyett SetRepr<int>-ből származik.

```
class ArraySet : SetRepr<int>
{
    private bool[] vect;
    int size;
    public ArraySet(int n)      { ... }
    public ArraySet(ArraySet)   { ... }
    public object Clone()       { ... }
    public void Insert(T e)     { ... }
    public void Remove(T e)     { ... }
    public T    Select()        { ... }
    public bool In(T e)         { ... }
}
```



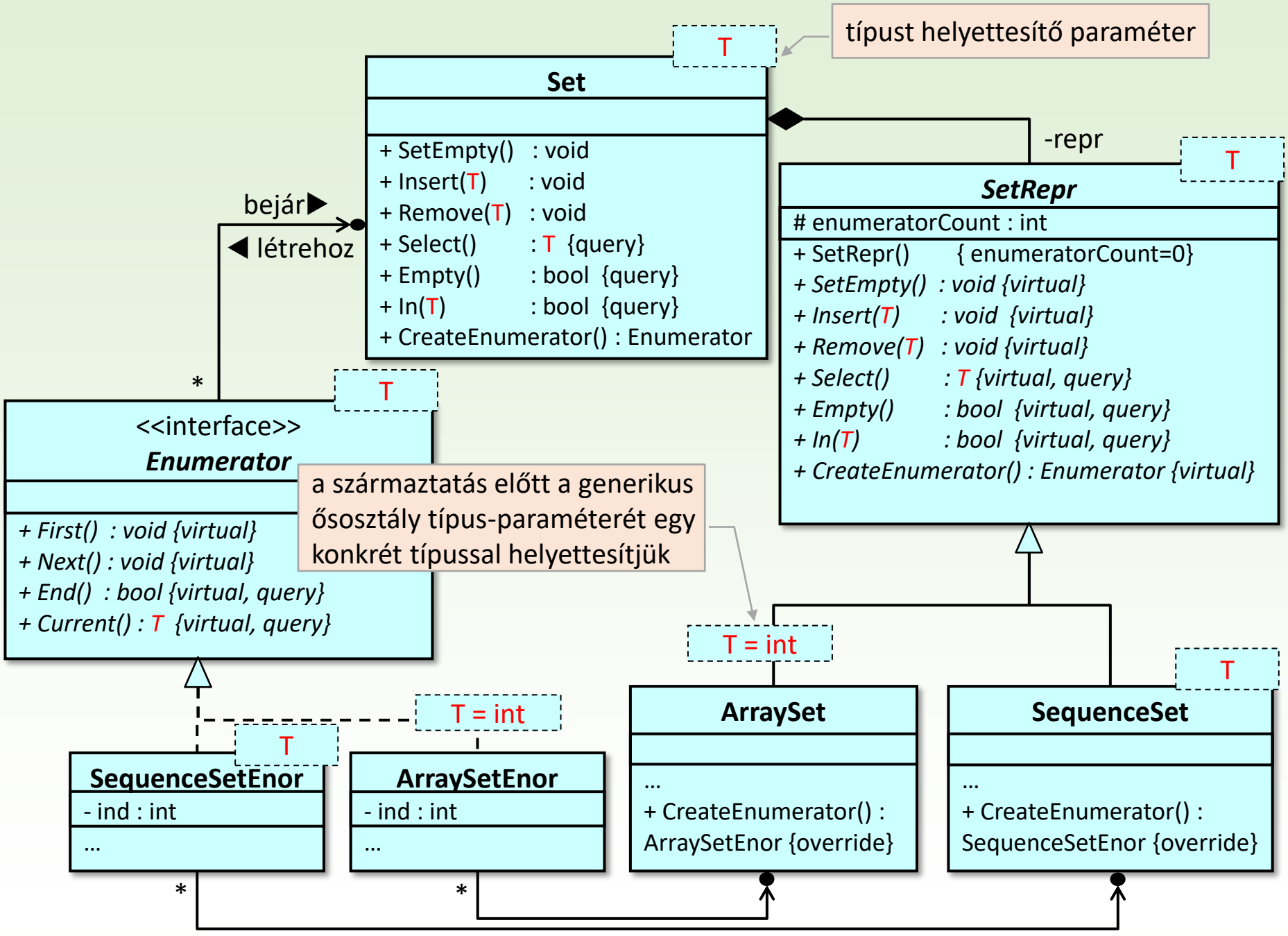
Set osztály konstruktorai

```
public Set(int n = 0)
{
   object o = null;
   if (typeof(T) == typeof(int))
   {
      if (0 == n) o = new SequenceSet<int>();
      else        o = new ArraySet(n);
   }
   else
   {
      if (n != 0) throw new MeaninglessConstructorParameter();
      else o = new SequenceSet<T>();
   }
   repr = (SetRepr<T>)o;
}
public Set(Set<T> other)
{
   object o = null;
   if (typeof(T) == typeof(int))
   {
      if (other.repr is SequenceSet<int> seqrepr) o = new SequenceSet<int>(seqrepr);
      else if (other.repr is ArraySet arrayrepr)  o = new ArraySet(arrayrepr);
   }
   else o = new SequenceSet<T>(other.repr as SequenceSet<T>);
   repr = (SetRepr<T>)o;
}
```

típus-paraméter vizsgálata

ha az elemek típusa nem int, akkor nem lehet az elemekre korlátot adni

az other.repr-re SequenceSet<T>-re kasztolja, ha nem sikerül, null-t ad vissza


típust helyettesítő paraméter
T
Set
+ SetEmpty() : void
+ Insert(T) : void
+ Remove(T) : void
+ Select() : T {query}
+ Empty() : bool {query}
+ In(T) : bool {query}
+ CreateEnumerator() : Enumerator
-repr
T
SetRepr
enumeratorCount : int
+ SetRepr() { enumeratorCount=0}
+ SetEmpty() : void {virtual}
+ Insert(T) : void {virtual}
+ Remove(T) : void {virtual}
+ Select() : T {virtual, query}
+ Empty() : bool {virtual, query}
+ In(T) : bool {virtual, query}
+ CreateEnumerator() : Enumerator {virtual}
bejár
létrehoz
*
T
<<interface>>
Enumerator
+ First() : void {virtual}
+ Next() : void {virtual}
+ End() : bool {virtual, query}
+ Current() : T {virtual, query}
a származtatás előtt a generikus ősosztály típus-paraméterét egy konkrét típussal helyettesítjük
T = int
T = int
T
SequenceSetEnor
- ind : int
...
ArraySetEnor
- ind : int
...
ArraySet
...
+ CreateEnumerator() : ArraySetEnor {override}
T
SequenceSet
...
+ CreateEnumerator() : SequenceSetEnor {override}
*
*