

Típus és osztály

Gregorics Tibor

`gt@inf.elte.hu`

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

Procedurális vs. objektumelvű paradigma

- ❑ **Procedurális szemléletmód:** Egy probléma megoldásához a problémát részfeladatokra bontjuk, és az ezeket megoldó tevékenységeket önálló egységekbe, ún. **procedúrákba** (részprogram, makró, eljárás, függvény) szervezzük. A problémát megoldó folyamatot a procedúrák közötti vezérlés-átadásoknak (eljárások, függvények esetében hívásoknak) láncolata határozza meg.
- ❑ **Objektumelvű szemléletmód:** Egy probléma megoldáshoz szükséges adatok egy-egy részét a hozzájuk kapcsolódó tevékenységekkel (az ún. metódusokkal) együtt önálló egységekbe, ún. **objektumokba** zárjuk. A problémát megoldó folyamatot az objektumok metódusai közötti vezérlés-átadások (közvetlen hívások vagy szignál-küldések) jelöli ki.

Feladat

Egy nem üres tömbben 0 és m közé eső természetes számok találhatók. Melyik a tömb leggyakoribb eleme?

❑ Procedurális megoldás: **maximum kiválasztás** és **számlálás**

- Rendre megszámoljuk, hogy a tömb elemei hányszor fordulnak elő a tömbben, és megkeressük a legnagyobb előfordulás-számmal rendelkező tömbelemet.

❑ Objektumelvű megoldás: **tároló objektum**

- Készítünk egy olyan tárolót, amelyben egy elem elhelyezése is, és a leggyakoribb elemének lekérdezése is gyors. Elhelyezzük a tömb elemeit ebben tárolóban, majd lekérdezzük a leggyakoribb elemét.

Típus és osztály

1.rész

Végrehajtható specifikáció

Gregorics Tibor

gt@inf.elte.hu

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

Elemzés

A feladatot **változók** segítségével specifikáljuk. Az Ef azt írja le, hogy e változók kezdetben milyen értékeket tartalmazhatnak, az Uf pedig azt, hogy mi a célunk: milyen értékeknek kell majd megjelenni a változókban figyelembe véve azok kezdőértékeit.

$A = (x:\mathbb{N}^n, m:\mathbb{N}, \text{elem}:\mathbb{N})$

jelöléseket vezetünk be az input-változók kezdőértékeire: $x_0 \in \mathbb{N}^n, m_0 \in \mathbb{N}$

x legalább egy elemű
x elemei 0 és m közé esnek

$Ef = (x = x_0 \wedge m = m_0 \wedge n \geq 1 \wedge \forall i \in [1..n]: x[i] \in [0..m])$

az input-változók
őrzik kezdőértékeiket

bedobáljuk x elemeit a zsákba

$Uf = (x = x_0 \wedge m = m_0 \wedge b:\text{Zsák} \wedge b = \bar{\cup}_{i=1..n} [x[i]] \wedge \text{elem} = \text{leggyakoribb}(b))$

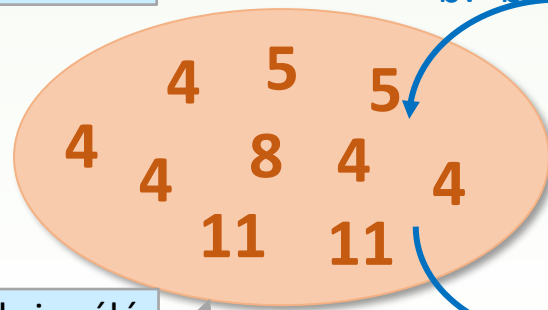
b egy zsák objektumra
hivatkozó segédváltozó

lekérjük a zsáktól
a leggyakoribb elemét

$\bar{\cup}$ egy zsákot egy zsákkal egyesítő művelet,
amelynek neutrális eleme az üres zsák (\emptyset).
[e] az e-t tartalmazó egyelemű zsákot jelöli.

Változók a típusaikkal
 $x:\mathbb{N}^n \sim x$ egy 1-től n-ig indexelt
természetes számokból
álló tömb típusú változó
 $m:\mathbb{N} \sim m$ egy természetes szám
típusú változó

$b =$



tárolóként funkcionáló
zsák objektum

$b := b \bar{\cup} [e]$

$e := \text{leggyakoribb}(b)$

Tervezés

$A = (x:\mathbb{N}^n, m:\mathbb{N}, b:\text{Zsák}, \text{elem}:\mathbb{N})$

$Ef = (m = m_0 \wedge x = x_0 \wedge n \geq 1 \wedge \forall i \in [1 .. n]: x[i] \in [0 .. m])$

$Uf = (Ef \wedge b = \bigcup_{i=1..n} [x[i]] \wedge \text{elem} = \text{leggyakoribb}(b))$

Összegzés algoritmus minta

$s = \sum_{i=m..n} f(i)$

$f:[m..n] \rightarrow H$

$+: H \times H \rightarrow H$

bal neutrális elem a 0

$s : H$

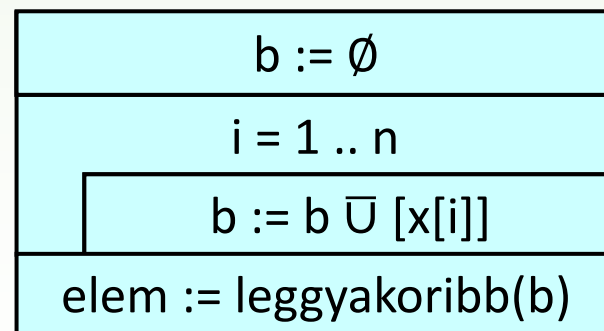
végrehajtható specifikáció: nemcsak azt írja le, hogy mi a feladat, hanem azt is, hogyan oldható meg. Elmosódik az elemzés és a tervezés közötti határ.

Visszavezetjük a bezsákolást az összegzésre :
egyezés:

$$s = \sum_{i=m..n} f(i) \sim b = \bigcup_{i=1..n} [x[i]]$$

eltérés:

eredmény:	$s : H$	\sim	$b : \text{Zsák}$
művelet:	$H, +, 0$	\sim	$\text{Zsák}, \bigcup, \emptyset$
elem:	$f(i)$	\sim	$[x[i]]$
felsorolás:	$i = m .. n$	\sim	$i = 1 .. n$



Megvalósítás input-output nélkül

$b := \emptyset$
$i = 1 \dots n$
$b := b \cup [x[i]]$
$\text{elem} := \text{leggyakoribb}(b)$

```
int n, m;  
...  
int[] x = new int[n];  
...  
Bag bag = new(m);  
bag.Erase();  
for( int i = 0; i < x.Count; ++i)  
{  
    bag.PutIn(x[i]);  
}  
int elem = bag.MostFrequent();
```

Diagram illustrating the mapping of C# code to set operations:

- `b:Zsák` points to `new(m)`
- `b := ∅` points to `bag.Erase()`
- `b := b ∪ [e]` points to `bag.PutIn(x[i])`
- `leggyakoribb(b)` points to `bag.MostFrequent()`

Hogyan adjuk meg a Bag és a műveleteinek a jelentését?
Ehhez a Zsák típust kell megtervezni, majd kódolni.

Típus és osztály

2.rész

Zsák típus

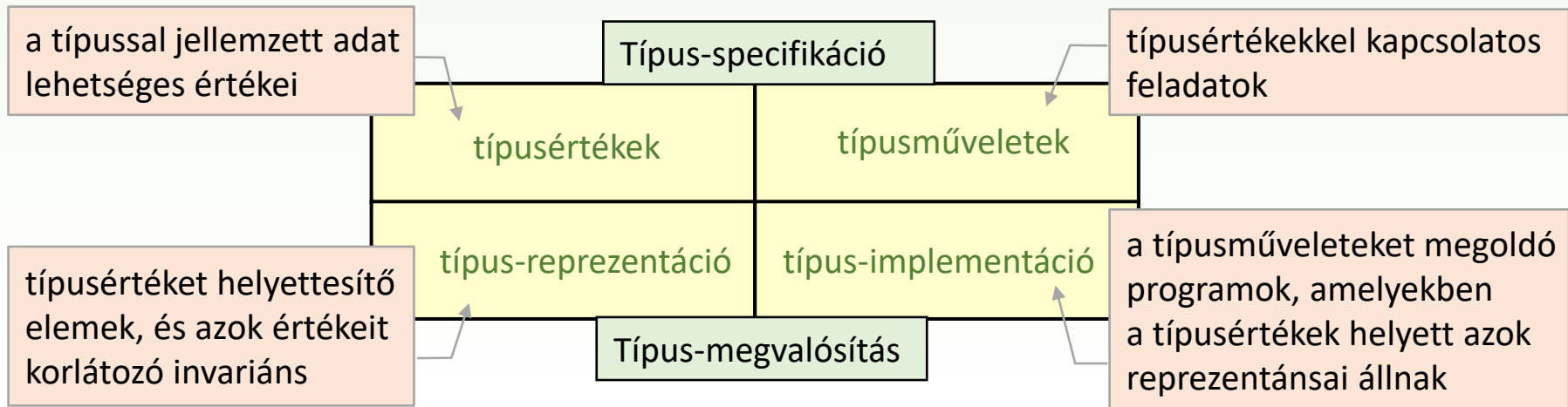
Gregorics Tibor

gt@inf.elte.hu

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

Adattípus fogalma

- ❑ Egy adat (változó) típusának definiálásához szükség van a típus specifikációjára és annak megvalósítására.
- ❑ A típus-specifikáció megadja:
 - az adat által felvehető értékek halmazát: **típusértékek**
 - a típusértékekkel végezhető műveleteket: **típusműveletek**
- ❑ A típus-megvalósítás megmutatja:
 - hogyan ábrázoljuk (**reprezentáljuk**) a típusértékeket
 - milyen programok helyettesítsék (**implementálják**) a műveleteket



Nevezetes adattípusok

bizonyos műveletek nem
mindenhol értelmezettek

- ❑ Természetes típus: $(\mathbb{N}, \{+, -, \cdot, /, \text{mod}\})$
- ❑ Egész típus: $(\mathbb{Z}, \{+, -, \cdot, /, \text{mod}\})$
- ❑ Valós típus: $(\mathbb{R}, \{+, -, \cdot, /\})$
- ❑ Logikai típus: $(\mathbb{L}, \{\wedge, \vee, \neg\})$
- ❑ Karakter típus: $(\mathbb{K}, \{\dots\})$
- ❑ Vektor típus: $(E^{m..n}, \{[.]\})$

az E halmaz elemeiből képzett egy dimenziós tömbök, ahol az elemeket m-től n-ig indexeljük ($m, n \in \mathbb{Z}, m \leq n+1$); $m=1$ esetén használjuk az E^n jelölést

- ❑ Mátrix típus: $(E^{l..n, k..m}, \{[. , .]\})$

az E halmaz elemeiből képzett két dimenziós tömbök, ahol a sorokat l-től n-ig, oszlopokat k-tól m-ig indexeljük ($l, n, k, m \in \mathbb{Z}, l \leq n+1, k \leq m+1$); $l=k=1$ esetén $E^{n \times m}$

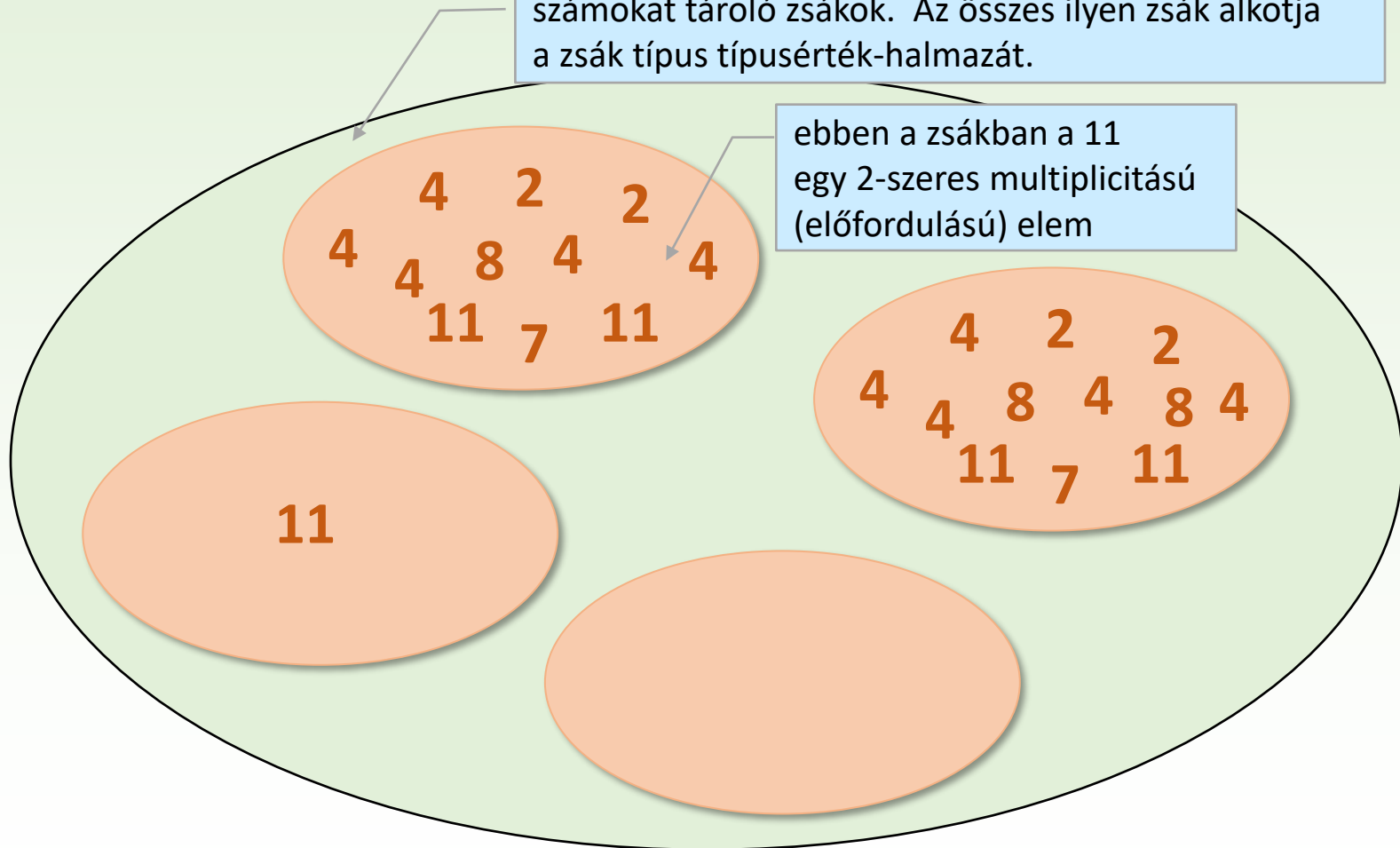
- ❑ Sorozat típus: $(E^*, \{[.], |. |, \oplus, \text{Add}(), \text{Remove}(), \dots\})$ ahol $E^* = \bigcup_{n=0.. \infty} E^n$
- ❑ Szöveg típus: $(\mathbb{S}, \{[.], |. |, \dots\})$ ahol $\mathbb{S} = \mathbb{K}^*$

általában a típusokra a
típusértékeik halmazára
bevezetett jelöléssel
szoktunk hivatkozni

Zsák típus típusérték-halmaza

Egy természetes számokat tároló zsák típusú adatnak (változónak) az értékei (típusértékei) természetes számokat tároló zsákok. Az összes ilyen zsák alkotja a zsák típus típusérték-halmazát.

ebben a zsákban a 11 egy 2-szeres multiplicitású (előfordulású) elem



Zsák típus műveletei

Kiüríti a zsákot:

$b := \emptyset$

$b:\text{Zsák}$

Betesz egy elemet a zsákba:

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

$b:\text{Zsák}, e:\mathbb{N}$

adjon hibajelzést,
ha $e \notin [0 .. m]$

Zsák leggyakoribb eleme:

$e := \text{leggyakoribb}(b)$ $b:\text{Zsák}, e:\mathbb{N}$

adjon hibajelzést,
ha a zsák üres

Zsák típus reprezentációja

b:

4 2 2
4 4 8 4 4
11 7 11

Egy zsákban $0..m$ közötti egészek lehetnek, amelyek előfordulási gyakoriságait egy $0..m$ indextartományú tömbben tároljuk.

vec:

0	1	2	3	4	5	6	7	8	9	10	11	...	<i>m</i>
0	0	2	0	5	0	0	1	1	0	0	2	...	0

 : $\mathbb{N}^{0..m}$

max:

4

 : \mathbb{N}

külön nyilvántartjuk a leggyakoribb elemet

típusinvariáns:

$\max \in [0..m] \wedge$

$\vec{vec}[\max] = \max_{i=0}^m \vec{vec}[i]$

egy típusérték reprezentálásához használt adatok közötti kapcsolat

hasznos melléktermék:

$\vec{vec}[\max]=0$ jelzi, hogy a zsák üres

Mikor jó egy reprezentáció?

Ha bármelyik típusértéket (zsákot) olyan elemek együttesével (*vec-max* párral) helyettesít, amelyek kielégítik a típusinvariánst; továbbá: a típusinvariánst kielégítő elemek együttese (*vec-max* pár) egy típusértéket (zsákot) helyettesít.

Zsák típus implementációja

hiba, ha az $e \notin [0..m]$

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

$0 \leq e \leq m$

$vec[e] := vec[e] + 1$

$vec[e] > vec[max]$

$max := e$

—

HIBA

Mikor jó az implementáció?

Ha minden típusművelethez megad egy olyan programot, amelyben a típusértékeket (zsákokat) a típusinvariánst kielégítő reprezentánsok (*vec-max* párok) helyettesítik.

típusinvariáns:

$max \in [0..m] \wedge$

$vec[max] = \max_{i=0}^m vec[i]$

$b := \emptyset$

$i = 0 .. m$

$vec[i] := 0$

$max := 0$

a típusinvariáns biztosításához a max a $0..m$ bármelyik eleme lehet, mert $\forall i \in [0..m]: vec[i] = 0$

típusinvariáns miatt kell

nem kell ellenőrizni a típusinvariánst

$e := leggyakoribb(b)$

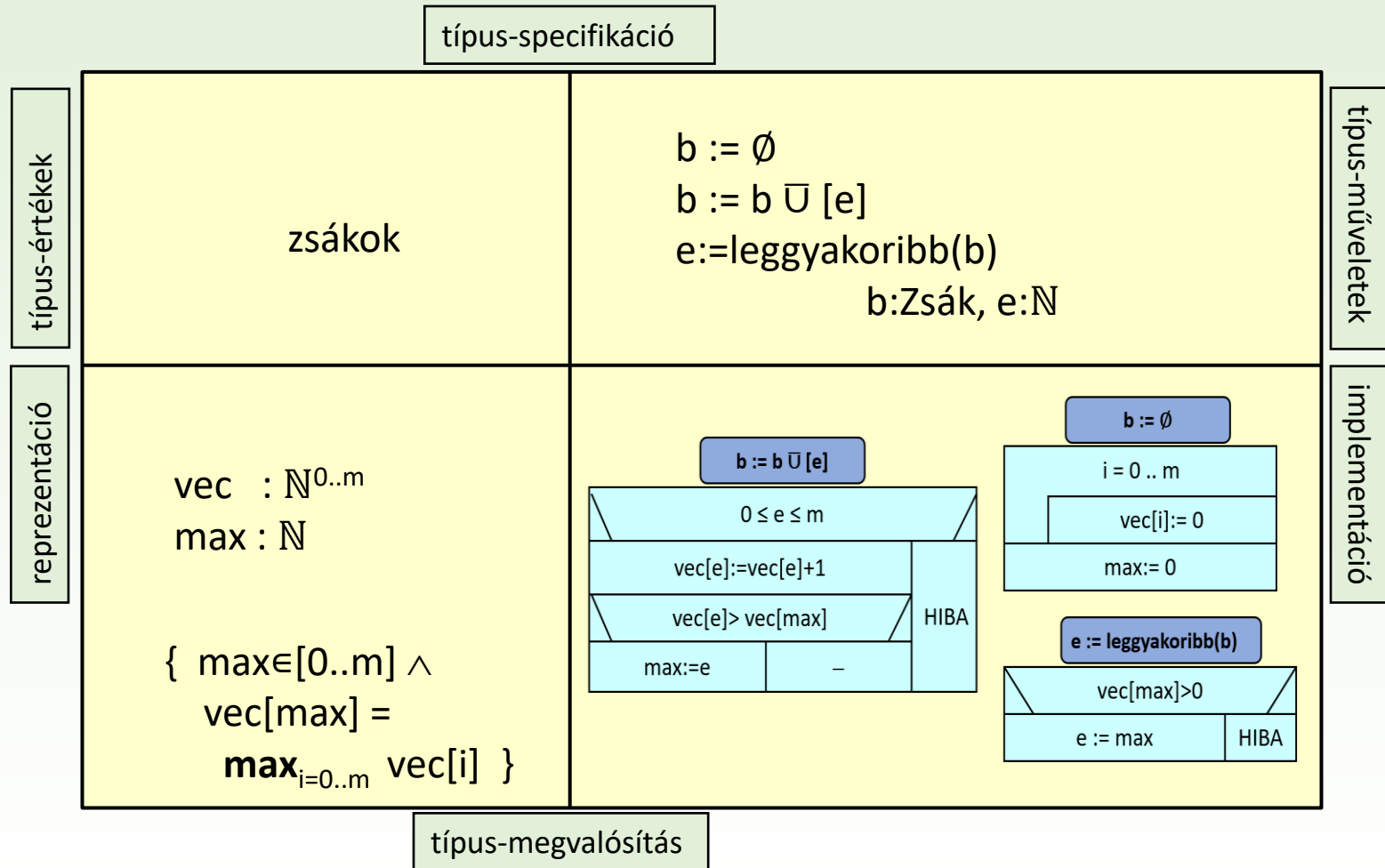
$vec[max] > 0$

$e := max$

HIBA

hiba, ha a zsák üres

Zsák típus



Típus és osztály

3.rész

Zsák osztály és a kódja

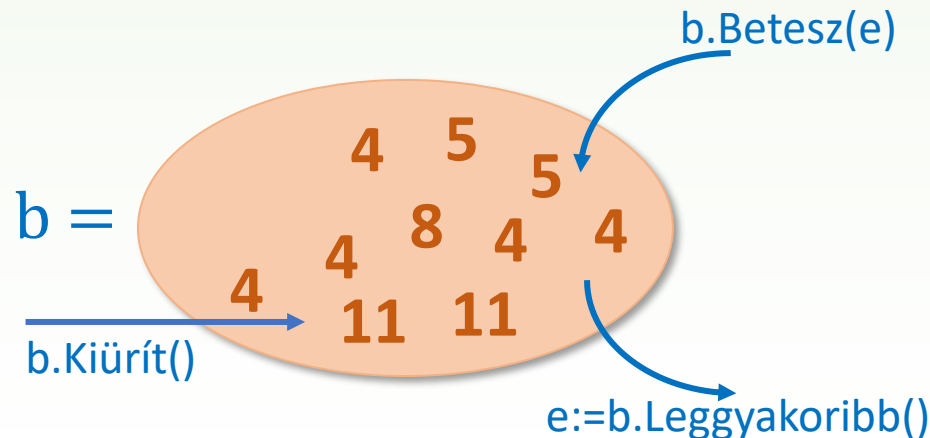
Gregorics Tibor

`gt@inf.elte.hu`

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

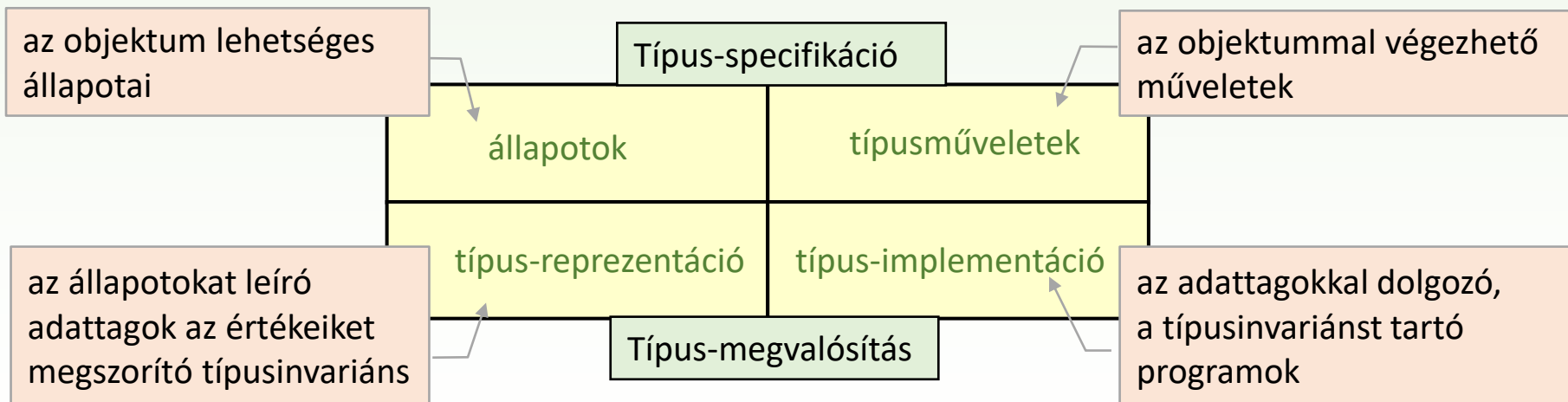
Objektum

- ❑ Objektumnak egy feladat megoldásának azon **önálló egyedként** azonosított részét nevezzük, amely magába foglalja az adott részért felelős **adatokat**, és az ezekkel kapcsolatos **műveleteket**.
- ❑ Egy objektumra – miután azt létrehoztuk (példányosítottuk) – csak közvetett módon, egy változó segítségével tudunk majd hivatkozni. Ez az **objektum-változó** annak a memória-területnek a címét tartalmazza, amely a program futása során (az objektum példányosításakor) jön létre azért, hogy ott az objektum adattagjait eltárolhassuk.



Objektum típusa

- ❑ Egy objektum típusa az azonos tulajdonságú objektumokat jellemzi úgy, hogy definiálja ezen objektumok lehetséges állapotait, és az azokkal végezhető műveleteket.
- ❑ Egy-egy állapotot az objektum adattagjainak olyan értékegyüttese reprezentál, amely kielégíti a típusinvariáns feltételt.
- ❑ A műveleteket az adattagokkal dolgozó típusinvariáns feltételt tartó programok implementálják.



Osztály

- ❑ Az osztály egy **objektum szerkezetének és viselkedésének a mintáját adja meg**, azaz
 - felsorolja az objektum **adattagjait** azok nevének, típusának, és láthatóságának (rejtett (-,#) vagy publikus (+)) megadásával, kiegészítve az esetleges típusinvariánssal
 - megadja az objektumra meghívható **metódusokat** (tagfüggvény, művelet) a nevükkel, paraméterlistájukkal, visszatérési értékük típusával, törzsükkel, és a láthatóságukkal
- ❑ Az osztály lényegében az **objektum típusa**: az objektumot az osztálya alapján hozzuk létre, azaz példányosítjuk.
- ❑ Egy osztályhoz több objektum is példányosítható: minden objektum rendelkezik az osztályleírás által leírt adattagokkal és metódusokkal.

Típus és Osztály

- Az objektumelvű tervezés során osztályként adjuk meg az egyedi, vagy ún. felhasználói típusokat (*custom type*).

Zsák típus:

típus-specifikáció

zsákok

$b := \emptyset$
 $b := b \cup [e]$
 $e := \text{leggyakoribb}(b)$
 $b: \text{Zsák}, e: \mathbb{N}$

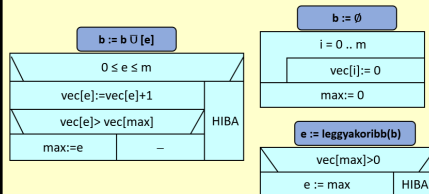
típus-műveletek

típus-értékek

reprezentáció

$\text{vec} : \mathbb{N}^{0..m}$
 $\text{max} : \mathbb{N}$
 $\{ \text{max} \in [0..m] \wedge \text{vec}[\text{max}] = \max_{i=0..m} \text{vec}[i] \}$

típus-megvalósítás



Zsák osztály:

$\text{max} \in [0..m] \wedge$
 $\text{vec}[\text{max}] = \max_{i=0..m} \text{vec}[i]$

Bag

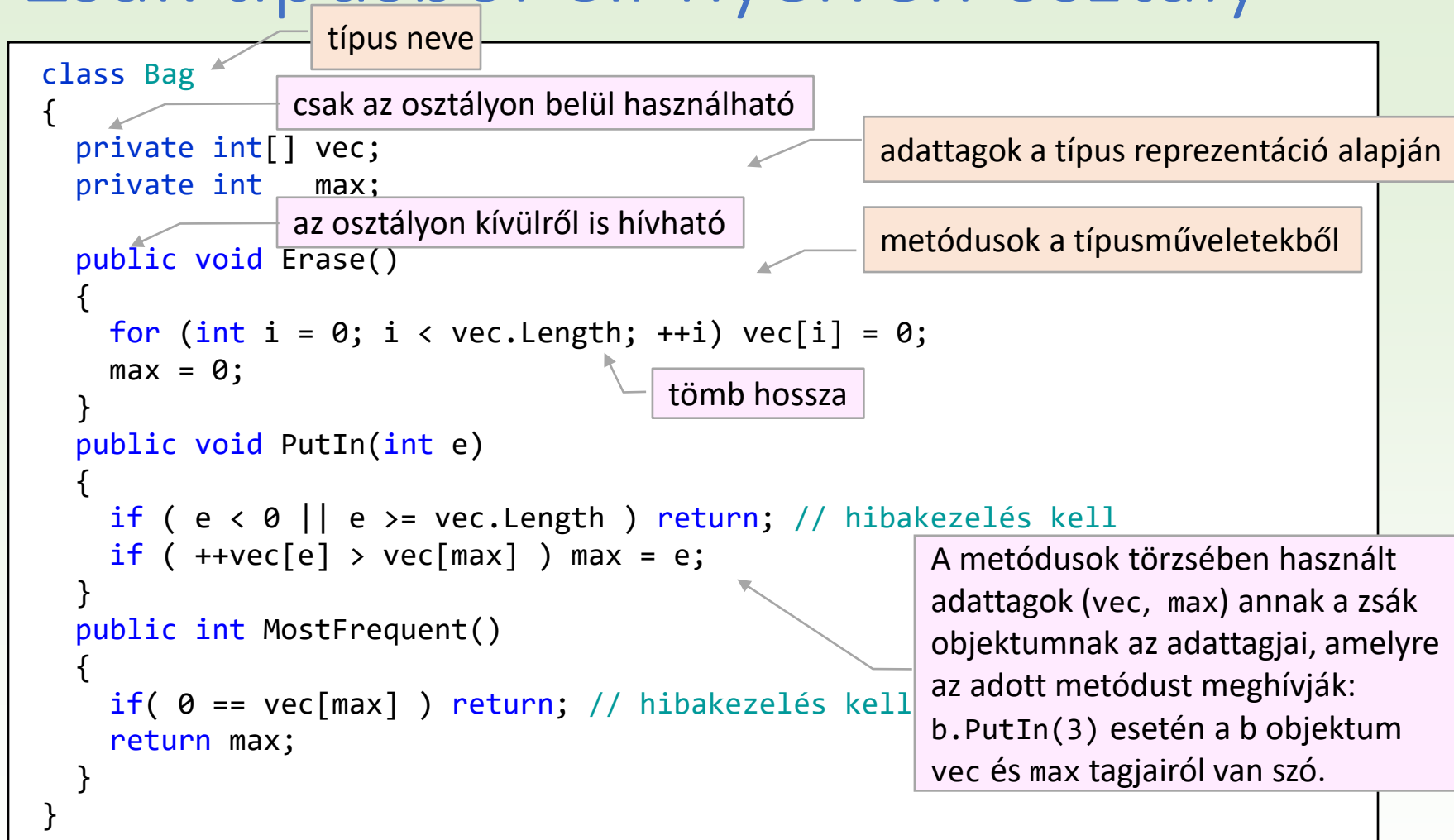
$- \text{vec} : \text{int}[0..m]$
 $- \text{max} : \text{int}$
 $+ \text{Erase}() : \text{void}$
 $+ \text{PutIn}(e:\text{int}) : \text{void}$
 $+ \text{MostFrequent}() : \text{int}$

for $i=0..m$ **loop** $\text{vec}[i] := 0$ **endloop**
 $\text{max} := 0$

if $\text{vec}[\text{max}] \leq 0$ **then error endif**
return max

if not $(0 \leq e \leq m)$ **then error endif**
 $++\text{vec}[i]$
if $\text{vec}[e] > \text{vec}[\text{max}]$ **then** $\text{max} := e$ **endif**

Zsák típusból C# nyelven osztály



1

Az objektum-orientált nyelvek lényeges ismérve az **egységbezárás**: egy adott feladatkör megvalósításához szükséges adatokat és az azokat manipuláló programrészeket a program többi részétől elkülönítve adhatjuk meg.

Konstruktor

Az **objektum példányosítását** (létrehozását) speciális metódus, a konstruktor végzi: memóriát foglal az objektum adattagjai számára (ha egy adattag maga is objektum-hivatkozás, akkor azt is példányosítja), és kezdeti értéket ad az adattagoknak.

Ha mást konstruktort nem definiálunk, akkor is rendelkezünk egy (paraméter nélküli és üres törzsű) ún. **üres konstruktorral**.

nincs visszatérési típusa
neve: az osztályának neve

```
class Bag
{
    private int[] vec;
    private int max;

    public Bag(int m)
    {
        vec = new int[m+1];
        for (int i = 0; i <= m; ++i) vec[i]=0;
        max = 0;
    }
    ...
}
```

A **Bag b = new()** utasítás az üres konstruktort hívná, ami nem nyújt lehetőséget arra, hogy megadjuk a zsákot reprezentáló tömb hosszát (az *m* értékét), és nem garantálja, hogy azt sem, hogy adattagokat az invariánsnak megfelelően inicializálja.

Készítsünk olyan konstruktort, amely megkapja a *vec* tömb hosszát (*m*) paraméterként, így lefoglalhatja annak tárhelyét.
Bag b = new(35)

Sőt, úgy kell inicializálni az új zsák adattagjait, hogy azok elégítsék ki a típus invariánst. Ehhez elég a *vec* elemeinek és a *max* értékének is nullát adni, amivel egy üres zsákot példányosítunk, mintha csak az *Erase()*-t hívnánk.

Hivatkozás egy objektum tagjaira

```
class Bag
{
    private int[] vec;
    private int max;
    public Bag(int m)        { ... }
    public void Erase()      { ... }
    public void PutIn(int e) { ... }
    public int MostFrequent() { ... }
}
```

```
Bag b1 = new(5);
Bag b2 = new(23);
b1.Erase();
b2.PutIn(5);
int a = b2.MostFrequent();
b1.max = 0;
b1.vec[5]++;
```

Amikor egy objektum egy tagját (adattagot vagy metódust) használni akarjuk, akkor az objektumot (pontosabban az arra hivatkozó változót) a használni kívánt tag elé kell írni. A metódus elé írt objektum a metódus egy kitüntetett extra paramétere lesz.

Egy objektum rejtett (privát, védett) tagjaira csak az objektum metódusainak törzsében hivatkozhatunk, máshol ezeket közvetlenül nem használhatjuk.

2

Az objektum orientált nyelvek fontos ismérve az **elrejtés**: az egységbe zárt elemek láthatóságának korlátozása. (Általában az adattagok rejtettek, azok értékéhez csak közvetetten, a publikus metódusokkal férünk hozzá.)

Típus és osztály

4.rész

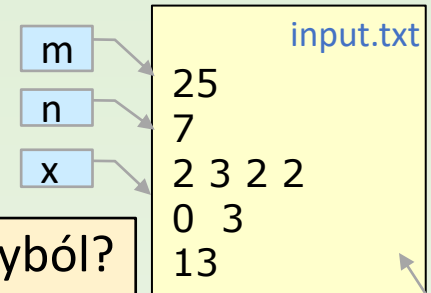
Megvalósítás

Gregorics Tibor

gt@inf.elte.hu

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

Bemenő adatok beolvasása



Hogyan olvassunk be egész számokat egy szöveges állományból?

1. A honlapról letöltött TextFile projekt lefordított kódja: `TextFile.dll`
2. Helyezzük el a dll-t a forrás fájlok közé
Build Action: `Content`
3. Add/Project Reference : `TextFile.dll`
4. A programkód elejére : `using TextFile`

1. Helyezzük az `input.txt` fájlt a forrásfájlok között (létrehozhatjuk a VS-sel is)
2. A Properties ablakban állítsuk be a fájlra:
Copy to Output Directory: `Copy if newer`
Build Action: `Content`

```
TextFileReader reader = new ("input.txt");
```

```
reader.ReadInt(out int m);
```

```
reader.ReadInt(out int n);
```

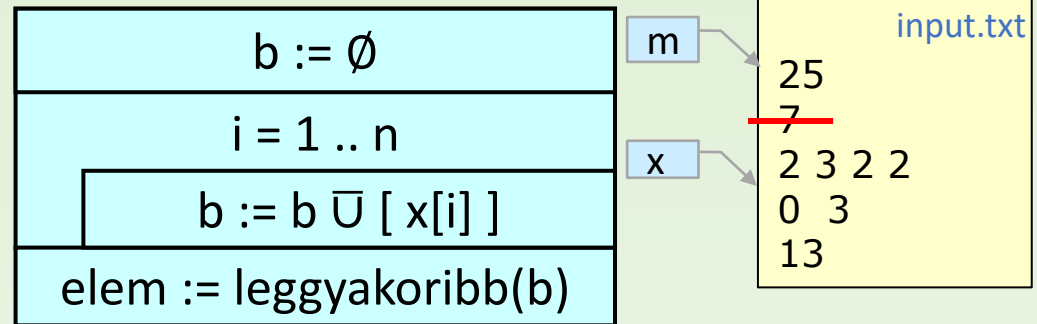
```
int[] x = new int[n];
```

```
for ( int i = 0; i < n; ++i )  
{  
    reader.ReadInt(out x[i]);  
}
```

inline változó deklaráció

A reader objektum adatcsatornát nyit a szöveges állomány és az alkalmazás között. Először példányosítjuk, majd meghívjuk rá többször is a `ReadInt()` metódusát, amely következő egész számot olvassa be; ha sikerül, igaz értéket ad vissza, különben hamisat

Megvalósítás



```
TextFileReader reader = new ("input.txt");
```

```
reader.ReadInt(out int m);
```

```
reader.ReadInt(out int n);
int[] x = new int[n];
for ( int i = 0; i < n; ++i )
{
    reader.ReadInt(out x[i]);
}
```

```
Bag bag = new(m);
for( int i = 0; i < x.Count; ++i)
{
    bag.PutIn(x[i]);
}
```

```
Console.WriteLine($"Most frequent element: { bag.MostFrequent() }");
```

Közvetlenül is elhelyezhetjük a fájl számait a zsákban az n elemű x tömb kiiktatásával. Ekkor az inputfájlban nem kell megadni az n értékét sem: fájl végéig olvasunk.

```
Bag bag = new Bag(m);
while ( reader.ReadInt(out int e) )
{
    bag.PutIn(e);
}
```

C# megoldás szerkezete

solution: Frequency
project: Frequency
Program.cs
Bag.cs
TextFile.dll

namespace Frequency

```
class Program
{
    static void Main()
    {
        TextFileReader reader = new ...
        reader.ReadInt(out int m);
        Bag b = new (m);
        ...
    }
}
```

osztályszintű metódus,
amelyik hívásához nem
kell objektum

Program.cs

```
class Bag
{
    private int[] vec;
    private int max;

    public Bag(int m){...}
    public void Erase(){...}
    public void PutIn(int e){...}
    public int MostFrequent(){...}
}
```

Bag.cs

namespace TextFile

```
public class TextFileReader
{
    public bool ReadInt(out int n);
    ...
}
```

publikus, hogy másik
névtérben látható legyen

namespace System

...

.dll

Főprogram

kivétel-kezelés: ha egy utasítás (pl. metódus hívás) valamilyen hibát észlel, akkor dobjon egy kivételt; ettől a program futása megszakad, de ha ez egy try blokkban történik, akkor lehetőségünk van a blokk után elhelyezett catch ágakban – ahová ilyenkor átkerül a vezérlés – reagálni a kivételt kiváltó okra.

```
using System;
using TextFile;

namespace Frequency
```

```
{
    class Program
```

```
{
    static void Main()
```

```
{
    try
```

kivételek figyelése

```
{
    TextFileReader reader = new ("input.txt");
    reader.ReadInt(out int m);
```

kivételt dob, ha nem találja a textfájlt

```
    Bag bag = new (m);
```

kivételt dob, ha m negatív

```
    while ( reader.ReadInt(out int e) )
```

```
{
    try { bag.PutIn(e); } catch( ... ) { ... }
```

kivételt dob, ha e nem esik 0 és m közé

```
}
```

```
    Console.WriteLine($"Most frequent element: {bag.MostFrequent()}");
```

```
}
```

kivételt dob, ha b üres

```
catch( ... ) { ... }
```

kivételek elkapása és lekezelése

Program.cs

Kivétel definiálása

```
using System;
```

```
namespace Frequency
```

```
{
```

```
    class Bag
```

```
    {
```

```
        public class NegativeSizeException : Exception { }
```

```
        public class EmptyBagException : Exception { }
```

```
        public class IllegalElementException : Exception { }
```

```
        private int[] vec;
```

```
        private int max;
```

```
        public Bag(int m) { ... }
```

```
        public void Erase() { ... }
```

```
        public void PutIn(int e) { ... }
```

```
        public int MostFrequent() { ... }
```

```
    }
```

```
}
```

egy zsáktípusú objektum működése
esetén előforduló hibák

az előforduló hiba eseteket „kivételként”
származtatással definiáljuk

Bag.cs

Kivétel dobása

a hiba észlelése különbözik a hiba kezelésétől

```
public Bag(int m)
{
    if (m < 0) throw new NegativeSizeException();
    vec = new int[m+1];
    for (int i = 0; i <= m; ++i) vec[i]=0;
    max = 0;
}
public void Erase()
{
    for (int i = 0; i < vec.Length; ++i) vec[i] = 0;
    max = 0;
}
public void PutIn(int e)
{
    if ( e<0 || e>=vec.Length ) throw new IllegalElementException();
    if ( ++vec[e] > vec[max] ) max = e;
}
public int MostFrequent()
{
    if( 0 == vec[max] ) throw new EmptyBagException();
    return max;
}
```

kivételt dob, ha m értéke negatív, és az objektum példányosítása megszakad

kivételt dob, ha a paraméter értéke nincs 0 és m között

kivételt dob, ha b üres

Bag.cs

Kivétel kezelése

```
try
{
    ...
    while ( reader.ReadInt(out int e) )
    {
        try { bag.PutIn(e); }
        catch (Bag.IllegalElementException)
        {
            Console.WriteLine($"The element of the bag must be in [0..{m}].");
        }
    }
    ...
}
catch (Bag.NegativeSizeException)
{
    Console.WriteLine("Upper limit of elements must be natural.");
}
catch (Bag.EmptyBagException)
{
    Console.WriteLine("There is no most frequented element.");
}
catch (System.IO.FileNotFoundException)
{
    Console.WriteLine("Input file does not exist.");
}
}
```

beágyazott try blokk:
a hiba lekezelése után folytatódik
a blokkot tartalmazó ciklus

ha van a try blokkban észlelt kivételhez
illeszkedő catch ág, akkor ide kerül át
a program vezérlése

Program.cs