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 ezen 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 ezen objektumok metódusai közötti vezérlés-átadások (közvetlen hívások vagy szignálok küldései) 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.

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.

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

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

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

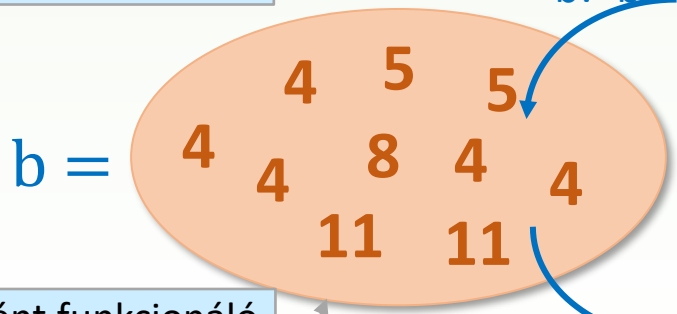
b egy zsák objektumra hivatkozó segédváltozó

lekérjük a zsáktól a leggyakoribb elemét

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

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

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



tárolóként funkcionáló zsák objektum

e:=leggyakoribb(b)

Tervezés

A = (x:$\mathbb{N}^n$, m:$\mathbb{N}$, b:Zsák, 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 = ($Ef \wedge b = \bar{\cup}_{i=1..n}[x[i]] \wedge$ elem = leggyakoribb(b))

Összegzés algoritmus minta
$s = \Sigma_{i=m..n} f(i)$
$f:[m..n] \longrightarrow H$
$+:H \times H \longrightarrow H$
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:

$$s = \Sigma_{i=m..n} f(i) \sim b = \bar{\cup}_{i=1..n} [x[i]]$$

eltérés:

eredmény:	s : H	~	b : Zsák
művelet:	H, +, 0	~	Zsák, $\bar{\cup}$, Ø
elem:	f(i)	~	[x[i]]
felsorolás:	i = m .. n	~	i = 1 .. n

```
b := Ø
i = 1 .. n
    b := b ∪̄ [x[i]]
elem := leggyakoribb(b)
```

Megvalósítás előkészítése

Hogyan olvassunk be egész számokat egy szöveges állományból?

input.txt
2 25 13 0 2
0 35 13 2

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. Legyen input.txt fájl 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

olvasó objektum, amely adatcsatornát nyit a szöveges állomány és az alkalmazás között

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

 while ( reader.ReadInt(out int e) )
 {
    … e …
 }
```

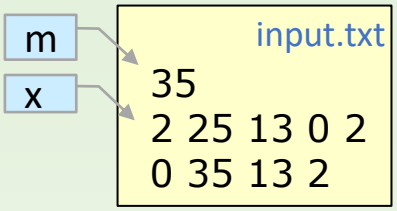
objektum példányosítás

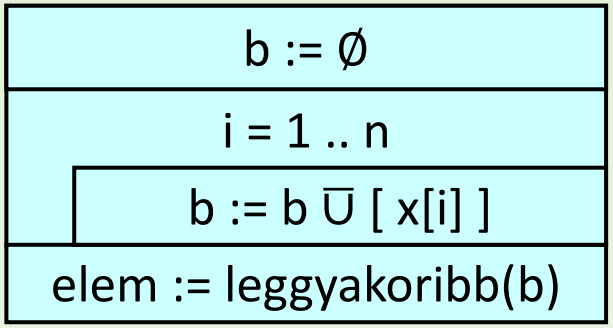
inline változó deklaráció

A következő egész számot olvassa be; ha sikerül, igaz értéket ad vissza, különben hamisat

objektum-orientált metódushívás

Megvalósítás






```
using System;
using TextFile;
...

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

reader.ReadInt(out int m);

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

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

ezek hiányában később a System.Console.WriteLine(…), vagy a TextFile.TextFileReader hivatkozást kellene használnunk

az első egész számot olvassa a fájlból

b:Zsák

b := Ø

Beolvassa a fájlból az egész számokat, és azokat közvetlenül a zsákban helyezi el, kiiktatva a megoldásból az n elemű x tömböt.

a megvalósítás során módosulhat a terv

b := b ∪ [e]

leggyakoribb(b)

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

2.rész
Zsák típus

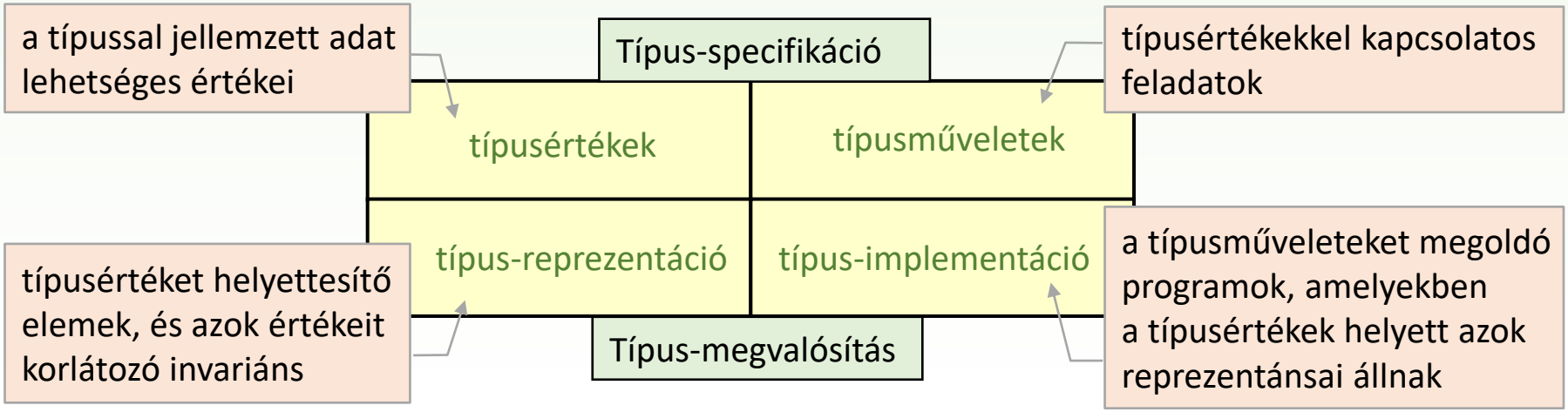
Gregorics Tibor

gt@inf.elte.hu

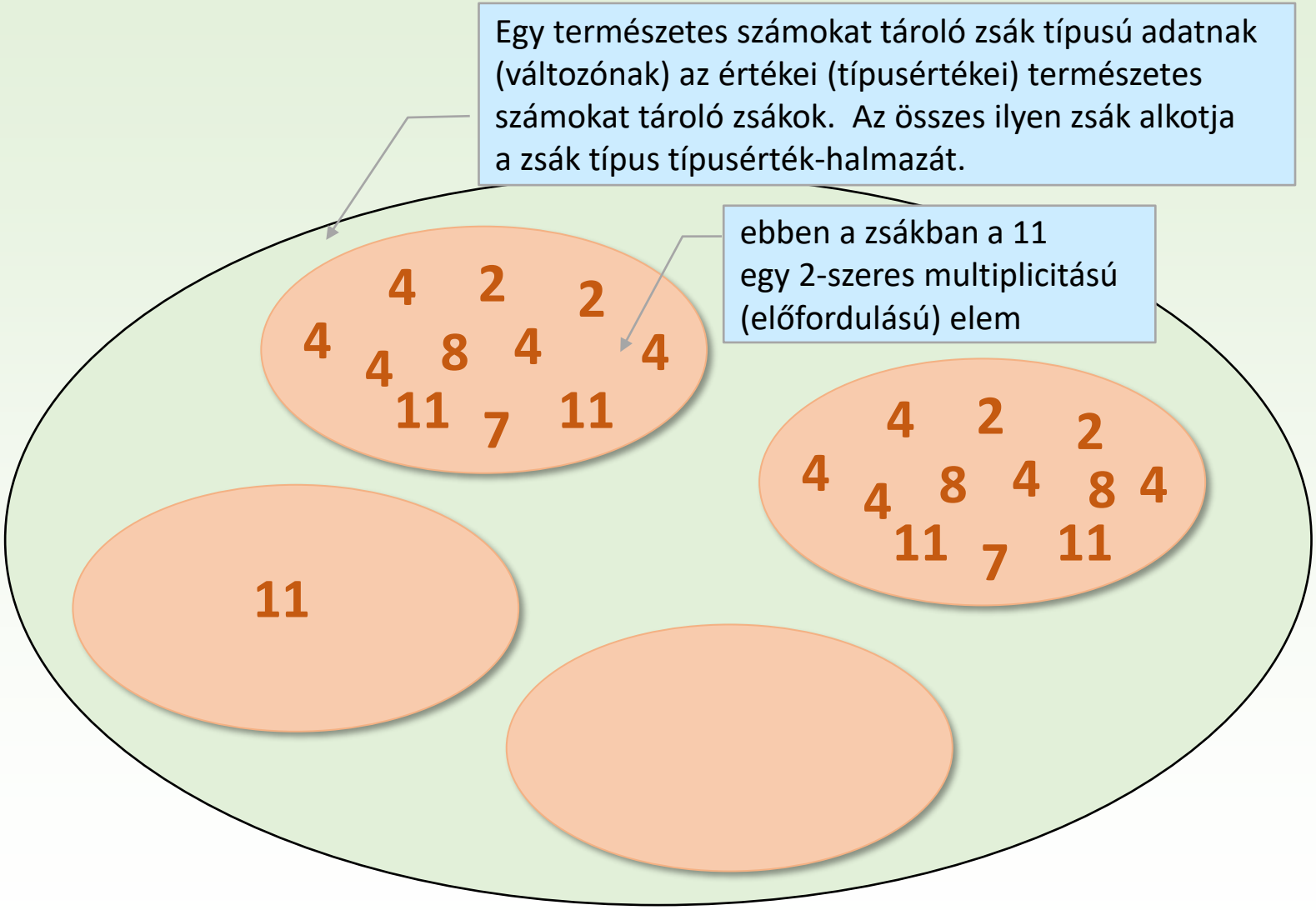
http://people.inf.elte.hu/gt/oep

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



Zsák típus típusérték-halmaza



Zsák típus műveletei

Kiüríti a zsákot:

b := ∅ b:Zsák

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

Betesz egy elemet a zsákba:

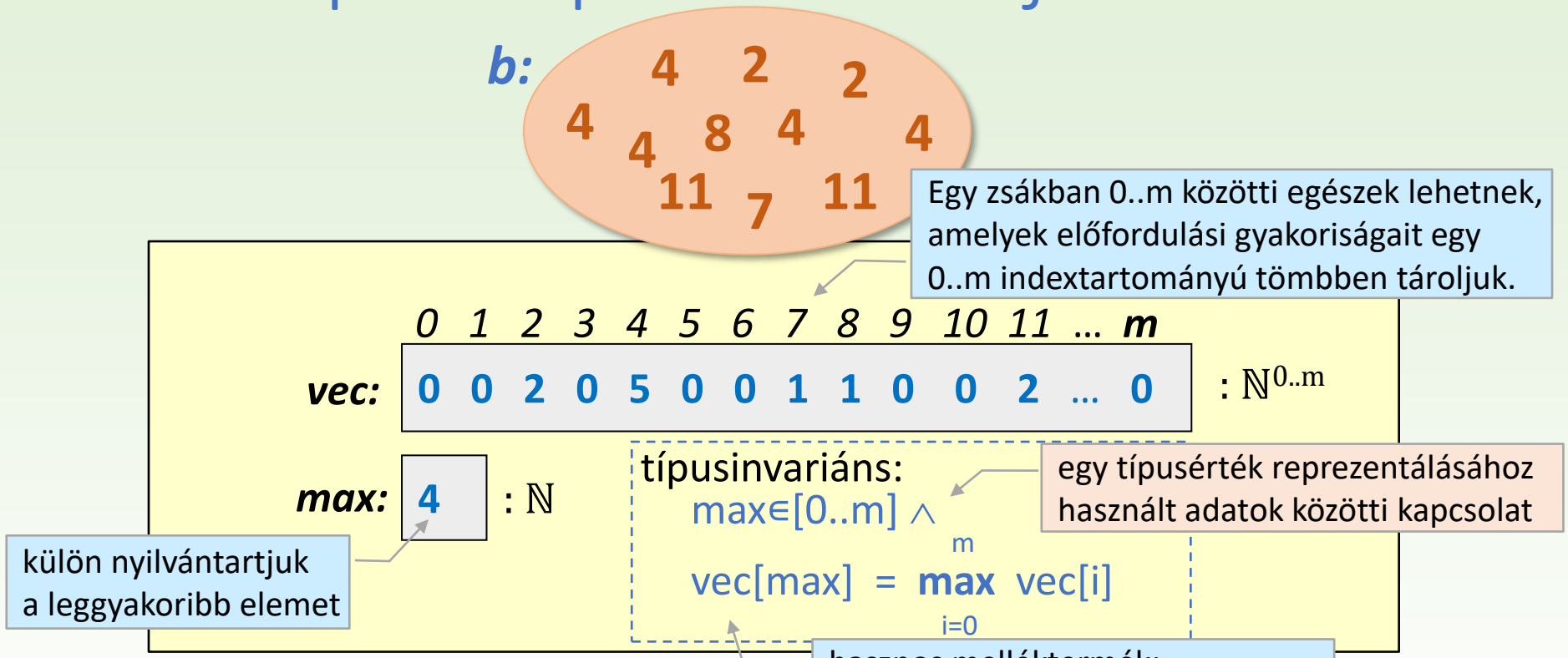
b := b ⊍ [e] b:Zsák , e:ℕ

adjon hibajelzést, ha a zsák üres

Zsák leggyakoribb eleme:

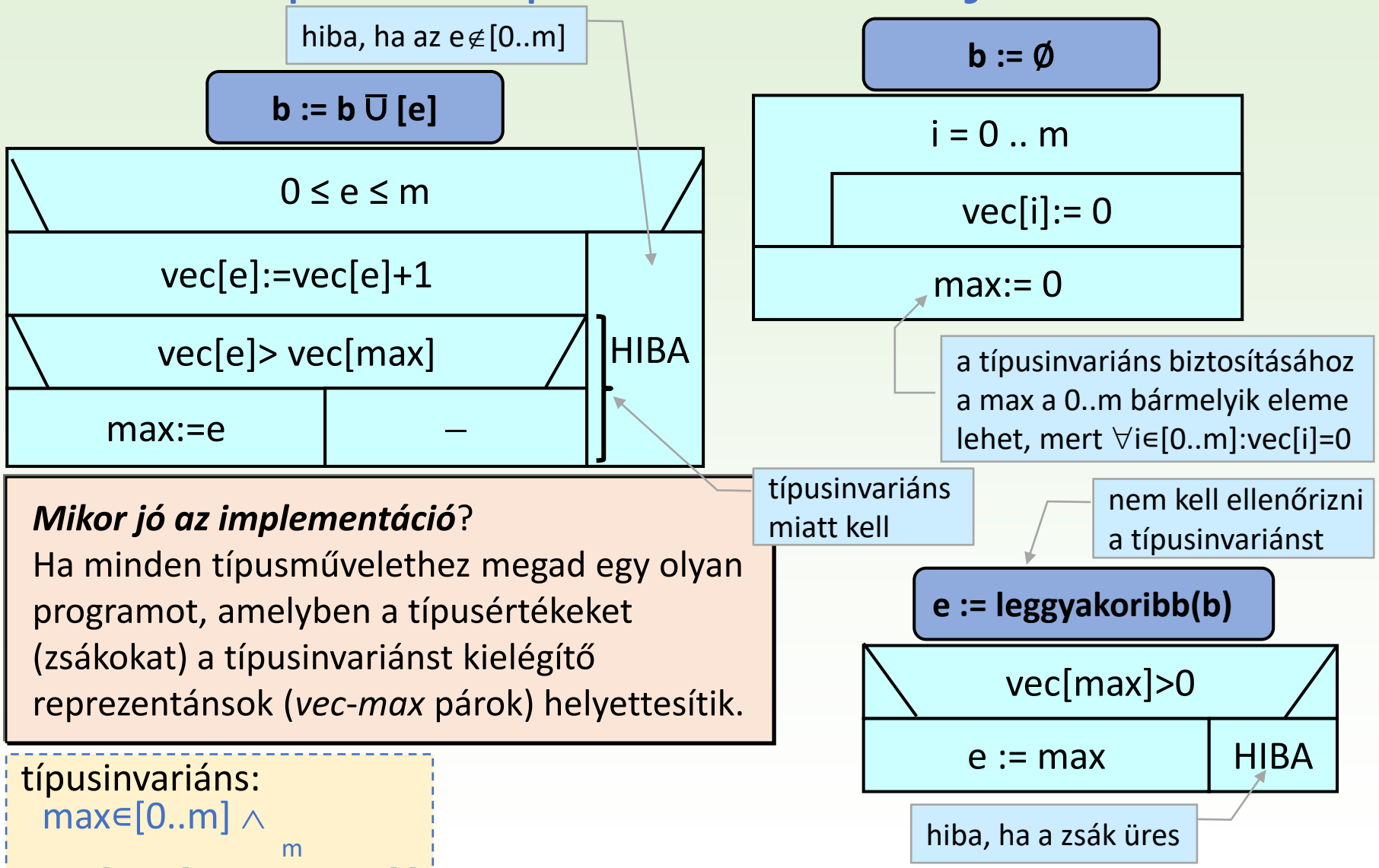
e:=leggyakoribb(b) b:Zsák , e:ℕ

Zsák típus reprezentációja



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



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

típusinvariáns:

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

Zsák típus

típus-specifikáció

típus-értékek

zsákok

típus-műveletek

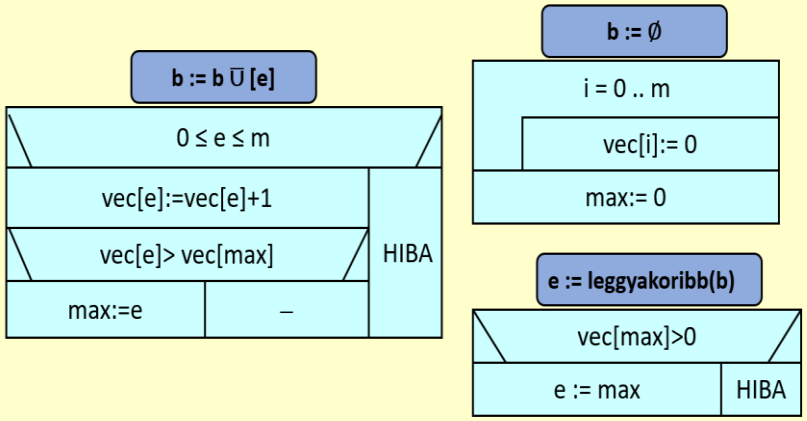
b := ∅
b := b ⊍ [e]
e:=leggyakoribb(b)
b:Zsák, e:ℕ

reprezentáció

vec : $\mathbb{N}^{0..m}$
max : ℕ

{ max∈[0..m] ∧
vec[max] =
max$_{i=0..m}$ vec[i] }

implementáció



típus-megvalósítás

3.rész
Zsák osztály

Gregorics Tibor

gt@inf.elte.hu

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

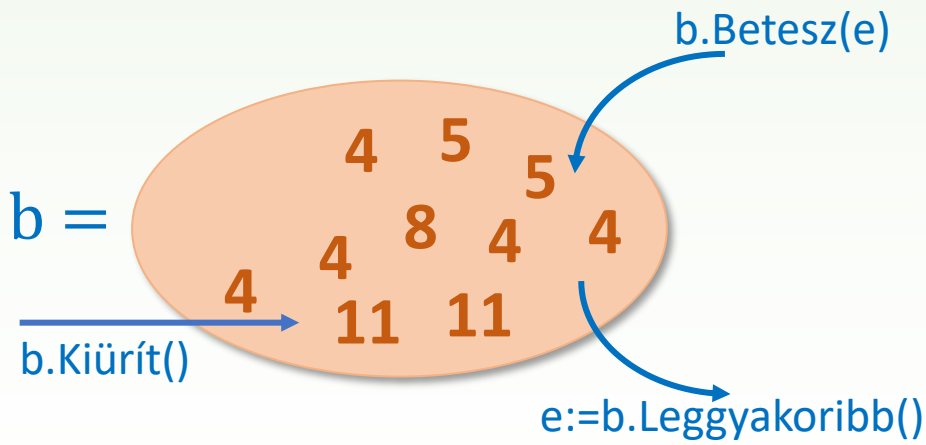
Objektum

például a b zsák

- Objektumnak egy feladat megoldásának olyan önálló egyedként azonosított részét nevezzük, amely a megoldás adott részéért felelős adatokat, és az ezekkel kapcsolatos műveleteket foglalja magába.

vec, max

kiürít, betesz, leggyakoribb



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.

Zsák típusból C# nyelven osztály

típus neve

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

  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 ) return; // hibakezelés kell
    if ( ++vec[e] > vec[max] ) max = e;
  }
  public int MostFrequent()
  {
    if( 0 == vec[max] ) return; // hibakezelés kell
    return max;
  }
}
```

csak az osztályon belül használható

adattagok a típus reprezentáció alapján

az osztályon kívülről is hívható

metódusok a típusműveletekből

tömb hossza

A metódusok törzsében használt adattagok (`vec`, `max`) annak a zsák objektumnak az adattagjai, amelyre az adott metódust meghívják: `b.PutIn(3)` esetén a b objektum `vec` és `max` tagjairól van szó.

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, amely memóriát foglal az objektum, azaz az adattagjai számára (lefutnak az adattagok konstruktorai is), amelyek kezdeti értéket is kapnak.

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 Bag()` utasítás az üres konstruktort hívná, de ezzel nem tudnánk beállítani a zsák elemeinek maximumát (ez a `vec` hosszának megadásához kell), és nem inicializálná az adattagokat az invariánsnak megfelelően.

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

objektumok példányosítása rövidíthető: `Bag b1 = new(5)`

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

Amikor egy objektum egy tagjával (adattaggal vagy metódussal) műveletet akarunk végezni, akkor az objektumot (pontosabban az arra hivatkozó változót) a tag elé kell írni.
A metódus elé írt objektum egyben a metódus egy kitüntetett extra paramétere.

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.)

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);
     ...
  }
}
```

Program.cs

osztályszintű metódus, amelyik hívásához nem kell objektum

```
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

.dll

```
namespace TextFile
```

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

publikus, hogy másik névtérben látható legyen

```
namespace System
```

```
...
```

4.rész
Hibakezelés, tesztelés

Gregorics Tibor

gt@inf.elte.hu

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

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. Ez a technika elválasztja a hiba észlelését a hiba lekezelésétől.

```
using System;
using TextFile;

namespace Frequency
{
  class Program
  {
    static void Main()
    {
      try
      {
        TextFileReader reader = new ("input.txt");
        reader.ReadInt(out int m);

        Bag bag = new (m);
        while ( reader.ReadInt(out int e) )
        {
          try {  bag.PutIn(e); } catch( … ) { … }
        }
        Console.WriteLine($"Most frequent element: {bag.MostFrequent()}");
      }
      catch( … ) { … }
    }
  }
}
```

kivételek figyelése

kivételt dob, ha nem találja a textfájlt

kivételt dob, ha m negatív

kivételt dob, ha e nem esik 0 és m közé

kivételt dob, ha b üres

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

```
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

Tesztkörnyezet

solution: Frequency
project: Frequency
project: BagTest

- Olyan tesztelő projektet (BagTest) készítünk .NET-ben, amelyik a feladatot megoldó projekt Bag osztályának metódusait ellenőrzi (unit test).

a BagTest projektnek el kell tudni érni a Frequency projektet ehhez: "Add/Project Reference: Frequency"

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Frequency;

namespace BagTest
{
  [TestClass]
  public class BagTest
  {
    [TestMethod]
    public void TestPutIn_NewElement() { ... }

    [TestMethod]
    public void TestPutIn_ExistingElement() { ... }
      ...
  }
}
```

Jelzi a Test Explorer-nek, hogy ennek az osztálynak egy objektumát létre kell hozni a teszteléshez.

Jelzi a Test Explorer-nek, hogy le kell futtatni ezt a metódust a tesztelés során.

BagTest.cs

Egység tesztek

```
[TestMethod]
public void TestPutIn_PutInNewElement()
{
  Bag bag = new (2);
  bag.PutIn(1);
  Assert.AreEqual( bag.MostFrequent() , 1 );
}

[TestMethod]
public void TestPutIn_ExistingElement()
{
  Bag bag = new (2);
  bag.PutIn(1);
  Assert.AreEqual( bag.MostFrequent() , 1 );
  bag.PutIn(2);
  bag.PutIn(2);
  Assert.AreEqual( bag.MostFrequent() , 2 );
}
```

összeveti a számított és a várt értéket

BagTest.cs

Kivételek tesztelése

```
[TestMethod]
public void TestBag_NegativeParam()
{
  Assert.ThrowsException<Bag.NegativeSizeException>( ()=>new Bag(-2) );
}

[TestMethod]
public void TestPutIn_IllegalElement()
{
  Bag bag = new (2);
  Assert.ThrowsException<Bag.IllegalElementException>( ()=>bag.PutIn(3) );
}

[TestMethod]
public void TestMostFrequent_EmptyBag()
{
  Bag bag = new (2);
  Assert.ThrowsException<Bag.EmptyBagException>( ()=>bag.MostFrequent() );
}
```

kivételek keletkezése is tesztelhető

paraméter nélküli lambda kifejezés

BagTest.cs