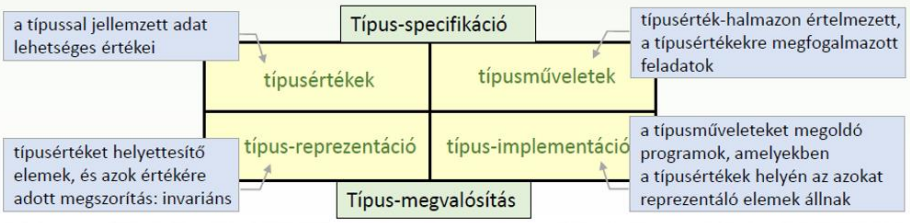
OEP 2. gyakorlat

Adattípus fogalma (1. előadás)

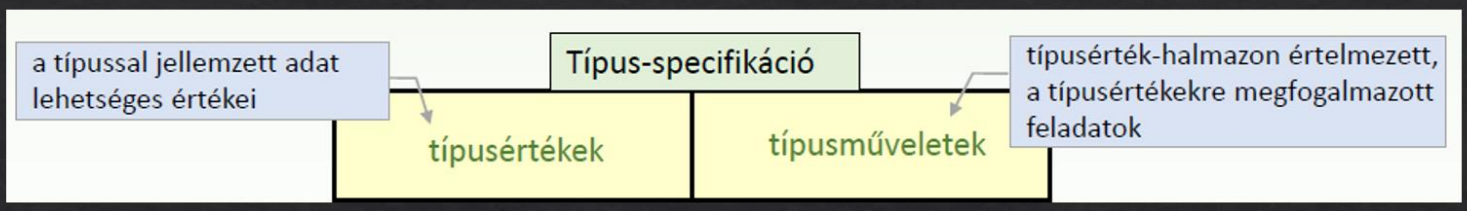
- Típus-specifikáció
  - Típusértékek
  - Típusműveletek
- Típus megvalósítás
  - Típus-reprezentáció
  - Típus-implementáció

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
  - a típusértékekkel végezhető műveleteket
- A típus-megvalósítás megmutatja:
  - hogyan ábrázoljuk (reprezentáljuk) a típus értékeit
  - milyen programok helyettesítsék (implementálják) a műveleteket

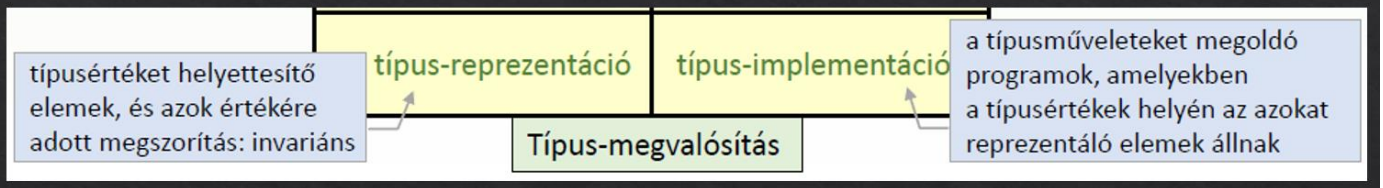


2. feladat: racionális számok típusa specifikáció



$\mathbb{Q}$	• összeadás/kivonás $(a: \mathbb{Q}, b: \mathbb{Q}, c: \mathbb{Q})$ $c := a \pm b$ • szorzás/osztás $(a: \mathbb{Q}, b: \mathbb{Q}, c: \mathbb{Q})$ $c := a * b$ $c := \frac{a}{b}$

2. feladat: racionális számok típusa
típus reprezentáció



- Ötlet: ábrázoljuk két egész szám hányadosával
  $\frac{x}{y}$, x,y: $\mathbb{Z}$
- Nullával nem lehet osztani $\Rightarrow$ y$\neq$ 0
  (típus invariáns tulajdonság)

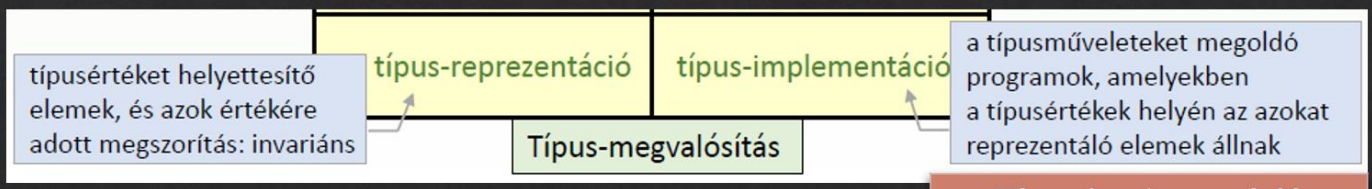
Típus-reprezentáció

$x, y : \mathbb{Z}$

$//\frac{x}{y}$

Invariáns: $y \neq 0$

2. feladat: racionális számok típusa
típus implementáció



- Összeadás/kivonás:

$$\frac{a.x}{a.y} \pm \frac{b.x}{b.y} = \frac{a.x * b.y \pm a.y * b.x}{a.y * b.y}$$

- Szorzás:

$$\frac{a.x}{a.y} * \frac{b.x}{b.y} = \frac{a.x * b.x}{a.y * b.y}$$

- Osztás: (b.x$\neq$0):

$$\frac{\frac{a.x}{a.y}}{\frac{b.x}{b.y}} = \frac{a.x}{a.y} * \frac{b.y}{b.x} = \frac{a.x * b.y}{a.y * b.x}$$

Típus-implementáció

Összeadás/kivonás

$c.x, c.y$
$:= (a.x * b.y \pm a.y$
$* b.x), a.y * b.y$

Szorzás

$c.x, c.y$
$:= a.x * b.x, a.y * b.y$

Osztás (b.x $\neq$ 0)

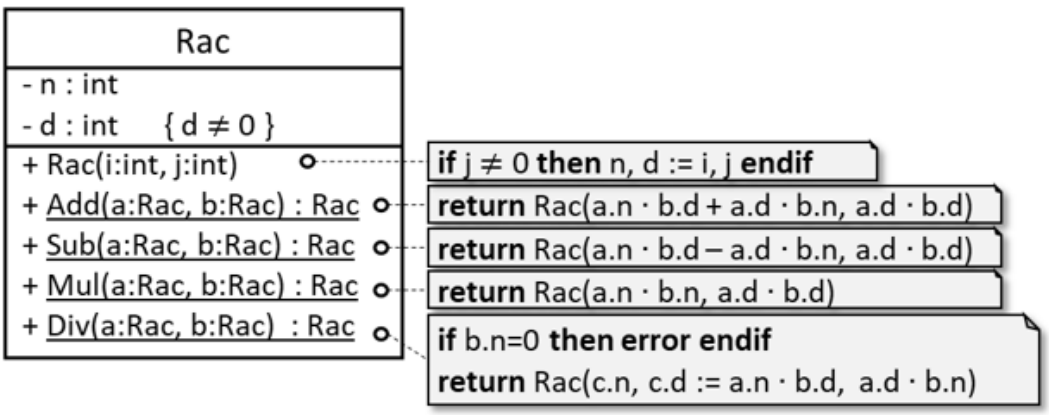
$c.x, c.y$
$:= a.x * b.y, a.y * b.x$

Racionális szám típus implementálás, UML ábra

$\mathbb{Q}$	c := a±b (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
	c := a*b (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
	c := a/b (b≠0) (a:$\mathbb{Q}$, b:$\mathbb{Q}$, c:$\mathbb{Q}$)
x,y: $\mathbb{Z}$ (Inv: y≠0) // $\frac{x}{y}$	c.x, c.y := a.x*b.y ± a.y*b.x, a.y*b.y
	c.x, c.y := a.x * b.x, a.y * b.y
	c.x, c.y := a.x * b.y, a.y * b.x (b.x≠0)

Osztálydiagram:

n := x, d := y, a.n := a.x, b.n := b.x, a.d := a.y, b.d := b.y



Spirál füzet típusa.

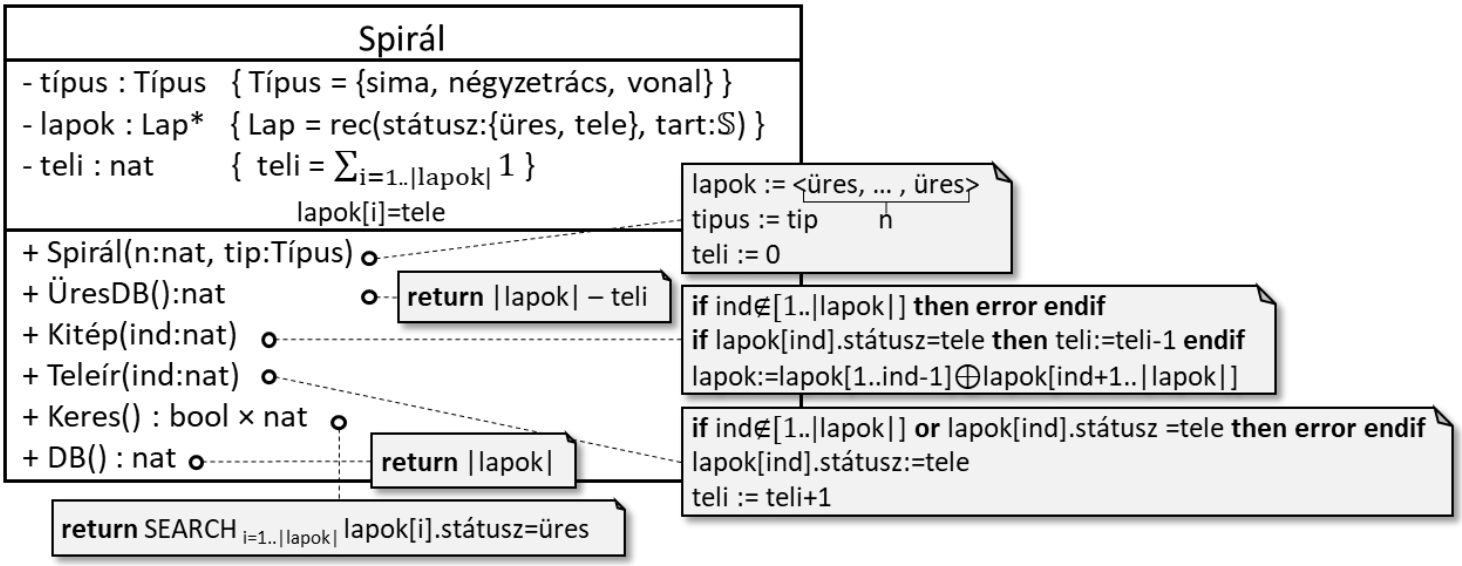
- Egy füzet azonos típusú (vagy négyzetrácsos, vagy sima, vagy vonalas) lapokból áll.
- Bizonyos lapjai már tele vannak írva, a többi még üres.
- Le lehet kérdezni, hogy hány üres lap van még a füzetben.
- Ki lehet tépni valahányadik lapot (üreset is, teleírtat is).
- Tele lehet írni egy kiválasztott lapot, ha az üres.
- Megkereshetjük az első üres lap sorszámát.

Típusdefiníció: **Spirál**

<table>
<tr><td rowspan="5">spirálfüzetek</td><td>a) db := ÜresDB(s) (s:Spirál, db:$\mathbb{N}$)</td></tr>
<tr><td>b) s := Kitép(s, ind) (s:Spirál, ind:$\mathbb{N}$)</td></tr>
<tr><td>c) s := Teleír(s, ind) (s:Spirál, ind:$\mathbb{N}$)</td></tr>
<tr><td>d) l, ind := KeresÜres(s) (s:Spirál, l:$\mathbb{L}$, ind:$\mathbb{N}$)</td></tr>
<tr><td>e) db := DB(s) (s:Spirál, db:$\mathbb{N}$)</td></tr>
<tr><td rowspan="5">típus : {sima, négyzetrács, vonal}
lapok : Lap*
teli : $\mathbb{L}$ // teli = $\sum_{i=1..|lapok|,\ lapok[i]=tele} 1$

Lap = rec(státusz : {üres, tele},
tart : $\mathbb{S}$)</td><td>a) db := |lapok| – teli</td></tr>
<tr><td>b) if ind$\notin$[1 .. |lapok|] then error endif
if lapok[ind].státusz=tele then teli:=teli-1 endif
lapok:=lapok[1..ind-1]$\oplus$lapok[ind+1..|lapok|]</td></tr>
<tr><td>c) if ind$\notin$[1 .. lapok]
or lapok[ind].státusz = tele then error endif
lapok[ind].státusz:=tele
teli := teli+1</td></tr>
<tr><td>d) l, ind := SEARCH$_{i=1..|lapok|}$ (lapok[i].státusz = üres)</td></tr>
<tr><td>e) db := |lapok|</td></tr>
</table>

Osztály:



Az üres lapot kereső művelet itteni megadása tipikus példája a végrehajtható specifikáció alkalmazásának.

Ez a specifikáció egyértelműen visszavezethető a lineáris keresés algoritmus mintára, így annak algoritmusával megoldható.

Labirintus

- Egy nxm-es négyzetrács alaprajzú labirintus i. sorának j. mezője vagy egy fal, vagy üres hely, vagy kincset tartalmaz, vagy szellem van ott.

- Kérdezhessük le a labirintus i. sorának j. mezőjéről, hogy megadott irányban (fel, le, jobbra, balra) tovább lépve falba ütközünk-e, szellemmel találkozunk-e, kincset tálálunk-e.

- Definiáljuk a kincs begyűjtés műveletét is: ez a labirintus i. sorának j. mezőjéről törli a kincset, így az üres lesz.

Típusdefiníció: **Labirintus**

labirintusok	a) k := MiVanOtt(a, x, y, ir) (a:Labirintus, x, y:ℕ, ir:Irány, k:Tartalom) Irány = { fel, le, jobb, bal } Tartalom = { üres, fal, kincs, szellem }
	b) a := Begyűjt(a, x, y) (a:Labirintus, x, y:ℕ, k:Tartalom)
térkép : Tartalom$^{n \times m}$ Tartalom = { üres, fal, kincs, szellem }	a) **switch** (ir) **case** fel: **if** (x=1) **then error endif**; x := x-1; **case** le: **if** (x=n) **then error endif**; x := x+1; **case** bal: **if** (y=1) **then error endif**; y := y-1; **case** jobb: **if** (y=m) **then error endif**; y := y+1; **endswitch** k := térkép[x, y]
	b) **if** térkép[x, y] ≠ kincs **then error endif** térkép[x, y] := üres

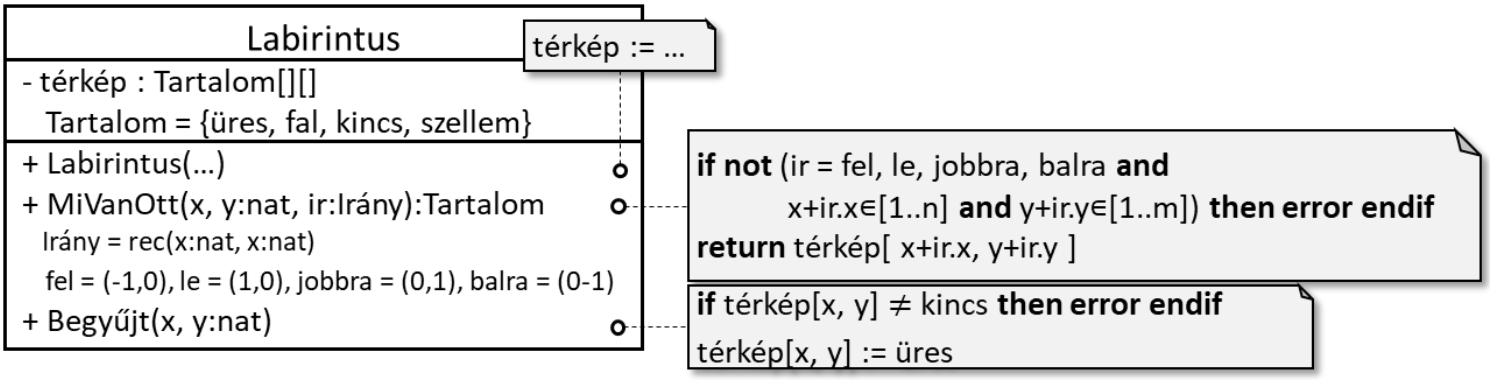
A műveletek gyakran tartalmaznak hibaellenőrzéseket.

A MiVanOtt() programja egyszerűbb lesz, ha az irányokat koordináta párokként kezeljük.

Ha Irány=**rec**(x,y : [-1..+1]), ahol fel = (-1,0), le = (1,0), jobbra = (0,1), balra = (0,-1), akkor a művelet:

if not (ir = fel, le, jobb, bal **and** x+ir.x∈[1..n] **and** y+ir.y∈[1..m]) **then error endif**
k := a.térkép[x+ir.x, y+ir.y]

Osztály:



A térkép adattag inicializálásához a konstruktor kaphat paraméterként egy Tartalom[][] típusú mátrixot,

vagy akár be is olvashatja a térkép adatait például egy szöveges állományból.

A metódusok törzsében a „**this**.” prefixszel hivatkozhatunk annak az objektumnak az adattagjaira, amelyre a metódust meghívták.

Ha a MiVanOtt() metódusban az x+ir.x helyett a **this**.x+ir.x kifejezést használnánk, még egyértelműbb lenne, hogy itt két eltérő objektumnak az x nevű adattagjait adjuk össze.