1. *Egyszerű osztályok I.*

1. Folyékonyszappan-adagoló. Az adagolónak ismert a maximális kapacitása, az abban tárolt szappan aktuális mennyisége, és az, hogy az adagolófej egy nyomásra mennyi adag szappant tud kinyomni (mindegyik milliliterben). Legyen lehetőség megnyomni az adagolót (ekkor az egy adag szappant kinyom); illetve feltölteni folyékony szappannal az adagolót.

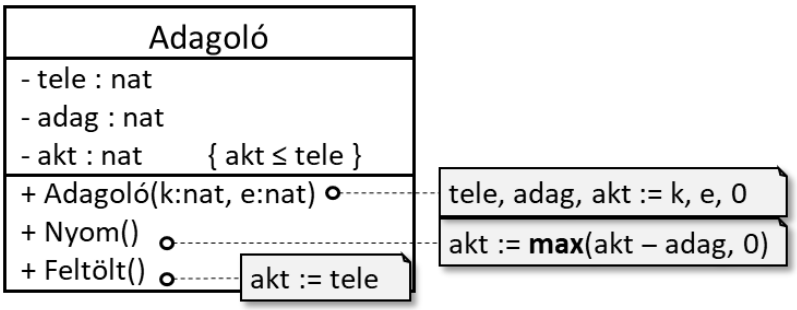
Típusdefiníció: **Adagoló**

adagolók	a) a := Nyom(a) (a:Adagoló)	a, e := Nyom(a) e:ℕ
	b) a := Feltölt(a) (a:Adagoló)	a:=Feltölt(a, f) f:ℕ
tele : ℕ adag : ℕ akt : ℕ // Inv: akt≤tele	a) akt := **max**(akt – adag, 0)	
	b) akt := tele	

Láthatjuk, hogy sem az adagoló fogalmát, sem a műveleteit nem definiáljuk a típusspecifikációban, csak a műveletek szintaxisát jelzi egy-egy értékadás, amiből leolvasható a művelet inputja és outputja. (Így nem kell majd bizonyítani, hogy a típusmegvalósítás megfelel a típusspecifikációnak.) A típust közvetlenül a megvalósításával adjuk meg: a reprezentációt adó adattagokkal, és az ezeken operáló programokkal. A típusinvariánst kielégítő adattagok együttes értéke reprezentál egy típusértékét (egy adagolót), a programok pedig a műveleteket helyettesítik. (A programokban az akt, adag, tele helyett írhatnánk a.akt, a.adag, a.tele kifejezést is arra utalva, hogy ezek az 'a' objektum adattagjai.) Egy ilyen program megadását bonyolultabb esetben megelőzheti annak elő- utófeltételes specifikációja. Például:

A = (tele : ℕ, adag : ℕ, akt : ℕ)
Ef = (tele=$tele_0$ ∧ adag=$adag_0$ ∧ akt=akt_0 ∧ akt≤ tele) [fennáll az invariáns]
Uf = (tele=$tele_0$ ∧ adag=$adag_0$ ∧ akt=**max**(akt_0– $adag_0$, 0) ∧ akt≤ tele)) [őrzi az invariánst]

Osztály:



Az adattagok rejtettek (privát), de ha meg akarjuk engedni az olvasásukat, akkor ezt a diagramban a getter bejegyzéssel jelezhetjük.

A konstruktor feladata egy objektum (itt egy üres adagoló) létrehozása. Ennek végrehajtása egy

Adagoló a = **new** Adagoló(100, 5)

utasítással történik, ahol 'a' az a változó, amely a létrehozott (példányosított) objektumra hivatkozik majd. A metódusok törzsében használt tele, adag, akt hivatkozások annak az objektumnak az adattagjai, amelyet az adott konstruktor létrehoz, vagy amelyre egy metódust meghívnak. Például az a.Nyom() vagy a.Feltölt() esetén az 'a' (vagy 'a'-val hivatkozott) objektumé.

2. 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**

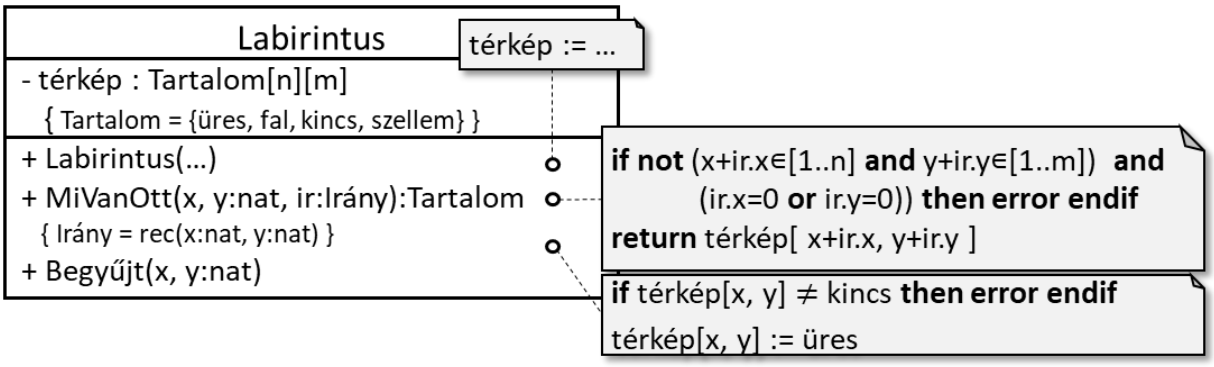
labirintusok	a) k := MiVanOtt(a, x, y, ir) (a:Labirintus, x, y:ℕ, ir:Irány, k:Tartalom) Irány = { fel, le, jobb, bal } Tartalom = { üres, fal, kincs, szellem }
	b) a := Begyűjt(a, x, y) (a:Labirintus, x, y:ℕ, k:Tartalom)
térkép : Tartalom$^{n\times m}$	a) **switch** (ir) **case** fel: **if** (x=1) **then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

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

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.

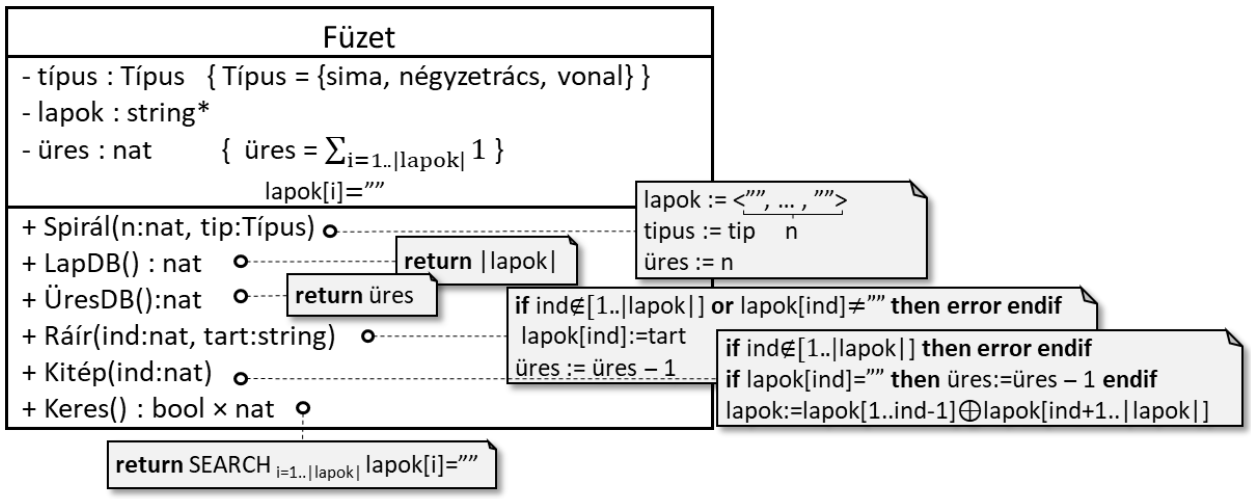
A metódusok törzsében a „**this**.” prefixszel hivatkozhatunk annak az objektumnak az adattagjaira, amelyre a metódust meghívták. Ha a MiVanOtt() metódusban az x+ir.x helyett a **this**.x+ir.x kifejezést használnánk, még egyértelműbb lenne, hogy itt két eltérő objektumnak az x nevű adattagjait adjuk össze.

3. Spirál füzet típusa. Egy füzet azonos típusú (vagy négyzetrácsos, vagy sima, vagy vonalas) lapokból áll; bizonyos lapjaira már írtak, a többi még üres. Le lehet kérdezni, hogy hány üres lap van még a füzetben; ki lehet tépni valahányadik lapot; rá lehet írni egy kiválasztott lapra, ha az még üres; megkereshetjük az első üres lap sorszámát.

Típusdefiníció: **Füzet**

füzetek	a) db := LapDB(f) (f:Füzet, db:$\mathbb{N}$) b) db := ÜresLapDB(f) (f:Füzet, db:$\mathbb{N}$) c) f := Ráír(f, ind, tart) (f:Füzet, ind:$\mathbb{N}$, tart:$\mathbb{S}$) d) f := Kitép(f, ind) (f:Füzet, ind:$\mathbb{N}$) e) l, ind := KeresÜres(f) (f:Füzet, l:$\mathbb{L}$, ind:$\mathbb{N}$)
típus : Típus lapok : $\mathbb{S}^*$ üres : $\mathbb{N}$ // üres = $\sum_{i=1..\lvert lapok\rvert,\ lapok[i]=""} 1$ Típus = {sima, négyzetrács, vonal}	a) db := |lapok| b) db := üres c) **if** ind∉[1 .. lapok] **or** lapok[ind]≠"" **then error endif** lapok[ind]:=tart üres := üres – 1 d) **if** ind∉[1 .. |lapok|] **then error endif** **if** lapok[ind]="" **then** üres:=üres –1 **endif** lapok:=lapok[1..ind-1]⊕lapok[ind+1..|lapok|] e) l, ind := **SEARCH**$_{i=1..\lvert lapok\rvert}$ (lapok[i] = "")

Osztály:



Az üres lapot kereső művelet tipikus példája a végrehajtható specifikáció alkalmazásának. Ezt a specifikációt egyértelműen visszavezethetjük a lineáris keresés algoritmus mintára.

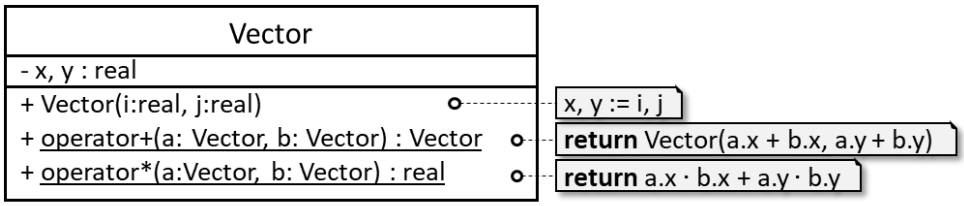
4. Síkvektorok típusa (összeg, nyújtás, skaláris szorzat műveleteivel). Döntsük el, hogy adott síkvektorok összege merőleges-e egy másik síkvektorra (a skaláris szorzatuk nulla-e).

Típusdefiníció: **Vector**

síkvektorok	c := a+b	(a, b, c : Vector)
	s := a·b	(a, b: Vector, s:ℝ)
x, y:ℝ	c.x, c.y := a.x+b.x, a.y+b.y	
	s := a.x·b.x + a.y·b.y	

A programok leírásában meg kell különböztetnünk, hogy mikor beszélünk az 'a', a 'b', vagy a 'c' vektor x koordinátájáról: a.x, b.x, illetve c.x.

Osztály:



Az összeadás és a skaláris szorzás bemenete nem egy vektorról szól: a bemenetük két vektor, az összeadásnak a kimenete egy harmadik. Nem lenne elegáns (bár megtehetnénk), ha ezeket a műveleteket egyetlen Vector típusú objektum műveleteiként vezetnénk be. Ehelyett ezek a Vector osztály (osztályszintű) metódusai lesznek, és ezeket nem egy kitüntetett vektor objektumra kell meghívni úgy, hogy paraméterként adjuk a másik vektort, hanem olyan metódusként, amelynek két vektor partamétere van, az összeadás esetében pedig a Vector típusú visszatérési értéke.

Feladat:

Adott síkvektorok összege merőleges-e egy adott síkvektorra (skaláris szorzatuk nulla-e).

Specifikáció:

A = (v:Vector^n, w:Vector, l: $\mathbb{L}$)
Ef = (v=v' $\wedge$ w=w')
Uf = (Ef $\wedge$ l = (($\Sigma_{i=1..n}$ v[i])*w = 0.0))

Algoritmus: