2. Óra

Miről lesz szó?

User data types

struct

- ℹ️ Érték típus (szimplább, kicsi adatstruktúrákra)
- ℹ️ Deklarálásnál létrejön a struct a stacken.

```
public struct Coords {
    public int x;
    public int y;
}

Coords coords; // Deklarálás
coords.x = 2;
coords.y = 3;
```

class

- ℹ️ Referencia típus

Stack? *Heap*? (Runtime memory areas)

Stack

- Lokális változók
- Érték típusok (primitív típusok + structok, enumok)

Heap

- Objektumok
  - Stringek
  - Arrayek
  - Listek
  - ...

Miért fontos?

`Wallet.cs`

```
public enum Banknote { Ft20k, Ft10k, Ft5k, Ft1k }
public class Wallet {
    private List<Banknote> _banknotes;

    public Wallet(List<Banknote> banknotes) {
        _banknotes = banknotes;
    }
```

```
    public void ListBanknotes() {
        foreach (Banknote banknote in _banknotes) {
            Console.WriteLine(banknote);
        }
    }
}
```

`Program.cs`

```
public class Program {
    static void Main(string[] args) {
        List<Banknote> list = new(){ Banknote.Ft1k };
        Wallet w = new(list);

        w.ListBanknotes(); // Ft1k

        list.Add(Banknote.Ft20k);
        w.ListBanknotes(); // Ft1k, Ft20k
    }
}
```

Megoldás: Defensive copying

```
_banknotes = new List<Banknote>(banknotes);
```

Array vs *List*

```
int[] a = ...
List<int> b = ...
```

Array

- Fix méret
- Folyamatos blokk a heapen

List

- Belül array
- Dinamikus (reallokál ha kell)
- Több memória
- SOKKAL több funkcionalitás
- Folyamatos blokk a heapen

Overriding

- `virtual`: van valamilyen implementáció, de felül **lehet** írni
- `abstract`: nincs implementáció, felül **kell** írni

```
public override string ToString() {
    return "...";
```

```
}
```

Object: `ToString()`, `Equals()`, `GetHashCode()`

Overloading

- Metódusok
- Konstruktorok

```
public int Add(int a, int b) => a + b;
public int Add(int a, int b, int c) => a + b + c;
```

Interpolált Stringek

- Behelyettesítésnél meghívja az objektumok `ToString()` metódusát

```
string a = "(" + _n.ToString() + "," + _d.ToString() + ")";

string b = $"({_n},{_d})";
```

`String.Format()`

- mértékegységek formázása
- indexelt placeholderes behelyettesítés

```
String.Format("{0:0.0}, {1:0.0#}", x, y);
```

(https://learn.microsoft.com/en-us/dotnet/fundamentals/runtime-libraries/system-string-format#get-started-with-the-stringformat-method)

Interpolált stringekben is:

```
$"{x:0.0#}, {y:0.0#}"
```

Operátor definíció

```
public static Rational operator +(Rational a, Rational b) {
    return new Rational(a._n * b._d + a._d * b._n, a._d * b._d);
}
```

`out` keyword

- Ha egy metódus több mindent szeretne visszaadni

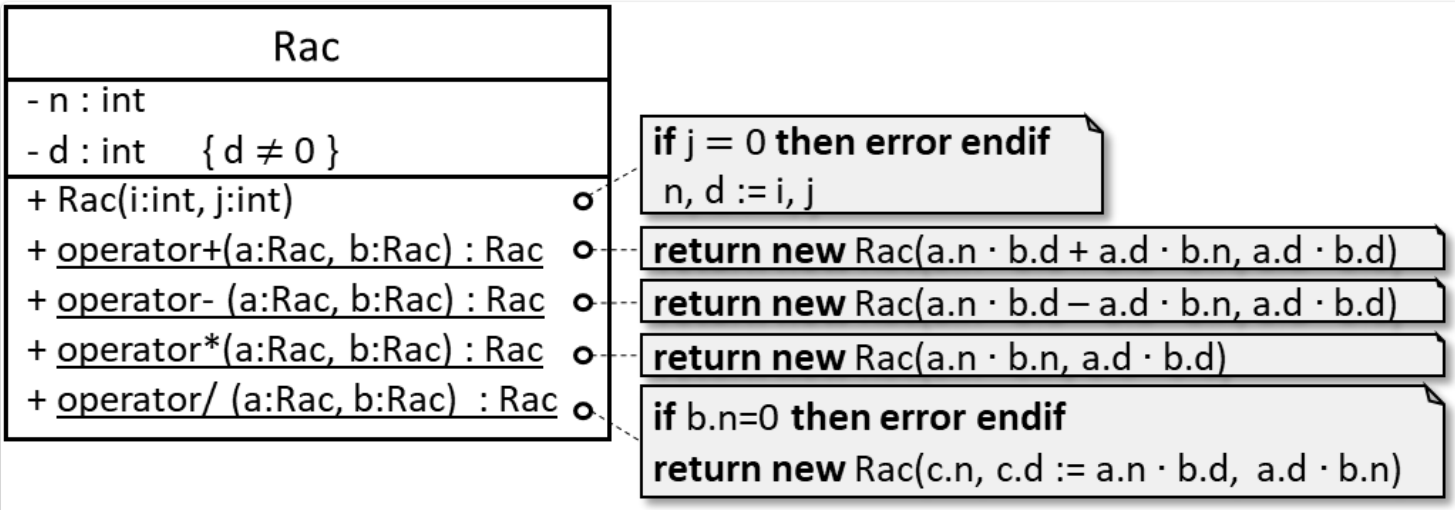
```
public class Notebook {

    ...

    public bool Search(out int ind) {
        ind = 0; // kötelező inicializálni mielőtt return-ölünk
```

```
        for (int i = 0; i < PageCount(); ++i) {
            if (_pages[i] == "") {
                ind = i + 1;
                return true;
            }
        }
        return false;
    }
}
```

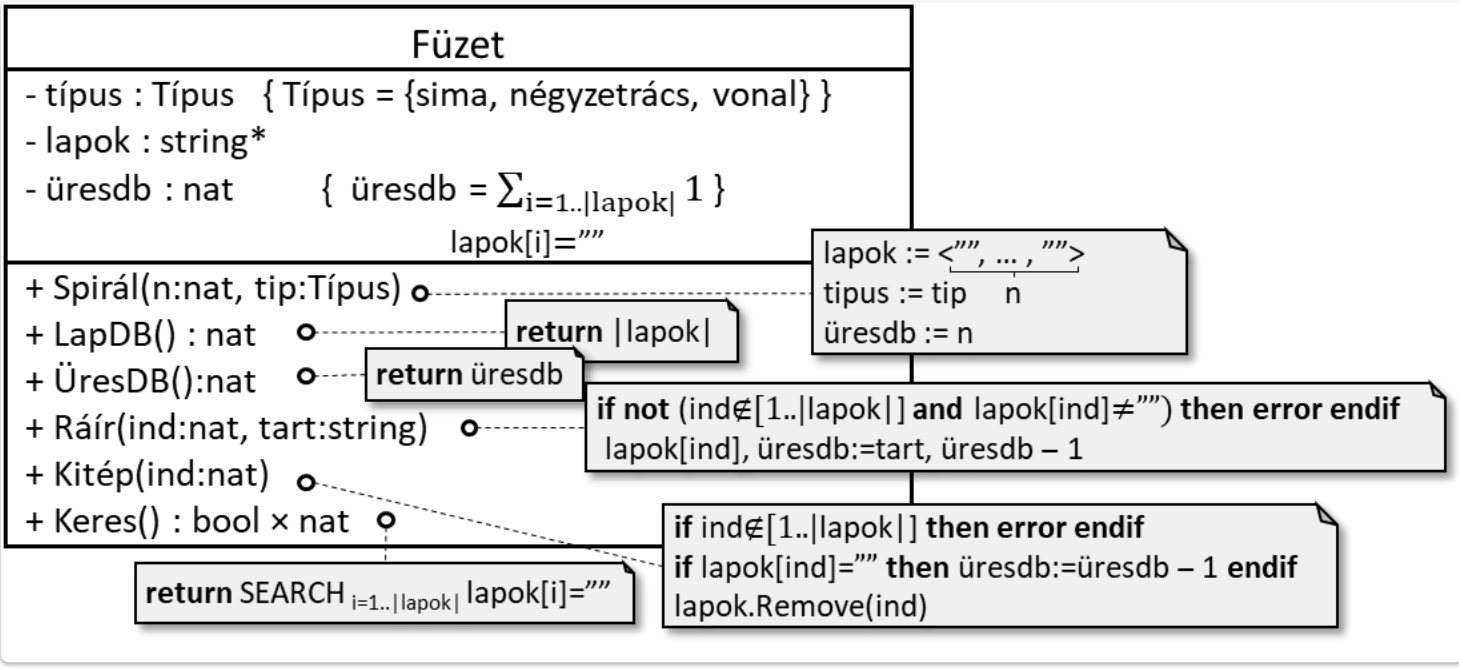
```
Notebook notebook = new(...);

bool l = notebook.Search(out int ind); // nem kell inicializálni
```

Órai feladatok

Racionális számok

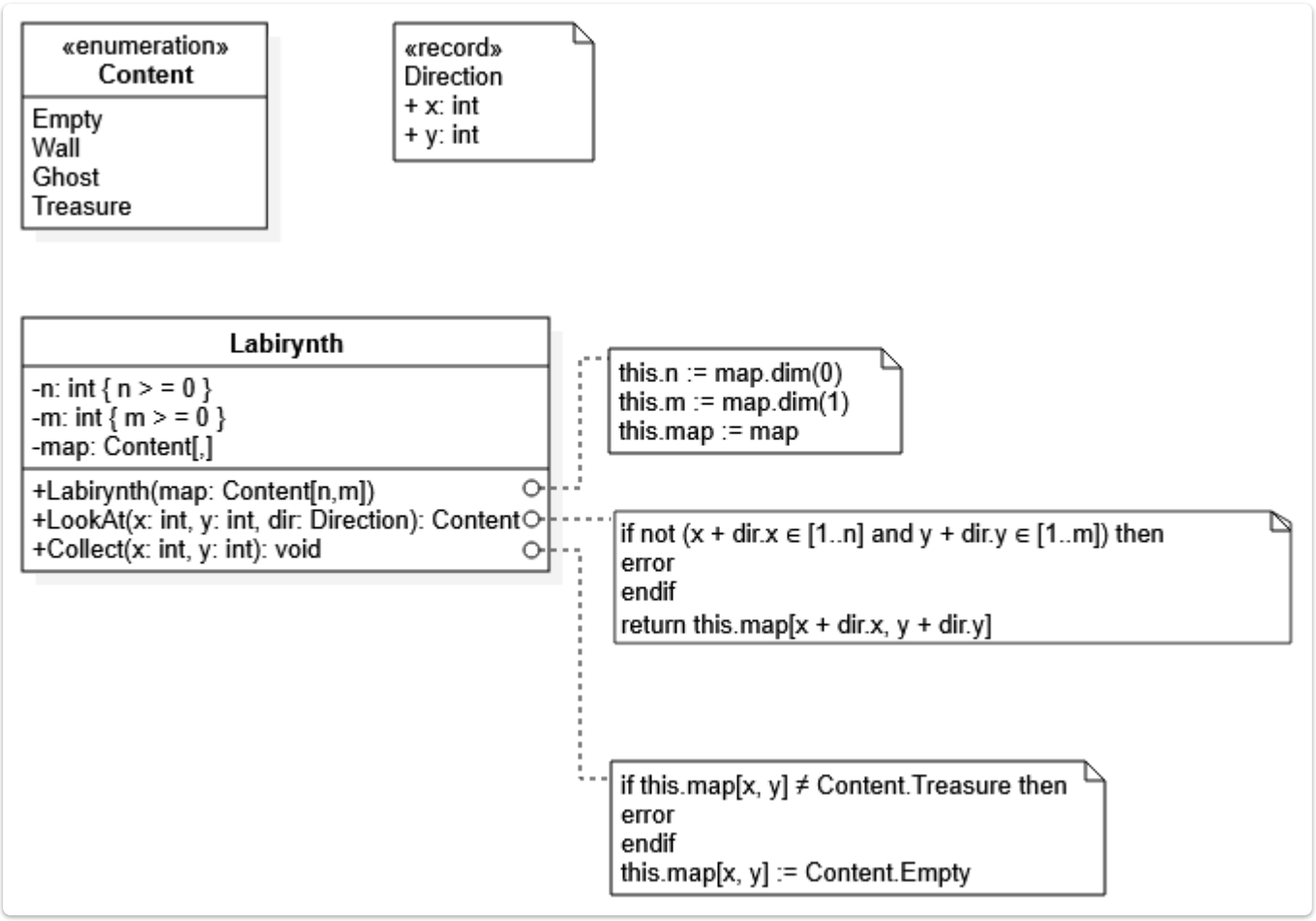


Spirálfüzet



Kisbeadandó (2/10) (1. batch)

★ Labirintus


«enumeration»
Content
Empty
Wall
Ghost
Treasure
«record»
Direction
+ x: int
+ y: int
Labirynth
-n: int { n > = 0 }
-m: int { m > = 0 }
-map: Content[,]
+Labirynth(map: Content[n,m])
+LookAt(x: int, y: int, dir: Direction): Content
+Collect(x: int, y: int): void
this.n := map.dim(0)
this.m := map.dim(1)
this.map := map
if not (x + dir.x ∈ [1..n] and y + dir.y ∈ [1..m]) then
error
endif
return this.map[x + dir.x, y + dir.y]
if this.map[x, y] ≠ Content.Treasure then
error
endif
this.map[x, y] := Content.Empty