

Vermek és Fák

1.rész

Lengyel forma és kiértékelése

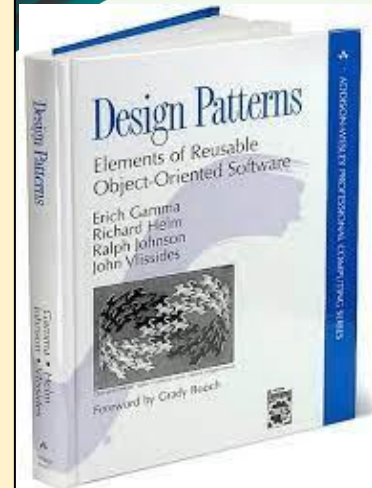
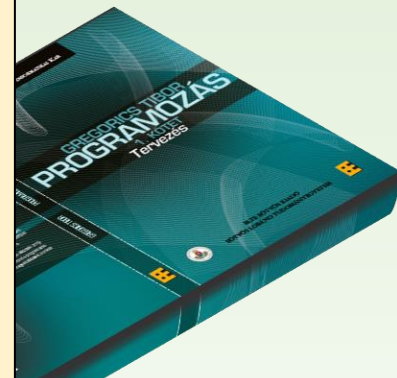
Gregorics Tibor

gt@inf.elte.hu

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

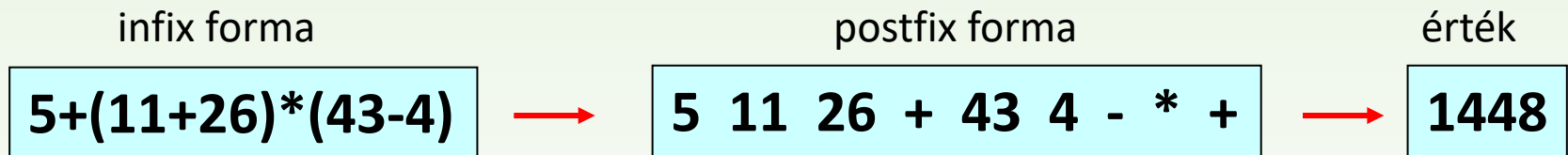
Programozási minták

- ❑ Egy programozási feladat megoldása gyorsabb, az előállított program biztonságosabb, ha a megoldást korábbi, hasonló feladatok megoldásainál bevált minták alapján állítjuk elő.
- ❑ A szoftvertechnológiában az ilyen programozási mintáknak számos csoportja jött létre. Ilyenek például
 - az algoritmus minták (programozási tételek),
 - a tervezési minták (tervminták, design patterns).
- ❑ A **tervezési minták** az objektumelvű modellezést támogatják: az osztály diagram tervezése során alkalmazzuk őket abból a célból, hogy a modell újrafelhasználható, könnyen módosítható, biztonságosan működő, és hatékony legyen, valamint – nem utolsósorban – megfeleljen a SOLID tervezési elveknek.



Feladat

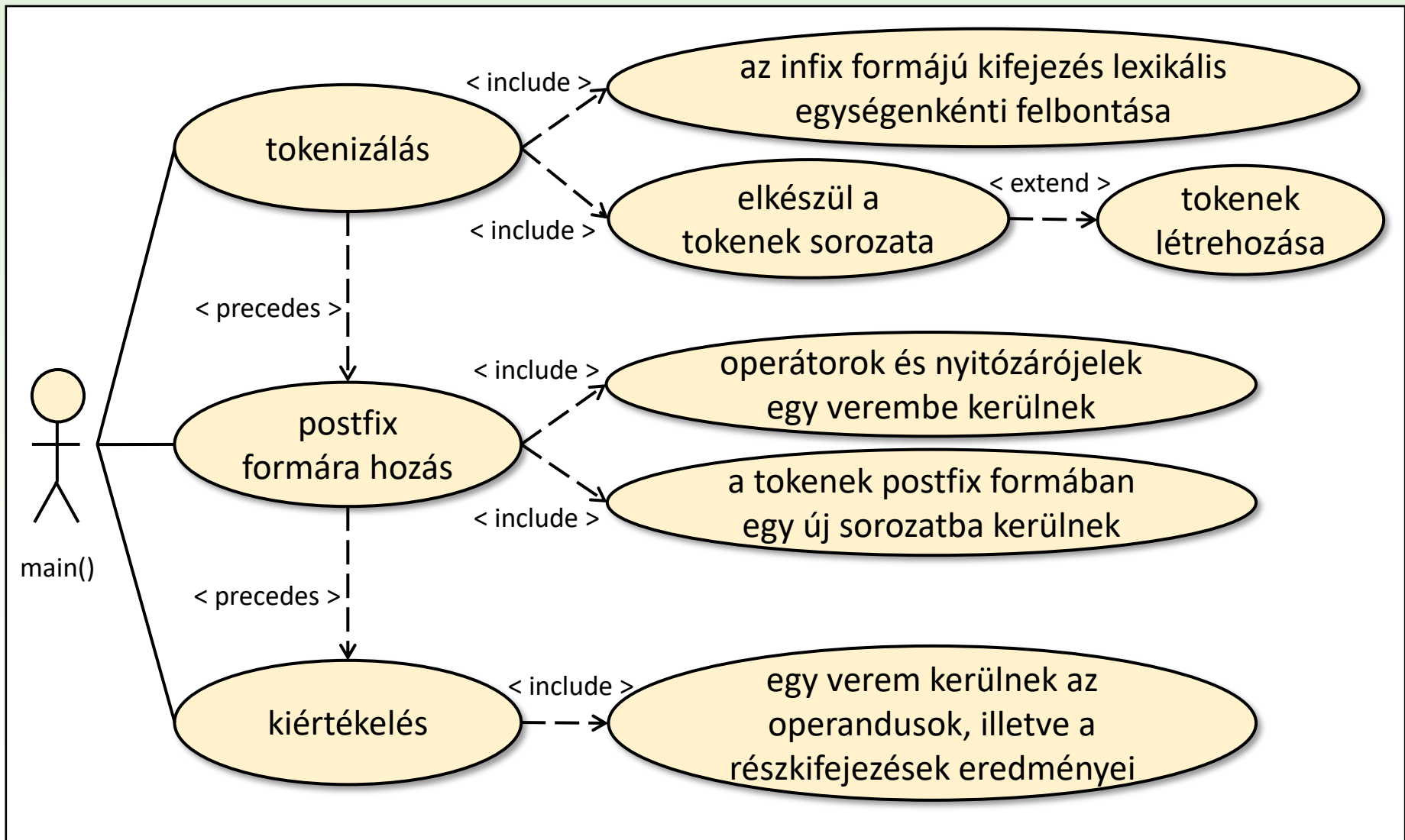
Alakítsunk át egy **infix** formájú aritmetikai kifejezést **postfix** formájúra (RPN), és számoljuk ki az **értékét**. (A vizsgált kifejezésben most csak természetes számok fordulhatnak elő.)



Mindkét lépéshez egy-egy **vermet** szoktak használni. Az átalakításnál a formula operátorait (műveleti jeleit) és nyitózárojeleket tároljuk egy veremben, a kiértékeléskor operandusokat és részeredményeket, azaz számokat.

Szükség lesz arra, hogy a kifejezéseinkben egymástól elválasztva tudjuk kezelni a lexikális egységeket (tokeneket), azaz külön-külön a műveleti jeleket, a zárójeleket, és a számokat.

Elemzés



Objektumok

- tokenek:** operátorok (**Operator**), operandusok (**Operand**), zárójelek (**LeftP**, **RightP**) mind a **Token** osztály leszármazottjai
- sorozatok:** tokenek (hivatkozásai) gyűjteményei (**List<Token>**):
 - egyik az infix formájú tokenizált kifejezést tárolja (infix)
 - másik a postfix formára hozott tokenizált kifejezést tárolja
- vermek:** tokeneket (hivatkozásait) tároló verem (**Stack<Token>**), egész számokat tároló verem (**Stack<int>**)
- sztring:** az infix formájú kifejezés a szabványos bemeneten (**string**)

Főprogram

```
public class Interrupt : Exception { }
static void Main()
{
    ConsoleKeyInfo cki;
    do {
        Console.WriteLine("\nGive me an arithmetic expression!\n");
        string input = Console.ReadLine();
        try
        {
            // Tokenization
            List<Token> infix = new();

            ...

            // Transforming into RPN
            Stack<Token> stackToken = new();
            List<Token> postfix = new();

            ...

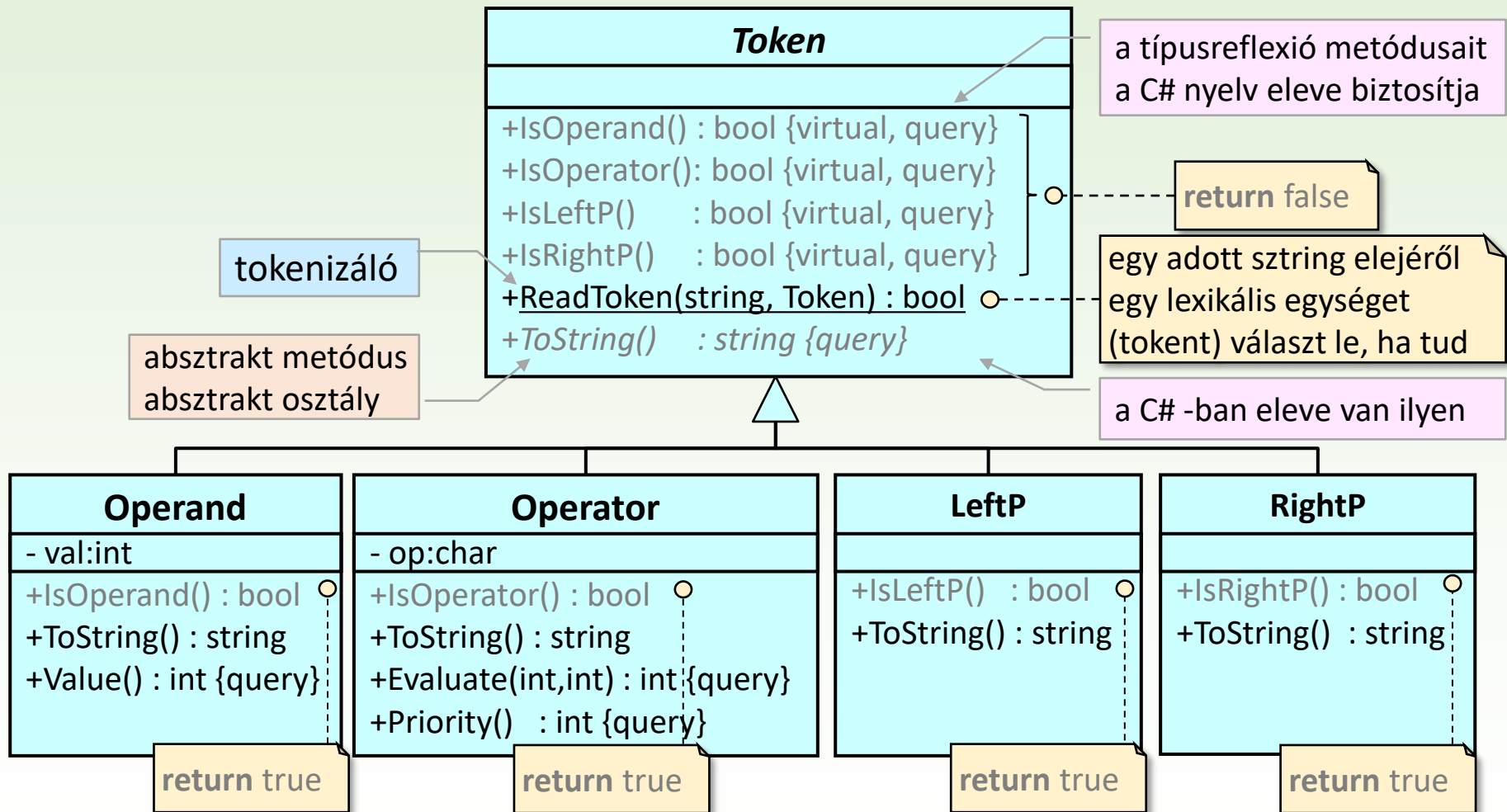
            // Evaluation
            Stack<int> stackInt = new();
            int result;

            ...
        }
        catch (Interrupt) { }

        Console.Write("\nDo you continue? I/N");
        cki = Console.ReadKey();
    } while (cki.KeyChar != 'n' && cki.KeyChar != 'N');
}
```

biztos nem lesz az értéke null

Token osztályok (első próbálkozás)



Token osztály gyártó függvénye

```
const string digits = "0123456789";
```

```
public static bool ReadToken(ref string input, out Token? token)
```

```
{
```

```
    token = null;
```

```
    if (input.Length == 0) return false;
```

```
    int i = 1;
```

```
    switch (input[0])
```

```
    {
```

```
        case '+': token = new Operator(...); break;
```

```
        case '-': token = new Operator(...); break;
```

```
        case '*': token = new Operator(...); break;
```

```
        case '/': token = new Operator(...); break;
```

```
        case '(': token = new LeftP(); break;
```

```
        case ')': token = new RightP(); break;
```

```
        case '0': case '1': case '2': case '3': case '4':
```

```
        case '5': case '6': case '7': case '8': case '9':
```

```
            string number = "";
```

```
            for (i = 0; i < input.Length && digits.Contains(input[i]); ++i)
```

```
            { number += input[i]; }
```

```
            token = new Operand(int.Parse(number));
```

```
            break;
```

```
        default: throw new IllegalElementException;
```

```
    }
```

```
    input = input[i..];
```

```
    return true;
```

```
}
```

lehet null értékű is

input-output paraméter

feltétel fennállásáig tartó
összefűzés (összegzés)

levágja az input elejéről a tokenné alakított karaktereket

Operator osztály (első próbálkozás)

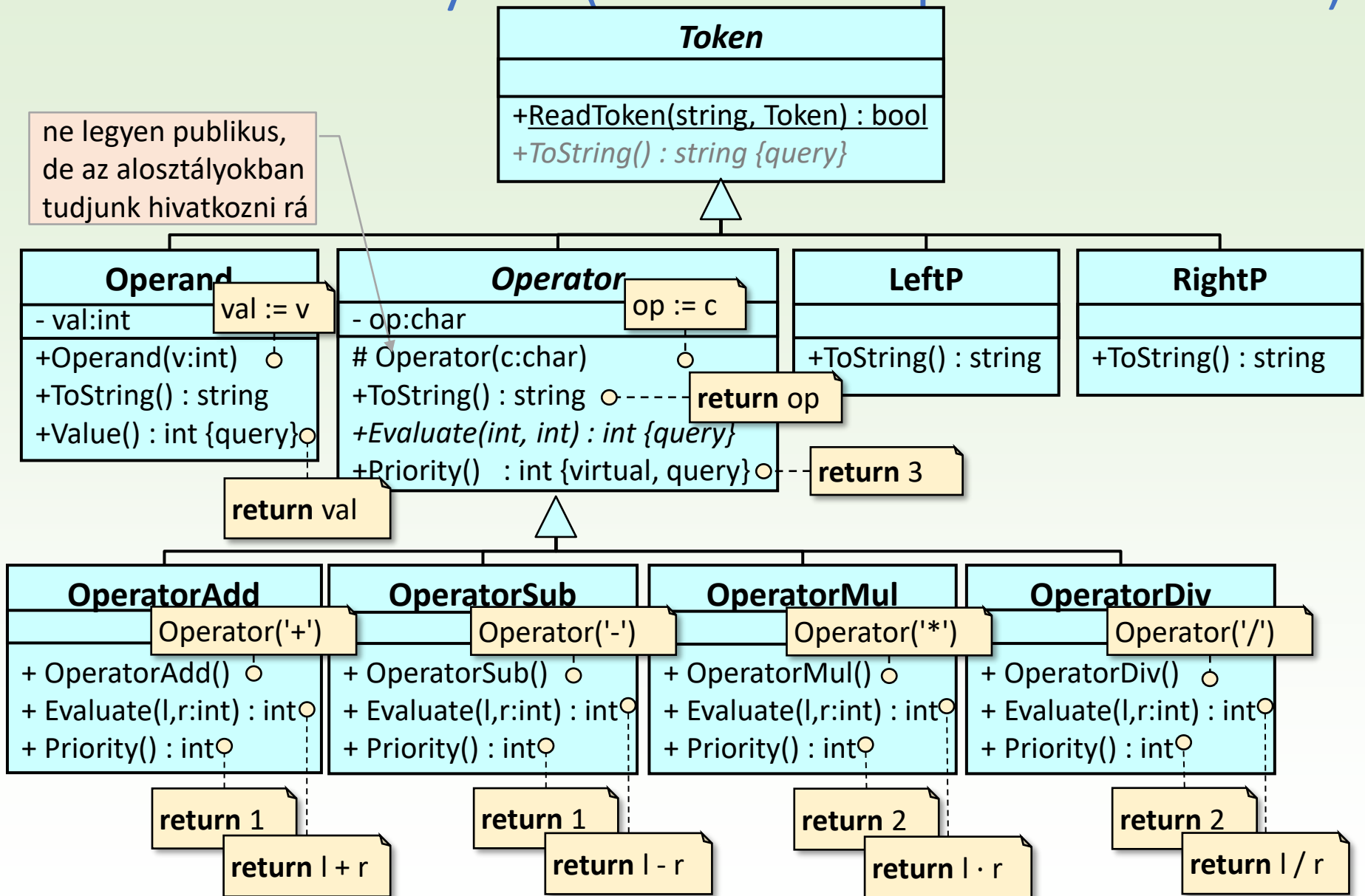
```
public class Operator : Token
{
    private readonly char op;

    public Operator(char o) { op = o; }
    public override string ToString() { return op.ToString(); }
    public int Evaluate(int leftValue, int rightValue)
    {
        switch(_op) {
            case '+': return leftValue + rightValue;
            case '-': return leftValue - rightValue;
            case '*': return leftValue * rightValue;
            case '/': return leftValue / rightValue;
            default: return 0;
        }
    }
    public int Priority()
    {
        switch(_op) {
            case '+': case '-': return 1;
            case '*': case '/': return 2;
            default: return 3;
        }
    }
}
```

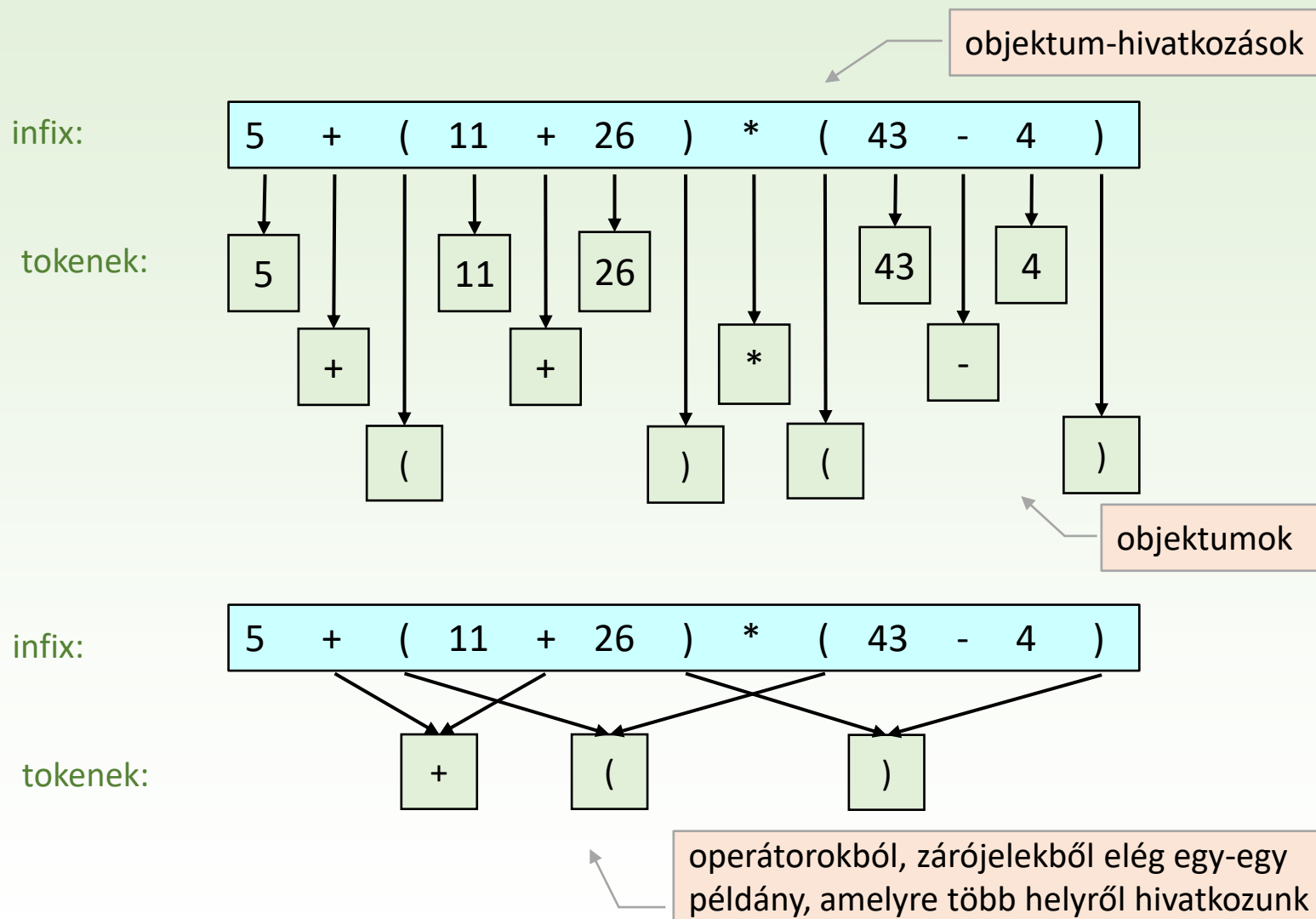
Single responsibility
Open-Closed
Liskov's substitution
I
D

ez a kód nem felel meg a nyitott-zárt elvnek:
újabb műveleti jelek bevezetése esetén több
metódus kódját is módosítani kellene

Token osztályok (második próbálkozás)

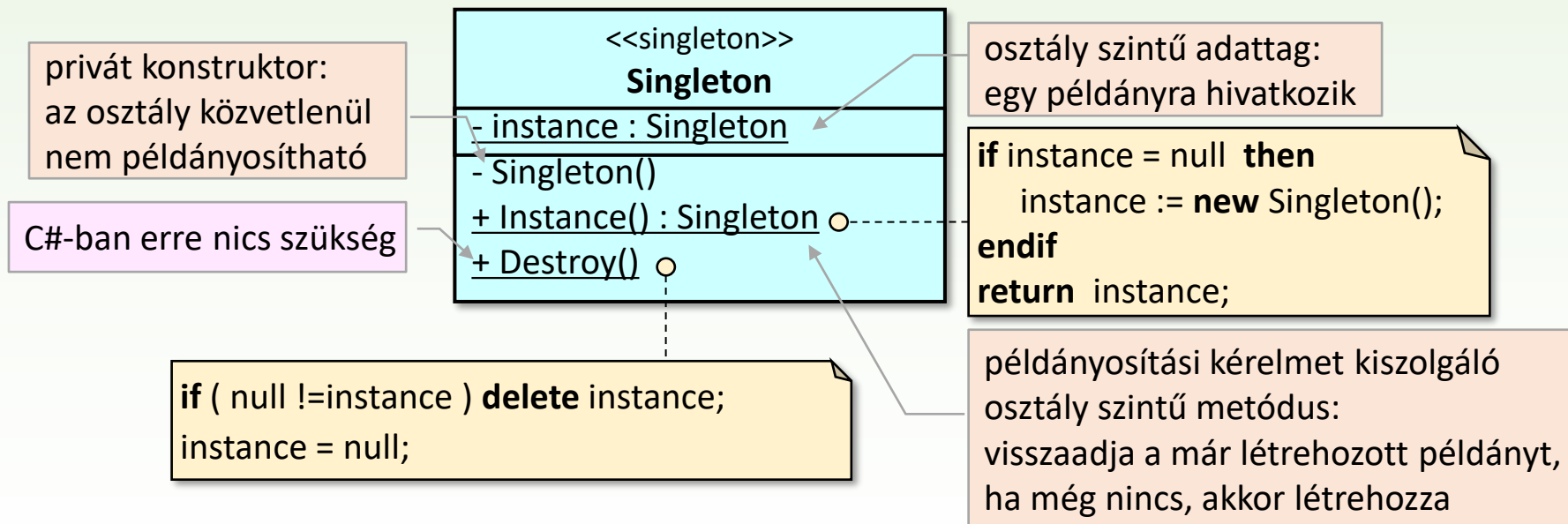


Memória igény

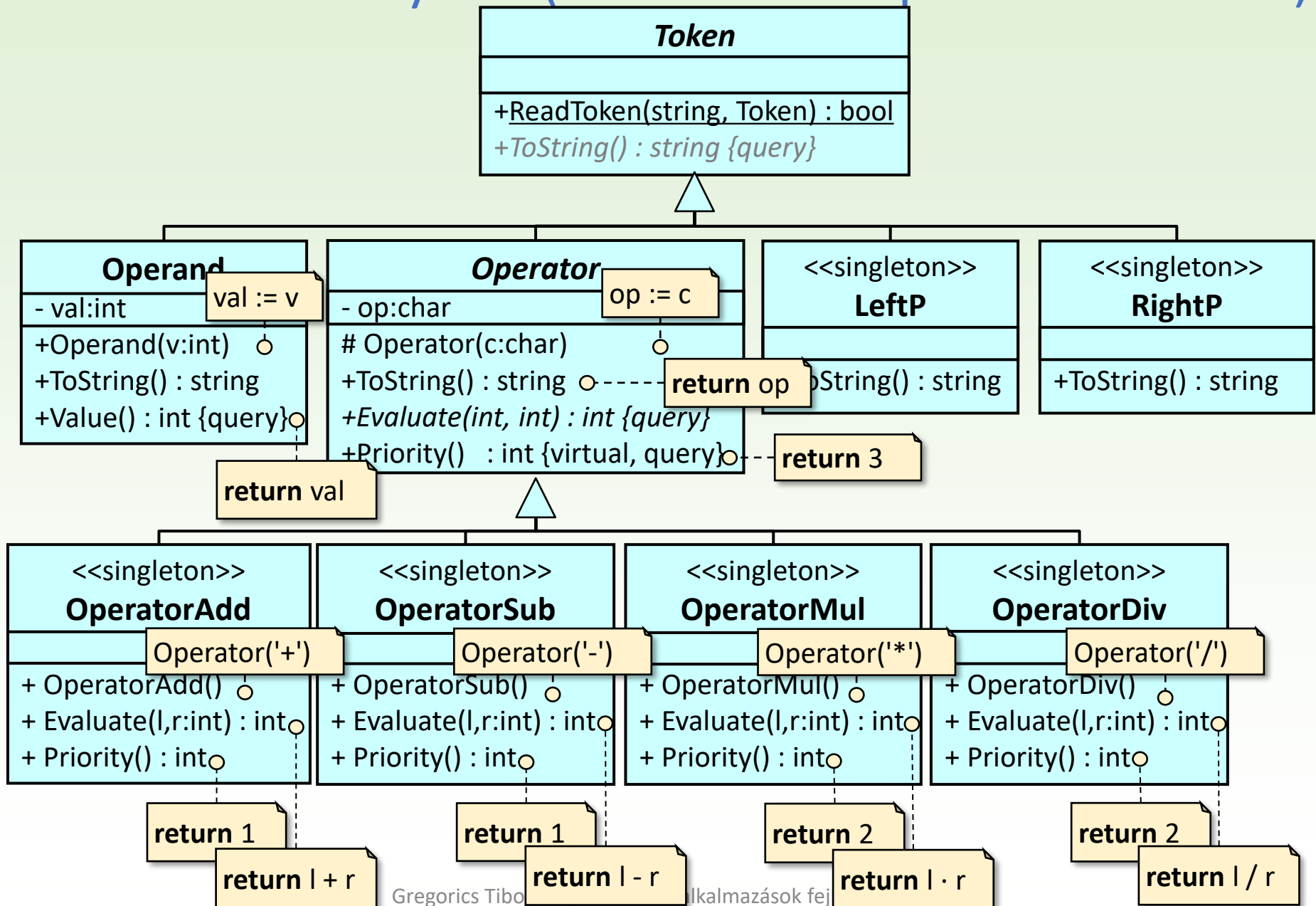


Egyke (singleton) tervezési minta

- Amikor egy osztályhoz legfeljebb egy objektumot kell példányosítani függetlenül a példányosítási kérések számától.



Token osztályok (harmadik próbálkozás)



LeftP és RightP osztály egykeként

```
public class RightP : Token
{
    private RightP() { }
```

```
public class LeftP : Token
{
    private LeftP() { }
```

kívülről nem hívható

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

a már létrehozott egyetlen példányra hivatkozik, vagy null

```
    private static LeftP? instance;
```

ezt a gyártó függvényt kell hívni a konstruktor helyett

```
    public static LeftP Instance()
    {
        instance ??= new LeftP();
        return instance;
    }
}
```

if (instance == null) instance = new LeftP()

Operator osztály és egyke alosztályai

```
abstract public class Operator : Token
{
    protected Operator(char o) { op = o; }
    public override string ToString() { return op.ToString(); }
    public abstract int Evaluate(int leftValue, int rightValue);
    public virtual int Priority(){ return 3; }
}
```

```
public class OperatorAdd : Operator
```

```
{
```

```
    public class OperatorSub : Operator
```

```
    {
```

```
        public class OperatorMul : Operator
```

```
        {
```

```
            public class OperatorDiv : Operator
```

```
            {
```

```
                private OperatorDiv() : base('/') { }
```

őszosztály konstruktorának hívása

```
                public override int Evaluate(int leftValue, int rightValue)
                { return leftValue / rightValue; }
```

```
                public override int Priority() { return 2; }
```

eltűntek az elágazások

```
                private static OperatorDiv? instance;
                public static OperatorDiv Instance() {
                    instance ??= new OperatorDiv();
                    return instance;
                }
            }
        }
    }
}
```

Tokenizálást végző metódus újra

```
public static bool ReadTok
{
    token = null;
    if (input.Length == 0)
    int i = 1;
    switch (input[0])
    {
        case '+': token = OperatorAdd.Instance(); break;
        case '-': token = OperatorSub.Instance(); break;
        case '*': token = OperatorMul.Instance(); break;
        case '/': token = OperatorDiv.Instance(); break;
        case '(': token = LeftP.Instance(); break;
        case ')': token = RightP.Instance(); break;
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            string str = "";
            for (i = 0; i < input.Length && digits.Contains(input[i]); ++i)
            { str += input[i]; }
            token = new Operand(int.Parse(str));
            break;
        default: throw new IllegalElementException(input[0]);
    }
    input = input[i..];
    return true;
}
```

```
public class IllegalElementException : Exception
{
    private readonly char ch;
    public IllegalElementException(char c) { ch = c; }
    public string What() { return ch.ToString(); }
}
```

a kivétel-objektumnak
adattagjai és metódusai
is lehetnek

Főprogram: kifejezés tokenizálása

```
// Tokenization
List<Token> infix = new ();

try
{
    while (Token.ReadToken(ref input, out Token? token))
    {
        infix.Add(token);
    }
}
catch (Token.IllegalElementException ex)
{
    Console.WriteLine("Illegal character: {0} ", ex.What() );
    throw new Interrupt();
}
if (0 == infix.Count)
{
    Console.WriteLine("Empty expression");
    throw new Interrupt();
}
```

tokent olvas

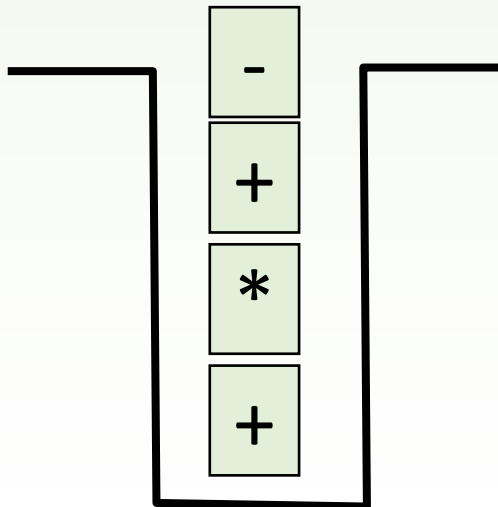
az olvasás dobja

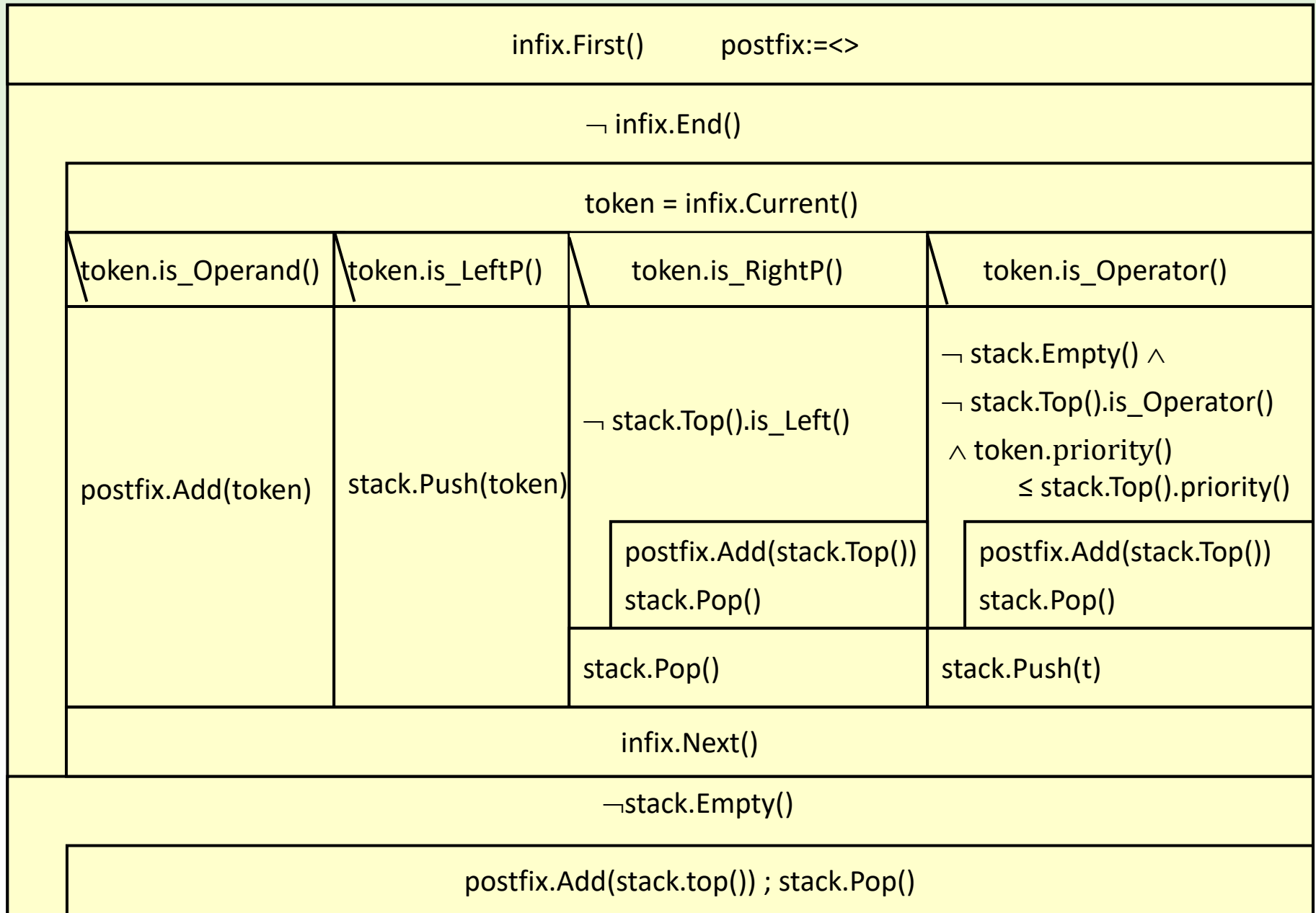
további műveletet nem végzünk

Infix formából postfix forma

A bemenő sorozat nyitó zárójeleit és műveleti jeleit egy verembe tesszük (az alacsonyabb precedenciájú műveleti jel mindig helyet cserél az alatta levő magasabb precedenciájúval), minden más jelet közvetlenül a kimenő sorozatba másolunk. Csukó zárójel olvasása esetén illetve a bemenő sorozat feldolgozásának végén kiürítjük a verem tartalmát a leg(f)első nyitózárrójeléig a kimenő sorozatba.

5 + (11 + 26) * (43 - 4)





Főprogram: Postfix formára hozás

```
// Transforming into RPN
Stack<Token> stackToken = new ();
List<Token> postfix = new ();
foreach (Token token in infix)
{
    if (token is Operand) postfix.Add(token);
    else if (token is LeftP) stackToken.Push(token);
    else if (token is RightP)
    {
        try
        {
            while (!(stackToken.Peek() is LeftP))
            {
                postfix.Add(stackToken.Peek());
                stackToken.Pop();
            }
            stackToken.Pop();
        }
        catch (InvalidOperationException)
        {
            Console.WriteLine("Syntax error: less left parenthesis than right ones");
            throw new Interrupt();
        }
    }
    else if (token is Operator operator1) ...
}
```

felsorolás

a négy-ágú elágazás a következő dián

típus reflexió

hiba lehetőség:
több nyitó zárójel, mint csukó

Postfix formára hozás

```
else if (token is Operator operator1)
{
    while ( stackToken.Count!=0 && (stackToken.Peek() is Operator operator2) &&
        operator1.Priority() <= operator2.Priority() )
    {
        postfix.Add(stackToken.Peek());
        stackToken.Pop();
    }
    stackToken.Push(token);
}
else
{
    Console.WriteLine("Syntax error: others");
    throw new Interrupt();
}
}
while (stackToken.Count!=0)
{
    if (stackToken.Peek() is LeftP)
    {
        Console.WriteLine("Syntax error: more left parenthesis than right ones");
        throw new Interrupt();
    }
    else postfix.Add(stackToken.Peek());
    stackToken.Pop();
}
```

operator1 hívkozik az **Operator** típusúra castolt tokenre

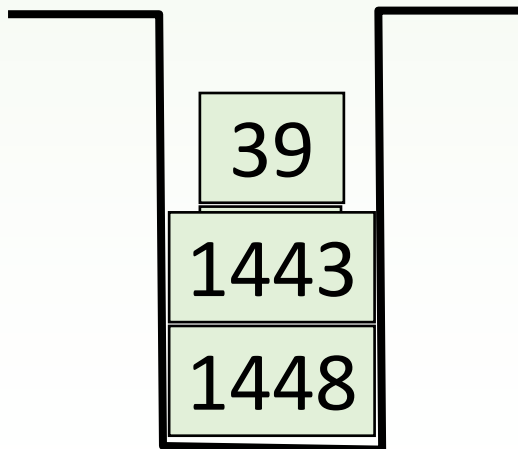
token.Priority() és stackToken.Peek().Priority() nem jó, mert Token-ben nincs Priority(), ezért castoljuk ezeket Operator típusúra: a castolások eredménye az operator1 és operator2

hiba lehetőség: több a csukó zárójel, mint nyitó

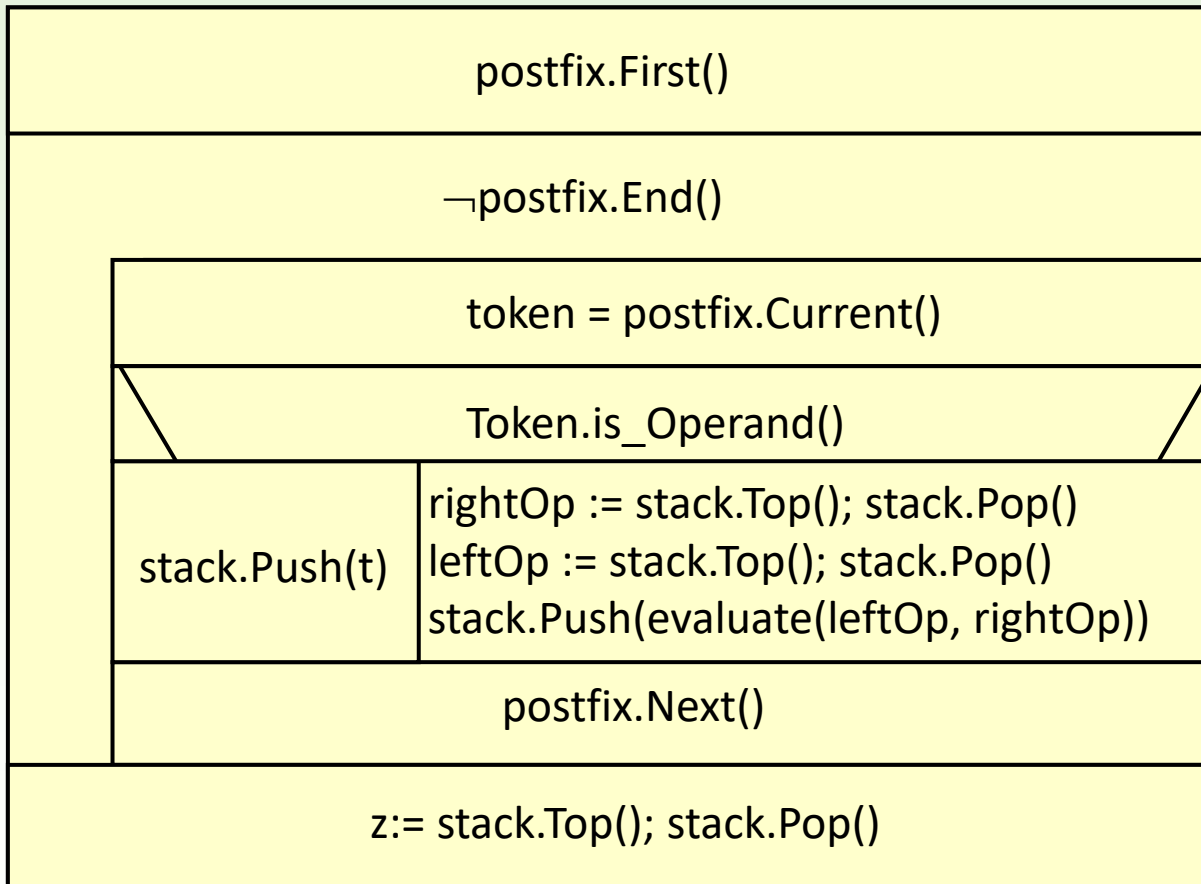
Postfix forma kiértékelése

A postfix forma operandusait (olvasásuk sorrendjében) egy verembe tesszük. Műveleti jel olvasása esetén a verem tetején levő két értéket (csak bináris műveleteink vannak) kivesszük, azokat a szóban forgó művelettel feldolgozzuk, és az eredményt visszatesszük a verembe. A feldolgozás végén a veremben találjuk kifejezés értékét.

5	11	26	+	43	4	-	*	+
---	----	----	---	----	---	---	---	---



Kiértékelés algoritmus



Kiértékelés

```
// Evaluation
Stack<int> stackInt = new();
try
{
    foreach (Token token in postfix)
    {
        if ( token is Operand operand ) { stackInt.Push((operand.Value())); }
        else if ( token is Operator operator0 )
        {
            int rightOp = stackInt.Peek(); stackInt.Pop();
            int leftOp = stackInt.Peek(); stackInt.Pop();
            stackInt.Push(operator0.Evaluate(leftOp, rightOp));
        }
    }
    int result = stackInt.Peek(); stackInt.Pop();
    if (stackInt.Count!=0)
    {
        Console.WriteLine("Syntax error: more operands than operators");
        throw new Interrupt();
    }
    Console.WriteLine($"value: {result}");
}
catch (InvalidOperationException)
{
    Console.WriteLine("Syntax error: less operands than operators");
    throw new Interrupt();
}
```

felsorolás

operand hívkozik az Operand típusúra castolt tokenre

hiba lehetőség: több operandus

hiba lehetőség: kevés operandus

Vermek és Fák

2.rész

Bináris fa és bejárása

Gregorics Tibor

gt@inf.elte.hu

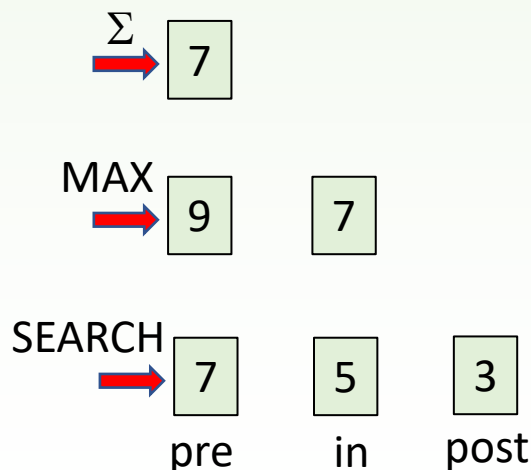
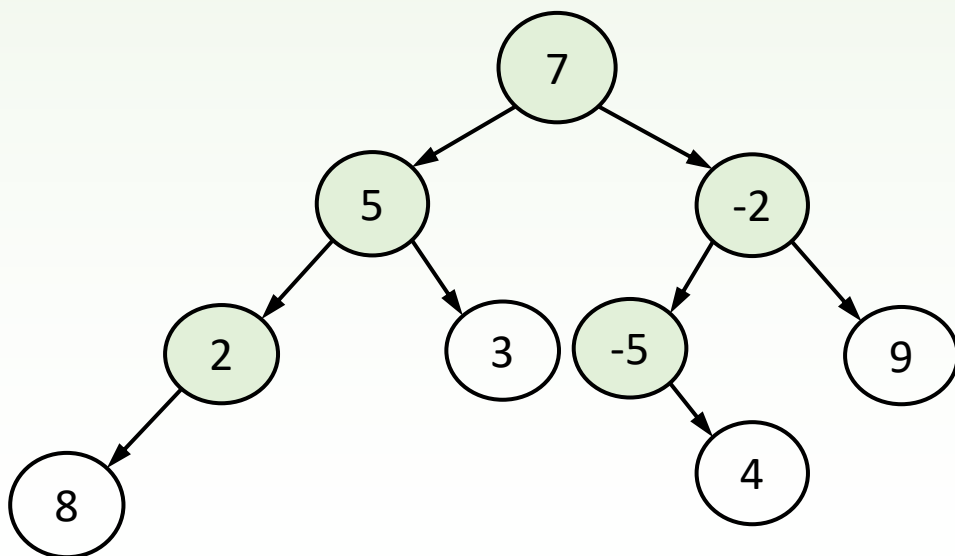
<http://people.inf.elte.hu/gt/oep>

Feladat: Bináris fa bejárása

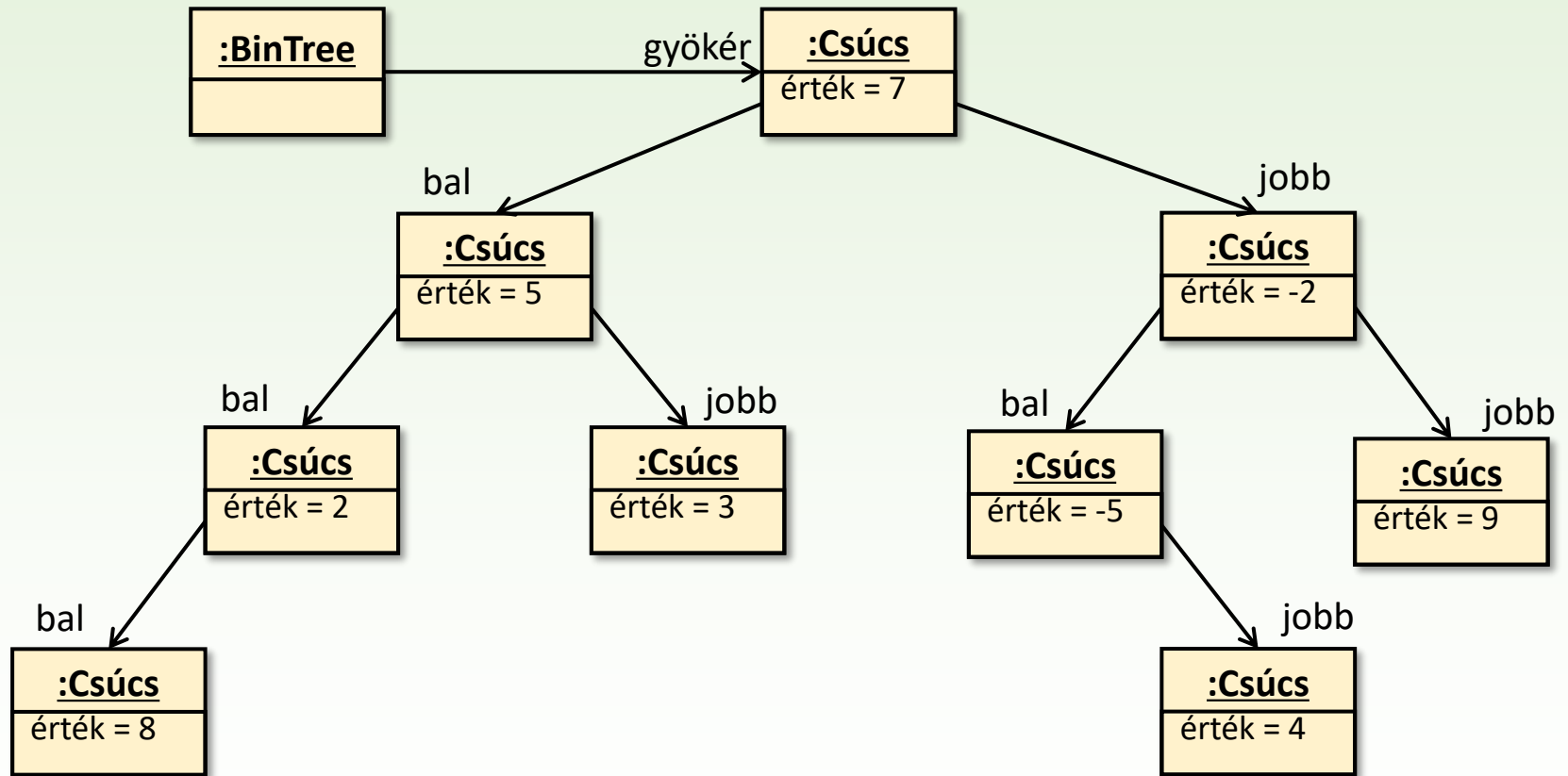
Építsünk fel megadott egész számokból egy **irányított bináris fát** úgy, hogy minden érték más-más csúcsba kerüljön, de, hogy melyikbe, az véletlen módon dőljön el.

Írassuk ki a csúcsokban tárolt értékeket különféle **bejárási stratégiák** mentén.

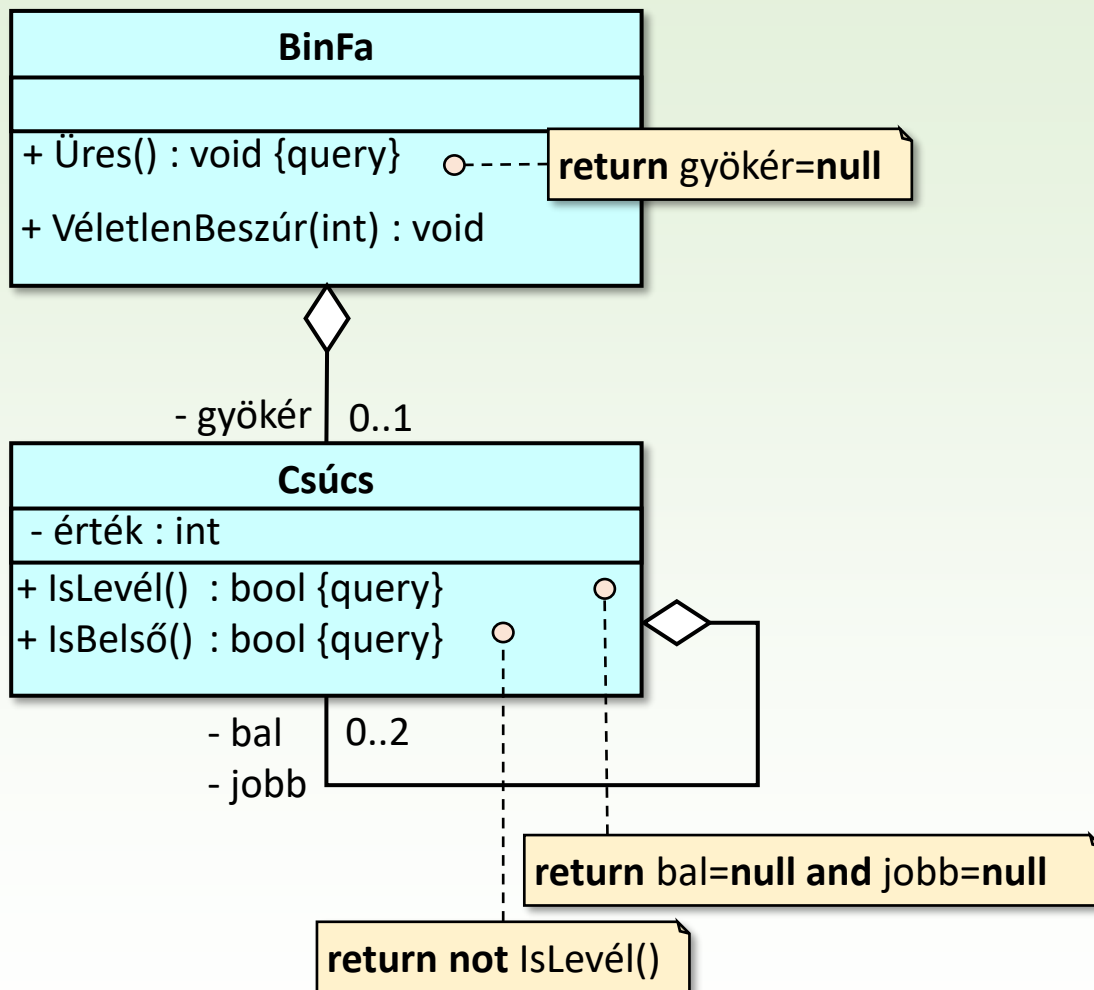
Végezzük el egy fa belső (nem levél) csúcsai értékeinek **összegzését**, az összes csúcs vagy csak a belső csúcsok értékei közötti **maximum keresést**, valamint valamelyik bejárási stratégia szerinti „első” páratlan szám **megkeresését**.



Bináris fa objektum diagramja



Osztálydiagram

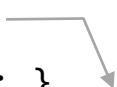


```

public class BinTree
{
    public class NoRootException : Exception { }
    private Node? root;
    public BinTree() { root = null; }
    public int Root
    {
        get
        {
            if (root == null) throw new NoRootException();
            return root.Value;
        }
    }
    public bool Empty() { return root == null; }
    private readonly Random rand = new (DateTime.Now.Millisecond);
    public void RandomInsert(int e) { ... }
}

```

véletlenszám generáló inicializálása

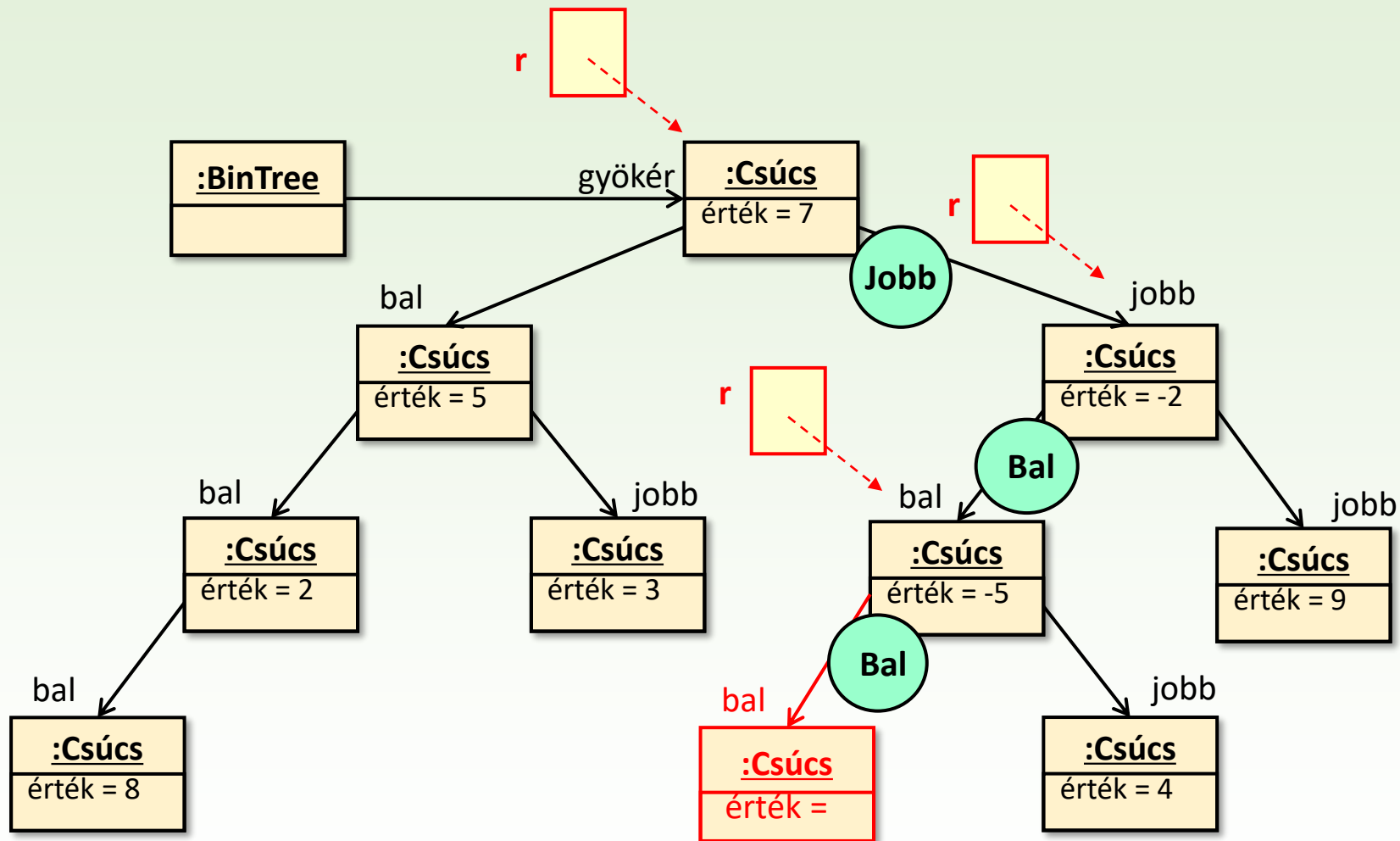


```

public class Node
{
    public int Value { get; }
    public Node? Left { get; private set; }
    public Node? Right { get; private set; }
    public Node(int v)
    {
        Value = v;
        Left = null;
        Right = null;
    }
    public bool IsLeaf { get { return Left == null && Right == null; } }
    public bool IsInternal { get { return !IsLeaf; } }
}

```

Új csúcs beszúrása a bináris fába



Új csúcs beszúrása a bináris fába

```
public void RandomInsert(int e)
{
    if (root == null) root = new Node(e);
    else
    {
        Node r = root;
        bool l = rand.Next() % 2 != 0;
        while ( l ? r.Left != null : r.Right != null)
        {
            if (l) r = r.Left;
            else r = r.Right;
            l = rand.Next() % 2 != 0;
        }
        if (l) r.Left = new Node(e);
        else r.Right = new Node(e);
    }
}
```

feltételes kifejezés

```
BinTree t = new();

Console.WriteLine("Give the elements of the tree: ");
int i;
while ( (i = int.Parse(Console.ReadLine())) != 0 )
{
    t.RandomInsert(i);
}
```

fa felépítése a
főprogramban

```
public class BinTree
{
```

```
...
```

```
public void PreOrder(IAction todo) { root.PreOrder(todo); }
```

```
public void InsOrder(IAction todo) { root.InOrder(todo); }
```

```
public void PostOrder(IAction todo){ root.PostOrder(todo); }
```

```
public interface IAction
{
    public void Exec(Node node);
}
```

BinFa

+ Üres() : void {query}

+ VéletlenBeszúr(int) : void

+ PreOrder(t:Tevékenység) : void

+ InOrder(t:Tevékenység) : void

+ PostOrder(t:Tevékenység) : void

return gyökér.PreOrder(t)

return gyökér.InOrder(t)

return gyökér.PostOrder(t)

<<interface>>

Tevékenység

+Végrehajt(Csúcs) : void



Konkrét tevékenység

+ Végrehajt(Csúcs) : void

- gyökér 0..1

Csúcs

- érték : int

+ IsLevél() : bool {virtual, query}

+ IsBelső() : bool {query}

+ PreOrder(t:Tevékenység) : void

+ InOrder(t:Tevékenység) : void

+ PostOrder(t:Tevékenység) : void

t.Végrehajt(this)

- bal 0..2

- jobb

```
public class Node
{
```

```
...
```

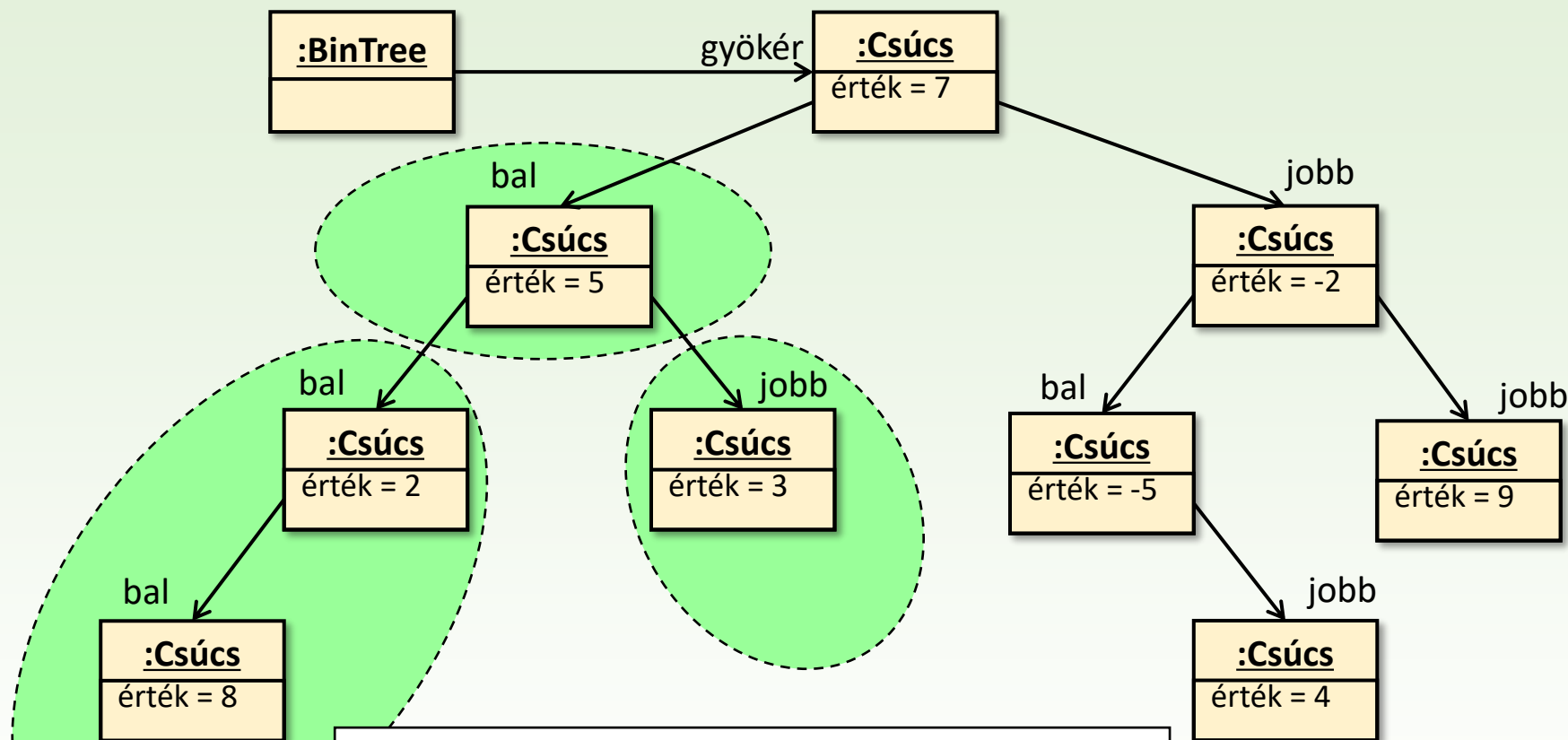
```
public void PreOrder(IAction todo) { ... }
```

```
public void InsOrder(IAction todo) { ... }
```

```
public void PostOrder(IAction todo){ ... }
```


preorder bejárás

7 5 2 8 3 -2 -5 4 9

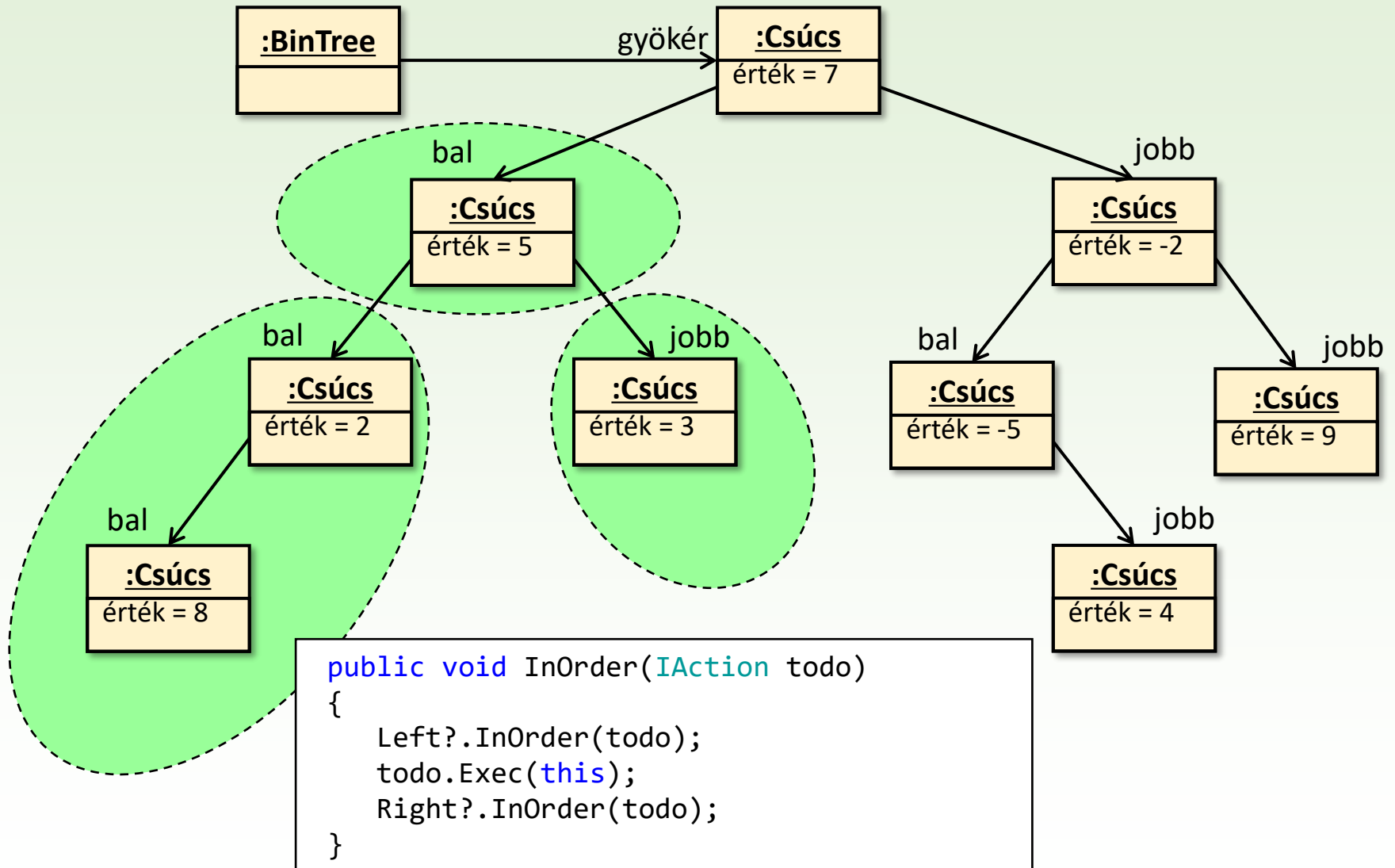


```
public void PreOrder(IAction todo)
{
    todo.Exec(this);
    Left?.PreOrder(todo);
    Right?.PreOrder(todo);
}
```

nem hívja meg, ha a Left értéke null
`if (Left!=null) Left.PreOrder(todo)`

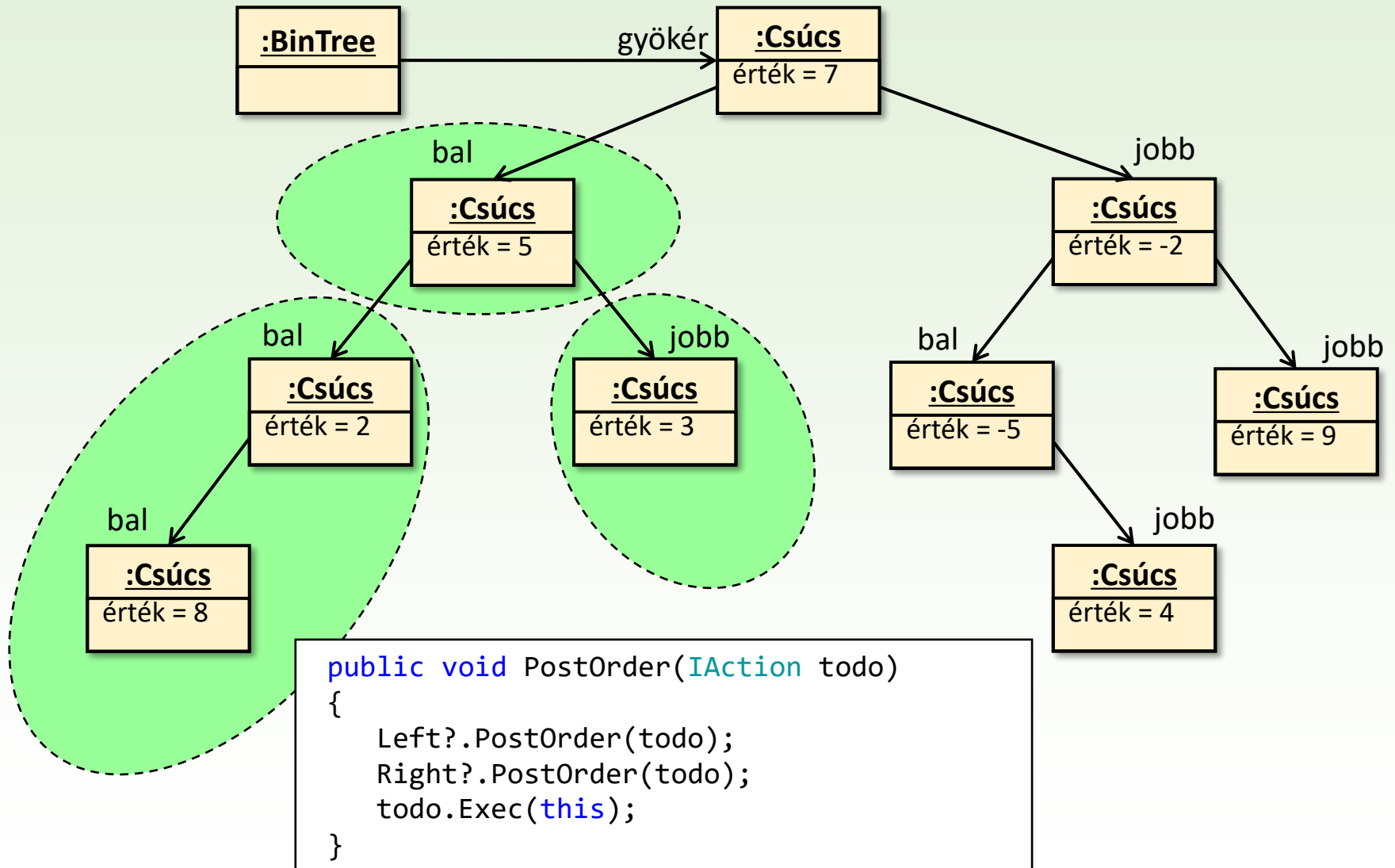
inorder bejárás

8 2 5 3 7 -5 4 -2 9



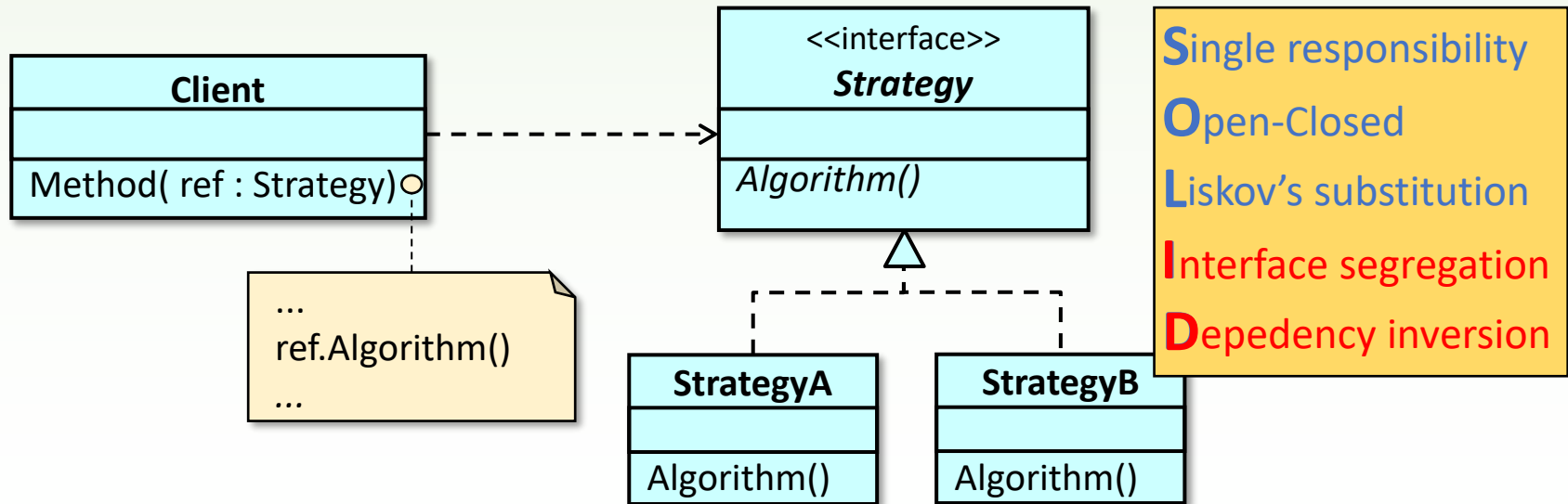
postorder bejárás

8 2 3 5 4 -5 9 -2 7



Stratégia (strategy) tervezési minta

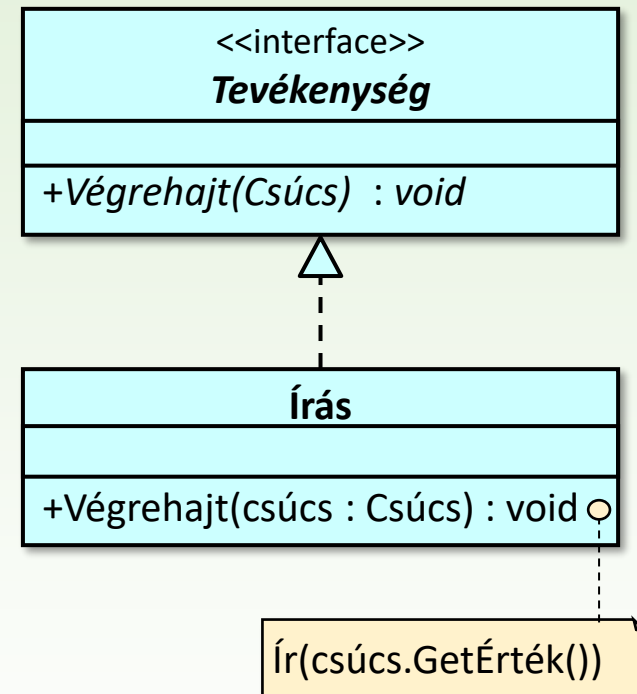
- Tevékenységek (algoritmusok, stratégiák) családját olyan osztályok metódusaként vezetünk be, amelyek egy közös interfészt valósítanak meg azért, hogy majd futási időben dőljön el, hogy mikor melyiket használja fel egy másik (kliens) osztály metódusa. Ezeket a tevékenységeket egy-egy (Strategy típusú) objektum hordozza a saját metódusaként, és ezt az objektumot kapja meg a kliens osztály metódusa paraméterként (vagy akár be is aggregálható a kliens objektumba).



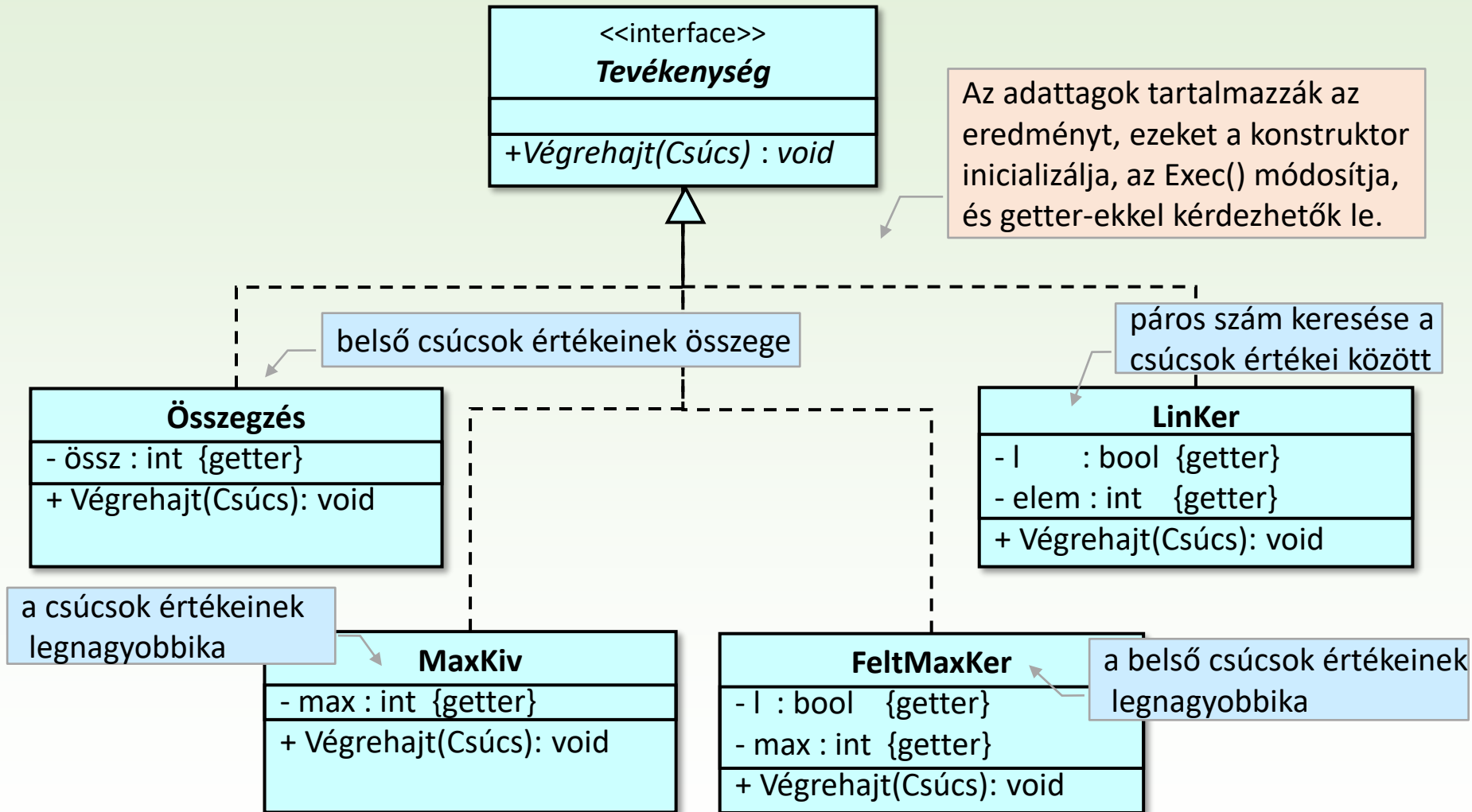
Kiírás tevékenység osztálysablonja

```
class Writer : IAction
{
    public void Exec(Node node)
    {
        Console.Write($" [{node.Value}] ");
    }
}
```

```
BinTree t = new();
...
Writer print = new();
Console.Write("\nPreorder traversal: ");
t.PreOrder(print);
Console.Write("\nInorder traversal: ");
t.InsOrder(print);
Console.Write("\nPostorder traversal:");
t.PostOrder(print);
Console.WriteLine();
```

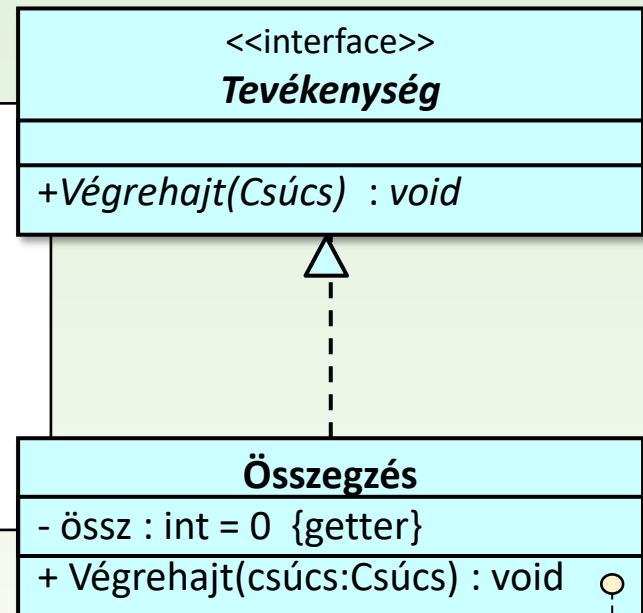


Egyéb tevékenység osztályok



Összegzés tevékenység osztálya

```
class Summation : IAction
{
    public int Sum { get; private set; }
    public Summation() { Sum = 0; }
    public void Exec(Node node)
    {
        if (node.IsInternal) Sum += node.Value;
    }
}
```

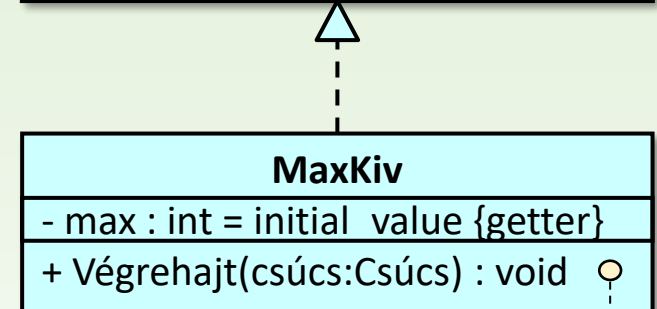
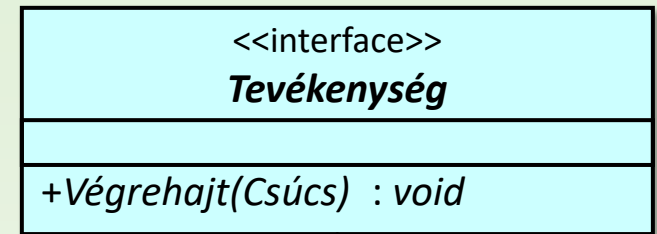


```
BinTree t = new();
...
Summation sum = new();
t.PreOrder(sum);
Console.WriteLine($"Sum of elements: {sum.Sum}");
```

```
if csúcs.IsBelső() then
    össz := össz + csúcs.GetÉrték()
endif
```

Maximum kiválasztás

```
class MaxSelect : IAction
{
    public int Max { get; private set; }
    public MaxSelect(int i) { Max = i; }
    public void Exec(Node node)
    {
        Max = Math.Max(Max, node.Value);
    }
}
```



```
BinTree t = new();
...
try
{
    MaxSelect max1 = new (t.Root);
    t.PreOrder(max1);
    Console.WriteLine($"\\nMaxima of elements: {max1.Max}");
}
catch (BinTree.NoRootException)
{
    Console.WriteLine("Empty tree.");
}
```

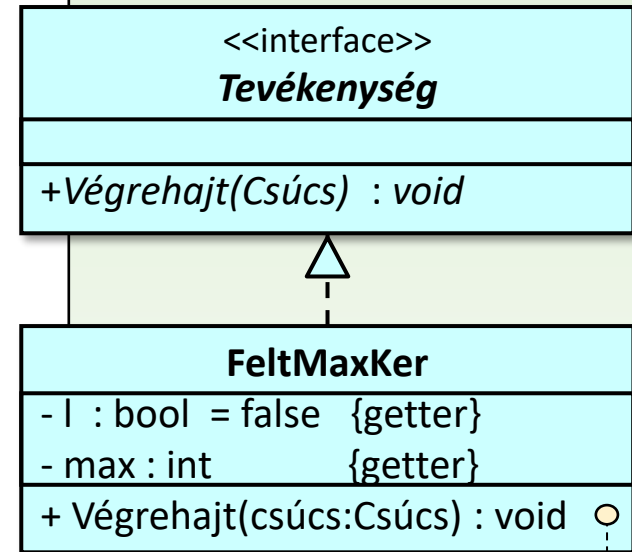
kivételt dob, ha a bináris fa üres

```
max := max(max, csúcs.GetÉrték())
```


Feltételes maximum keresés

```
class CondMaxSearch : IAction
{
    public bool Found { get; private set; }
    public int Max { get; private set; }

    public CondMaxSearch() { Found = false; }
    public void Exec(Node node)
    {
        if (node.IsInternal)
        {
            if (!Found) { Found = true; Max = node.Value; }
            else { Max = Math.Max(Max, node.Value); }
        }
    }
}
```



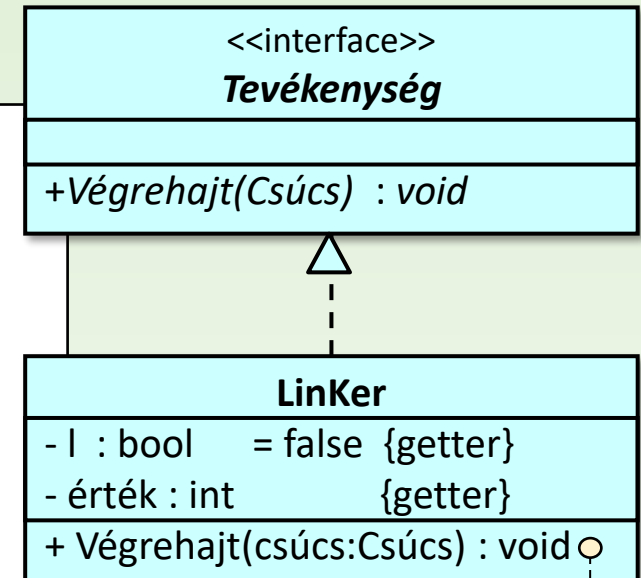
```
BinTree t;
...
CondMaxSearch max2 = new();
t.PreOrder(max2);
Console.WriteLine("\nMaxima of internal elements: ");
if (max2.Found) Console.WriteLine(max2.Max);
else Console.WriteLine("none");
```

```
if node.IsLevél() then return endif
if not l then
    l, max := true, csúcs.GetÉrték()
else max := max(max, csúcs.GetÉrték())
endif
```

Lineáris keresés

```
class Found : Exception { }  
class Search : IAction  
{  
    public int Value { get; private set; }  
    public void Exec(Node node)  
    {  
        if (node.Value % 2 == 0)  
        {  
            Value = node.Value;  
            throw new Found();  
        }  
    }  
}
```

```
BinTree t = new();  
...  
Search search = new();  
try {  
    t.PreOrder(search);  
    Console.WriteLine("\nNo even numbers");  
} catch (Found) {  
    Console.WriteLine($"First even number: {search.Value}");  
}
```



if csúcs.GetÉrték() mod 2 = 0 then
 l := true
 érték := csúcs.GetÉrték()
endif

Összetétel (composite) tervezési minta

- Objektumok fastruktúrába rendezéséhez ajánlják különösen akkor, ha a rész-egész (gyerek-szülő) viszonyokat kell kezelni. A módszer révén a levélcsúcsokat és a részfákat egységesen kezelhetjük.

